GUARDIAN

M0

M Extensions

Security Assessment

August 5th, 2025

Summary

Audit Firm Guardian

Prepared By Curiousapple, Wafflemakr, Cosine, Osman Özdemir, Vladimir Zotov

Client Firm M0

Final Report Date August 5th, 2025

Audit Summary

M0 engaged Guardian to review the security of their M Extensions. From the 23rd of June to the 5th of August, a team of 5 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the lack of critical issues detected and minimal code changes following the main review, Guardian assigns a Confidence Ranking of 4 to the protocol. Guardian advises the protocol to consider periodic review with future changes. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

🔗 Blockchain network: **Ethereum, Arbitrum, Optimism**

✅ Verify the authenticity of this report on Guardian’s GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite:
https://github.com/GuardianOrg/m-extensions-m0-m-extensions-team1

https://github.com/GuardianOrg/m-extensions-m0-m-extensions-team2

https://github.com/GuardianOrg/m-extensions-m0-m-extensions-fuzz

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. **Recommendation:** Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only Low or Medium-severity issues were discovered. Design patterns are sound, and test coverage is reasonable. Small changes, such as modifying rounding logic, may introduce new vulnerabilities and should be carefully reviewed. **Recommendation:** Suitable for deployment after remediations; consider periodic review with changes.	0 High/Critical findings. Varied Low/Medium severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. **Recommendation:** Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1 High finding and ≥ 3 Medium. Varied Low severity findings.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities (~2/week). Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. **Recommendation:** Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. **Recommendation:** Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Overview

Project Summary

Project Name	M0
Language	Solidity
Codebase	https://github.com/m0-foundation/m-extensions
Commit(s)	Initial commit: 66eb4710e73ce1ffcf198b593e9e81848f9385cc Final commit: 34717d18d3b4019226479d50cd9ad030ab3cdeb8

Audit Summary

Delivery Date	August 5, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0
● High	1	0	0	0	0	1
● Medium	4	0	0	2	0	2
● Low	33	0	0	21	0	12
● Info	0	0	0	0	0	0

Audit Scope & Methodology

```
Scope and details:
contract,source,total,comment
m-extensions/src/MExtension.sol,101,271,116
m-extensions/src/interfaces/IMExtension.sol,12,58,32
m-extensions/src/interfaces/IMTokenLike.sol,5,27,16
m-extensions/src/swap/SwapFacility.sol,131,280,91
m-extensions/src/swap/UniswapV3SwapAdapter.sol,122,218,57
m-extensions/src/libs/IndexingMath.sol,32,97,51
m-extensions/src/components/Blacklistable.sol,61,135,45
m-extensions/src/components/IBlacklistable.sol,8,51,33
m-extensions/src/projects/yieldToOne/IMYieldToOne.sol,9,41,21
m-extensions/src/projects/yieldToOne/MYieldToOne.sol,106,231,75
m-extensions/src/projects/yieldToAllWithFee/MSpokeYieldFee.sol,42,94,38
m-extensions/src/projects/yieldToAllWithFee/MYieldFee.sol,249,508,147
m-extensions/src/projects/earnerManager/IMEarnerManager.sol,16,79,45
m-extensions/src/projects/earnerManager/MEarnerManager.sol,214,441,136
m-extensions/src/swap/interfaces/IRegistrarLike.sol,3,13,7
m-extensions/src/swap/interfaces/ISwapFacility.sol,11,44,20
m-extensions/src/swap/interfaces/IUniswapV3SwapAdapter.sol,14,72,42
m-extensions/src/projects/yieldToAllWithFee/interfaces/IContinuousIndexing.sol,4,25,16
m-extensions/src/projects/yieldToAllWithFee/interfaces/IMSpokeYieldFee.sol,4,18,9
m-extensions/src/projects/yieldToAllWithFee/interfaces/IMYieldFee.sol,15,81,49
m-extensions/src/projects/yieldToAllWithFee/interfaces/IRateOracle.sol,3,15,9
m-extensions/src/swap/interfaces/uniswap/IUniswapV3SwapCallback.sol,3,16,12
m-extensions/src/swap/interfaces/uniswap/IV3SwapRouter.sol,35,62,21

source count: {
  total: 2877,
  source: 1200,
  comment: 1088,
  single: 333,
  block: 755,
  mixed: 0,
  empty: 590,
  todo: 0,
  blockEmpty: 1,
  commentToSourceRatio: 0.9066666666666666
}
```

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	● Critical	● High	● Medium
Likelihood: *Medium*	● High	● Medium	● Low
Likelihood: *Low*	● Medium	● Low	● Low

Impact

High	Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.
Medium	A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.
Low	Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High	The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.
Medium	An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.
Low	Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Invariants Assessed

During Guardian's review of M0, fuzz-testing was performed on the protocol's main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000,000+ runs with a prepared fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
MYF-01	MYieldFee extension mToken Balance must be greater or equal than projectedSupply	✅	❌	❌	10M+
MYF-02	MYieldFee extension mToken Balance must be greater or equal than projectedSupply + fee	✅	❌	❌	10M+
SWAP-01-00	YTO-TO-YTO: MYieldToOne yield must not change after swaps	✅	✅	✅	10M+
SWAP-01-01	YFEE-TO-YFEE: MYieldFee yield must not change after swaps	✅	✅	✅	10M+
SWAP-01-02	MEARN-TO-MEARN: MEarnerManager yield must not change after swaps	✅	✅	✅	10M+
SWAP-02	Swap facility M0 balance must be 0 after swap out	✅	✅	✅	10M+
SWAP-03	Total M0 balance of all users must not change after swap	✅	✅	✅	10M+
SWAP-04	Received amount of M0 must be greater or equal than slippage	✅	✅	✅	10M+
SWAP-05	Received amount of USDC must be greater or equal than slippage	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
MEARN-01	MEarnerManager extension mToken Balance must be greater or equal than projectedTotalSupply	✅	❌	✅	10M+
ERR-01	Unexpected Error	✅	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	MYieldFee: Yield Accrual	Rewards	● High	Resolved
M-01	MYieldFee : Extension May Drift, Causing Insolvency Risk	Rewards	● Medium	Acknowledged
M-02	Wrong Swap Recipient Set	Logical Error	● Medium	Resolved
M-03	MEarnerManager: Old Fee Recipient Address Not Reset	Logical Error	● Medium	Acknowledged
M-04	MEarnerManager: Extraction Of Fake Yield Via DeFi	Rewards	● Medium	Resolved
L-01	swapInToken Steals Tokens From User	Logical Error	● Low	Resolved
L-02	Contracts Can't Be Upgraded	Upgradeability	● Low	Resolved
L-03	Swap Facility Blocks Actions	DoS	● Low	Acknowledged
L-04	Incompatibility Of M Extensions	Logical Error	● Low	Acknowledged
L-05	Race Condition On Access Control List	Access Control	● Low	Acknowledged
L-06	Design For L2 Oracle Sequencer Resilience	Best Practices	● Low	Acknowledged
L-07	MEarnerManager: Missing Claim	Rewards	● Low	Resolved
L-08	MYieldFee: Claim Yield	Rewards	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-09	MSpokeYieldFee: Stepwise Jumps	Rewards	● Low	Acknowledged
L-10	Blacklisted Tokens Permanently Locked	Warning	● Low	Acknowledged
L-11	Token Path Validation Bypassed	Validation	● Low	Acknowledged
L-12	Entire Balance Swaps Between Extensions Fail	DoS	● Low	Acknowledged
L-13	MYieldFee: Earning Disabled	Unexpected Behavior	● Low	Resolved
L-14	MEarnerManager: Yield Fee Rounds Against The Protocol	Logical Error	● Low	Acknowledged
L-15	MYieldFee: Accruing Yield When Earning Stops	Logical Error	● Low	Acknowledged
L-16	Users Could Own M Tokens	Warning	● Low	Acknowledged
L-17	Excess Yield Can't Be Claimed	Logical Error	● Low	Acknowledged
L-18	Consider Using UniversalRouter For Swaps	Informational	● Low	Acknowledged
L-19	Unbacked Extension Balance Due To Rounding	Rounding	● Low	Acknowledged
L-20	Whitelist Check Limits Compatibility	Validation	● Low	Acknowledged
L-21	MExtension Implementations Can Be Initialized	Logical Error	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-22	Missing Fee Recipient Check In Blacklist	Validation	● Low	Acknowledged
L-23	MYieldOne: Claim Yield	Logical Error	● Low	Resolved
L-24	Interface feeRate Not Declared As View	Informational	● Low	Resolved
L-25	MSpokeYieldFee: Spoke Rate Update	Logical Error	● Low	Acknowledged
L-26	Stale balanceOf DoS	DoS	● Low	Acknowledged
L-27	Blacklisted Users Accrue Yield	Warning	● Low	Acknowledged
L-28	IMExtension Claim Function	Informational	● Low	Acknowledged
L-29	Incorrect Comments	Informational	● Low	Resolved
L-30	Incorrect Return Param Name	Informational	● Low	Resolved
L-31	Rounding Creates Insolvent Scenarios	Informational	● Low	Acknowledged
L-32	baseToken Fetched In Every Iteration	Gas Optimization	● Low	Resolved
L-33	MYieldFee:Unnecessary Index Update	Validation	● Low	Resolved

H-01 | MYieldFee : Yield Accrual

Category	Severity	Location	Status
Rewards	● High	MYieldFee.sol: 198-199	Resolved

Description PoC

The MYieldFee M Extension currently allows user balances to increase even when earning is disabled, because calling updateIndex resets _latestRate to a non-zero value—bypassing the isEarningEnabled() check in currentIndex.

As a result, claimYieldFor can inflate user balances even when the extension isn't receiving yield from the M token, leading to inaccurate accounting and potential insolvency.

While SwapFacility blocks further actions during such states, M Extensions are designed to function independently and integrate with DeFi—making this behavior risky.

Examples:

- Lending protocols: Users could borrow against artificially inflated balances not backed by real yield.
- DEXs: Balance-based logic (e.g., in minting or routing) could misbehave or be exploited.

Additionally, if an extension is disabled and later re-enabled, updateIndex would incorrectly assume yield accrued throughout the entire disabled period.

Recommendation

Update the updateIndex function to respect the earning status.

Consider adding

```
if (isEarningEnabled()) return $.latestIndex;
```

Resolution

M0 Team: Resolved.

M-01 | MYieldFee : Extension May Drift, Causing Insolvency Risk

Category	Severity	Location	Status
Rewards	● Medium	Global	Acknowledged

Description PoC

The MYieldFee extension mimics the yield accrual curve of the original M Token by independently tracking the earner rate and current index. However, the earner rate on the original M Token can be updated by MinterGateway.

1. MinterGateway updates collateral.
2. This triggers a call to mToken.updateIndex().
3. Inside mToken.updateIndex(), _latestRate is refreshed using rateModel().rate().
4. earnerRate() of M Token subsequently reflects this new _latestRate.

Until the extension's updateIndex() function is manually invoked, it will continue operating based on stale parameters. This can create a discrepancy between the extension's internal index and the actual yield curve of the M Token. Additionally, if mToken.stopEarning is triggered directly (without invoking the extension's logic), the extension keeps accruing "fake" yield.

This leads to minting of unbacked extension tokens. Over time, this mismatch can result in excess yield being distributed via the extension. If sustained long enough, and if the extension owner's fee reserves are insufficient to absorb the difference, it could theoretically lead to insolvency.

Recommendation

• Extension owners should be made explicitly aware of the need to call updateIndex() whenever the underlying earner rate is updated.
• Consider building a proactive alerting or notification mechanism to inform extension owners when a rate change is about to occur or has occurred.
• Alternatively, explore automating index updates as part of key interactions or lifecycle hooks to minimize reliance on manual upkeep.
• Ensure that extension.stopEarning() is invoked before or concurrently with mToken.stopEarning() to trigger any necessary extension-side logic (such as halting internal index updates).

Resolution

M0 Team: Acknowledged.

M-02 | Wrong Swap Recipient Set

Category	Severity	Location	Status
Logical Error	● Medium	UniswapV3SwapAdapter.sol: 94	Resolved

Description

If a path is given to the `swapIn` function in the `UniswapV3SwapAdapter` contract `msg.sender` is used as recipient instead of the given `recipient` parameter.

Recommendation

Use the given `recipient` parameter.

Resolution

M0 Team: Resolved.

M-03 | MEarnerManager: Old Fee Recipient Address Not Reset

Category	Severity	Location	Status
Logical Error	● Medium	MEarnerManager.sol: 338-351	Acknowledged

Description

In the MEarnerManager contract the fee recipient gets a fee rate of 0 (does not pay fees). When a new fee recipient is set the old fee recipient account is not reset therefore the old account still pays 0 fees.

This can lead to loss of yield if not noticed. For example if the fee recipient was changed due to the address being compromised or due the employee with this address left the company etc.

Recommendation

Reset the old fee recipient account when a new on is set.

Resolution

M0 Team: Acknowledged.

M-04 | MEarnerManager: Extraction Of Fake Yield Via DeFi

Category	Severity	Location	Status
Rewards	● Medium	MEarnerManager.sol: 169-170	Resolved

Description

Even if earning is disabled for `MEarnerManager`, it continues to accrue yield as per the `currentIndex`. Users will no longer be able to exit via the swap facility, since it restricts the use of non-earning tokens in extensions.

That said, extension tokens are expected to be used independently in DeFi. As a result, users may still exit through other protocols using their token balances, which now include yield accrued after earning was disabled—effectively allowing them to extract fake yield.

Recommendation

Consider implementing a snapshot of the index at the time earning is disabled to prevent post-disablement yield accrual from being misused.

Resolution

M0 Team: Resolved.

L-01 | swapInToken Steals Tokens From User

Category	Severity	Location	Status
Logical Error	● Low	SwapFacility.sol: 161	Resolved

Description

The swapInToken function is the main entry point for users to enter an extension. It allows users to trade external tokens (e.g., USDC) for extension tokens, and this core functionality is currently broken.

The flow differs slightly depending on whether the desired extension token is $wM or another extension token. For non-$wM tokens, the flow looks like this:

• The user calls the swapInToken function.
• The external tokens (e.g., USDC) are transferred into the contract.
• These tokens are swapped for $wM tokens on Uniswap, with the SwapFacility contract—not the user—set as the recipient.
• The _swap function is then called to convert the received $wM tokens into the desired extension token using an unwrap/wrap flow that assumes the tokens belong to the user.

The issue is that _swap attempts to use the user's $wM tokens, even though $wM tokens from the initial swap were sent to the SwapFacility contract. This leads to two potential problems:

• The call will revert (DoS) if the user doesn't already hold enough $wM.
• Or, it will steal $wM from the user—causing them to effectively pay twice (e.g., swapping $100 USDC results in ~$200 worth of user tokens being deducted).

The M0 team later clarified that this behavior was designed to support the current deployment of $wM, which uses msg.sender in the unwrap function and doesn't account for SwapFacility as the receiver. They intend to update the $wM implementation in the future to MExtension Interface. Above issue is only applicable consider wM to be one of MExtensions.

Recommendation

Be aware of this issue when developing the new $wM implementation.

Resolution

M0 Team: Resolved.

L-02 | Contracts Can't Be Upgraded

Category	Severity	Location	Status
Upgradeability	● Low	Global	Resolved

Description

As observed in tests and confirmed in discussions with the M0 team, UUPS proxies are intended to be used for upgrades. However, UUPS proxies rely on `ERC1967Proxy` and are not upgradeable by default.

The implementation contract must include upgrade logic and explicitly authorize upgrades by inheriting from `OpenZeppelin`'s `UUPSUpgradeable` and implementing the necessary authorization functions.

Currently, neither the Extensions nor the `SwapFacility` contracts implement this logic. As a result, despite using a UUPS proxy structure, these contracts are not actually upgradeable.

Recommendation

To enable proper upgradeability, have the implementation contracts inherit from `OpenZeppelin`'s `UUPSUpgradeable.sol` and implement the required upgrade authorization functions.

Reference: OpenZeppelin UUPSUpgradeable Docs

Resolution

M0 Team: Resolved.

L-03 | Swap Facility Blocks Actions

Category	Severity	Location	Status
DoS	● Low	SwapFacility.sol: 287-288	Acknowledged

Description

All M Extensions include a toggle to enable or disable yield earning. Any user interaction such as wrapping, unwrapping, or swapping must go through the `SwapFacility`.

However, the `SwapFacility` restricts actions via `_revertIfNotApprovedExtension`, which only permits extensions that are currently earning.

This effectively blocks all interactions with an M Extension once earning is disabled, even though the extension may still serve utility beyond yield accrual. (Swap with wM/USDC, Unwrap, Wrap)

Recommendation

Consider maintaining a dedicated list of approved extensions within `SwapFacility` that is independent of the active earners list. This allows for greater flexibility and separation of concerns between approval and earning status.

Resolution

M0 Team: Acknowledged.

L-04 | Incompatibility Of M Extensions

Category	Severity	Location	Status
Logical Error	● Low	MExtension.sol: 31-35	Acknowledged

Description

As per M0's documentation, M Extensions are intended to act as wrappers that do not rebase, i.e., their `balanceOf()` should remain constant unless explicitly changed via user actions. However, due to the nature of continuous yield distribution, balances increase over time when `claimFor(address)` is called. While this isn't technically rebasing (since it requires explicit action), the end result — a growing balance over time — introduces similar challenges. M Extensions are designed to be used independently across DeFi, but this yield-accruing behavior causes incompatibilities with many DeFi protocols, which generally assume that a token's balance is static unless changed by transfer/mint/burn operations.

Perpetual Protocols (e.g., GMX)
• Unaccounted Balance as Deposit: Call `claimFor(address)` and then `createDeposit()`
GMX Deposit Logic
DEXs (e.g., Uniswap)
• Unaccounted Balance as Deposit: Call `claimFor()` within the callback of a `mint()` operation.
Uniswap V3 Pool Mint Logic
Lending Protocols (e.g., Aave)
• If `claimFor()` is called:
• The collateral balance silently increases.
• The protocol may overstate collateral value, or miscalculate interest.
• See Aave–Lido integration spec for a similar class of issues with `stETH` and the design changes needed to accommodate it.
Vaults (e.g., Yearn)
• Unaccounted Balance as Deposit:Attack Vector: Call `claimFor()` just before `deposit()` into a vault.
Yearn Deposit Logic

We later understood that the M0 team has accounted for this issue and introduced a mitigation via the permissioned `claimRecipient` feature, which allows designated role holders to set custom recipients for DeFi pools. While this does help address many of the risks, it comes with certain limitations:
• It introduces protocol-specific setup, requiring the M0 team to manually configure a different recipient for each new DeFi pool that integrates M Extensions.
• It places the burden of yield redistribution on the integrating protocol, which may necessitate non-trivial contract modifications—such as tracking user entries and exits to ensure fair distribution.
• This setup introduces a layer of centralization and operational friction, as each integration becomes dependent on coordination with the M0 team.

Recommendation

To ensure smoother and safer integration across DeFi:
1. Consider wrapping M Extensions in a token that behaves like `wstETH` or `cTokens`, where:
• `balanceOf()` is constant
• Yield is reflected via an `exchangeRate()` model
• All rebasing or claiming is internalized and deterministic
2. Allow self-directed recipient configuration:
• Let users (e.g., DeFi pool creators) set their own yield recipient without needing M0's intervention.
• This reduces bottlenecks and central dependency.
3. Proactively document integration considerations:
• Highlight the `claimFor` behavior clearly.
• Offer best practices for protocols to adapt (e.g., use wrapper, handle rebasing-like logic off-chain).

Resolution

M0 Team: Acknowledged.

L-05 | Race Condition On Access Control List

Category	Severity	Location	Status
Access Control	● Low	Global	Acknowledged

Description

M0 intends to implement blacklist/whitelist controls in its extensions, but inclusion or exclusion transactions can be frontrun by sophisticated attackers.

They might consider DeFi pools as an escape hatch—for example, attackers have repeatedly used Curve's 3-pool to circumvent a USDT blacklist.

Recommendation

Be aware of these scenarios and recommend frontrun-resistant RPC endpoints for extension-owner transactions that modify whitelist/blacklist status.

Resolution

M0 Team: Acknowledged.

L-06 | Design For L2 Oracle Sequencer Resilience

Category	Severity	Location	Status
Best Practices	● Low	Global	Acknowledged

Description

The L2 rate oracle hasn’t been implemented yet. Because L2 updates will occur in discrete jumps, sequencer uptime will be critical to ensure timely and accurate rate feeds once the oracle goes live.

Recommendation

Factor in sequencer availability when building the L2 rate oracle and design retry or fallback logic to mitigate downtime.

Resolution

M0 Team: Acknowledged.

L-07 | MEarnerManager: Missing Claim

Category	Severity	Location	Status
Rewards	● Low	MEarnerManager.sol: 315-316	Resolved

Description

In MEarnerManager, if a user is removed from the whitelist and then re-added without an intermediate claimFor call, they will retroactively accrue yield for the period during which they were not whitelisted.

Recommendation

Before returning early on a whitelist-status change, invoke an internal claimFor to settle any owed yield for the user. (100% fee)

Resolution

M0 Team: Resolved.

L-08 | MYieldFee: Claim Yield

Category	Severity	Location	Status
Rewards	● Low	MYieldFee.sol: 232-233	Resolved

Description

When setting a new claim recipient in MYieldFee, the yield accrued to date is not claimed for the current recipient—instead, all that yield immediately goes to the new recipient. (M0 has commented out this behavior and marked it optional.)

Recommendation

Revisit this behavior and, if it wasn't intentional, insert a claim for the existing recipient before updating to the new one.

Resolution

M0 Team: Resolved.

L-09 | MSpokeYieldFee: Stepwise Jumps

Category	Severity	Location	Status
Rewards	● Low	MSpokeYieldFee.sol	Acknowledged

Description

MSpokeYieldFee uses a stepwise accrual curve rather than the continuous curve of its L1 counterpart.

As a result, if the oracle rate moves from point A to B, yield will accrue for the entire A–B interval—even if, in reality, only a portion of that time should count.

Recommendation

Be aware of this stepwise behavior; as long as rates remain low and the oracle updates frequently, the risk should be minimal.

Resolution

M0 Team: Acknowledged.

L-10 | Blacklisted Tokens Permanently Locked

Category	Severity	Location	Status
Warning	● Low	Global	Acknowledged

Description

Protocol has a blacklist feature; however, once tokens are blacklisted there is no way for the extension owner or admin to seize those assets, effectively locking them in perpetuity.

Recommendation

Consider adding an option for the admin or extension owner to seize blacklisted assets.

Resolution

M0 Team: Acknowledged.

L-11 | Token Path Validation Bypassed

Category	Severity	Location	Status
Validation	● Low	UniswapV3SwapAdapter.sol: 214-215	Acknowledged

Description

Although M0 checks that the input and output tokens of a swap belong to the whitelist, an on-chain path could route through arbitrary intermediary tokens—circumventing those checks entirely.

Recommendation

Beware of this scenario, and if not intentional consider adding validations for all tokens included in path.

Resolution

M0 Team: Acknowledged.

L-12 | Entire Balance Swaps Between Extensions Fail

Category	Severity	Location	Status
DoS	● Low	MExtension.sol: 226	Acknowledged

Description

M0 roundings during transfers are always in favor of the protocol. However, this can cause a revert due to insufficient balance in edge-case scenarios where the entire balance needs to be swapped, such as during migrations between extensions, potentially affecting the last swapper.

During the first step of the swap, `mTokens` need to be transferred from `extensionIn` to the `swapFacility`. Since `extensionIn` is an earner but the `swapFacility` is non-earner, this transfer requires rounding up.

When the last user tries to swap their entire balance, the rounded-up `mToken` amount to transfer becomes greater than the `mToken` balance of the extension, causing the swap to fail.

It is expected that a few extra wei will be deducted from the extension balance during `_unwrap`. However, unexpected reverts should be prevented when the entire balance is being swapped.

Recommendation

Document this behavior and inform users about the implications of swapping their entire balance.

Resolution

M0 Team: Acknowledged.

L-13 | MYieldFee: Earning Disabled

Category	Severity	Location	Status
Unexpected Behavior	● Low	MYieldFee.sol: 294	Resolved

Description

The MYieldFee extensions allows partners to set a fee for the yield generated on $M holdings. This fee can be update by the FEE_MANAGER_ROLE, with values ranging from 0 to 100%.

If the fee rate is set to 100%, the updateIndex will set the latestRate to 0, as the new earnerRate is 100 - 100 = 0. Therefore, the isEarningEnabled function will return false.

This is an unexpected behavior of the isEarningEnabled, as the $M tokens are still generating yield for the fee recipient, just not for the users. Worth comparing this behavior to the result of disableEarning, which will set both the latestRate to zero and effectively stop earning yield in$ M.

Additionally, the currentIndex calls will always early return with the latestIndex so yield will never accrue even if the $M latestUpdateTimestamp changes.

Recommendation

Consider updating the isEarningEnabled function to return:

IMTokenLike(mToken()).isEarning(address(this)).

Alternatively, consider not allowing fee rate to be set to 100%.

Resolution

M0 Team: Resolved.

L-14 | MEarnerManager: Yield Fee Rounds Against The Protocol

Category	Severity	Location	Status
Logical Error	● Low	MEarnerManager.sol: 175	Acknowledged

Description

In the `accruedYieldAndFeeOf` function, the fee is calculated as `fee = (yieldWithFee * feeRate_) / ONE_HUNDRED_PERCENT`, which rounds down in favor of the user and against the protocol. This also results in a zero fee when claims are made with small amounts.

Recommendation

Round up the fee calculation in the `accruedYieldAndFeeOf` function.

Resolution

M0 Team: Acknowledged.

L-15 | MYieldFee: Accruing Yield When Earning Stops

Category	Severity	Location	Status
Logical Error	● Low	MYieldFee.sol: 175	Acknowledged

Description

When an extension is removed from the earner's list, any user can call the permissionless mToken.stopEarning function with the extension's address, instead of extension.stopEarning.

Regarding the MYieldFee extensions, this is an issue as it calculates its own index based on the time delta and rate. Therefore, users will be earning "fake" yield, minting unbacked extension tokens.

Although the swapFacility won't allow users to unwrap once earning has been disabled in the registrar, the extension token might be added to a Uniswap Pool, so users may exit here, while LPs are stuck with funds they can never unwrap.

Recommendation

Make sure the extension.stopEarning is executed at the exact same time the $M earnings are disabled.

Resolution

M0 Team: Acknowledged.

L-16 | Users Could Own M Tokens

Category	Severity	Location	Status
Warning	● Low	Global	Acknowledged

Description

The SwapFacility contract tries to prevent any normal user from receiving $M tokens. This invariant can be broken by malicious extensions.

Even if fine in the first place an extension could upgrade it's code to be able to send out $M tokens to regular users as there is nothing blocking it.

Recommendation

Consider creating a mapping of addresses which are allowed to hold $M tokens to make sure this invariant holds.

Resolution

M0 Team: Acknowledged.

L-17 | Excess Yield Can't Be Claimed

Category	Severity	Location	Status
Logical Error	● Low	MEarnerManager.sol	Acknowledged

Description

Extensions can accumulate excess yield ($M balance greater than projected supply) due to different factors, like rounding, donations, or rate updates, specially in L2.

Both the `MYieldOne` and `MYieldFee` rely on the $M balance of the contract. However, the `MEarnerManager` does not rely on the token balance but principal and current index calculations.

The only way to claim fees is using `claimFor`, that claims both the user's yield and protocol's fee. Therefore, owner can't claim any excess yield accumulated in the extension.

Recommendation

Consider adding a `claimExcess` manager function to claim these yields.

Resolution

M0 Team: Acknowledged.

L-18 | Consider Using UniversalRouter For Swaps

Category	Severity	Location	Status
Informational	● Low	UniswapV3SwapAdapter.sol	Acknowledged

Description

The UniswapV3SwapAdapter contract creates swap parameters compatible with SwapRouter02, which is different than UniV3 SwapRouter and does not include a deadline parameter in ExactInputSingleParams and ExactInputParams.

However, the official Uniswap documentation recommends using the UniversalRouter instead (reference):

The UniversalRouter contract is the current preferred entrypoint for ERC20 and NFT swaps, replacing, among other contracts, SwapRouter02. An up-to-date list of deploy addresses by chain is hosted on GitHub.

Recommendation

Consider using UniversalRouter instead of SwapRouter02.

Resolution

M0 Team: Acknowledged.

L-19 | Unbacked Extension Balance Due To Rounding

Category	Severity	Location	Status
Rounding	● Low	MYieldFee.sol: 441	Acknowledged

Description PoC

The MYieldFee and MEarnerManager extensions implement dual accounting, tracking both balance and principal. When transferring tokens between accounts or burning them, the principal subtracted from the sender is slightly overestimated. The recipient also receives this overestimated principal amount.

These extensions use getSafePrincipalAmountRoundedUp to perform this overestimation. However, this can result in an account having a non-zero balance but a zero principal, especially after transfers involving very small amounts. Additionally, the transfer and burn functionality does not restrict zero-principal transfers.

These non-zero balances can be moved between accounts even when the actual principal transferred is zero. More importantly, they can be used in swaps to another extension. During such swaps, a non-zero balance with zero principal is burned on extensionIn, reducing the mBalance of extensionIn, while both a non-zero balance and a non-zero principal are minted on extensionOut.

While we couldn't identify a large amount of profit, it is possible to hold balances without any backing principal and use those balances to mint additional principal out of thin air. This introduces an insolvency risk in edge-case scenarios where all users attempt to claim their yields.

Recommendation

One option to consider is disallowing any transfer or burn when the transferred principal is zero. This would prevent swapping a balance with zero principal to another extension. However, this alone does not prevent accounts from holding a non-zero balance with zero principal. Additionally, consider clearing an account's balance when the return value of getSafePrincipalAmountRoundedUp equals the sender's principal balance.

Another option is to use getPrincipalAmountRoundedUp and revert if the principal balance is insufficient, similar to the underlying M0 behavior. However, this approach may introduce unexpected reverts during wrap or unwrap operations.

Resolution

M0 Team: Acknowledged.

L-20 | Whitelist Check Limits Compatibility

Category	Severity	Location	Status
Validation	● Low	MEarnerManager.sol: 279	Acknowledged

Description

The _beforeTransfer function of the MEarnerManager contract does not only revert if the sender or recipient is not whitelisted, but also if the executor is not whitelisted.

This limits compatibility with many DeFi protocols and may prevents things like gasless transactions, multisig transactions, etc.

Recommendation

Beware of this behavior, and revisit if not intended.

Resolution

M0 Team: Acknowledged.

L-21 | MExtension Implementations Can Be Initialized

Category	Severity	Location	Status
Logical Error	● Low	MExtension.sol	Resolved

Description

The swapFacility is an upgradeable contract and its implementation correctly calls _disableInitializers in the constructor.

However, MExtensions will also be upgradeable, but there is no _disableInitializers to prevent someone initializing the implementation.

Although no potential harm its evident at the contract level, these extensions could be widely known, and their contracts should not be manipulated in any way, as they may lose reputation.

Recommendation

Consider adding _disableInitializers in the constructor of MExtension.

Resolution

M0 Team: Resolved.

L-22 | Missing Fee Recipient Check In Blacklist

Category	Severity	Location	Status
Validation	● Low	MYieldToOne.sol: 230	Acknowledged

Description

The MYieldToOne manager can change the fee recipient using the setYieldRecipient function. This function reverts if yieldRecipient_ = address(0) but does not check if this address is blacklisted.

Recommendation

Validate if yieldRecipient address is blacklisted before updating the recipient.

Resolution

M0 Team: Acknowledged.

L-23 | MYieldOne: Claim Yield

Category	Severity	Location	Status
Logical Error	● Low	MYieldToOne.sol: 94	Resolved

Description

The setYieldRecipient function in the MYieldToOne contract changes the recipient without first claiming the yield for the previous recipient.

As a result, all accumulated yield is attributed to the new recipient, even though it belongs to the previous one.

Recommendation

Consider claiming the yield before changing the recipient, similar to how it's handled when changing the fee recipient in the MYieldFee contract.

Resolution

M0 Team: Resolved.

L-24 | Interface feeRate Not Declared As View

Category	Severity	Location	Status
Informational	● Low	IMYieldFee.sol: 164	Resolved

Description

The IMYieldFee interface declares the feeRate function, but is not declared as view. Using this interface in a contract function declared as view will throw compilation errors.

Recommendation

Declare feeRate function as view in IMYieldFee.sol.

Resolution

M0 Team: Resolved.

L-25 | MSpokeYieldFee: Spoke Rate Update

Category	Severity	Location	Status
Logical Error	● Low	MYieldFee.sol: 190	Acknowledged

Description

The MSpokeYieldFee is an extension that will be deployed in L2s. Therefore, it relies on the latestUpdateTimestamp of $M contract in L2 (which is propagated every 1-2 hours) and earnerRate of an oracle contract, to compute its own extension index. The $M index is propagation to L2 takes about 15 minutes.

This delay will cause some issues when the earner rate is updated. Consider the following, where t is in minutes:
• t=0 $M index was propagated (current rate 4%)
• t=100 Earner rate increased to 5% (also increased in L2 RateOracle)
• t=100 Index propagated through Wormhole starts
• t =115 $M index is updated on L2
• t=115 call extension's updateIndex(). This will increase the current index, using 5% from [0,115] , while the L1 real yield was 4% from [0,100].

Therefore, the $M real yield will increase less than the yield calculated on the extension, creating unbacked tokens or affecting the protocol's fees.

Even if the RateOracle is updated after the index is propagated, there will always be some accounting issues depending if the rate increase or decreases, the time it takes to propagate $M index, or the time where updateIndex is finally executed.

Recommendation

Current design prevents yield accruing to be completely in sync due to the Wormhole propagation delay. However, to avoid insolvency in extensions when rate increases, make sure $M index is propagated first, and update the RateOracle rate once the $M index updates in L2.

Resolution

M0 Team: Acknowledged.

L-26 | Stale balanceOf DoS

Category	Severity	Location	Status
DoS	● Low	Global	Acknowledged

Description

• The ERC20 functions balanceOf & totalSupply in the MEarnerManager & MYieldFee contract are usually stale as the correct values are only returned right after a yield claim.
• The ERC20 and wrap/unwrap flows in the MExtension contract use the balanceOf function to check if the user owns enough funds to perform the wished action and revert otherwise.
This can lead to DoS and makes it hard to unwrap all tokens (dust is most likely lost). For example:
• User once wrapped 100 tokens and now owns 110 tokens at the current yield index
• The user wants to unwrap all tokens and therefore calls the unwrap function with 110 tokens
• The call reverts with an InsufficientBalance error as the system uses the balanceOf function instead of the balanceWithYieldOf function and therefore thinks the user only owns 100 tokens
• Therefore the user needs to claim the yield and call unwrap again and now the call goes through but in the meantime more yield was accrued and a dust amt is left in the extension

When users unwraps or transfers their current balance without claiming yield before it is even possible to reach a 0 balance and positive principal state.

This can lead to users likely miss that they still own some stablecoins as their wallet will read 0 balanceOf and therefore not show the token anymore.

Recommendation

Always execute the normal yield claiming function for the given users before performing any action with them.

Also consider allowing users to use their total balance for a operation by supplying type(uint256).max for example.

Resolution

M0 Team: Acknowledged.

L-27 | Blacklisted Users Accrue Yield

Category	Severity	Location	Status
Warning	● Low	Global	Acknowledged

Description

Blacklisted users continue to accrue yield.

Recommendation

Consider to change that if this behavior is not intended.

Resolution

M0 Team: Acknowledged.

L-28 | IMExtension Claim Function

Category	Severity	Location	Status
Informational	● Low	IMExtension.sol	Acknowledged

Description

IMExtension should have a common claim function, overridden by each extension (either to revert if claim is not available like in MYieldToOne or claiming for account). This will avoid different claim signatures like claimFor and claimYieldFor.

Recommendation

Consider creating a common claim function in IMExtension.

Resolution

M0 Team: Acknowledged.

L-29 | Incorrect Comments

Category	Severity	Location	Status
Informational	● Low	Global	Resolved

Description

• swapOutM(): mentions "exntesiom" in the comments
• _revertIfInsufficientBalance : says that it reverts if account balance is below balance, but it should be amount.

Recommendation

Fix the incorrect comments.

Resolution

M0 Team: Resolved.

L-30 | Incorrect Return Param Name

Category	Severity	Location	Status
Informational	● Low	ISwapFacility.sol: 143	Resolved

Description

In the `NatSpec` for the `ISwapFacility .swapAdapter()` function, the `@return` parameter name is incorrect:

```
function swapAdapter() external view returns (address registrar);
```

Recommendation

Update return param name from `registrar` to `swapAdapter`

Resolution

M0 Team: Resolved.

L-31 | Rounding Creates Insolvent Scenarios

Category	Severity	Location	Status
Informational	● Low	Global	Acknowledged

Description

In order to remain solvent, the $M balance should always be greater or equal to the extension's total supply (or projected total supply), so users are able to withdraw all funds, including recipient fees.

However, when minting/wrapping, $M is transferred from the swap facility (non-earner) to the extension (earner).

Depending on the amount and current index, this can yield to the extension receiving 1-2 wei less $M than expected, but the full amount of extension tokens are minted to the user.

Due to the fact that the fees are calculated based on `$M balance - extension token supply`, this will affect extension owners directly.

Recommendation

Document this behavior so extension owners are aware of this fee reduction.

Resolution

M0 Team: Acknowledged.

L-32 | baseToken Fetched In Every Iteration

Category	Severity	Location	Status
Gas Optimization	● Low	SwapFacility.sol	Resolved

Description

Both swapInToken and swapOutToken fetches baseToken from the swapAdapter. The baseToken is an immutable param in the adapter, while the swapAdapter address is immutable in the swapFacility. These calls are unnecessary and increase the transaction gas costs.

Recommendation

Consider making baseToken immutable also in swap facility and avoid gas spent to fetch this address.

Resolution

M0 Team: Resolved.

L-33 | MYieldFee:Unnecessary Index Update

Category	Severity	Location	Status
Validation	● Low	MYieldFee.sol: 194	Resolved

Description

The `MYieldFee.updateIndex()` will update the index as long as the `block.timestamp` or `earnerRate` changed since the last update.

Although this seems correct for L1 deployments, L2 deployments will not work as expected. This is due to the fact that the `_latestEarnerRateAccrualTimestamp` on L2 is the `latestUpdateTimestamp` of $M, but `block.timestamp` on L1.

Therefore, every time the `updateIndex` is called on L2, the same variables will be assigned in storage, emitting the same `IndexUpdated` params.

Recommendation

Compare the `latestUpdateTimestamp` with the `_latestEarnerRateAccrualTimestamp` instead:

`if ($.latestUpdateTimestamp = _latestEarnerRateAccrualTimestamp() $.latestRate = rate_) return $.latestIndex;`

Resolution

M0 Team: Resolved.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits