BLOCKSEC

Security Audit
Report for AsyncVault Contracts

Date: April 28, 2026 **Version:** 1.0
Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	R25
Target	AsyncVault Contracts

Version History

Version	Date	Description
1.0	April 28, 2026	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-04-28 09:11:49+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About the Audit Target

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The audit target (hereinafter referred to as the Target) of this audit is the code repository [1] of AsyncVault Contracts of R25.

The `AsyncVault` contracts are an upgradeable tokenized asset vault system of R25. `AsyncVault` follows an asynchronous redemption model, where deposits mint time-locked shares and redemptions proceed through a request-based flow that is settled after maturity, with withdrawal fulfillment handled through a separate queue component. Across this module, deposited assets are forwarded to a custodian address, withdrawals are funded by a separate payer account through the queue, and vault instances are created and tracked by the factory.

Note this audit only focuses on the smart contracts in the following directories/files:

- src/r25-pc/AsyncVault

Other files are not within the scope of the audit. Additionally, all dependencies of the Target are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 2`), as well as new code (in the following versions) to fix issues in the audit report. Code prior to and including the baseline version (`Version 0`), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
solidity-contract	`Version 1`	`33133de16b62f68e646bf38983a3b93509109f48`
	`Version 2`	`f3e56320fe7b86d9d332f17b8d1987cd2f3f4e76`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

[1] https://github.com/RiftSwapLabs/solidity-contract

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the Target, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of the Target.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan the Target with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of the Target and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency
* Improper use of the proxy system
* Reentrancy
* Denial of Service (DoS)
* Untrusted external call and control flow
* Exception handling
* Data handling and flow
* Events operation
* Error-prone randomness
* Oracle security
* Business logic correctness
* Semantic and functional consistency
* Emergency mechanism
* Economic and incentive impact

1.3.2 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **six** potential security issues. Besides, we have **nine** recommendations and **nine** notes.

- Low Risk: 6
- Recommendation: 9
- Note: 9

ID	Severity	Description	Category	Status
1	Low	Potential gas griefing attacks	Security Issue	Confirmed
2	Low	Lack of check on `assetsAfterFee` in function `batchRedeem()`	Security Issue	Confirmed
3	Low	Potential gas griefing in function `batchAutoRenew()`	Security Issue	Confirmed
4	Low	Incorrect implementation of ERC-4626 maximum limit functions	Security Issue	Partially Fixed
5	Low	Incompatibility with the ERC-7540 standard	Security Issue	Partially Fixed
6	Low	Lack of check on `maturityPeriod`	Security Issue	Fixed
7	-	Add an upper-bound check in `setMaxFluctuationBps()`	Recommendation	Confirmed
8	-	Fix storage gap size	Recommendation	Confirmed
9	-	Add input validation in function `createAsyncVault()`	Recommendation	Confirmed
10	-	Emit events before deleting request data	Recommendation	Fixed
11	-	Revise the incorrect parameters	Recommendation	Confirmed
12	-	Use different boundary conditions for cancellation and renewal	Recommendation	Confirmed
13	-	Revise incorrect annotations	Recommendation	Confirmed
14	-	Grant required `WithdrawQueue` roles during deployment	Recommendation	Confirmed
15	-	Add a minimum check on `redeemNoticePeriod`	Recommendation	Confirmed
16	-	Token assumption	Note	-
17	-	Inconsistency among vault logic due to implementation updates	Note	-
18	-	Deviations from the EIP-4626 standard	Note	-
19	-	Centralized withdrawal mechanism	Note	-
20	-	Ensure the correctness and timeliness of Oracle reports	Note	-
21	-	Unused `controller` field	Note	-
22	-	Share token transfer restriction	Note	-

23	-	Potential centralization risks	Note	-
24	-	Off-chain NAV treatment of pending redemption shares	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Potential gas griefing attacks

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The contract `AsyncVault` stores each user's time-locked share positions as an ordered linked list and linearly traverses these batches across multiple execution paths. As the number of batches grows, the gas cost of these operations increases. However, there is no upper bound on the number of timelock batches. This allows an attacker to repeatedly transfer shares on N distinct days to inject nodes into a benign user's list. This griefing attack gradually increases the victim's list size and makes subsequent related query paths increasingly gas-expensive.

```
1827   function getAvailableShares(
1828       address owner,
1829       uint64 currentTime
1830   ) public view returns (uint256 availableShares) {
1831       uint64 unlockTime = _nextTimelock[owner][GUARD];
1832       while (unlockTime != GUARD && unlockTime <= currentTime) {
1833           availableShares += _sharesLocks[owner][unlockTime];
1834           unlockTime = _nextTimelock[owner][unlockTime];
1835       }
1836   }
```

Listing 2.1: src/r25-pc/AsyncVault/AsyncVault.sol

Impact The attacker could intentionally increase the gas costs of a benign user's related query paths over time.

Suggestion Revise the logic accordingly.

Feedback from the project The project considered this risk acceptable, as a user is expected to hold only one share record per day. Exploiting this issue would require the attacker to transfer shares to the victim continuously over a long period.

2.1.2 Lack of check on `assetsAfterFee` in function `batchRedeem()`

Severity Low

Status Confirmed

Introduced by Version 1

Description In the contract AsyncVault, the function batchRedeem() converts redeemed shares into assets and applies a withdrawal fee. Under certain conditions, e.g., a very small share amount, a low settlementNav, assetsAfterFee may become zero. If swaper.withdrawByAgent() reverts when the withdraw value is zero, the function will revert, blocking normal settlement for all requests in that batch. Furthermore, retrying the affected request individually will also revert, as the settlement amount for that unlock time remains zero, leaving the request permanently stuck. Conversely, if the swaper accepts such withdrawals, the shares are burned while the user receives no assets.

```
1247          uint256 assets = request.shares.mulDiv(settlementNav, NAV_PRECISION, Math.Rounding.
                  Floor);
1248          uint256 fee = assets.mulDiv(
1249              withdrawFeeBps,
1250              FEE_PRECISION,
1251              Math.Rounding.Ceil
1252          );
1253
1254          uint256 assetsAfterFee = assets - fee;
1255
1256          _transferByUnlockTime(address(this), address(0), request.shares, request.unlockTime);
1257          uint256 withdrawalId = withdrawQueue.requestWithdrawal(
1258              assetsAfterFee,
1259              request.receiver,
1260              request.owner,
1261              requestId
1262          );
1263
1264          totalFeesCollected += fee;
1265          _unlockTimeToRequestIds[request.unlockTime].remove(requestId);
1266          _ownerToRequestIds[request.owner].remove(requestId);
1267          delete redeemRequests[requestId];
1268
1269          // Withdraw the underlying tokens from swaper
1270          swaper.withdrawByAgent(requestId, custodian, assetsAfterFee);
```

Listing 2.2: src/r25-pc/AsyncVault/AsyncVault.sol

Impact A redemption request may become permanently stuck, or a user may lose shares while receiving zero assets.

Suggestion Add a check to ensure that assetsAfterFee is greater than 0, and add corresponding processing logic.

Feedback from the project The project indicated that the withdrawal fee would not be enabled in practice, and settlementNav would always be greater than NAV_PRECISION. Therefore, the edge case would not ocurr.

2.1.3 Potential gas griefing in function batchAutoRenew()

Severity Low

Status Confirmed

Introduced by Version 1

Description In the contract `AsyncVault`, the function `batchAutoRenew()` processes shares under a given `oldUnlockTime` that have not entered a redeem request and automatically rolls these shares into a new unlock time. The function iterates through the address set in the variable `_unlockTimeToUserAddress[oldUnlockTime]`. Therefore, the gas cost of the function is related to the number of holders. However, the lack of restrictions on holder count exposes the function to gas griefing. Specifically, the transfer does not enforce a minimum transfer amount, nor does it have a whitelist restriction. This allows an attacker to distribute tiny amounts of shares with the same unlock time across many addresses. As a result, the variable `_unlockTimeToUserAddress[oldUnlockTime]` can become significantly larger, leading to higher gas consumption required by the function `batchAutoRenew()`.

```
864   function batchAutoRenew(
865       uint64 oldUnlockTime,
866       uint256 batchSize
867   ) external onlyRole(BizRole.CONTROLLER) returns (uint16 processedCount) {
868       if (batchSize == 0) {
869           batchSize = DEFAULT_BATCH_SIZE;
870       }
871
872       if (batchSize > MAX_REQUEST_BATCH) {
873           revert BatchSizeTooLarge(batchSize);
874       }
875
876       if (block.timestamp < oldUnlockTime - redeemNoticePeriod) {
877           revert InvalidRenewTimestamp(oldUnlockTime);
878       }
879
880       EnumerableSet.AddressSet storage users = _unlockTimeToUserAddress[oldUnlockTime];
881
882       uint16 userCount = type(uint16).max;
883       if (EnumerableSet.length(users) < userCount) {
884           userCount = uint16(EnumerableSet.length(users));
885       }
886
887       // Calculate how many users to process
888       uint256 toProcess = batchSize;
889       if (batchSize > userCount) {
890           toProcess = userCount;
891       }
892
893       address[] memory userList = new address[](toProcess);
894       for (uint256 i = 0; i < toProcess; i++) {
895           userList[i] = users.at(i);
896       }
897
898       // Process each user
899       for (uint256 i = 0; i < toProcess; i++) {
900           // Skip self
901           address owner = userList[i];
```

```
902             if (owner == address(this)) {
903                 continue;
904             }
905
906             uint256 shares = _sharesLocks[owner][oldUnlockTime];
907             if (shares > 0) {
908                 _autoRenew(owner, oldUnlockTime, shares);
909                 processedCount++;
910             }
911         }
912
913         emit BatchAutoRenewProcessed(oldUnlockTime, processedCount);
914         return processedCount;
915     }
```

Listing 2.3: src/r25-pc/AsyncVault/AsyncVault.sol

Impact This vulnerability may increase the gas cost of the function `batchAutoRenew()`.

Suggestion Revise the code logic accordingly.

Feedback from the project The project confirmed that the impact was acceptable.

2.1.4 Incorrect implementation of ERC-4626 maximum limit functions

Severity Low

Status Partially Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `AsyncVault`, ERC-4626 maximum limit functions are implemented incorrectly. Specifically, functions `maxDeposit()` and `maxMint()` do not reflect the `maxSupply` constraint, and the function `maxWithdraw()` does not account for fees. Additionally, none of the max functions return `0` when the vault is paused, as required by the ERC-4626 specification [1]. As a result, integrators may receive incorrect capacity signals, leading to failed transactions and inconsistent ERC-4626 behavior.

```
617     function maxWithdraw(address owner) public view returns (uint256) {
618         (uint256 sharesCanRequestRedeem, ) = _getFirstEligibleShares(owner, uint64(block.timestamp)
                );
619         return _convertToAssets(sharesCanRequestRedeem, Math.Rounding.Floor);
620     }
```

Listing 2.4: src/r25-pc/AsyncVault/AsyncVault.sol

```
596     function maxDeposit(address) public view returns (uint256) {
597         return maxDepositLimit;
598     }
599
600     /**
601      * @notice Returns the maximum amount of shares that can be minted by a given address
602      * @dev Calculated from maxDeposit limit
```

[1] https://eips.ethereum.org/EIPS/eip-4626

```
603     * @return The maximum mint amount in share units
604     */
605    function maxMint(address) public view returns (uint256) {
606        return _convertToShares(maxDepositLimit, Math.Rounding.Floor);
607    }
```

Listing 2.5: src/r25-pc/AsyncVault/AsyncVault.sol

Impact This may cause failed transactions due to incorrect capacity signals returned by these functions.

Suggestion Align these functions with the specification of EIP4626.

Feedback from the project The project clarified that they temporarily do not consider the pause situation and will upgrade the implementation in the future.

2.1.5 Incompatibility with the ERC-7540 standard

Severity Low

Status Partially Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `AsyncVault` declares itself as "ERC7540-compliant" (line 34). However, the implementation is not fully compatible with ERC-7540 specification [2] and may break valid integrations.

1. ERC-7540 vaults should return `true` for the ERC-165 interface IDs `0xe3bc4e65` and `0x2f0a18c5`. However, the implementation does not satisfy this requirement.

2. ERC-7540 requires implementing the ERC-7575 interface as well. In particular, the vault must expose the function `share()` to return the share token address. The contract `AsyncVault` does not implement this function.

3. ERC-7540 requires operator-related functionality. However, the contract `AsyncVault` only imports and inherits the interface `IERC7540Redeem` and does not implement the required functionality.

4. The function `claimableRedeemRequest()` does not consider whether the required oracle report is available. When the oracle report is not ready, this function may still return a non-zero value, indicating that redemption is claimable even though the subsequent redemption process cannot be completed.

```
1852    /// @notice Checks if the contract supports the given interface.
1853    /// @param interfaceId The interface ID.
1854    /// @return True if the contract supports the interface, false otherwise.
1855    function supportsInterface(
1856        bytes4 interfaceId
1857    ) public view virtual override(AccessControlUpgradeable, ERC165Upgradeable) returns (bool) {
1858        return
1859            interfaceId == bytes4(0x620ee8e4) ||
1860            super.supportsInterface(interfaceId);
1861    }
```

[2] https://eips.ethereum.org/EIPS/eip-7540

Listing 2.6: src/r25-pc/AsyncVault/AsyncVault.sol

```
function claimableRedeemRequest(
    uint256 requestId,
    address controller
) public view returns (uint256) {
    // 1. Verify request exists and belongs to controller
    RedeemRequestDetail memory request = redeemRequests[requestId];
    if (request.controller != controller) {
        return 0;
    }

    // 2. Check if in claimable state (based on time)
    if (block.timestamp >= request.unlockTime) {
        return request.shares;
    }
    return 0;
}
```

Listing 2.7: src/r25-pc/AsyncVault/AsyncVault.sol

Impact This incompatibility may cause failed integrations and incorrect redemption assumptions for standards-based integrators.

Suggestion Revise the logic accordingly.

Feedback from the project The project indicated that they temporarily do not support operator methods of ERC-7540 and the related interface ID (i.e., `0xe3bc4e65`).

2.1.6 Lack of check on `maturityPeriod`

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `AsyncVault`, the function `batchRedeem()` uses the NAV at the settlement time (i.e., `request.unlockTime - 1 days`) to process matured redeem requests. If the NAV value is zero, the function reverts. Meanwhile, the function `handleOracleReport()` only allows the Oracle to report NAV at strict 1-day intervals. This means that the settlement time needs to be an integer multiple of 1 day. The settlement time is derived from `lastAccountingTimestamp + maturityPeriod + 1 day` for a new issuance, and from `oldUnlockTime + maturityPeriod` for a renewal.

However, the function `initialize()` only checks that `maturityPeriod` is non-zero, and `maturityPeriod` cannot be modified once initialized. If `maturityPeriod` is not a multiple of `1 days`, the settlement time will not align to a day boundary, causing the redemption invocation to fail permanently.

```
function initialize(IAsyncVault.InitializeParams calldata params) public initializer {
    if (
        params.admin == address(0) ||
```

```
515             params.asset == address(0) ||
516             params.withdrawQueue == address(0) ||
517             params.whitelist == address(0) ||
518             params.swaper == address(0) ||
519             params.maturityPeriod == 0 ||
520             IERC20Metadata(params.asset).decimals() != 6
521         ) {
522             revert InvalidInput();
523         }
```

Listing 2.8: src/r25-pc/AsyncVault/AsyncVault.sol

```
1496    function handleOracleReport(
1497        ReportData calldata report
1498    ) external onlyRole(BizRole.ORACLE_REPORTER_ROLE) {
1499        // Sequence validation
1500        if (accounting.lastAccountingTimestamp != 0 && accounting.lastReportId != 0) {
1501            if (
1502                (report.accountingTimestamp - accounting.lastAccountingTimestamp != 1 days) ||
1503                (report.reportId - accounting.lastReportId != 1)
1504            ) {
1505                revert InvalidReport(
1506                    report.reportId,
1507                    report.accountingTimestamp,
1508                    report.todayNav
1509                );
1510            }
1511        }
1512
1513        if (block.timestamp < report.accountingTimestamp) {
1514            revert InvalidReport(
1515                report.reportId,
1516                report.accountingTimestamp,
1517                report.todayNav
1518            );
1519        }
1520
1521        // Fluctuation validation: based on latestNav vs report.todayNav
1522        if (accounting.latestNav > 0 && totalSupply() > 0) {
1523            uint256 navDifference = report.todayNav > accounting.latestNav
1524                ? report.todayNav - accounting.latestNav
1525                : accounting.latestNav - report.todayNav;
1526
1527            uint256 fluctuationBps = navDifference.mulDiv(
1528                FEE_PRECISION,
1529                accounting.latestNav,
1530                Math.Rounding.Floor
1531            );
1532
1533            if (fluctuationBps > maxFluctuationBps) {
1534                revert InvalidReport(
1535                    report.reportId,
1536                    report.accountingTimestamp,
```

```
1537                 report.todayNav
1538             );
1539         }
1540     }
1541
1542     // Record historical NAV
1543     dailyNav[report.accountingTimestamp] = report.todayNav;
1544     uint256 preNav = accounting.latestNav;
```

Listing 2.9: src/r25-pc/AsyncVault/AsyncVault.sol

```
1241         uint64 settlementTime = request.unlockTime - 1 days;
1242         uint256 settlementNav = dailyNav[settlementTime];
1243         if (settlementNav == 0) {
1244             revert NavNotUpdate(settlementTime);
1245         }
```

Listing 2.10: src/r25-pc/AsyncVault/AsyncVault.sol

```
1995     if (from == address(0) && to != address(0)) {
1996         // deposit/mint or other mint scenarios
1997         if (accounting.lastAccountingTimestamp == 0) {
1998             revert AccountingNotSet();
1999         }
2000         uint64 unlockTime = SafeCast.toUint64(
2001             accounting.lastAccountingTimestamp + maturityPeriod + 1 days
2002         );
```

Listing 2.11: src/r25-pc/AsyncVault/AsyncVault.sol

```
918  function _autoRenew(
919      address owner,
920      uint64 oldUnlockTime,
921      uint256 shares
922  ) internal {
923      uint64 newUnlockTime = SafeCast.toUint64(oldUnlockTime + maturityPeriod);
924
925      _removeShare(owner, oldUnlockTime);
926      _onReceiveShares(owner, shares, newUnlockTime, 0);
```

Listing 2.12: src/r25-pc/AsyncVault/AsyncVault.sol

Impact This vulnerability may lead to the unavailability of redemption functionality.

Suggestion Validate that `maturityPeriod` is a non-zero multiple of `1 days` during initialization.

2.2 Recommendation

2.2.1 Add an upper-bound check in `setMaxFluctuationBps()`

Status Confirmed

Introduced by `Version 1`

Description In contract `AsyncVault`, functions `setMaxSupply()`, `setMaxDepositLimit()`, and `setMaxFluctuationBps()` lack boundary validation on crucial configurations.

Specifically, functions `setMaxSupply()` and `setMaxDepositLimit()` permit a value of zero, which would cause every subsequent `deposit()` and `mint()` invocation to revert immediately. Function `setMaxFluctuationBps()` has no lower or upper bound. Setting it to zero causes every oracle report to be rejected as the computed `fluctuationBps` will always exceed the limit, permanently blocking NAV updates. Setting it above `FEE_PRECISION` renders the fluctuation check trivially satisfiable, negating its intended protection against abrupt NAV manipulation.

```
1669    function setMaxSupply(
1670        uint256 maxSupply_
1671    ) external onlyRole(BizRole.CONTROLLER) {
1672        uint256 preMaxSupply = maxSupply;
1673        maxSupply = maxSupply_;
1674        emit ConfigUpdated(preMaxSupply, maxSupply_, ConfigType.MAX_SUPPLY);
1675    }
```

Listing 2.13: src/r25-pc/AsyncVault/AsyncVault.sol

```
1684    function setMaxDepositLimit(
1685        uint256 maxDepositLimit_
1686    ) external onlyRole(BizRole.CONTROLLER) {
1687        uint256 preMaxDepositLimit = maxDepositLimit;
1688        maxDepositLimit = maxDepositLimit_;
1689        emit ConfigUpdated(
1690            preMaxDepositLimit,
1691            maxDepositLimit_,
1692            ConfigType.MAX_DEPOSIT_LIMIT
1693        );
1694    }
```

Listing 2.14: src/r25-pc/AsyncVault/AsyncVault.sol

```
1703    function setMaxFluctuationBps(
1704        uint256 maxFluctuationBps_
1705    ) external onlyRole(BizRole.CONTROLLER) {
1706        uint256 preMaxFluctuationBps = maxFluctuationBps;
1707        maxFluctuationBps = maxFluctuationBps_;
1708        emit ConfigUpdated(
1709            preMaxFluctuationBps,
1710            maxFluctuationBps_,
1711            ConfigType.MAX_FLUCTUATION_BPS
1712        );
1713    }
```

Listing 2.15: src/r25-pc/AsyncVault/AsyncVault.sol

Suggestion Enforce clear boundary checks for the functions.

2.2.2 Fix storage gap size

Status Confirmed

Introduced by Version 1

Description The contract AsyncVault declares a storage gap to reserve space for future upgrades. However, since the contract declares some state variables. Therefore, the size of __gap should not be 50 considering OpenZeppelin's convention.

```
2250    /// forge-lint: disable-next-line(mixed-case-variable)
2251    uint256[50] private __gap;
```

Listing 2.16: src/r25-pc/AsyncVault/AsyncVault.sol

Suggestion Recalculate and adjust the __gap length to conform to the standard 50-slot layout.

2.2.3 Add input validation in function createAsyncVault()

Status Confirmed

Introduced by Version 1

Description In the contract AsyncVaultFactory, function createAsyncVault() is expected to enforce meaningful identity parameters for registry keys and ERC-20 metadata. However, it forwards tokenName and tokenSymbol without checks and derives the registry key from keccak256(abi.encodePacked(organization, tokenName)) even when organization is zero and the strings are empty. This lack of validation allows a vault with empty metadata and a zero-organization namespace to be deployed.

```
109    function createAsyncVault(
110        InitVaultParam calldata initVaultParam
111    ) external onlyRole(DEFAULT_ADMIN_ROLE) returns (address) {
112        // Validation and preparation
113        bytes32 tokenNameHashed = _prepareDeployment(initVaultParam);
114
115        // Deploy all contracts
116        DeployedContracts memory deployed = _deployContracts(initVaultParam, tokenNameHashed);
117
118        // Register and emit events
119        _finalizeDeployment(initVaultParam, tokenNameHashed, deployed);
120
121        return deployed.vaultProxy;
122    }
123
124    function _prepareDeployment(
125        InitVaultParam calldata initVaultParam
126    ) internal view returns (bytes32 tokenNameHashed) {
127        tokenNameHashed = keccak256(abi.encodePacked(
128            initVaultParam.organization,
129            initVaultParam.tokenName
130        ));
```

Listing 2.17: src/r25-pc/AsyncVault/AsyncVaultFactory.sol

Suggestion Reject empty tokenName, empty tokenSymbol, and organization == bytes32(0) before deployment.

2.2.4 Emit events before deleting request data

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the function `batchRedeem()`, `request` is a storage reference to `redeemRequests[requestId]`. Once `redeemRequests[requestId]` is deleted from storage, subsequent reads through `request` return default zero values. As a result, events `CustomWithdraw` and `Withdraw` emitted afterward may contain incorrect fields (e.g., `request.shares`). This creates an inconsistency between the actual withdrawal execution and the event data.

```
1230          RedeemRequestDetail storage request = redeemRequests[requestId];
```

Listing 2.18: src/r25-pc/AsyncVault/AsyncVault.sol

```
1267          delete redeemRequests[requestId];
1268
1269          // Withdraw the underlying tokens from swaper
1270          swaper.withdrawByAgent(requestId, custodian, assetsAfterFee);
1271
1272          emit CustomWithdraw(
1273              msg.sender,
1274              request.receiver,
1275              request.owner,
1276              assetsAfterFee,
1277              request.shares,
1278              fee,
1279              requestId,
1280              withdrawalId
1281          );
1282
1283          emit Withdraw(
1284              msg.sender,
1285              request.receiver,
1286              request.owner,
1287              assetsAfterFee,
1288              request.shares
1289          );
```

Listing 2.19: src/r25-pc/AsyncVault/AsyncVault.sol

Suggestion Emit the events before the delete operation.

2.2.5 Revise the incorrect parameters

Status Confirmed

Introduced by `Version 1`

Description 1. In the interface `IERC7540`, event `RedeemRequest` declares the last parameter as `assets`. However, the actual value passed is `shares` in the contract `AsyncVault`. According to the ERC-7540 specification, this parameter semantically represents shares, creating a mismatch between the ABI label and the on-chain data.

2. In contract `AsyncVault`, the error `ExceededMaxSupply` defines its second parameter as `assets`, while the functions `deposit()` and `mint()` actually pass `shares`.

```
102   event RedeemRequest(
103       address indexed controller, address indexed owner, uint256 indexed requestId, address
              sender, uint256 assets
104   );
```

Listing 2.20: src/r25-pc/interface/IERC7540.sol

```
1186      emit RedeemRequest(controller, owner, requestId, msg.sender, shares);
```

Listing 2.21: src/r25-pc/AsyncVault/AsyncVault.sol

```
227   error ExceededMaxSupply(
228       address receiver,
229       uint256 assets,
230       uint256 maxSupply
231   );
```

Listing 2.22: src/r25-pc/AsyncVault/AsyncVault.sol

Suggestion Rename the error parameter from `assets` to `shares`.

2.2.6 Use different boundary conditions for cancellation and renewal

Status Confirmed

Introduced by `Version 1`

Description In the contract `AsyncVault`, functions `batchAutoRenew()`, `renewByUsers()`, `requestRedeem()`, and `cancelRedeemRequest()` are callable at the exact same boundary timestamp of `unlockTime - redeemNoticePeriod`. As a result, both the request, cancellation, and renewal operations are valid simultaneously at the exact boundary point.

```
1196  function _getFirstEligibleShares(
1197      address owner,
1198      uint64 currentTime
1199  ) internal view returns (uint256 shares, uint64 timestamp) {
1200      uint64 unlockTime = _nextTimelock[owner][GUARD];
1201      while (unlockTime != GUARD) {
1202          if (unlockTime > currentTime && unlockTime - currentTime >= redeemNoticePeriod) {
1203              shares = _sharesLocks[owner][unlockTime];
1204              timestamp = unlockTime;
1205              break;
1206          }
1207          unlockTime = _nextTimelock[owner][unlockTime];
1208      }
1209  }
```

Listing 2.23: src/r25-pc/AsyncVault/AsyncVault.sol

```
1325  function _validateAndCancelRequest(
1326      uint256 requestId
```

```
1327    ) internal returns (address owner, uint256 shares, uint64 unlockTime) {
1328        // 1. Get request from storage
1329        RedeemRequestDetail storage request = redeemRequests[requestId];
1330
1331        // 2. Check if there are pending shares (validates request exists)
1332        if (request.shares == 0) {
1333            revert InvalidInput();
1334        }
1335
1336        // 3. Validate caller is the request owner
1337        if (msg.sender != request.owner) {
1338            revert NotAuthorized();
1339        }
1340
1341        // 4. Validate advance notice period
1342        uint64 currentTime = SafeCast.toUint64(block.timestamp);
1343        if (request.unlockTime <= currentTime || request.unlockTime - currentTime <
                redeemNoticePeriod) {
1344            revert OperationTooLate(
1345                request.owner,
1346                request.unlockTime,
1347                currentTime,
1348                redeemNoticePeriod
1349            );
1350        }
```

Listing 2.24: src/r25-pc/AsyncVault/AsyncVault.sol

```
864    function batchAutoRenew(
865        uint64 oldUnlockTime,
866        uint256 batchSize
867    ) external onlyRole(BizRole.CONTROLLER) returns (uint16 processedCount) {
868        if (batchSize == 0) {
869            batchSize = DEFAULT_BATCH_SIZE;
870        }
871
872        if (batchSize > MAX_REQUEST_BATCH) {
873            revert BatchSizeTooLarge(batchSize);
874        }
875
876        if (block.timestamp < oldUnlockTime - redeemNoticePeriod) {
877            revert InvalidRenewTimestamp(oldUnlockTime);
878        }
```

Listing 2.25: src/r25-pc/AsyncVault/AsyncVault.sol

Suggestion Adjust the boundary conditions so that the redeem-request window and the auto-renewal window do not overlap.

2.2.7 Revise incorrect annotations

Status Confirmed

Introduced by Version 1

Description There are some annotations inconsistent with implementations, as follows:

1. In the contract `AsyncVault`, the function `handleOracleReport()` lists `InvalidAccountingTimestamp` and `InvalidNavFluctuation` in `@custom:throws`, while the actual error is `InvalidReport`. In addition, the function `batchRedeem()` lists `MAX_REQUEST_BATCH`, which is a constant rather than an error, and omits the actual error `NavNotUpdate`.

2. Several function descriptions do not match the actual behavior. In the contract `AsyncVault`, the function `toggleReceiverEqSender()` states that the restriction applies to four operations, while it applies only to `deposit()` and `mint()` in the current implementation.

3. The event reference is inconsistent with the emitted event. In the contract `AsyncVault`, the function `setRedeemNoticePeriod()` references `RedeemNoticePeriodUpdated` in `@custom:emits`, while the emitted event is `ConfigUpdated()`.

4. Several NatSpec parameter descriptions are incomplete or inaccurate. In the contract `AsyncVault`, the function `batchAutoRenew()` states that the batch size is "default 50, max 100", while `MAX_REQUEST_BATCH` is actually 50 and `DEFAULT_BATCH_SIZE` is 20. In addition, the contract `AsyncVault` still describes non-existent features such as "Instant withdrawal" and "daily quota".

5. Some minor documentation and naming issues remain. The field `ReportData.submitTimestamp` in the contract `AsyncVault` is defined but never read and `swaper` is a typo. The license header in `AsyncVault` contains the typo "govered".

```
55 * Withdrawal Options:
56 * 1. **Instant withdrawal**: Uses daily quota, processed immediately
57 * 2. **Queued withdrawal**: For amounts exceeding quota, processed in batches
58 *
```

Listing 2.26: src/r25-pc/AsyncVault/AsyncVault.sol

```
1492     * @custom:throws InvalidAccountingTimestamp if report timestamp is in the future
1493     * @custom:throws InvalidNavFluctuation if NAV fluctuation exceeds configured threshold
```

Listing 2.27: src/r25-pc/AsyncVault/AsyncVault.sol

```
1581     * @dev When enabled, requires that the receiver address must equal the sender/owner address
1582     *      in deposit, mint, withdraw, and redeem operations to prevent third-party deposits/
             withdrawals
```

Listing 2.28: src/r25-pc/AsyncVault/AsyncVault.sol

```
1216     * @custom:throws MAX_REQUEST_BATCH if batch size exceed the limitation
```

Listing 2.29: src/r25-pc/AsyncVault/AsyncVault.sol

```
1645     * @custom:emits RedeemNoticePeriodUpdated upon successful update
```

Listing 2.30: src/r25-pc/AsyncVault/AsyncVault.sol

```
405    struct ReportData {
406        // Off-chain report accounting timestamp
407        uint64 accountingTimestamp;
408        // Off-chain report submission timestamp
```

```
409       // depends on Oracle's reporting settlement completion
410       uint64 submitTimestamp;
```

Listing 2.31: src/r25-pc/AsyncVault/AsyncVault.sol

```
3// Use of this software is govered by the Business Source License included in the LICENSE.TXT file
      and at r25.xyz/busl11.
```

Listing 2.32: src/r25-pc/AsyncVault/AsyncVault.sol

Suggestion Revise the annotations above so that they match the actual implementation.

2.2.8 Grant required `WithdrawQueue` roles during deployment

Status Confirmed

Introduced by `Version 1`

Description In the contract `AsyncVaultFactory`, the function `_deployContracts()` deploys and initializes variables `WithdrawQueue`, `Whitelist`, and `AsyncVault`, binding the newly deployed vault to its corresponding `WithdrawQueue`. This binding cannot be modified afterward via the contract. However, the deployment flow does not automatically grant the role `BizRole.VAULT` to the vault proxy in the corresponding `WithdrawQueue` after deployment. This design means that the deployment result still depends on additional manual role configuration.

```
145   function _deployContracts(
146       InitVaultParam calldata initVaultParam,
147       bytes32 tokenNameHashed
148   ) internal returns (DeployedContracts memory deployed) {
149       // Set logic
150       deployer.setLogic(tokenNameHashed, vaultAsyncLogic);
151
152       bytes32 wqNameHashed = keccak256(abi.encodePacked(
153           initVaultParam.organization,
154           initVaultParam.tokenName,
155           "_WQ"
156       ));
157       deployer.setLogic(wqNameHashed, withdrawQueueLogic);
158
159       bytes32 whitelistNameHashed = keccak256(abi.encodePacked(
160           initVaultParam.organization,
161           initVaultParam.tokenName,
162           "_WL"
163       ));
164       deployer.setLogic(whitelistNameHashed, whitelistLogic);
165
166       deployed.withdrawQueueProxy = _deploy(
167           wqNameHashed,
168           initVaultParam.organization,
169           abi.encodeCall(
170               IWithdrawQueue.initialize,
171               (initVaultParam.owner, initVaultParam.asset)
172           )
```

```
173         );
174
175         deployed.whitelistProxy = _deploy(
176             whitelistNameHashed,
177             initVaultParam.organization,
178             abi.encodeCall(
179                 IWhitelist.initialize,
180                 (initVaultParam.owner)
181             )
182         );
183
184         deployed.vaultProxy = _deploy(
185             tokenNameHashed,
186             initVaultParam.organization,
187             abi.encodeCall(
188                 IAsyncVault.initialize,
189                 IAsyncVault.InitializeParams({
190                     admin: initVaultParam.owner,
191                     asset: initVaultParam.asset,
192                     swaper: initVaultParam.swaper,
193                     withdrawQueue: deployed.withdrawQueueProxy,
194                     whitelist: deployed.whitelistProxy,
195                     maturityPeriod: initVaultParam.maturityPeriod,
196                     tokenName: initVaultParam.tokenName,
197                     tokenSymbol: initVaultParam.tokenSymbol
198                 })
199             )
200         );
201     }
```

Listing 2.33: src/r25-pc/AsyncVault/AsyncVaultFactory.sol

```
134     function requestWithdrawal(
135         uint256 amount,
136         address recipient,
137         address owner,
138         uint256 redeemRequestId
139     ) external override onlyRole(BizRole.VAULT) returns (uint256 requestId) {
```

Listing 2.34: src/r25-pc/AsyncVault/WithdrawQueue.sol

Suggestion Grant the role `BizRole.VAULT` to the vault proxy in the corresponding `WithdrawQueue` during deployment.

Feedback from the project The project confirmed that role configuration will be implemented atomically during deployment.

2.2.9 Add a minimum check on `redeemNoticePeriod`

Status Confirmed

Introduced by `Version 1`

Description In the contract `AsyncVault`, users may submit or cancel redeem requests before `unlockTime - redeemNoticePeriod`. This parameter `redeemNoticePeriod` is intended to lock the user's redemption decision before settlement. Meanwhile, the protocol settles redemption with the value of `dailyNav` at `unlockTime - 1 days`. However, function `setRedeemNoticePeriod()` does not enforce a lower bound of 1 day on `redeemNoticePeriod`. If `redeemNoticePeriod` is set to less than `1 days`, users can know the settlement NAV and decide whether to submit, maintain, or cancel redeem requests. This behavior may conflict with the notice-based redemption model that is commonly expected in fund-like or bond-like products.

```
1647    function setRedeemNoticePeriod(
1648        uint64 redeemNoticePeriod_
1649    ) external onlyRole(BizRole.CONTROLLER) {
1650        if (redeemNoticePeriod_ > maturityPeriod) {
1651            revert InvalidInput();
1652        }
1653        uint64 preRedeemNoticePeriod = redeemNoticePeriod;
1654        redeemNoticePeriod = redeemNoticePeriod_;
1655        emit ConfigUpdated(
1656            preRedeemNoticePeriod,
1657            redeemNoticePeriod_,
1658            ConfigType.REDEEM_NOTICE_PERIOD
1659        );
1660    }
```

Listing 2.35: src/r25-pc/AsyncVault/AsyncVault.sol

Suggestion Add a minimum value check for `redeemNoticePeriod`.

Feedback from the project The project stated that they would ensure the `redeemNoticePeriod` is greater than 1 day.

2.3 Note

2.3.1 Token assumption

Introduced by `Version 1`

Description This project only supports assets with a decimal place hardcoded to `6`. Specifically, the contract `AsyncVault` enforces a 6-decimal asset during initialization, and the share token decimals are also fixed to `6`. Meanwhile, the project assumes that the underlying asset follows standard ERC20 semantics with exact nominal accounting. Therefore, non-standard tokens (e.g., Fee-on-Transfer tokens or rebasing tokens) are outside the supported scope. If such tokens are used, the related deposit, withdrawal, redemption, and accounting flows may deviate from the intended behavior and produce unexpected results.

2.3.2 Inconsistency among vault logic due to implementation updates

Introduced by `Version 1`

Description In contract `AsyncVaultFactory`, function `setVaultAsyncLogic()` allows modifying the implementation logic of vault contracts. However, only vaults deployed after this change

adopt the updated logic, leaving inconsistencies with previously deployed vaults. Therefore, the project should carefully perform upgrades to existing vaults via privileged roles if all vault instances should retain the same implementation logic.

2.3.3 Deviations from the EIP-4626 standard

Introduced by Version 1

Description The contract `AsyncVault` deviates from the EIP-4626 specification as a deliberate design decision. It introduces a state variable `isReceiverEqSender` to control whether the receiver must equal the caller for certain operations. When enabled, function `deposit()` and `mint()` enforce the receiver to be `_msgSender()`. Integrators should account for these constraints accordingly.

2.3.4 Centralized withdrawal mechanism

Introduced by Version 1

Description In this project, the vault does not hold user assets. The withdrawal mechanism relies on a centralized arrangement involving the `swaper`, and `payer`. Upon deposit, user assets are forwarded from the vault to the `swaper` rather than retained in the vault. It is assumed that the `swaper` manages the deposited assets off-chain and periodically replenishes the `payer` to fulfill withdrawal requests. As a result, successful withdrawals require that this address consistently maintain sufficient liquidity and grant adequate allowance to the vault. This creates a single point of operational dependency, and the funds and approval configuration must therefore be managed with strict care at all times.

2.3.5 Ensure the correctness and timeliness of Oracle reports

Introduced by Version 1

Description In contract `AsyncVault`, the `ORACLE_REPORTER` can invoke the function `handleOracleReport()` to submit updated NAV data. Sequential report validation is only enforced after both `lastAccountingTimestamp` and `lastReportId` are non-zero, and NAV deviation checks apply only after a non-zero NAV has been submitted. Additionally, since NAV updates are applied asynchronously on-chain, a front-running attack may arise where an attacker can observe a pending NAV-increasing report transaction in the mempool. The attacker could deposit before the report is executed to mint shares at the old NAV, and then profit immediately from the increased NAV after execution.

Therefore, the project must ensure that `ORACLE_REPORTER` submits reports in a timely manner to preserve correct NAV-based conversion, and configure fees and lock-up periods appropriately to minimize the impact of front-running attacks.

Feedback from the project The project stated that the impact of front-running is acceptable.

2.3.6 Unused `controller` field

Introduced by Version 1

Description In contract `AsyncVault`, the `controller` address passed to function `requestRedeem()` can be specified arbitrarily, and it is later written into the request state and events. However, subsequent request processing is performed by the project-side role `BizRole.CONTROLLER`, while the user-specified `controller` does not participate in authorization checks or execution logic. Meanwhile, the code comment already states that `RedeemRequestDetail.controller` is not used. Therefore, integrators and developers should not assume that this field has operational meaning.

2.3.7 Share token transfer restriction

Introduced by `Version 1`

Description Contract `AsyncVault` overrides the function `_update()` but does not apply the modifiers `whenNotPaused` and `onlyWhitelisted` to the function. This means that the transfer operation is not subject to emergency pause mechanisms and whitelist access controls.

```
1983    function _update(
1984        address from,
1985        address to,
1986        uint256 value
1987    ) internal override {
```

Listing 2.36: src/r25-pc/AsyncVault/AsyncVault.sol

2.3.8 Potential centralization risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `DEFAULT_ADMIN_ROLE`, `CONTROLLER`, `ORACLE_REPORTER_ROLE`, and the upgrade operators in `TransparentUpgradeDeployer`) can conduct sensitive operations, which introduces potential centralization risks. For example, the `DEFAULT_ADMIN_ROLE` can grant any role to any address, including itself, giving it ultimate control over all role assignments. The `CONTROLLER` role can widen or disable NAV fluctuation checks via `maxFluctuationBps`, pause all vault operations, and extend user share lock periods via functions `batchAutoRenew()` or `renewByUsers()`. Through the contract `WithdrawQueue`, privileged operators can also selectively skip or fulfill withdrawal requests out of order. The `ORACLE_REPORTER_ROLE` directly controls NAV updates that determine share valuation, and since `CONTROLLER` can widen the fluctuation threshold arbitrarily, the Oracle restraint is ultimately discretionary. Upgrade operators can replace all vault logic via the transparent proxy pattern. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.9 Off-chain NAV treatment of pending redemption shares

Introduced by `Version 1`

Description In contract `AsyncVault`, submitted redemption shares are transferred to the vault itself. These shares remain counted in function `totalSupply()` until burned via function `batchRedeem()`, which executes only after the settlement NAV is available.

Since redemptions settle against the NAV of `unlockTime - 1 days`, the project must ensure pending shares are included in the NAV calculation through `unlockTime - 1 days`, but excluded from periods after that point.

Feedback from the project The project confirmed that NAV quotes are produced independently off-chain, and will exclude pending shares from NAV calculation after `unlockTime - 1 days`.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS