SHERLOCK

Security Review For Zyfai

Collaborative Audit Prepared For: **Zyfai**
Lead Security Expert(s): **0x3b**
defsec
Date Audited: **March 2 - March 4, 2026**

Introduction

Zyfai is a DeFi agent built to optimize yields using real-time data and automated on-chain actions. Leveraging ERC-7579 Safe modular accounts, for example with session modules, it provides a non-custodial solution for open and personalized financial rebalancing across DeFi.

Zyfai is pushing the boundaries of on-chain capital allocation with a more tailored, efficient, and customizable protocol.

Scope

Repository: ondefy/erc7540-wrapper

Audited Commit: a1f65cf4267d400d54cc2654d5d19b77a0cf75d8

Final Commit: b92b778148b5d2af2a0a422b295ed3a3c914d151

Files:

- src/SemiAsyncRedeemVault.sol
- src/SmartAccountProxy.sol
- src/SmartAccountWrapper.sol

Final Commit Hash

b92b778148b5d2af2a0a422b295ed3a3c914d151

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	13

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: No `cancelRedeem` irrevocable share burn leaves users trapped [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/6

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

Summary

There is no `cancelRedeemRequest`, `cancelRedeem`, or any cancellation mechanism. When `requestRedeem` burns shares, the user is left with only a non-transferable, non-cancellable `WithdrawRequest` struct and no escape hatch. ERC-7887 standardizes this cancellation flow but is entirely absent.

Vulnerability Detail

When a user calls `requestRedeem`, shares are burned immediately at line 364:

```
_burn(owner, sharesToRequest); // Irrevocable
```

The user's position transforms from:

- **Before:** Shares (fungible, transferable, yield-bearing)
- **After:** `WithdrawRequest` struct (non-fungible, non-transferable, non-cancellable, non-yield-bearing, no timeout)

Missing ERC-7887 functions:

- `cancelRedeemRequest(uint256, address)`
- `pendingCancelRedeemRequest(uint256, address)`
- `claimableCancelRedeemRequest(uint256, address)`
- `claimCancelRedeemRequest(uint256, address, address)`
- ERC-165 interface ID `0xe76cffc7`

Impact

Users are held hostage by the operator/owner timeline with zero self-service recovery. In vault insolvency, users have no shares to participate in recovery.

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SemiAsyncRedeemVault.sol#L350-L390

Tool Used

Manual Review

Recommendation

Implement ERC-7887 cancellation support. Add `cancelRedeemRequest` that re-mints `requestedShares` to the controller, subtracts `requestedAssets` from `cumulativeRequestedWithdrawalAssets`, and blocks new `requestRedeem` calls for the cancelling controller while pending.

Discussion

0xSulpiride

This is acceptable

defsec

Risk accepted.

Issue M-2: `previewRedeem` and `previewWithdraw` do not revert for async vault, violates ERC-7540 [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/7

Summary

ERC-7540 states that for async redeem vaults, `previewRedeem` and `previewWithdraw` MUST revert. Neither function is overridden, so both remain operational and return values.

Vulnerability Detail

The contract inherits from `ERC4626Upgradeable` without overriding `previewRedeem` or `previewWithdraw`. Both remain fully operational.

Additionally, `previewRedeem` is used internally in `requestRedeem` (line 317):

```
uint256 assets = previewRedeem(shares); // locks exchange rate at request time
```

Link : https://eips.ethereum.org/EIPS/eip-7540

Impact

Misleading values for external consumers. Stale exchange rate locking for async requests.

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SemiAsyncRedeemVault.sol#L317

Tool Used

Manual Review

Recommendation

Override both functions to revert:

```
function previewRedeem(uint256) public pure override returns (uint256) {
    revert("ERC7540: async vault");
```

```
}
function previewWithdraw(uint256) public pure override returns (uint256) {
    revert("ERC7540: async vault");
}
```

Use an internal helper for the `requestRedeem` conversion logic.

Discussion

0xSulpiride

fixed - https://github.com/ondefy/erc7540-wrapper/commit/9133612dbedce802f98730707b7dabad9988bd9c

Issue M-3: Users can front run losses on strategies [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/12

Summary

Users can front run losses on strategies

Vulnerability Detail

If a loss on any vaults that we operate occurs `transmitAllocatedAssets` is called to set out `allocatedAssets` to the actual amount.

```
function transmitAllocatedAssets(uint256 assets) public onlySmartAccount {
    if (pendingWithdrawals() > 0) revert SA__PendingWithdrawals();
    _checkMaxDeviationRate(assets);
    _getSmartAccountWrapperStorage().allocatedAssets = assets;
    emit AllocatedAssetsTransmitted(assets);
}
```

That function cannot be triggered if there are any withdraws scheduled. However this does not stop users from evading losses. They can front-run `transmitAllocatedAssets`:

- if there are any idle assets and directly withdraw
- even if there are none, they can still request a withdraw and their request will not be impacted by loss

Note that this can also be done for positive yield, however if the function is called often enough it won't be practical for users to farm it.

Example:

1. One of the vaults we stake in gets hacked
2. We need to report a 5% loss
3. Users can withdraw instantly or request a withdraw
4. Loss is applied
5. The user deposits again

The user has essentially avoided the loss since his assets to withdraw are calculated when the request is made and thus even after applying a loss he will withdraw 100% of his assets.

Impact

Users can evading losses by front running `transmitAllocatedAssets`.

Tool Used

Manual Review

Recommendation

Consider making the requests in shares and not assets and convert when the claim happens. If you don't want to let users earn APY when in a withdraw cycle then keep the original assets and award the user `min(shares * currectExchangeRate, assetsToRequest)`.

Discussion

defsec

Great catch!

Also, Both `requestRedeem` and `requestWithdraw` convert between shares and assets at the current exchange rate with no minimum output or deadline parameter. If the share price changes between transaction submission and execution (e.g., via a preceding `transmitAllocatedAssets` call), the user receives a different amount than expected.

0xSulpiride

in _requestWithdraw we should store only shares

0xSulpiride

1. return the minimum of actual assets vs assetsToRequest

0xSulpiride

1. return the minimum of actual assets vs assetsToRequest

added this and unit tests for such cases - https://github.com/ondefy/erc7540-wrapper/commit/54d0da51a922b19ffc80e1b4865740f42aed243e

Issue L-1: `RedeemRequest` event emits `assetsToRequest` instead of `sharesToRequest` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/4

Summary

The `RedeemRequest` event in `SemiAsyncRedeemVault._requestWithdraw` emits `assetsToRequest` as the last parameter, but the ERC-7540 specification requires the last parameter to be `shares` for redeem requests.

Vulnerability Detail

ERC-7540 Specification:

```
event RedeemRequest(
    address indexed controller,
    address indexed owner,
    uint256 indexed requestId,
    address sender,
    uint256 shares   // <-- MUST be shares for redeem requests
);
```

Actual Code (`SemiAsyncRedeemVault.sol:385`):

```
emit RedeemRequest(controller, owner, uint256(withdrawKey), caller,
 ↪  assetsToRequest);
//
//                                                              audit: emits
 ↪  assets, not shares
```

According to the specification, the last non-indexed parameter should be the number of vault shares being redeemed. The code emits `assetsToRequest` (the underlying asset amount) instead of `sharesToRequest`. Since shares and assets can have very different magnitudes (especially as yield accrues and the exchange rate diverges), **every off-chain indexer, aggregator, and integrator** consuming this event will get incorrect data.

Impact

Incompatibility with the ERC-7540 spec.

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SemiAsyncRedeemVault.sol#L385

Tool Used

Manual Review

Recommendation

Change line to:

```
emit RedeemRequest(controller, owner, uint256(withdrawKey), caller,
↪  sharesToRequest);
```

Discussion

0xSulpiride

fixed - https://github.com/ondefy/erc7540-wrapper/commit/48b98fe6e42ffe683e6218f29cdff86437580212
allowbreak #diff-19989ae45006de749b6bc6fea651cbed6961fcee36d20ecebe7f6263e5117334R221-L395

defsec

Fix is confirmed.

Issue L-2: Deviation rate check bypassed when `allocatedAssets == 0` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/8

Summary

The `_checkMaxDeviationRate` function is entirely skipped when `allocatedAssets()` is 0, allowing the smart account to set `allocatedAssets` to any value via `transmitAllocatedAssets`.

Vulnerability Detail

```
// SmartAccountWrapper.sol:143-151
function _checkMaxDeviationRate(uint256 assets) internal view {
    uint256 _allocatedAssets = allocatedAssets();
    if (_allocatedAssets > 0) { // BYPASSED when allocatedAssets == 0
        uint256 deviationAbs = assets > _allocatedAssets ? assets -
        ↪  _allocatedAssets : _allocatedAssets - assets;
        uint256 deviationRate = deviationAbs.mulDiv(1 ether, _allocatedAssets);
        if (deviationRate > MAX_DEVIATION_RATE) revert
        ↪  SA__ExceededMaxDeviationRate();
    }
}
```

`allocatedAssets` can be 0 at deployment, after `forceTransmitAllocatedAssets(0)`, or after `transmitDeallocatedAssets(0)`. In this state, the smart account can call `transmitAllocatedAssets(type(uint256).max)` with zero checks.

Impact

Share price inflation bypassing the 0.25% deviation safety mechanism.

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SmartAccountWrapper.sol#L143-L151

Tool Used

Manual Review

Recommendation

When `allocatedAssets == 0`, revert if `assets > 0`.

Discussion

0xSulpiride

removed deviation check - https://github.com/ondefy/erc7540-wrapper/commit/9133612dbedce802f98730707b7dabad9988bd9c

defsec

Fix is confirmed.

Issue L-3: No emergency pause mechanism [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/9

Summary

Neither contract uses `PausableUpgradeable`. There is no way to halt deposits, withdrawals, or claims during a crisis.

Vulnerability Detail

If the smart account is compromised, new deposits continue flowing to the compromised address because `_deposit` always calls `_transferToSmartAccount`. There is no way to stop this, `setSmartAccount` redirects future deposits but doesn't prevent the immediate one in progress.

Impact

Inability to respond to emergencies in a controlled manner.

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SmartAccountWrapper.sol#L112-L123

Tool Used

Manual Review

Recommendation

Add `PausableUpgradeable` and apply `whenNotPaused`.

Discussion

0xSulpiride

added - https://github.com/ondefy/erc7540-wrapper/commit/4c1fd3227c8c8e9fcf61bb1b3fe853531f85d416

defsec

Fix is confirmed.

Issue L-4: No two-step ownership transfer [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/10

Summary

Uses `OwnableUpgradeable` instead of `Ownable2StepUpgradeable`. An accidental ownership transfer is irrecoverable.

Vulnerability Detail

A single `transferOwnership` call with a wrong address permanently loses all admin functions and beacon upgrade capability.

Impact

Irrecoverable loss of protocol control.

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SmartAccountWrapper.sol#L16

Tool Used

Manual Review

Recommendation

Use `Ownable2StepUpgradeable`.

Discussion

0xSulpiride

fixed - https://github.com/ondefy/erc7540-wrapper/commit/48b98fe6e42ffe683e6218f29cdff86437580212

defsec

Fix is confirmed.

Issue L-5: `transmitDeallocatedAssets` does not verify actual token transfer occurred [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/11

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

Summary

The function updates `allocatedAssets` accounting without verifying that the corresponding tokens were actually transferred to the vault.

Vulnerability Detail

```
function transmitDeallocatedAssets(uint256 remainingAllocatedAssets) public
↪  onlySmartAccount {
    uint256 current = allocatedAssets();
    if (remainingAllocatedAssets > current) revert
    ↪  SA__DeallocatedAssetsExceedAllocated(remainingAllocatedAssets, current);
    _getSmartAccountWrapperStorage().allocatedAssets = remainingAllocatedAssets;
}
```

The expected pattern is `transfer` then `transmitDeallocatedAssets`, but this ordering is not enforced on-chain.

Impact

Potential accounting discrepancy between onchain state and actual token balances.

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SmartAccountWrapper.sol#L160-L167

Tool Used

Manual Review

Recommendation

Check `balanceOf(address(this))` before and after, or combine the transfer and accounting update in a single function.

Discussion

0xSulpiride

this is also ok, transmitDeallocatedAssets reports how much of underlying asset is deposited into strategies, and we can't track them onchain without integrating each protocol in the strategy

defsec

Acknowledged.

Issue L-6: `totalAssets()` can revert [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/13

Summary

`totalAssets()` contains an explicit `revert` when `obligations > gross`, violating the ERC-4626 requirement that `totalAssets` MUST NOT revert.

Reference : https://eips.ethereum.org/EIPS/eip-4626

Vulnerability Detail

```
// SemiAsyncRedeemVault.sol:143-151
function totalAssets() public view override(ERC4626Upgradeable, IERC4626) returns
↪  (uint256) {
    uint256 vaultBalance = IERC20(asset()).balanceOf(address(this));
    uint256 gross = vaultBalance + allocatedAssets() + pendingDeallocationAssets()
    ↪  + claimableFromStrategies();
    uint256 obligations = _outstandingObligations();
    if (obligations > gross) revert SA__TotalAssetsUnderflow(); // <-- VIOLATES
    ↪  ERC-4626
    unchecked { return gross - obligations; }
}
```

Impact

ERC4626 Violation

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SemiAsyncRedeemVault.sol#L143-L151

Tool Used

Manual Review

Recommendation

Return 0 instead of reverting.

Discussion

0xSulpiride

report 0 if obligations > gross

0xSulpiride

fixed it here - https://github.com/ondefy/erc7540-wrapper/commit/48b98fe6e42ffe683e6218f29cdff86437580212

defsec

Fix is confirmed.

Issue L-7: EIP7540 and EIP7575 are not fully compliant [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/14

Summary

EIP7540 and EIP7575 are not fully compliant

Vulnerability Detail

1 `requestRedeem` short circuits and does not call claim

`requestRedeem` calls `_processRequest`, where if enough idle assets are inside the vault will directly call `_withdraw`.

```
function _processRequest(uint256 assets, uint256 shares, address controller,
↪  address receiver, address owner)
    internal
    returns (bytes32)
{
    // ...
    if (assetsToWithdraw > 0) _withdraw(_msgSender(), receiver, owner,
    ↪  assetsToWithdraw, sharesToRedeem);
```

However EIP7549 states that it MUST trigger the claim-able state before the withdraw happens.

> A Request MAY transition straight to Claimable state but MUST NOT skip the Claimable state.

The EIP allows the function to withdraw immediately, but it needs to trigger claim first. Consider modifying it to trigger `claim` instead of strait to `_withdraw`.

2 `supportsInterface` does not trigger for `0x2f0a18c5`

> All asynchronous Vaults MUST return the constant value true if either 0xe3bc4e65 (representing the operator methods that all ERC-7540 Vaults implement) or 0x2f0a18c5 (representing the ERC-7575 interface) is passed through the interfaceID argument.

```
function supportsInterface(bytes4 interfaceId) external pure returns (bool) {
    return interfaceId == type(IERC165).interfaceId || interfaceId ==
    ↪  type(IERC7540Redeem).interfaceId
```

```
        || interfaceId == type(IERC7540Operator).interfaceId;
}
```

Consider adding it.

Impact

Not compliant to EIP7540.

Tool Used

Manual Review

Recommendation

These issues do not pose a thread, but would be nice to be fixed in order to be more compliant with the EIPs.

Discussion

defsec

Probably I can add this issue (https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/5) into that one.

0x3b33

yep, I was thinking of writing a comment, but decided to open a new one

0xSulpiride

I guess this one is ok since we don't support ERC-7575 (our vault is not multi-asset). Do I need to add `interfaceId == type(IERC4626).interfaceId` though?

0x3b33

Yes, this went overlooked on my part, you don't need to fix the second part - "2 supportsInterface does not trigger for 0x2f0a18c5"

0xSulpiride

removed sync withdraws completely - https://github.com/ondefy/erc7540-wrapper/commit/7ebe1d83b2329492b4ee9f8dd303b613996127c3

Issue L-8: Smart account can compound 0.25% deviation to arbitrary share price manipulation [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/15

Summary

`transmitAllocatedAssets` enforces a per-call 0.25% deviation check but has no cooldown, per-block limit, or cumulative cap. The smart account can call it hundreds of times in a single transaction, compounding 0.25% each time to achieve arbitrary share price inflation.

Vulnerability Detail

```
// SmartAccountWrapper.sol:153-158
function transmitAllocatedAssets(uint256 assets) public onlySmartAccount {
    if (pendingWithdrawals() > 0) revert SA__PendingWithdrawals();
    _checkMaxDeviationRate(assets);  // 0.25% per call - no limit on frequency
    _getSmartAccountWrapperStorage().allocatedAssets = assets;
}
```

The deviation check is relative to the *current* `allocatedAssets`, which is updated each call. Each call's 0.25% compounds on the previous value:

```
Starting:    1,000,000
Call 1:      1,002,500  (+0.25%)
Call 2:      1,005,006  (+0.25% of 1,002,500)
Call 100:    1,284,025  (+28.4% total)
Call 277:    2,000,000  (+100% total)
Call 400:    2,714,568  (+171% total)
```

All 400 calls can be batched in a single transaction via a wrapper contract. The smart account address just needs to be (or route through) a contract that calls `transmitAllocatedAssets` in a loop.

Impact

Share price can be inflated arbitrarily while technically staying within the 0.25% deviation per-call safety check.

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SmartAccountWrapper.sol#L153-L158
https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SmartAccountWrapper.sol#L143-L151

Tool Used

Manual Review

Recommendation

Add a cooldown period (e.g., one call per block) or a cumulative deviation cap per time window.

Discussion

0xSulpiride

We need to check if we need deviation check at all

0xSulpiride

removed deviation check completely - https://github.com/ondefy/erc7540-wrapper/commit/9133612dbedce802f98730707b7dabad9988bd9c

defsec

Fix is confirmed.

Issue L-9: Permanent claim lockout if receiver address is USDC blacklisted [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/16

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

Summary

The receiver address is set immutably at request time and cannot be changed. Since the deployed asset is USDC (which has a blacklist), if the receiver gets blacklisted between request and claim, the `safeTransfer` in `claim()` permanently reverts. With no cancel function and no way to change the receiver, the assets are locked forever.

Vulnerability Detail

he receiver is locked at request time:

```
// SemiAsyncRedeemVault.sol:373-382
$.withdrawRequests[withdrawKey] = WithdrawRequest({
    receiver: receiver,    // immutable after creation - no setter exists
    // ...
});
```

The claim function always sends to this locked receiver:

```
// SemiAsyncRedeemVault.sol:430
IERC20(asset()).safeTransfer(request.receiver, amount);
```

The deployed asset is USDC on Base (`0x833589fCD6eDb6E08f4c7C32D4f71b54bdA02913`). USDC has a blacklist controlled by Circle. If the receiver address gets blacklisted between request creation and claim execution.

Impact

Permanent, irrecoverable loss of user funds. The user has zero shares (burned) and cannot receive assets (blacklisted).

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SemiAsyncRedeemVault.sol#L373-L382
https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SemiAsyncRedeemVault.sol#L407-L434

Tool Used

Manual Review

Recommendation

Allow the controller (or an admin) to update the receiver address for an unclaimed request.

Issue L-10: No timeout on pending withdrawal requests [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/19

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

Summary

`WithdrawRequest.requestTimestamp` is set at request time but never read anywhere in the codebase. There is no timeout, deadline, or escalation mechanism. If the off-chain operator goes offline, withdrawal requests remain pending indefinitely with no self service recovery.

Vulnerability Detail

```
// SemiAsyncRedeemVault.sol:53-63
struct WithdrawRequest {
    uint256 requestedAssets;
    uint256 cumulativeRequestedWithdrawalAssets;
    uint256 requestTimestamp;  // @audit never used
    address owner;
    address receiver;
    bool isClaimed;
    uint256 requestedShares;
    address controller;
}
```

Request fulfillment depends entirely on the off-chain smart account operator calling `transmitDeallocatedAssets` and transferring tokens back. The only recovery path is the owner calling `forceTransmitDeallocatedAssets`, which requires the owner key to be available.

Impact

Users' funds can be locked indefinitely with no self-service recovery.

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SemiAsyncRedeemVault.sol#L53-L63
https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SemiAsyncRedeemVault.sol#L373-L382

Tool Used

Manual Review

Recommendation

Implement a timeout that allows users to cancel or auto-reclaim shares after a configurable period.

Discussion

0xSulpiride

Will delete `uint256 requestTimestamp`

defsec

acknowledged.

Issue L-11: `setSmartAccount` doesn't adjust `allocatedAssets` [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/20

Summary

When the owner calls `setSmartAccount(newAddress)`, the `allocatedAssets` value is not adjusted. It continues reflecting tokens held by the old smart account. The new smart account has zero tokens, but `totalAssets()` still includes the old `allocatedAssets`.

Vulnerability Detail

```
// SmartAccountWrapper.sol:176-180
function setSmartAccount(address smartAccount_) public onlyOwner {
    if (smartAccount_ == address(0)) revert SA__ZeroAddress();
    _getSmartAccountWrapperStorage().smartAccount = smartAccount_;
    // @audit  allocatedAssets is not adjusted, still reflects old SA's balance
    emit SmartAccountSet(smartAccount_);
}
```

- Old smart account can no longer call `transmitDeallocatedAssets()` =>`onlySmartAccount` modifier checks against the new address.
- Owner must call `forceTransmitAllocatedAssets(actual_in_new_SA)` => but this reverts if `pendingWithdrawals() > 0`.
- Owner can try `forceTransmitDeallocatedAssets(actual_in_new_SA)` => but this has the flawed guard (M-01)
- Tokens in old smart account are permanently stranded outside the vault's control.

Impact

Share price is inflated by the stranded `allocatedAssets` amount.

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SmartAccountWrapper.sol#L176-L180

Tool Used

Manual Review

Recommendation

Require `allocatedAssets == 0` before allowing smart account change, or automatically reset it.

Discussion

0xSulpiride

Will add allocatedAssets argument in the function

defsec

acknowledged.

0xSulpiride

> Will add allocatedAssets argument in the function

added this, we can mark this as fixed - https://github.com/ondefy/erc7540-wrapper/blob/main/src/SmartAccountWrapper.sol
allowbreak #L195-L200

Issue L-12: `maxDeposit` and `maxMint` return `type(uint256).max` when `smartAccount` is not set [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/21

Summary

When `smartAccount == address(0)`, deposits revert in `_transferToSmartAccount`, but `maxDeposit/maxMint` still return `type(uint256).max`. ERC-4626 requires these to return 0 when deposits are disabled.

Vulnerability Detail

```
// _transferToSmartAccount reverts when smartAccount is zero:
function _transferToSmartAccount(uint256 assets) internal {
    address _smartAccount = smartAccount();
    if (_smartAccount == address(0)) revert SA__SmartAccountNotSet();
    // ...
}

// But maxDeposit (inherited, not overridden) returns type(uint256).max
```

ERC-4626: "MUST return the maximum amount of assets deposit would allow... MUST return 0 if deposits are disabled."

Impact

Misleading return values violate ERC-4626.

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SmartAccountWrapper.sol#L118-L123

Tool Used

Manual Review

Recommendation

Override `maxDeposit` and `maxMint` to return 0 when `smartAccount == address(0)`.

Discussion

0xSulpiride

fixed here - https://github.com/ondefy/erc7540-wrapper/commit/48b98fe6e42ffe683e6218f29cdff86437580212

defsec

Fix is confirmed.

Issue L-13: `initialize` performs no validation on the parameters [RESOLVED]

Source: https://github.com/sherlock-audit/2026-02-zyfai-mar-2nd-2026/issues/22

Summary

The `initialize` function accepts `address(0)` for `underlyingToken_` and `smartAccount_` without validation, unlike `setSmartAccount` which checks for zero address. Since `initialize` can only be called once (guarded by `initializer`), misconfiguration requires proxy redeployment.

Vulnerability Detail

```
// SmartAccountWrapper.sol:83-94
function initialize(
    address owner_,
    address smartAccount_,
    address underlyingToken_,
    string memory name_,
    string memory symbol_
) public initializer {
    __Ownable_init(owner_);                              // OZ validates owner != 0
    __ERC20_init_unchained(name_, symbol_);               // no validation
    __ERC4626_init_unchained(IERC20(underlyingToken_));   // no zero check
    _getSmartAccountWrapperStorage().smartAccount = smartAccount_;  // no zero check
}
```

Compare with `setSmartAccount` which validates:

```
function setSmartAccount(address smartAccount_) public onlyOwner {
    if (smartAccount_ == address(0)) revert SA__ZeroAddress();  // validates
    // ...
}
```

Impact

Misconfigured initialization deploys a non-functional vault.

Code Snippet

https://github.com/ondefy/erc7540-wrapper/blob/a1f65cf4267d400d54cc2654d5d19b77a0cf75d8/src/SmartAccountWrapper.sol#L83-L94

Tool Used

Manual Review

Recommendation

Add zero-address checks for `underlyingToken_` and `smartAccount_` in `initialize`.

Discussion

0xSulpiride

smartAccountAddress will be zeroAddress at first

validated underlyingToken - https://github.com/ondefy/erc7540-wrapper/commit/b4fbcf38f1ac386608e2a838221a861c7e8f216f

defsec

Fix is confirmed.

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.