BLOCKSEC

Security Audit

Report for xaue-lending

Date: June 11, 2026 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction **1**

1.1 About the Audit Target . . . 1
1.2 Disclaimer . . . 2
1.3 Procedure of Auditing . . . 2
1.3.1 Security Issues . . . 2
1.3.2 Additional Recommendation . . . 3
1.4 Security Model . . . 3

Chapter 2 Findings **5**

2.1 Security Issue . . . 5
2.1.1 Potential DoS due to mismatched function signatures . . . 5
2.1.2 Incorrect return value in function `previewMint()` . . . 6
2.1.3 Incorrect logic in function `claimRejectedRedemption()` . . . 7
2.1.4 Stale pricing due to immutable oracle . . . 8
2.1.5 Lack of `protocolSurplusXaue` restoration on rejection . . . 9
2.1.6 Potential overwrite of the `vaultByPartner` mapping . . . 10
2.1.7 Arbitrary receiver of protocol surplus . . . 10
2.2 Recommendation . . . 11
2.2.1 Add allowance revocation logic . . . 11
2.2.2 Return redemption ID and expose redemption status . . . 12
2.2.3 Add zero-address validations in contract `LendingShare` . . . 13
2.2.4 Add errors for pause status . . . 13
2.2.5 Remove redundant code . . . 14
2.3 Note . . . 14
2.3.1 Potential centralization risks . . . 14
2.3.2 Assumptions on external contracts . . . 15
2.3.3 NAV should be monotonically increasing . . . 15

Report Manifest

Item	Description
Client	XAUE Lab
Target	xaue-lending

Version History

Version	Date	Description
1.0	June 11, 2026	First release

Signature

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

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2026-06-12 03:57:54+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 xaue-lending of XAUE Lab.

XAUE-lending is a tokenized gold lending vault that allows users to deposit Tether Gold (i.e., XAUT) and receive XAUE-backed lending shares (i.e., xLEND), which can be used as collateral in other protocols. The protocol converts deposited XAUT into XAUE via an external Agent Router, tracks value growth through a Cobo Fund Oracle NAV price feed, and splits any NAV appreciation surplus between users and the protocol. Redemptions are asynchronous: users request redemption, the vault routes XAUE back through the Router, and users later claim XAUT.

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

- contracts/core/libraries/FundConvert.sol
- contracts/core/LendingFactory.sol
- contracts/core/LendingShare.sol
- contracts/core/LendingVault.sol
- contracts/products/xaue-lending/XaueLendingRoles.sol

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 1`), 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
xaue-lending	`Version 1`	`5cdec4303912b3d91e35108b6d1600bb9ebfe5a4`
	`Version 2`	`6ba721b64abf7d3cf923c20d9cd32ff45b0a4cef`

[1] https://github.com/XAUElab/xaue-lending

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.

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.

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

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

- **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.

Chapter 2 Findings

In total, we found **seven** potential security issues. Besides, we have **five** recommendations and **three** notes.

- High Risk: 1
- Medium Risk: 3
- Low Risk: 3
- Recommendation: 5
- Note: 3

ID	Severity	Description	Category	Status
1	High	Potential DoS due to mismatched function signatures	Security Issue	Fixed
2	Medium	Incorrect return value in function `previewMint()`	Security Issue	Fixed
3	Medium	Incorrect logic in function `claimRejectedRedemption()`	Security Issue	Fixed
4	Medium	Stale pricing due to immutable oracle	Security Issue	Fixed
5	Low	Lack of `protocolSurplusXaue` restoration on rejection	Security Issue	Fixed
6	Low	Potential overwrite of the `vaultByPartner` mapping	Security Issue	Fixed
7	Low	Arbitrary receiver of protocol surplus	Security Issue	Fixed
8	-	Add allowance revocation logic	Recommendation	Fixed
9	-	Return redemption ID and expose redemption status	Recommendation	Fixed
10	-	Add zero-address validations in contract `LendingShare`	Recommendation	Fixed
11	-	Add errors for pause status	Recommendation	Fixed
12	-	Remove redundant code	Recommendation	Fixed
13	-	Potential centralization risks	Note	-
14	-	Assumptions on external contracts	Note	-
15	-	NAV should be monotonically increasing	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Potential DoS due to mismatched function signatures

Severity High

Status Fixed in `Version 2`

Introduced by Version 1

Description In contract LendingVault, functions _routerMintXaue() and _routerRequestRedeem() invoke mint(uint256 xautAmount) and requestRedeem(uint256 xaueAmount) on the contract AgentRouter through the interface IXAUERouter. However, contract AgentRouter declares mint(uint256 xautAmount, uint256 minXaueOut) and requestRedeem(uint256 xaueAmount, uint256 minXautOut). Since Solidity derives the call selector from the interface declaration, the mismatched function signatures target a non-existent function. As a result, both functions always revert, disabling all deposit and redemption functionality.

```
 5interface IXAUERouter {
 6    function mint(uint256 xautAmount) external returns (uint256 xaueAmount);
 7
 8    function requestRedeem(uint256 xaueAmount) external returns (uint256 routerReqId);
 9
10    function getUnderlyingRequestId(uint256 routerReqId) external view returns (uint256);
11
12    function claimXaut(uint256 routerReqId) external;
13
14    function claimRejectedShares(uint256 routerReqId) external;
15}
```

Listing 2.1: contracts/core/interfaces/IXAUERouter.sol

Impact All deposits and redemptions revert, resulting in a DoS.

Suggestion Align the interface IXAUERouter with the contract AgentRouter.

2.1.2 Incorrect return value in function previewMint()

Severity Medium

Status Fixed in Version 2

Introduced by Version 1

Description In the contract LendingVault, function previewMint() does not comply with the EIP-4626 requirement to use ceiling rounding. Specifically, function previewMint() invokes _previewMint() to compute and return required assets, which uses floor rounding. Meanwhile, the function mint() passes _previewMint()'s return value to function _deposit(), which re-calculates shares that can be minted (i.e., minted). It reverts when minted is less than input shares. Consequently, function _previewMint() may return a non-zero asset amount for a share quantity that would be rejected by function mint().

```
285    function mint(uint256 shares, address receiver) external nonReentrant returns (uint256 assets)
           {
286        if (depositPaused) revert LendingVaultBadInput();
287        if (shares == 0) revert LendingVaultBadInput();
288        assets = _previewMint(shares);
289        uint256 minted = _deposit(assets, receiver);
290        if (minted < shares) revert LendingVaultBadInput();
291        return assets;
292    }
```

Listing 2.2: contracts/core/LendingVault.sol

```
262   function _previewMint(uint256 shares) internal view returns (uint256) {
263       if (shares == 0) return 0;
264       return _sharesToAssetsByIndex(shares, _currentXlendIndex());
265   }
266
267   function previewMint(uint256 shares) external view returns (uint256) {
268       return _previewMint(shares);
269   }
```

Listing 2.3: contracts/core/LendingVault.sol

```
458   function _deposit(uint256 xautAmount, address receiver) internal returns (uint256 sharesOut) {
459       if (depositPaused) revert LendingVaultBadInput();
460       if (xautAmount == 0 || receiver == address(0)) revert LendingVaultBadInput();
461
462       _accrueOracleSurplus();
463
464       sharesOut = _assetsToSharesByIndex(xautAmount, xlendIndex);
465       if (sharesOut == 0) revert LendingVaultBadInput();
466
467       require(xaut.transferFrom(msg.sender, address(this), xautAmount), "vault: xaut pull");
468       uint256 xaueReceived = _routerMintXaue(xautAmount);
469
470       totalXaue += xaueReceived;
471       share.mint(receiver, sharesOut);
472
473       emit Deposited(receiver, xautAmount, xaueReceived, sharesOut);
474   }
```

Listing 2.4: contracts/core/LendingVault.sol

Impact Integrators who compute the approval amount using function `previewMint()` may encounter reverts during the mint process.

Suggestion Revise the code logic accordingly.

2.1.3 Incorrect logic in function `claimRejectedRedemption()`

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `LendingVault`, the function `claimRejectedRedemption()` restores the returned `XAUE` and mints `xLEND` to the user. However, `xlendIndex` may increase during the pending period, and this increase does not include the NAV appreciation accrued on the `XAUE` in the pending redemption request. When the request is rejected, the function `claimRejectedRedemption()` directly mints the original `xLEND` share amount (i.e., the recorded `r.routedShares`), instead of recalculating the `xLEND` amount based on the current value of the returned `XAUE`

and the current `xlendIndex`. Meanwhile, it does not allocate the returned `XAUE` appreciation to `protocolSurplusXaue`. As a result, part of the appreciation in the returned `XAUE` is not allocated to users or the protocol.

```
572   function claimRejectedRedemption(uint256 id) external nonReentrant {
573       Redemption storage r = redemptions[id];
574       if (r.user != msg.sender) revert LendingVaultUnauthorized();
575       if (r.status != RedemptionStatus.RouterPending) revert LendingVaultBadStatus();
576
577       _accrueOracleSurplus();
578       uint256 beforeBal = xaue.balanceOf(address(this));
579       router.claimRejectedShares(r.routerReqId);
580       uint256 xaueReturned = xaue.balanceOf(address(this)) - beforeBal;
581       if (xaueReturned == 0) revert LendingVaultBadInput();
582
583       totalXaue += xaueReturned;
584       uint256 returnedShares = r.routedShares;
585       r.routedShares = 0;
586       r.routedSupply = 0;
587       r.status = RedemptionStatus.Rejected;
588       share.mint(msg.sender, returnedShares);
589
590       emit RedemptionRejected(id, xaueReturned, returnedShares);
591   }
```

Listing 2.5: contracts/core/LendingVault.sol

Impact Part of the appreciation in the `XAUE` returned after rejection is not allocated to users or the protocol and remains stuck in the vault.

Suggestion Revise the code logic accordingly.

2.1.4 Stale pricing due to immutable oracle

Severity Medium

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `LendingVault`, the variable `oracle` is immutable and should match the oracle used by the `XAUE` token, since the vault uses `oracle.getLatestPrice()` to price shares. However, the `XAUE` token can update its oracle, while the vault cannot. As a result, after the token updates its oracle, the vault keeps reading a stale one and misprices shares during deposits and redemptions.

```
14contract LendingVault is IERC4626Minimal {
15   uint8 internal constant XAUT_DECIMALS = 6;
16   uint8 internal constant XAUE_DECIMALS = 18;
17
18   IERC20Minimal public immutable xaut;
19   IERC20Minimal public immutable xaue;
20   IXAUERouter public immutable router;
21   ICoboFundOracle public immutable oracle;
```

Listing 2.6: contracts/core/LendingVault.sol

```
159   function _navPricing() internal view returns (uint256 nav, uint8 assetDec, uint8 shareDec) {
160       nav = oracle.getLatestPrice();
161       assetDec = XAUT_DECIMALS;
162       shareDec = XAUE_DECIMALS;
163   }
```

Listing 2.7: contracts/core/LendingVault.sol

Impact Shares may be mispriced using a stale oracle.

Suggestion Read the current oracle from `XAUE` instead of an immutable variable.

2.1.5 Lack of `protocolSurplusXaue` restoration on rejection

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `LendingVault`, the function `_routeRedemptionToRouter()` may use `protocolSurplusXaue` to cover the `XAUE` gap when a redemption exceeds the user reserve (line 516). It decreases both `protocolSurplusXaue` and `totalXaue` in this case. On rejection, the function `claimRejectedRedemption()` restores `totalXaue` and re-mints the shares. However, it does not restore the consumed `protocolSurplusXaue`. As a result, `userXaueReserve` includes the unrecovered gap, crediting the protocol surplus to users.

```
512       uint256 userReserve = _userXaueReserve();
513       if (xaueOut > userReserve) {
514           uint256 reserveGap = xaueOut - userReserve;
515           if (reserveGap > protocolSurplusXaue) revert LendingVaultBadInput();
516           protocolSurplusXaue -= reserveGap;
517       }
```

Listing 2.8: contracts/core/LendingVault.sol

```
572   function claimRejectedRedemption(uint256 id) external nonReentrant {
573       Redemption storage r = redemptions[id];
574       if (r.user != msg.sender) revert LendingVaultUnauthorized();
575       if (r.status != RedemptionStatus.RouterPending) revert LendingVaultBadStatus();
576
577       _accrueOracleSurplus();
578       uint256 beforeBal = xaue.balanceOf(address(this));
579       router.claimRejectedShares(r.routerReqId);
580       uint256 xaueReturned = xaue.balanceOf(address(this)) - beforeBal;
581       if (xaueReturned == 0) revert LendingVaultBadInput();
582
583       totalXaue += xaueReturned;
584       uint256 returnedShares = r.routedShares;
585       r.routedShares = 0;
586       r.routedSupply = 0;
```

```
587         r.status = RedemptionStatus.Rejected;
588         share.mint(msg.sender, returnedShares);
589
590         emit RedemptionRejected(id, xaueReturned, returnedShares);
```

Listing 2.9: contracts/core/LendingVault.sol

Impact The protocol surplus is permanently credited to users, inflating the user reserve.

Suggestion In function `claimRejectedRedemption()`, restore `protocolSurplusXaue` by the same gap consumed during routing.

2.1.6 Potential overwrite of the `vaultByPartner` mapping

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `LendingFactory`, function `createPool()` assigns a vault to `vaultByPartner[cfg.partnerId]` whenever `cfg.partnerId` is non-zero. However, it does not verify whether `cfg.partnerId` is already mapped to a vault. As a result, creating a vault with an existing `partnerId` silently overwrites the previous vault address.

```
81          if (cfg.partnerId != bytes32(0)) {
82              vaultByPartner[cfg.partnerId] = vault;
83          }
```

Listing 2.10: contracts/core/LendingFactory.sol

Impact An existing partner-to-vault mapping can be overwritten, causing integrators that rely on `vaultByPartner` to resolve to the wrong vault.

Suggestion Add a duplicate check before assigning `vaultByPartnerId`.

2.1.7 Arbitrary receiver of protocol surplus

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In contract `LendingVault`, functions `harvestProtocolSurplusXaue()` and `harvest()` transfer protocol surplus to an arbitrary non-zero `to` address supplied by the admin instead of transferring it to the configured `treasury` address. Although the `treasury` is initialized to a non-zero address during deployment, it is never referenced in these functions. This allows the admin to direct protocol surplus to any address, and introduces the risk of fund misallocation if an incorrect address is provided.

```
103         if (xaut_ == address(0) || xaue_ == address(0) || router_ == address(0) || oracle_ ==
                address(0) || treasury_ == address(0) || admin_ == address(0)) {
104             revert LendingVaultBadInput();
105         }
```

```
106         if (surplusUserBps_ > 10_000) revert LendingVaultBadInput();
107         xaut = IERC20Minimal(xaut_);
108         xaue = IERC20Minimal(xaue_);
109         router = IXAUERouter(router_);
110         oracle = ICoboFundOracle(oracle_);
111         treasury = treasury_;
```

Listing 2.11: contracts/core/LendingVault.sol

```
616     function harvestProtocolSurplusXaue(address to, uint256 amount) external onlyAdmin
            nonReentrant {
617         if (to == address(0) || amount == 0 || amount > protocolSurplusXaue) revert
                LendingVaultBadInput();
618         protocolSurplusXaue -= amount;
619         totalXaue -= amount;
620         require(xaue.transfer(to, amount), "vault: surplus xaue xfer");
621         emit ProtocolSurplusHarvested(to, amount);
622     }
```

Listing 2.12: contracts/core/LendingVault.sol

```
599     function harvest(address to) external onlyAdmin nonReentrant {
600         if (to == address(0)) revert LendingVaultBadInput();
601         uint256 h = harvestable();
602         if (h == 0) revert LendingVaultBadInput();
603         require(xaut.transfer(to, h), "vault: harvest xfer");
604         emit Harvest(to, h);
605     }
```

Listing 2.13: contracts/core/LendingVault.sol

Impact The protocol surplus may be misallocated.

Suggestion Transfer protocol surplus to the `treasury` address.

2.2 Recommendation

2.2.1 Add allowance revocation logic

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `LendingVault`, the functions `_routerMintXaue()` and `_routerRequestRedeem()` approve the router to spend `XAUT` and `XAUE`. These approvals are set through the function `_approveMax()`, which may leave large residual allowances after the router operation completes. It is recommended to revoke the allowance after each mint or redemption request.

```
436     function _routerRequestRedeem(uint256 xaueAmount) internal returns (uint256 routerReqId) {
437         _approveMax(xaue, address(router));
438         routerReqId = router.requestRedeem(xaueAmount);
439     }
```

Listing 2.14: contracts/core/LendingVault.sol

```
428   function _routerMintXaue(uint256 xautAmount) internal returns (uint256 xaueOut) {
429       _approveMax(xaut, address(router));
430       uint256 beforeBal = xaue.balanceOf(address(this));
431       router.mint(xautAmount);
432       xaueOut = xaue.balanceOf(address(this)) - beforeBal;
433       if (xaueOut == 0) revert LendingVaultBadInput();
434   }
```

Listing 2.15: contracts/core/LendingVault.sol

Suggestion Reset the allowance to 0 after the router operation completes.

2.2.2 Return redemption ID and expose redemption status

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `LendingVault`, each redemption request is identified by an ID. However, the function `requestRedeem()` does not return it. Additionally, the contract does not provide a dedicated view function that exposes user-facing redemption information such as the current status and claimable amount in a convenient form.

```
477   function requestRedeem(uint256 shareAmount) external nonReentrant {
478       if (redeemPaused) revert LendingVaultBadInput();
479       if (shareAmount == 0) revert LendingVaultBadInput();
480
481       _accrueOracleSurplus();
482
483       require(share.transferFrom(msg.sender, address(this), shareAmount), "vault: share pull");
484
485       uint256 id = nextRedemptionId++;
486       redemptions[id] = Redemption({
487           user: msg.sender,
488           shares: shareAmount,
489           routedShares: 0,
490           routedSupply: 0,
491           routerReqId: 0,
492           claimableXaut: 0,
493           status: RedemptionStatus.RouterPending
494       });
495
496       _routeRedemptionToRouter(id);
497       emit RedemptionRequested(id, msg.sender, shareAmount);
498   }
```

Listing 2.16: contracts/core/LendingVault.sol

Suggestion Return the generated `id` from the function `requestRedeem()`, and add a view function to query whether the corresponding redemption is claimable and the claimable amount.

2.2.3 Add zero-address validations in contract LendingShare

Status Fixed in Version 2

Introduced by Version 1

Description In contract LendingShare, function _transfer() does not validate the recipient against the zero address. If the user transfers the share to address(0), the function increases balanceOf[address(0)] without decreasing totalSupply and emits the same Transfer(from, address(0)) event as a burn operation. An off-chain device would treat it as a burn event. Similarly, the function _mint() does not validate the recipient against the zero address. It is recommended to add a zero-address check in functions _transfer() and _mint().

```
80  function _transfer(address from, address to, uint256 value) internal {
81      require(balanceOf[from] >= value, "LendingShare: balance");
82      unchecked {
83          balanceOf[from] -= value;
84          balanceOf[to] += value;
85      }
86      emit Transfer(from, to, value);
87  }
```

Listing 2.17: contracts/core/LendingShare.sol

```
65  function _mint(address to, uint256 value) internal {
66      totalSupply += value;
67      balanceOf[to] += value;
68      emit Transfer(address(0), to, value);
69  }
```

Listing 2.18: contracts/core/LendingShare.sol

Suggestion Add a zero-address check in functions _transfer() and _mint().

2.2.4 Add errors for pause status

Status Fixed in Version 2

Introduced by Version 1

Description In contract LendingVault, functions _deposit() and requestRedeem() revert with the error LendingVaultBadInput when the deposit or withdrawal is paused. This error name is misleading as it implies a failure due to incorrect user input.

```
458  function _deposit(uint256 xautAmount, address receiver) internal returns (uint256 sharesOut) {
459      if (depositPaused) revert LendingVaultBadInput();
```

Listing 2.19: contracts/core/LendingVault.sol

```
477  function requestRedeem(uint256 shareAmount) external nonReentrant {
478      if (redeemPaused) revert LendingVaultBadInput();
```

Listing 2.20: contracts/core/LendingVault.sol

Suggestion Add an error for pause status and emit it in the functions.

2.2.5 Remove redundant code

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `LendingVault`, the function `claimRedemption()` contains a branch that handles redemptions with `r.status == RedemptionStatus.Claimable`. This branch decrements `reservedXaut` and pays `r.claimableXaut` to the caller. However, this branch is unreachable since no code path sets a redemption status to `Claimable`, and neither `r.claimableXaut` nor `reservedXaut` is ever incremented. It is recommended to remove the unreachable branch.

```
533   function claimRedemption(uint256 id) external nonReentrant {
534       Redemption storage r = redemptions[id];
535       if (r.user != msg.sender) revert LendingVaultUnauthorized();
536
537       uint256 payout;
538       if (r.status == RedemptionStatus.RouterPending) {
539           uint256 xautGross = _routerClaimXaut(r.routerReqId);
540           payout = _finalizeRedemptionPayout(id, r.routedShares, r.routedSupply, xautGross, msg.
                  sender);
541       } else if (r.status == RedemptionStatus.Claimable) {
542           payout = r.claimableXaut;
543           r.claimableXaut = 0;
544           r.status = RedemptionStatus.Claimed;
545           if (reservedXaut < payout) revert LendingVaultBadInput();
546           unchecked {
547               reservedXaut -= payout;
548           }
549           require(xaut.transfer(msg.sender, payout), "vault: claim xfer");
550       } else {
551           revert LendingVaultBadStatus();
552       }
553
554       emit RedemptionClaimed(id, msg.sender, payout);
555   }
```

Listing 2.21: contracts/core/LendingVault.sol

Suggestion Remove the redundant code.

2.3 Note

2.3.1 Potential centralization risks

Introduced by `Version 2`

Description In this project, several privileged roles (e.g., `admin`) can conduct sensitive operations, which introduces potential centralization risks. For example, the vault `admin` can transfer admin rights through the function `setAdmin()`, pause deposits and redemptions through the function `setPaused()`, adjust the surplus split through the function `setSurplusUserBps()`, update the treasury through the function `setTreasury()`, withdraw accrued surplus through the

functions `harvest()` and `harvestProtocolSurplusXaue()`, and rescue accidentally transferred tokens through the function `rescueERC20()`. Meanwhile, the factory `admin` can deploy new pools through the function `createPool()`. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

2.3.2 Assumptions on external contracts

Introduced by `Version 1`

Description In contract `LendingVault`, the deposit, redemption, and pricing flows rely on external contracts that are not implemented in the codebase. This includes the router interface `IXAUERouter`, and the underlying `XAUE` token represented by the interface `ICoboFundToken`. They are configured as immutable addresses during deployment, and the implementations are assumed to match the contracts audited in the following audit reports: blocksec_cobo_cobotokenization_signed_20260401 [1] and blocksec_baelor_agent_router_signed_20260527 [2].

2.3.3 NAV should be monotonically increasing

Introduced by `Version 1`

Description Contract `LendingVault` is implemented under the assumption that NAV values only increase. Specifically, the variable `xlendIndex` only increases when the oracle-reported NAV increases, and it is never decreased. If the real NAV decreases and later recovers, this index may cause the index-implied `XAUT` value to exceed the actual backing assets. To ensure proper conversion between shares and the underlying assets, the project must ensure that the NAV is strictly monotonically increasing in contract `CoboFundOracle`.

Feedback from the project The project states that contract `CoboFundOracle` was validated in a prior BlockSec audit [3], and enforces a strictly monotonically increasing NAV.

[1] https://github.com/blocksecteam/audit-reports/blob/main/solidity/blocksec_cobo_cobotokenization_signed_20260401.pdf

[2] https://github.com/blocksecteam/audit-reports/blob/main/solidity/blocksec_baelor_agent_router_signed_20260527.pdf

[3] https://github.com/blocksecteam/audit-reports/blob/main/solidity/blocksec_cobo_cobotokenization_signed_20260401.pdf


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS