OpenZeppelin | security

Origin ARM Audit

June 13, 2025

Table of Contents

Summary

Type	DeFi
Timeline	From 2025-05-13 To 2025-05-19
Languages	Solidity

Total Issues	24 (6 resolved, 4 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	1 (0 resolved, 1 partially resolved)
Low Severity Issues	7 (1 resolved)
Notes & Additional Information	15 (4 resolved, 3 partially resolved)

Scope

OpenZeppelin audited the OriginProtocol/arm-oeth repository at commit 700fa99 commit.

In scope were the following files:

```
src/contracts
├── markets
│   └── SiloMarket.sol
├── AbstractARM.sol
├── OriginARM.sol
├── ZapperARM.sol
└── SonicHarvester.sol
```

System Overview

The Automated Redemption Manager (ARM) is a protocol designed to manage liquidity and redemptions between Origin's yield-bearing assets and their liquid-wrapped counterparts. It operates across networks like Ethereum (e.g., OETH and WETH) and Sonic (e.g., Origin Sonic and Wrapped Sonic).

At the core of the ARM contract are two key tokens:

- **Base Asset**: A rebasing yield-bearing token, such as OETH or Origin Sonic.
- **Liquidity Asset**: A wrapped, non-rebasing token that is easier to use and trade, like WETH or Wrapped Sonic.

Users interact with the ARM in two primary ways:

- **Shares**: Users can deposit and redeem the **liquidity asset** to mint or burn ARM shares.
- **Swaps**: Users can also swap between the **base asset** and **liquidity asset** in either direction. This allows users to redeem base assets without waiting for vault delays or to acquire them at discounted rates, depending on direction.

Revenue Sources

ARM earns protocol-level profits through:

1. **Swap Pricing Advantage**: The swap mechanism is configured to always benefit the protocol by buying base assets at a discount and selling at a premium.
2. **Rebasing Yield**: Base asset holdings grow automatically as Origin assets accrue yield.
3. **Lending Market Allocation**: Excess liquidity can be deployed into whitelisted lending markets (e.g., Silo), dynamically chosen by the operator for optimal returns. Rewards accrued from these markets are collected by the Harvester and swapped into the liquidity asset, further increasing protocol revenue.

Profits are partially captured through a **performance fee**, which is forwarded to a fee recipient.

Auxiliary Components

Several contracts extend the ARM contract's capabilities:

- `SiloMarket`: A helper contract that abstracts interactions with Silo Finance's ERC-4626 markets, allowing ARM to deposit and withdraw while allowing the Harvester to claim rewards.
- `SonicHarvester`: Collects strategy rewards (e.g., from Silo) and swaps them into the liquidity asset using Magpie. It enforces slippage limits and can consult an oracle for price verification.
- `ZapperARM`: Lets users wrap native tokens (e.g., ETH or Sonic) and deposit into ARM in one step.

The ARM system maintains protocol efficiency and capital flexibility while enabling users to interact with Origin assets in a permissionless, yield-accruing environment.

Security Model and Trust Assumptions

The following considerations outline the protocol's security model and operational assumptions:

- Liquidity providers may attempt just-in-time (JIT) deposits ahead of profitable swaps or other actions that increase total protocol assets. This behavior is considered an acceptable risk, as the enforced claim delay ensures a minimum lock period that mitigates the potential for immediate profit extraction.
- When integrating with lending markets like `SiloMarket`, the underlying market must not fail silently. If it does, funds can become stuck in the intermediary contract.
- The `owner` should be a timelock contract, and it should set an `operator` to manage day-to-day actions for protocol profitability. The community must monitor its activity to ensure correct and profitable actions.
- The deployer that initializes the contracts must have sufficient liquidity and provide the correct configuration of `baseAsset` (e.g., Origin token) and `liquidityAsset` (e.g., wrapped token).
- The lending market must prevent share inflation, as it could lead to manipulation of accounting or potential read-only reentrancy attacks.

- If the underlying market is paused, `maxWithdraw` should return zero to prevent unexpected behavior during pause conditions.
- Total assets in the protocol should be ever-increasing. This should be managed by setting the correct parameters in the protocol to ensure profitability.
- The oracle used by the Harvester for benchmarking Magpie swap prices must be resistant to manipulation by the operator.
- If the Harvester's liquidity token is Fee-on-Transfer or Rebasing, it requires additional safeguards to ensure correct slippage validation using post-swap balance checks.
- The protocol assumes that tokens used will correctly revert on transfer failures. It does not check return values from `transfer` and `transferFrom`, relying on the token contract to bubble up errors.
- The current initialization logic is designed for `OriginARM`, which is intended to be newly deployed. If an existing ARM instance is to be upgraded to the new version, a separate initialization function is required. For example, an upgrading ARM does not require an initial deposit amount during setup. This assumption is accepted given that upgrades are expected to be managed appropriately.

Privileged Roles

Roles in the system that have privileged access to certain functionalities are listed below:

- **Owner**: A timelock contract that governs critical configuration like fees and market whitelisting.
- **Operator**: A semi-trusted multisig responsible for active operations, including redemption requests, managing market allocation, and interacting with vaults.

High Severity

H-01 The `_gap` Variable Is Incorrect

The `_gap` variable in `AbstractARM` is intended to preserve storage layout across contract upgrades by reducing its size for each new variable added. However, while three new variables (`activeMarket`, `supportedMarkets`, and `armBuffer`) have been introduced, the `_gap` array was only reduced from 41 to 39 instead of 38. This mismatch can lead to storage collisions in contracts that inherit from `AbstractARM`, potentially breaking functionality in already deployed instances.

Consider reducing the `_gap` array size from 39 to 38 to align with the number of added variables.

Update: *Resolved in pull request #89 at commit d8c2c27.*

Medium Severity

M-01 Operator Can Steal Rewards From the Harvester

After `AbstractARM` invests excess liquidity into an underlying lending market, it may accrue rewards in various tokens. It is the operator's responsibility to use the `SonicHarvester`'s `swap` function to convert these reward tokens into the `liquidityAsset` required by the ARM. The operator provides a `data` payload to the `swap` function, which is forwarded to the Magpie router to execute the swap. Before executing the swap, the Harvester validates that `data` contains the correct values for `recipient`, `fromAsset`, `toAsset`, and `fromAssetAmount`. Additionally, if slippage protection is enabled, the Harvester checks the change in the recipient's balance of the `liquidityAsset` to confirm a swap has occurred.

However, the slippage protection can be bypassed by crafting malicious `data` using Magpie's `CommandAction`s. The following caveats are relevant:

- The `toAssetAmount` returned by Magpie only reflects the recipient's `liquidityAsset` balance change after executing all commands and does not guarantee a swap occurred.
- Magpie's router can execute arbitrary calls defined in the provided `CommandAction`s.
- According to the documentation, initially the `rewardRecipient` is set to the ARM contract.

This setup allows the operator to steal rewards from the Harvester while bypassing the slippage check, as explained in the scenario below:

Initial State

`SonicHarvester` holds 10 units of token A that should be swapped to `liquidityAsset`.

Attack Sequence

1. The operator crafts `data` with the following: - A `TransferFrom` command that transfers all 10 token `A` to the attacker. - A `Call` command that initiates a swap in the ARM from `baseAsset` to `liquidityAsset`, thus inflating the ARM's `liquidityAsset` balance.
2. Since the slippage check only looks at the balance change in `liquidityAsset`, the swap appears valid even though the operator stole the reward tokens.

Final State

The attacker ends up with the 10 token A and only loses the cost of swapping `baseAsset` with `liquidityAsset` in the ARM. The slippage check passes because the ARM's `liquidityAsset` balance increased during the transaction.

This exploit path also applies to any function that increases the ARM's `liquidityAsset` balance, such as:

- `OriginARM`'s `claimOriginWithdrawals`, which redeems from the Origin Vault.
- The `_allocate` function, which may pull funds from the active lending market.

Consider using a more restrictive router with limited flexibility, or implement strict validation on each command in the `data` payload to prevent abuse.

Update: *Partially resolved in pull request #100 at commit de8ac5f. Even after the mitigation where rewards are returned to the Harvester after the swap, if any supported strategy yields*

rewards in `liquidityAsset`, the operator can redeem them during the Magpie router flow and incorrectly inflate the `toAssetAmount` of the swap.

Low Severity

L-01 `Token0` and `Token1` Might Be Incompatible With UniswapV2

A comment in the `AbstractARM` contract mentions that `token0` and `token1` should be compatible with UniswapV2. However, the `UniswapV2Factory` sorts the tokens before creating a pair contract. In contrast, the constructor does not sort the tokens, which may result in an incompatible configuration with UniswapV2 if the inputs are provided in the wrong order.

Consider sorting the token addresses in the same way as UniswapV2 to ensure compatibility with UniswapV2 contracts.

Update: *Acknowledged, not resolved. The ARM contract uses `traderate0`, set to 1, and `traderate1`, initialized with a value below 1. Under normal circumstances, it is expected that the `liquidityAsset` is assigned to `token0` and the staked asset to `token1`. Otherwise, users could potentially swap staked assets for liquidity assets at a 1-to-1 rate. This might lead to a scenario where `token0` and `token1` are set up in a way that contradicts the behavior of UniswapV2. The Origin team stated:*

> *The deployer of the ARM contract needs to set `token0` and `token1` in the correct order so that it is compatible with UniswapV2.*

L-02 `SonicHarvester` Does Not Validate Available Balance Before Swapping

The `swap` function of the `SonicHarvester` contract initiates a token swap without first verifying whether the contract holds a sufficient balance of `fromAssetAmount`. This omission may cause the transaction to revert later during the call to `magpieRouter`.

Consider adding an early check to ensure that `SonicHarvester` holds at least `fromAssetAmount` of the input token.

Update: *Acknowledged, not resolved. The Origin team stated:*

> *A revert in `SonicHarvest` will make it a little easier to debug, but it does not change the functional outcome that the transactions should revert if the Operator tries to swap more than what is in `SonicHarvester`. At this late, stage we are going to leave this out of the initial deployment.*

L-03 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity version with which the contracts will be compiled.

Throughout the codebase, multiple instances of floating pragma directives were identified:

- `AbstractARM.sol` has the `solidity ^0.8.23` floating pragma directive.
- `OriginARM.sol` has the `solidity ^0.8.23` floating pragma directive.
- `SiloMarket.sol` has the `solidity ^0.8.23` floating pragma directive.
- `SonicHarvester.sol` has the `solidity ^0.8.0` floating pragma directive.
- `ZapperARM.sol` has the `solidity ^0.8.23` floating pragma directive.

Consider using fixed pragma directives.

Update: *Acknowledged, not resolved. The Origin team stated:*

> *Fixing the version means that all the pragmas would have to be updated when we upgrade Solidity. We will keep the floating pragmas for now.*

L-04 Possible Duplicate Event Emissions

When a setter function does not check if the value has changed, it opens the possibility of spamming events that indicate a change, even when no actual change has occurred. Spamming identical values may confuse off-chain clients that rely on event data to track state changes.

Throughout the codebase, there are multiple instances of potential event spamming:

- The `setCrossPrice` function sets the `crossPrice` and emits an event without checking if the value has changed.
- The `_setFeeCollector` function sets the `feeCollector` and emits an event without checking if the value has changed.
- The `setCapManager` function sets the `capManager` and emits an event without checking if the value has changed.

- The `setARMBuffer` function sets the `armBuffer` and emits an event without checking if the value has changed.
- The `_setPriceProvider` function sets the `priceProvider` and emits an event without checking if the value has changed.
- The `_setAllowedSlippage` function sets the `allowedSlippageBps` and emits an event without checking if the value has changed.
- The `_setRewardRecipient` function sets the `rewardRecipient` and emits an event without checking if the value has changed.

Consider adding a check to revert the transaction if the new value is identical to the current value.

Update: *Acknowledged, not resolved. The Origin team stated:*

> *These setters can only be called by the permissions accounts, which are the Owner and/or the Operator. It is unlikely that these trusted accounts will spam the protocol. We also do not think that duplicate events are an issue for Squid, which is the off-chain indexing solution used by the Origin Protocol.*

L-05 Operator Can Block Market Removal by Setting It as Active

In case one of the `supportedMarkets` needs to be removed, the timelock set as the owner might initiate the removal. However, the operator can block this action by setting the target market as the `activeMarket`, which prevents its removal.

Consider implementing a mechanism to disable the `activeMarket` (e.g., setting it to the zero address) or introducing a fallback market that can be switched to when removing an active market.

Update: *Resolved in pull request #94 at commit cc0247e. The Origin team stated:*

> *The Owner, which will be a Timelock contract, can avoid the blocking action by calling `setActiveMarket` before calling `removeMarket`. This does open up a new blocking action whereby the Operator can set the active market to the same one the Owner is setting, causing the transaction to fail with `ARM: market in active`. One solution would be to change `setActiveMarket` by replacing `require(previousActiveMarket != _market, "ARM: already active market");` with `if (previousActiveMarket == _market) return;`*

L-06 Incorrect `_availableAssets` Calculation During Initialization

During the initialization of the `AbstractARM`, the contract calls the `_availableAssets` function to compute the total value held by the contract. However, at this stage, the `crossPrice` is not yet set and thus defaults to zero. This causes `_availableAssets` to ignore the `baseAsset` balance in its logic. If the contract holds any `baseAsset` during initialization, the resulting value calculation will be incorrect and could lead to existing `baseAssets` amount being charged performance fees on pre-existing balances.

Consider setting the `crossPrice` before invoking `_availableAssets` to ensure accurate valuation during initialization.

Update: *Acknowledged, not resolved. The Origin team stated:*

> *It is true that `crossPrice` is set after `_availableAssets` is used in the initializer. We think that this is ok. If someone donates base assets to the ARM proxy before it is initialized, then we do not want to count them in the initial `lastAvailableAssets`. When we call `collectFees` the first time, the protocol can earn a 20% performance fee from the donated base assets.*

L-07 Improper Handling of Paused or Compromised `activeMarket` in ARM

ARM contracts are augmented with the ability to invest their surplus `liquidityAsset` into an underlying lending market, referred to as the `activeMarket`. While this provides an opportunity to earn additional yield, the current design does not gracefully handle cases where the `activeMarket` becomes paused or compromised.

One issue arises when the `activeMarket` is paused. The ARM's `setActiveMarket` function, which is used to migrate to a different market, relies on a redeem call to exit the current one. If the market is paused, this redeem call fails, preventing the protocol from rotating to a new market.

A second issue stems from what happens if the `activeMarket` is hacked. In such a case, the balance of the ARM in the lending market drops, which in turn causes the total available assets to fall below the amount owed to users. The design of the withdrawal queue, where earlier claims must be fully satisfied before later ones, can lead to a system-wide freeze on redemptions.

A third issue arises from the logic in the `_allocate` function. When `availableAssets` is zero, the function exits early and does not attempt to withdraw funds from the market. While this is normally safe, if funds are stranded in the `activeMarket` following a compromise or pause, they remain inaccessible even though users are waiting for liquidity.

To clarify the second and third issues, below is a scenario where Alice is trying to redeem her shares from the ARM:

Initial State

- Alice and Bob are the only liquidity providers in the ARM. Alice has deposited 8 tokens and Bob has deposited 2.
- The ARM has a total of 10 tokens in `liquidityAsset`, and with a 20% liquidity buffer, it invests 8 tokens into the `activeMarket`, keeping 2 in the ARM.

Scenario Steps

1. Alice calls `requestRedeem` to withdraw 5 tokens. While the request is submitted, liquidity reallocation does not happen immediately, which leads to 8 tokens remaining in the `activeMarket`.
2. The `activeMarket` is hacked, and 6 tokens worth of value are lost. ARM is now left with only 4 tokens (2 in buffer, 2 salvaged from the market).
3. The total `assets` tracked by the ARM is now 4, which is less than the outstanding withdrawal requests (8).
4. The `_availableAssets` function now returns 0 due to its condition check.
5. The `_allocate` function refuses to withdraw funds from the `activeMarket` since `availableAssets` is 0.

Final State

- Alice cannot call `claimRedeem` because her 5-token request exceeds the protocol's current asset balance. To retrieve any tokens, she would have to send an extra token to ARM, effectively giving up 1 token to reclaim the other 4.
- Bob, who only requested 2 tokens, is blocked from claiming because Alice's request is earlier in the queue and must be fully satisfied before his is processed.
- The `_allocate` function does not take the money out of the `activeMarket` as it stops when available assets are zero.

To address this situation, consider implementing the following suggestions:

- Allow migrating between markets even when the `activeMarket` is paused so that the protocol can reclaim assets once the underlying lending market unpauses.

- Enable partial redeeming so that users can claim what is available, even if it does not fully satisfy their original request.
- Modify `_allocate` to fully withdraw funds from the `activeMarket` when `availableAssets` becomes zero.

Update: *Acknowledged, not resolved.*

Notes & Additional Information

N-01 Prefix Increment Operator (`++i`) Can Save Gas in Loops

Throughout the codebase, multiple opportunities for optimizing `for` loop iteration were identified:

- The `i++` increment in `AbstractARM.sol`.
- The `i++` increment in `SiloMarket.sol`.

Consider using the prefix increment operator `++i` instead of the post-increment operator `i++` in order to save gas. This optimization skips storing the value before the incremental operation, as the return value of the expression is ignored.

Update: *Resolved in pull request #95 at commit 52b43b9.*

N-02 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is beneficial as it allows the code owners to define the appropriate communication channel for vulnerability disclosure, reducing the risk of miscommunication or unreported issues. Additionally, in the event that a vulnerability is discovered in a third-party dependency, it becomes easier for maintainers to notify the relevant parties and share mitigation steps.

Throughout the codebase, multiple instances of contracts missing a security contact were identified:

- The `AbstractARM` abstract contract
- The `OriginARM` contract
- The `ISiloMarket` interface
- The `IHookReceiver` interface
- The `SiloMarket` contract
- The `SonicHarvester` contract
- The `ZapperARM` contract

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended, as it is supported by tools such as the OpenZeppelin Wizard and the ethereum-lists.

Update: *Acknowledged, not resolved. The Origin team stated:*

> *Origin Protocol uses Immunefi for its security issues. `@custom:security-contact` supports linking to a webpage but we will not make this change for the initial release.*

N-03 Missing Named Parameters in Mappings

Since Solidity 0.8.18, developers can utilize named parameters in mappings. This means that mappings can take the form of `mapping(KeyType KeyName? => ValueType ValueName?)`. This updated syntax provides a more transparent representation of a mapping's purpose.

Throughout the codebase, multiple instances of mappings without named parameters were identified:

- The `withdrawalRequests` state variable in the `AbstractARM` contract
- The `supportedStrategies` state variable in the `SonicHarvester` contract

Consider adding named parameters to mappings in order to improve the readability and maintainability of the codebase.

Update: *Acknowledged, not resolved. The Origin team stated:*

> *We will leave this for now.*

N-04 Outdated Solidity Version

Using an outdated version of Solidity can lead to vulnerabilities and unexpected behavior in contracts. As such, it is important to keep the Solidity compiler version up to date.

The `SonicHarvester.sol` file is using an outdated Solidity version (`^0.8.0`).

Consider taking advantage of the latest Solidity version to improve the overall readability and security of the codebase. Regardless of the Solidity version used, consider pinning the version consistently throughout the codebase to prevent bugs due to incompatible future releases.

Update: *Resolved in pull request #93 at commit 9759118.*

N-05 Lack of Indexed Event Parameters

Throughout the codebase, multiple instances of events missing indexed parameters were identified:

- The `TraderateChanged` event of `AbstractARM.sol`
- The `CrossPriceUpdated` event of `AbstractARM.sol`
- The `FeeUpdated` event of `AbstractARM.sol`
- The `ARMBufferUpdated` event of `AbstractARM.sol`
- The `RequestOriginWithdrawal` event of `OriginARM.sol`
- The `ClaimOriginWithdrawals` event of `OriginARM.sol`
- The `HarvesterUpdated` event of `SiloMarket.sol`
- The `CollectedRewards` event of `SiloMarket.sol`
- The `SupportedStrategyUpdate` event of `SonicHarvester.sol`
- The `RewardRecipientUpdated` event of `SonicHarvester.sol`
- The `AllowedSlippageUpdated` event of `SonicHarvester.sol`
- The `PriceProviderUpdated` event of `SonicHarvester.sol`

To improve the ability of off-chain services to search and filter for specific events, consider indexing event parameters.

Update: *Acknowledged, not resolved. The Origin team stated:*

> *Most of these are existing events that are already being indexed by Squid. Changing them now may make it worse for indexing.*

N-06 Inconsistent Order Within Contracts

Throughout the codebase, multiple instances of contracts having an inconsistent order of functions were identified:

- The `AbstractARM` contract in `AbstractARM.sol`
- The `SonicHarvester` contract in `SonicHarvester.sol`
- The `ZapperARM` contract in `ZapperARM.sol`

To improve the project's overall legibility, consider standardizing ordering throughout the codebase as recommended by the Solidity Style Guide (Order of Functions).

Update: *Acknowledged, not resolved. The Origin team stated:*

> *This will not be changed now.*

N-07 Inconsistent Use of Returns

Within `SiloMarket.sol`, multiple instances of functions having an inconsistent usage of returns were identified:

- In the `maxWithdraw` function
- In the `maxRedeem` function

Consider being consistent with the use of returns throughout the functions.

Update: *Acknowledged, not resolved. The Origin team stated:*

> *This has resulted from using a mix of standards. `maxWithdraw` and `maxRedeem` is from ERC-4626. `balanceOf` is from ERC-20 which was written before named return variables. We will leave this as it is.*

N-08 Magic Numbers in the Code

Within `SonicHarvester.sol`, multiple instances of literal values with unexplained meanings were identified:

- The `18` literal number
- The `1e4` literal number
- The `1e4` literal number
- The `1e18` literal number

- The `1000` literal number

Consider defining and using `constant` variables instead of using literals to improve the readability and maintainability of the codebase.

Update: *Acknowledged, not resolved. This issue is primarily a matter of coding style and would be perfectly acceptable if the magic number is clearly documented within the code. The Origin team stated:*

> *We would argue that using constants makes these harder to read. For example, we prefer* `if (IERC20Metadata(_liquidityAsset).decimals() != 18) revert InvalidDecimals();` *over* `if (IERC20Metadata(_liquidityAsset).decimals() != DECIMAL_NUM) revert InvalidDecimals();`

N-09 Function Visibility Overly Permissive

Throughout the codebase, multiple instances of functions with unnecessarily permissive visibility were identified:

- The `_inDeadline` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.
- The `_transferAsset` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.
- The `_transferAssetFrom` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.
- The `_swapExactTokensForTokens` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.
- The `_swapTokensForExactTokens` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.
- The `_deposit` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.
- The `_requireLiquidityAvailable` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.
- The `_availableAssets` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.
- The `_externalWithdrawQueue` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.

- The `_setFee` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.
- The `_setFeeCollector` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.
- The `_feesAccrued` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.
- The `_allocate` function in `AbstractARM.sol` with `internal` visibility could be limited to `private`.
- The `_externalWithdrawQueue` function in `OriginARM.sol` with `internal` visibility could be limited to `private`.
- The `_setHarvester` function in `SiloMarket.sol` with `internal` visibility could be limited to `private`.
- The `_doSwap` function in `SonicHarvester.sol` with `internal` visibility could be limited to `private`.
- The `_setPriceProvider` function in `SonicHarvester.sol` with `internal` visibility could be limited to `private`.
- The `_setAllowedSlippage` function in `SonicHarvester.sol` with `internal` visibility could be limited to `private`.
- The `_setRewardRecipient` function in `SonicHarvester.sol` with `internal` visibility could be limited to `private`.
- The `deposit` function in `ZapperARM.sol` with `public` visibility could be limited to `external`.

To better convey the intended use of functions and to potentially realize some additional gas savings, consider changing a function's visibility to be only as permissive as required.

Update: *Acknowledged, not resolved. The Origin team stated:*

> *We think that using `internal` over `private` is ok. It just makes it harder if the contract is to be overriden with inheritance. `ZapperARM` will not compile if `deposit` is changed to `external` as `deposit` is called in the `external` `receive` function.*

N-10 Multiple Optimizable State Reads

Throughout the codebase, multiple opportunities for optimizing storage reads were identified:

- The `crossPrice` variable in lines 417 and 418 in `AbstractARM.sol`.
- The `capManager` variable in lines 505 and 506 in `AbstractARM.sol`.
- The `feeCollector` variable in lines 754 and 756 in `AbstractARM.sol`.

- The `activeMarket` variable in lines 858, 859, 863, 871, 876, 878, and 882 in `AbstractARM.sol`.
- The `harvester` variable in lines 155 and 159 in `SiloMarket.sol`.
- The `rewardRecipient` variable in lines 121 and 127 in `SonicHarvester.sol`.
- The `priceProvider` variable in lines 135 and 138 in `SonicHarvester.sol`.

Consider reducing SLOAD operations that consume unnecessary amounts of gas by caching the values in a memory variable.

Update: *Partially resolved in pull request #96 at commit d963105. The Origin team stated:*

> *The extra 100 gas for a warm read in the admin functions is ok. We are mostly concerned with the swap functions when it comes to gas optimizations. There are a lot of extra reads of `activeMarket` in `_allocate`, so that has been changed.*
>
> ***Before***
> `text | Function Name | Min | Avg | Median | Max | # Calls | | allocate | 2525 | 64510 | 83603 | 101829 | 8 |`
>
> ***After***
> `text | Function Name | Min | Avg | Median | Max | # Calls | | allocate | 2525 | 64393 | 83429 | 101555 | 8 |`

N-11 Variables Initialized With Their Default Values

Throughout the codebase, multiple instances of variables being initialized with their default values were identified:

- In `AbstractARM.sol`, the `i` variable
- In `SiloMarket.sol`, the `i` variable
- In `SonicHarvester.sol`, the `i` variable

To avoid wasting gas, consider not initializing variables with their default values.

Update: *Acknowledged, not resolved. The Origin team stated:*

> *We do not see any gas savings with Solidity 8.0.23 with the optimizer on.*

N-12 Missing Docstrings

Throughout the codebase, multiple instances of missing docstrings were identified. While some of the instances listed below are events and the Origin team has previously noted that comments are typically not included for events, the following list has been provided in case the team wishes to add comments at this stage:

- In `AbstractARM.sol`, the `allocate` function
- In `AbstractARM.sol`, the `setARMBuffer` function
- In `SiloMarket.sol`, the `ISiloMarket` interface
- In `SiloMarket.sol`, the `hookReceiver` function
- In `SiloMarket.sol`, the `IHookReceiver` interface
- In `SiloMarket.sol`, the `configuredGauges` function
- In `ZapperARM.sol`, the `ws` state variable

Event Instances

Multiple events across `AbstractARM.sol`, `OriginARM.sol`, `SiloMarket.sol`, `SonicHarvester.sol`, and `ZapperARM.sol` lack docstrings. While event documentation is often omitted, adding NatSpec comments can help improve off-chain tooling and documentation clarity.

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Partially resolved in pull request #99 at commit 60b1111. Interfaces and their functions mentioned in the issue still lack docstrings.*

N-13 Incomplete Docstrings

Throughout the codebase, multiple instances of incomplete docstrings were identified. In many cases, parameters or return values are missing or not documented in the expected NatSpec format:

- In `AbstractARM.sol`, in the `claimable` function:
  - Not all return values are documented in the correct format.
- In `AbstractARM.sol`, in the `totalAssets` function:
  - Not all return values are documented.

- In `AbstractARM.sol`, in the `convertToShares` function:
  - The `assets` parameter is not documented.
  - The return value is not documented.
- In `AbstractARM.sol`, in the `convertToAssets` function
  - The `shares` parameter is not documented.
  - The return value is not documented.
- In `AbstractARM.sol`, in the `setFeeCollector` function:
  - The `_feeCollector` parameter is not documented.
- In `AbstractARM.sol`, in the `collectFees` function:
  - Not all return values are documented.
- In `AbstractARM.sol`, in the `feesAccrued` function:
  - The return value is not documented.
- In `AbstractARM.sol`, in the `setActiveMarket` function:
  - The `_market` parameter is not documented.
- In `AbstractARM.sol`, in the `setCapManager` function:
  - The `_capManager` parameter is not documented.
- In `OriginARM.sol`, in the `claimOriginWithdrawals` function:
  - The return value is not documented.
- In `SiloMarket.sol`, in the `previewRedeem` function:
  - The `shares` parameter is not documented.
- In `SiloMarket.sol`, in the `collectRewards` function:
  - Not all return values are documented.
- In `SonicHarvester.sol`, in the `initialize` function:
  - Parameters `_priceProvider`, `_allowedSlippageBps`, and `_rewardRecipient` are not documented.
- In `SonicHarvester.sol`, in the `collect` function:
  - The return value is not documented.
- In `SonicHarvester.sol`, in the `swap` function:
  - The return value is not documented.
- In `SonicHarvester.sol`, in the `setPriceProvider` function:
  - The `_priceProvider` parameter is not documented.
- In `SonicHarvester.sol`, the `setRewardRecipient` function:
  - The `_rewardRecipient` parameter is not documented.

- In `ZapperARM.sol`, the `deposit` function:
  - The `arm` parameter is not documented.
- In `ZapperARM.sol`, in the `rescueERC20` function:
  - The `amount` parameter is not documented.

Consider thoroughly documenting all public and sensitive functions, including their parameters and return values. When writing docstrings, use the Ethereum Natural Specification Format (NatSpec) to improve maintainability and compatibility with documentation and analysis tools.

Update: *Partially resolved in pull request #98 at commits 0bfb238, c67e212, and 70a5c65. The docstrings for the `SiloMarket.sol` contract, specifically for the `collectRewards` function, as well as for the `SonicHarvester.sol` contract and its `initialize` and `collect` functions, have not been added. The Origin team stated:*

> *Most of these have been fixed.*

N-14 Redundancies Throughout the Codebase

Throughout the codebase, multiple instances of redundant code were identified:

- This comparison with the `false` boolean literal within the `AbstractARM` contract is redundant.
- Cast to `address` is not needed for `token0` and `token1`.

Consider fixing these instances to avoid unnecessary operations and improve the clarity of the code.

Update: *Resolved. The Origin team stated:*

> *We think that the following is more readable `require(request.claimed == false, "Already claimed");` than `require(request.claimed, "Already claimed");` . We also think that the cast-to-address issue came up in the last audit of the ARM contract. Removing the `address` cast results in compilation failure as `token0` and `token1` are of type `IERC20`.*

N-15 Typographical Errors

Throughout the codebase, multiple instances of typographical errors were identified:

- `OriginARM`, line 81: "Vualt" → "Vault"

- `SiloMarket`, line 73: "a exact" → "an exact"
- `SiloMarket`, line 103: "a exact" → "an exact"
- `SonicHarvester`, line 73: The `@notice` tag is empty and can be removed or filled with a meaningful description.

Consider fixing the aforementioned instances to increase the clarity and readability of the codebase.

Update: *Resolved in pull request #97 at commit 0bfb238.*

Recommendations

Consider Adding Non-Reentrancy to `swap`

The `swap` function in the `SonicHarvester` contract is protected by the `onlyOperatorOrOwner` modifier, but it does not include a non-reentrancy guard. Given that the Magpie router allows for the execution of many steps through `CommandActions`, an operator could potentially craft calldata that causes the `SonicHarvester` contract to re-enter its own `swap` function. Re-entering this function does not serve any legitimate purpose and does not provide any functional advantage.

Since a single call is sufficient for executing a swap, and allowing reentrancy only increases the potential attack surface, consider adding a non-reentrancy modifier to the `swap` function of the `SonicHarvester` contract.

Conclusion

This audit focused on the newly added functionality in the **ARM (Automated Redemption Manager)**, which allows surplus liquidity to be deployed into an underlying lending market. The scope also included supporting contracts such as the `SonicHarvester`, which is responsible for collecting and converting reward tokens, and the `SiloMarket`, which facilitates interaction with Silo Finance markets.

During the review, several issues were identified. Most notably, a high-severity storage gap misconfiguration was found that could lead to upgradeability risks and potential storage collisions. Additionally, a medium-severity vulnerability was uncovered in the `SonicHarvester`, which could allow a privileged operator to exploit Magpie's command interface and redirect rewards for malicious purposes. These vulnerabilities should be addressed prior to deployment to ensure the security and robustness of the system.

Despite these findings, the codebase reflects **solid engineering practices** and a **clean, modular architecture**. The Origin team is appreciated for their responsiveness and collaboration throughout the audit process.