



Security Audit

Report for xAsset and yBridge Contracts

Date: October 8, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Chapter 1 Introduction	1
1.1 About Target Contracts	1
1.2 Disclaimer	1
1.3 Procedure of Auditing	2
1.3.1 Software Security	2
1.3.2 DeFi Security	2
1.3.3 NFT Security	3
1.3.4 Additional Recommendation	3
1.4 Security Model	3
Chapter 2 Findings	5
2.1 DeFi Security	6
2.1.1 Lack of chain ID in the signature for settlement-related contracts	6
2.1.2 Reusable signature in the <code>claimFees</code> function	7
2.1.3 Potential reentrancy risk due to the callback mechanism of the underlying token	9
2.1.4 Potentially disrupted fee collection due to the rescue of native tokens	11
2.1.5 User-controlled <code>aggregatorAdaptor</code> in the <code>SwapRequested</code> event	11
2.2 Additional Recommendation	13
2.2.1 Unify the method for native token transfers	13
2.2.2 Remove redundant value assignment	13
2.2.3 Add input validation check for the <code>refund</code> function	14
2.2.4 Initialize <code>completeDepositGasLimit</code> in <code>YBridgeVaultV3</code> 's constructor	15
2.2.5 Revise the misleading error message <code>NotEnoughGasFee</code>	15
2.2.6 Add checks to prevent zero transfers in the <code>YBridgeVaultV3</code> contract	16
2.2.7 Add a <code>Nonexist</code> status to the <code>RequestStatus</code> enum	16
2.2.8 Add a check for <code>threshold</code> in <code>Supervisor</code> 's constructor	17
2.3 Note	17
2.3.1 Potential centralization risks	17
2.3.2 Potential off-chain risks	18
2.3.3 Ensure the fund migration when changing a vault address	18
2.3.4 Avoid dependence on <code>RebalanceRewardCalculation</code> for reward calculation	18
2.3.5 Potential DoS due to insufficient check on the <code>_swapStatus</code>	18

Report Manifest

Item	Description
Client	XY Finance
Target	xAsset and yBridge Contracts

Version History

Version	Date	Description
1.0	October 8, 2024	First release

Signature

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 Target Contracts

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

The focus of this audit is on the xAsset Bridge¹ and yBridge² of the XY Finance. XY Finance is a cross-chain interoperability protocol that enables projects and users to bridge and swap assets across different chains. The xAsset Bridge allows projects to transfer their tokens to supported chains using Mint & Burn and Pool-based methods. yBridge supports cross-chain swap services between various peripheral chains through a network of validators, ensuring correct and secure settlement for all cross-chain requests. Additionally, yBridge allows liquidity providers to earn fees by contributing liquidity to the vault.

Please note that the audit scope is limited to smart contracts in the following folders:

- `xasset-contract/src`
- `ybridge-contracts-audit/contracts`

Files intended for testing purposes, specifically those found in the `xasset-contract/src/mocks` and `ybridge-contracts-audit/contracts/mocks` directories, are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of 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.

Project	Version	Commit Hash
xAsset Bridge	Version 1	62ab0f53c159b5e11dedfdb8cf2c8b10109207f
	Version 2	edd8837fe730284211728e5564122e7ce08c57f8
yBridge	Version 1	626f117bb1d298f6f34203ff27f94f8977c89f5b
	Version 2	fc93662b638c886ffcced318979fe48e6ae58074

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.

¹<https://github.com/XY-Finance/xasset-contract>

²<https://github.com/XY-Finance/ybridge-contracts-audit>

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 smart contracts, 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 smart contracts.

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 smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts 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 Software Security

- * Reentrancy
- * DoS
- * Access control
- * Data handling and data flow
- * Exception handling
- * Untrusted external call and control flow
- * Initialization consistency
- * Events operation
- * Error-prone randomness
- * Improper use of the proxy system

1.3.2 DeFi Security

- * Semantic consistency
- * Functionality consistency
- * Permission management
- * Business logic

- * Token operation
- * Emergency mechanism
- * Oracle security
- * Whitelist and blacklist
- * Economic impact
- * Batch transfer

1.3.3 NFT Security

- * Duplicated item
- * Verification of the token receiver
- * Off-chain metadata security

1.3.4 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 Methodology³ and Common Weakness Enumeration⁴. 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	High	High	Medium
	Low	Medium	Low
		High	Low
		Likelihood	

³https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

⁴<https://cwe.mitre.org/>

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 four 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.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

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

- Medium Risk: 1
- Low Risk: 4
- Recommendation: 8
- Note: 5

ID	Severity	Description	Category	Status
1	Medium	Lack of chain ID in the signature for settlement-related contracts	DeFi Security	Confirmed
2	Low	Reusable signature in the <code>claimFees</code> function	DeFi Security	Fixed
3	Low	Potential reentrancy risk due to the call-back mechanism of the underlying token	DeFi Security	Fixed
4	Low	Potentially disrupted fee collection due to the rescue of native tokens	DeFi Security	Fixed
5	Low	User-controlled <code>aggregatorAdaptor</code> in the <code>SwapRequested</code> event	DeFi Security	Fixed
6	-	Unify the method for native token transfers	Recommendation	Fixed
7	-	Remove redundant value assignment	Recommendation	Fixed
8	-	Add input validation check for the <code>refund</code> function	Recommendation	Fixed
9	-	Initialize <code>completeDepositGasLimit</code> in <code>YBridgeVaultV3</code> 's constructor	Recommendation	Fixed
10	-	Revise the misleading error message <code>NotEnoughGasFee</code>	Recommendation	Fixed
11	-	Add checks to prevent zero transfers in the <code>YBridgeVaultV3</code> contract	Recommendation	Fixed
12	-	Add a <code>Nonexist</code> status to the <code>RequestStatus</code> enum	Recommendation	Acknowledged
13	-	Add a check for <code>threshold</code> in <code>Supervisor</code> 's constructor	Recommendation	Fixed
14	-	Potential centralization risks	Note	-
15	-	Potential off-chain risks	Note	-
16	-	Ensure the fund migration when changing a vault address	Note	-
17	-	Avoid dependence on <code>RebalanceRewardCalculation</code> for reward calculation	Note	-

18	-	Potential DoS due to insufficient check on the <code>_swapStatus</code>	Note	-
----	---	---	------	---

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Lack of chain ID in the signature for settlement-related contracts

Severity Medium

Status Confirmed

Introduced by [Version 1](#)

Description In the [YBridgeVaultSettlementV3](#) and [FeeUtilityV2](#) contracts of the yBridge project, the domain separator for messages lacks a field related to the chain ID, potentially allowing signatures to be replayed on other chains.

```

785  function refund(
786      EVMRefundDescription calldata refundDesc,
787      bytes[] calldata signatures,
788      bytes[] calldata signaturesForRefund
789  ) public onlyRole(ROLE_SETTLEMENT_WORKER) {
790      // Check
791      bytes32 universalSwapId = _getUniversalId(refundDesc.srcChainId, refundDesc.srcChainSwapId)
792      ;
793      require(_swapStatus[universalSwapId] != SwapStatus.Completed, "ERR_INVALID_SWAP_STATUS");
794
795      _verifySignatures(
796          abi.encodePacked(
797              REFUND_IDENTIFIER, address(this), address(supervisor), refundDesc.srcChainId,
798              refundDesc.srcChainSwapId
799          ),
800          signatures
801      );
802
803      _verifySignatures(
804          bytes.concat(
805              abi.encodePacked(
806                  LOCK_CLOSE_SWAP_AND_REFUND_IDENTIFIER,
807                  refundDesc.yBridge,
808                  refundDesc.supervisor,
809                  refundDesc.srcChainId,
810                  refundDesc.srcChainSwapId
811              ),
812              abi.encodePacked(
813                  refundDesc.receiver,
814                  refundDesc.gasFeeReceiver,
815                  refundDesc.vaultToken,
816                  refundDesc.refundAmount,
817                  refundDesc.refundGasFee
818              )
819          )
820      );

```

```
816     ),
817     signaturesForRefund
818 );
819
820 // If has initiateCrossChainSwap, unlock liquidity on destination chain. But this case
821 // shouldn't happen.
821 if (_swapStatus[universalSwapId] == SwapStatus.Initiated) {
822     uint32 dstChainId = _swapInfo[universalSwapId].dstChainId;
823     chainLockedAmount[dstChainId] -= (
824         _swapInfo[universalSwapId].amountOut + _swapInfo[universalSwapId].gasFee
825         + _swapDaoFeeInfo[universalSwapId]
826     );
827 }
828 _swapStatus[universalSwapId] = SwapStatus.Completed;
829 emit Refunded(refundDesc, signaturesForRefund);
830 }
```

Listing 2.1: contracts/settlement-chain/YBridgeVaultSettlementV3.sol

```
169 function updateXyDaoReserveFeeRate(uint256 _xyDaoReserveFeeRate, bytes[] calldata signatures)
170     external {
171     require(_xyDaoReserveFeeRate <= 10 ** 8, "ERR_XY_DAO_RESERVE_FEE_RATE_TOO_LARGE");
172
173     _verifySignatures(
174         abi.encodePacked(
175             UPDATE_XY_DAO_RESERVE_FEE_RATE,
176             address(this),
177             address(supervisor),
178             _xyDaoReserveFeeRate,
179             nonceForUpdate++
180         ),
181         signatures
182     );
183
184     xyDaoReserveFeeRate = _xyDaoReserveFeeRate;
185     emit XyDaoReserveFeeRateUpdated(_xyDaoReserveFeeRate);
186 }
```

Listing 2.2: contracts/settlement-chain/FeeUtilityV2.sol

Impact The signatures can be reused on other chains, potentially affecting validators' operations in the future.

Suggestion Add the chain ID to the domain separator in messages to prevent signatures from being replayed on other chains.

Feedback from the project It's noted. But considering the current contract usage, no further actions would be done on this.

2.1.2 Reusable signature in the `claimFees` function

Severity Low

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the [YBridgeVaultSettlementV3](#) contract of the yBridge project, the reusable signature [signaturesForCollectFees](#) allows users to bypass the checks from Lines 849 to 852, enabling the inclusion of an outdated [ClaimFeeDesc](#) in the [FeesClaimed](#) event. This could potentially disrupt relayers' off-chain operations due to the emission of the [FeesClaimed](#) event with an outdated [claimFeeDesc](#).

```
804 function claimFees(  
805     uint32 chainId,  
806     uint256 nonceForCollectFeesAndRefuge,  
807     EVMClaimFeesDescription calldata descForSettle,  
808     EVMClaimFeesDescription calldata claimFeeDesc,  
809     bytes[] calldata signatures,  
810     bytes[] calldata signaturesForCollectFees  
811 ) external {  
812     require(feeUtility.withholdingFeeReceiver() != address(0), "ERR_INVALID_FEE_RECEIVER");  
813     require(feeUtility.closeSwapGasFeeReceiver() != address(0), "ERR_INVALID_FEE_RECEIVER");  
814     require(feeUtility.xyDaoReserveFeeReceiver() != address(0), "ERR_INVALID_FEE_RECEIVER");  
815  
816     _verifySignatures(  
817         bytes.concat(  
818             abi.encodePacked(  
819                 CLAIM_FEES_IDENTIFIER, address(this), address(supervisor), chainId, descForSettle  
820                     .yBridgeVault  
821             ),  
822             abi.encodePacked(  
823                 feeUtility.withholdingFeeReceiver(),  
824                 descForSettle.withholdingFee,  
825                 feeUtility.closeSwapGasFeeReceiver(),  
826                 descForSettle.closeSwapGasFee,  
827                 feeUtility.xyDaoReserveFeeReceiver(),  
828                 descForSettle.xyDaoReserveFee,  
829                 nonceForClaimFees++  
830             ),  
831             signatures  
832         );  
833     _verifySignatures(  
834         bytes.concat(  
835             abi.encodePacked(COLLECT_FEES_IDENTIFIER, claimFeeDesc.yBridgeVault, claimFeeDesc.  
836                 supervisor, chainId),  
837             abi.encodePacked(  
838                 feeUtility.withholdingFeeReceiver(),  
839                 claimFeeDesc.withholdingFee,  
840                 feeUtility.closeSwapGasFeeReceiver(),  
841                 claimFeeDesc.closeSwapGasFee,  
842                 feeUtility.xyDaoReserveFeeReceiver(),  
843                 claimFeeDesc.xyDaoReserveFee,  
844                 nonceForCollectFeesAndRefuge  
845             ),  
846             signaturesForCollectFees
```

```

847     );
848
849     require(
850         claimFeeDesc.withholdingFee + claimFeeDesc.closeSwapGasFee + claimFeeDesc.
            xyDaoReserveFee > 0,
851         "ERR_INVALID_FEE_AMOUNT"
852     );
853     require(withholdingFees[chainId] >= descForSettle.withholdingFee, "
            ERR_INVALID_DEPOSIT_AND_WITHDRAW_FEE");
854     require(closeSwapGasFees[chainId] >= descForSettle.closeSwapGasFee, "
            ERR_INVALID_CLOSE_SWAP_GAS_FEE");
855     require(xyDaoReserveFees[chainId] >= descForSettle.xyDaoReserveFee, "
            ERR_INVALID_XY_DAO_RESERVE_FEE");
856     withholdingFees[chainId] -= descForSettle.withholdingFee;
857     closeSwapGasFees[chainId] -= descForSettle.closeSwapGasFee;
858     xyDaoReserveFees[chainId] -= descForSettle.xyDaoReserveFee;
859     emit FeesClaimed(
860         chainId,
861         feeUtility.withholdingFeeReceiver(),
862         feeUtility.closeSwapGasFeeReceiver(),
863         feeUtility.xyDaoReserveFeeReceiver(),
864         descForSettle,
865         claimFeeDesc,
866         signaturesForCollectFees
867     );
868 }

```

Listing 2.3: contracts/settlement-chain/YBridgeVaultSettlementV3.sol

Impact The reusable signature could introduce an outdated `claimFeeDesc` in the `FeesClaimed` event, potentially impacting relayers' off-chain operations.

Suggestion Add a nonce to the `signaturesForCollectFees` to prevent the reuse of signatures.

2.1.3 Potential reentrancy risk due to the callback mechanism of the underlying token

Severity Low

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description The `completeCrossChainRequest` function of the `XYCrossChainRelay` contract in the xAsset bridge project is designed to fulfill users' cross-chain requests and transfer the corresponding underlying token (`underlyingToken`). However, if the underlying token employs a callback mechanism during the transfer, it may be susceptible to a reentrancy attack.

```

220     function completeCrossChainRequest(
221         uint256 requestId,
222         uint32 sourceChainId,
223         address receiver,
224         uint256 amount,
225         uint256 fee,

```

```
226     bytes[] memory signatures
227 ) external whenNotPaused onlyRole(ROLE_STAFF) {
228     require(!completedCrossChainRequest[sourceChainId][requestId], "
229         ERR_CROSS_CHAIN_REQUEST_ALREADY_COMPLETE");
229     require(amount > fee, "ERR_FEE_GREATER_THAN_AMOUNT");
230     require(amount <= maxCrossChainAmount, "ERR_INVALID_CROSS_CHAIN_AMOUNT");
231
232     bytes32 sigId = keccak256(
233         abi.encodePacked(
234             supervisor.VALIDATE_XY_CROSS_CHAIN_IDENTIFIER(),
235             address(this),
236             sourceChainId,
237             thisChainId,
238             requestId,
239             receiver,
240             amount,
241             fee
242         )
243     );
244     bytes32 sigIdHash = sigId.toEthSignedMessageHash();
245     supervisor.checkSignatures(sigIdHash, signatures);
246     uint256 amountSubFee = amount - fee;
247
248     bool isRedeemableTokenMinted = false;
249     if (isPrimitive) {
250         if (checkBridgeLiquidityEnough(amount)) {
251             _safeTransferTokenUnified(underlyingToken, receiver, amountSubFee);
252             _safeTransferTokenUnified(underlyingToken, treasury, fee);
253         } else {
254             require(isAbleToMintIfNoLiquidity, "ERR_NO_MINT_WHEN_NO_LIQUIDITY");
255             isRedeemableTokenMinted = true;
256             // Mint redeemable token to receiver
257             _mint(receiver, amountSubFee);
258             _mint(treasury, fee);
259         }
260     } else {
261         IERC20MintBurnable(underlyingToken).mint(receiver, amountSubFee);
262         IERC20MintBurnable(underlyingToken).mint(treasury, fee);
263     }
264
265     completedCrossChainRequest[sourceChainId][requestId] = true;
266     emit CrossChainCompleted(requestId, sourceChainId, thisChainId, receiver, amount, fee,
267         isRedeemableTokenMinted);
267 }
```

Listing 2.4: src/XYCrossChainRelay.sol

Impact The transfer of the underlying token with a callback mechanism could potentially trigger a reentrancy attack.

Suggestion Add a `ReentrancyGuard` modifier to the `completeCrossChainRequest` function.

2.1.4 Potentially disrupted fee collection due to the rescue of native tokens

Severity Low

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the [YBridgeVaultV3](#) contract of the yBridge project, the [rescue](#) function is designed solely to recover funds accidentally sent to the contract, specifically excluding deposit tokens and [xyWrappedToken](#). However, it does not restrict privileged users from rescuing the native token, which is used for gas fees and held in the [YBridgeVaultV3](#) contract. This could potentially lead to a Denial of Service (DoS) issue for the fee collection feature.

```

322 function rescue(IERC20[] memory tokens) external onlyRole(ROLE_OWNER) {
323     uint256 length = tokens.length;
324     for (uint256 i; i < length; i++) {
325         IERC20 token = tokens[i];
326         require(token != depositToken, "ERR_CAN_NOT_RESCUE_DEPOSIT_TOKEN");
327         require(address(token) != address(xyWrappedToken), "ERR_CAN_NOT_RESCUE_XY_WRAPPED_TOKEN");
328         uint256 _tokenBalance =
329             address(token) == ETHER_ADDRESS ? address(this).balance : token.balanceOf(address(
330                 this));
331         _safeTransferAsset(msg.sender, token, _tokenBalance);
332     }
333 }
```

Listing 2.5: contracts/periphery-chain/YBridgeVaultV3.sol

Impact This may potentially lead to a DoS issue for the fee collection feature.

Suggestion Revise the logic of the [rescue](#) function to ensure proper fee collection.

2.1.5 User-controlled aggregatorAdaptor in the SwapRequested event

Severity Low

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the [YBridgeV3](#) contract of the yBridge project, the [_swap](#) function manages the swap process for users' cross-chain requests. However, the [aggregatorAdaptor](#) recorded in the [SwapRequested](#) event can be inaccurate. Specifically, users can specify a non-zero [aggregatorAdaptor](#) in their requests even when [swapDesc.srcToken == swapDesc.dstToken](#). This could lead to the incorrect [aggregatorAdaptor](#) being logged in the [SwapRequested](#) event. Although the [aggregatorAdaptor](#) is not utilized in this scenario, the inaccurate recording could still affect relayers' off-chain operations.

```

444 function _swap(
445     address aggregatorAdaptor,
446     IDexAggregatorAdaptor.SwapDescription memory swapDesc,
447     bytes memory aggregatorData,
448     DstChainDescription memory dstChainDesc,
```

```
449     address referrer
450 ) private {
451     address receiver = swapDesc.receiver; // receiver on the dst chain
452     uint256 swappingYeild; // amount of the token after swap
453
454     // No swap needed
455     if (swapDesc.srcToken == swapDesc.dstToken) {
456         // Transfer directly to the YBridgeVault, since no swap needed
457         _transferLiquidity(YBridgeVaults[address(swapDesc.dstToken)], swapDesc.srcToken,
458             swapDesc.amount);
459         swappingYeild = swapDesc.amount;
460     } else {
461         // Swap needed
462         // Transfer to YBridgeV3 (this) first for swapping
463         _transferLiquidity(address(this), swapDesc.srcToken, swapDesc.amount);
464         if (!isWhitelistedAggregatorAdaptor[aggregatorAdaptor]) revert InvalidAggregatorAdaptor
465             ();
466         swappingYeild = getTokenBalance(swapDesc.dstToken, address(this));
467         swapDesc.receiver = address(this);
468
469         // swap
470         if (address(swapDesc.srcToken) != ETHER_ADDRESS) {
471             swapDesc.srcToken.safeApprove(aggregatorAdaptor, swapDesc.amount);
472             IDexAggregatorAdaptor(aggregatorAdaptor).swap{value: 0}(swapDesc, aggregatorData);
473             swapDesc.srcToken.safeApprove(aggregatorAdaptor, 0);
474         } else {
475             IDexAggregatorAdaptor(aggregatorAdaptor).swap{value: swapDesc.amount}(swapDesc,
476                 aggregatorData);
477         }
478         swappingYeild = getTokenBalance(swapDesc.dstToken, address(this)) - swappingYeild;
479         _transferTo(YBridgeVaults[address(swapDesc.dstToken)], swapDesc.dstToken, swappingYeild)
480         ;
481     }
482     if (swappingYeild < minVaultTokenSwapAmount[address(swapDesc.dstToken)]) {
483         revert SwapAmountTooSmall(minVaultTokenSwapAmount[address(swapDesc.dstToken)],
484             swappingYeild);
485     }
486     if (swappingYeild > maxVaultTokenSwapAmount[address(swapDesc.dstToken)]) {
487         revert SwapAmountTooLarge(maxVaultTokenSwapAmount[address(swapDesc.dstToken)],
488             swappingYeild);
489     }
490
491     emit SwapRequested(
492         swapId++,
493         msg.sender,
494         aggregatorAdaptor,
495         dstChainDesc,
496         swapDesc.srcToken,
497         swapDesc.dstToken,
498         swappingYeild,
499         receiver,
500         swapDesc.amount,
501         0, /*was ExpressFeeAmount*/
```

```
496         referrer
497     );
498 }
```

Listing 2.6: contracts/periphery-chain/YBridgeV3.sol

Impact The user-controlled `aggregatorAdaptor` parameter included in the `SwapRequested` event could potentially impact relayers' off-chain operations.

Suggestion Revise the code accordingly.

2.2 Additional Recommendation

2.2.1 Unify the method for native token transfers

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description XY Finance contracts currently use two different methods for native token transfers: `transfer` and `Address.sendValue`. The strict gas limits of the `transfer` function can lead to failures, especially for proxy contract users with fallback functions that consume more gas than allocated. Therefore, it is recommended to standardize on a single method for all native token transfers.

Impact May lead to unexpected results.

Suggestion Unify the method for native token transfers.

2.2.2 Remove redundant value assignment

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the `_handleAdditionalFee` function of the `YBridgeV3` contract, the value assignment for the variable `dstAdaptor_` (Lines 400-402) is redundant and can be removed.

```
398     uint256 _dstAdditionalFee = 0;
399     if (dstAdaptor_ != address(0)) {
400         _dstAdditionalFee = _getAdditionalFee(dstAdaptor_);
401     } else {
402         dstAdaptor_ = address(0);
403     }
```

Listing 2.7: ybridge-contracts-audit/contracts/periphery-chain/YBridgeV3.sol

Impact N/A

Suggestion Remove the redundant value assignment.

2.2.3 Add input validation check for the `refund` function

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the `refund` function of the `YBridgeV3` contract, the input values `refundAmount` and `refundGasFee` are not validated. It is recommended to implement checks for these inputs to prevent zero transfers.

```

849  function refund(
850      uint256 _swapId,
851      address receiver,
852      address gasFeeReceiver,
853      address vaultToken,
854      uint256 refundAmount,
855      uint256 refundGasFee,
856      bytes[] calldata signatures
857 ) external whenNotPaused onlyRole(ROLE_YPOOL_WORKER) {
858     if (_swapId >= swapId) {
859         revert InvalidSwapId();
860     }
861     if (refundStatus[_swapId]) {
862         revert AlreadyDone();
863     }
864     refundStatus[_swapId] = true;
865
866     {
867         bytes32 sigId = keccak256(
868             bytes.concat(
869                 abi.encodePacked(
870                     supervisor.LOCK_CLOSE_SWAP_AND_REFUND_IDENTIFIER(),
871                     address(this),
872                     address(supervisor),
873                     chainId,
874                     _swapId
875                 ),
876                 abi.encodePacked(receiver, gasFeeReceiver, vaultToken, refundAmount, refundGasFee)
877             )
878         );
879         bytes32 sigIdHash = sigId.toEthSignedMessageHash();
880         supervisor.checkSignatures(sigIdHash, signatures);
881     }
882     IYBridgeVault(YBridgeVaults[vaultToken]).transferTo(IERC20(vaultToken), receiver,
883         refundAmount);
884     IYBridgeVault(YBridgeVaults[vaultToken]).transferTo(IERC20(vaultToken), gasFeeReceiver,
885         refundGasFee);
886     emit SwapRefunded(_swapId, receiver, gasFeeReceiver, vaultToken, refundAmount, refundGasFee);
887 }

```

Listing 2.8: contracts/periphery-chain/YBridgeV3.sol

Impact N/A

Suggestion Implement checks on the input parameters of the `refund` function.

2.2.4 Initialize `completeDepositGasLimit` in `YBridgeVaultV3`'s constructor

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description The `completeDepositGasLimit` variable is not initialized in the constructor of the `YBridgeVaultV3` contract. Failing to initialize this variable allows users to bypass gas fee payments necessary to cover the deposit completion transaction cost. To ensure robust functionality, it is recommended to initialize critical variables in the constructor rather than relying on manual settings.

```

278 function _depositFor(address receiver, uint256 vaultTokenAmount) private {
279     if (!acceptDepositRequest || !acceptDepositAndWithdrawRequest) revert NotAcceptingRequest()
280     ;
281     if (vaultTokenAmount <= minDepositAmount) revert InvalidInputAmount();
282     // Get the native token amount required for this deposit
283     uint256 gasFee = completeDepositGasLimit * _getGasPrice();
284     uint256 requiredValue = (address(depositToken) == ETHER_ADDRESS) ? gasFee +
        vaultTokenAmount : gasFee;
285     if (msg.value < requiredValue) revert NotEnoughGasFee(gasFee);
286
287     // Transfer deposit token from sender to this contract
288     if (address(depositToken) != ETHER_ADDRESS) {
289         _safeTransferAssetFrom(depositToken, msg.sender, address(this), vaultTokenAmount);
290     }
291
292     // Refund extra native token
293     Address.sendValue(payable(msg.sender), msg.value - requiredValue);
294
295     emit DepositRequested(receiver, numDeposits++, vaultTokenAmount, gasFee);
296 }

```

Listing 2.9: contracts/periphery-chain/YBridgeVaultV3.sol

Impact Failing to configure `completeDepositGasLimit` upon contract deployment could allow depositors to bypass fee payments.

Suggestion Initialize `completeDepositGasLimit` in the constructor.

2.2.5 Revise the misleading error message `NotEnoughGasFee`

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the `_depositFor` function of the `YBridgeVaultV3` contract, the printed error message `NotEnoughGasFee` may be misleading. Specifically, when users attempt to deposit native tokens but provide an amount less than `vaultTokenAmount`, the function reverts with the error `NotEnoughGasFee`, which is semantically incorrect.

```

278 function _depositFor(address receiver, uint256 vaultTokenAmount) private {
279     if (!acceptDepositRequest || !acceptDepositAndWithdrawRequest) revert NotAcceptingRequest()
280     ;
281     if (vaultTokenAmount <= minDepositAmount) revert InvalidInputAmount();
282     // Get the native token amount required for this deposit
283     uint256 gasFee = completeDepositGasLimit * _getGasPrice();
284     uint256 requiredValue = (address(depositToken) == ETHER_ADDRESS) ? gasFee +
        vaultTokenAmount : gasFee;
285     if (msg.value < requiredValue) revert NotEnoughGasFee(gasFee);

```

Listing 2.10: contracts/periphery-chain/YBridgeVaultV3.sol

Impact The inaccurate error message may confuse users.

Suggestion Add a specific error message to handle cases of insufficient deposit amounts.

2.2.6 Add checks to prevent zero transfers in the YBridgeVaultV3 contract

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description

- In the `_depositFor` function, the refunding amount (i.e., `msg.value - requiredValue`) could potentially be zero. To optimize gas usage, it is recommended to implement zero-value checks to avoid redundant external calls.

```

292 // Refund extra native token
293 Address.sendValue(payable(msg.sender), msg.value - requiredValue);

```

Listing 2.11: contracts/periphery-chain/YBridgeVaultV3.sol

- In the `collectFees` function, it is recommended to add a check for the total sum of all fees to prevent zero transfers.

```

609 payable(withholdingFeesReceiver).transfer(withholdingFees);
610 emit WithholdingFeesCollected(withholdingFeesReceiver, withholdingFees);
611
612 _safeTransferAsset(closeSwapGasFeesReceiver, depositToken, closeSwapGasFees);
613 emit CloseSwapGasFeesCollected(depositToken, closeSwapGasFeesReceiver, closeSwapGasFees);
614
615 _safeTransferAsset(xyDaoReserveFeesReceiver, depositToken, xyDaoReserveFees);
616 emit XYDaoReserveFeesCollected(depositToken, xyDaoReserveFeesReceiver, xyDaoReserveFees);

```

Listing 2.12: contracts/periphery-chain/YBridgeVaultV3.sol

Impact N/A

Suggestion Add checks to prevent zero transfers.

2.2.7 Add a Nonexist status to the RequestStatus enum

Status Acknowledged

Introduced by [Version 1](#)

Description In the [RebalanceRewardsAsync](#) contract, the default status of the [RequestStatus](#) enum is set to pending, which is semantically incorrect.

```
17  enum RequestStatus {
18      Pending,
19      Completed
20  }
```

Listing 2.13: contracts/periphery-chain/RebalanceRewardsAsync.sol

Impact N/A

Suggestion Add a [Nonexist](#) status to the [RequestStatus](#) enum for better clarity.

Feedback from the project The [RequestStatus](#) enum in [RebalanceRewardsAsync.sol](#) is now deprecated, so no further updates are required.

2.2.8 Add a check for threshold in Supervisor's constructor

Status Fixed in [Version 2](#)

Introduced by [Version 1](#)

Description In the constructor of the [Supervisor](#) contract, the [threshold](#) value is not adequately validated. It is important to ensure that the [threshold](#) variable is not assigned a zero value.

```
34  constructor(uint32 _chainId, address [] memory _validators, uint256 _threshold) {
35      chainId = _chainId;
36
37      for (uint256 i; i < _validators.length; i++) {
38          validators[_validators[i]] = true;
39      }
40      validatorsNum = _validators.length;
41      require(_threshold <= validatorsNum, "ERR_INVALID_THRESHOLD");
42      threshold = _threshold;
43  }
```

Listing 2.14: src/Supervisor.sol

Impact A misconfiguration that sets the [threshold](#) to zero could allow bypassing signature verification processes.

Suggestion Add a non-zero check for [threshold](#).

2.3 Note

2.3.1 Potential centralization risks

Introduced by [Version 1](#)

Description XY Finance contracts include several critical functions, such as upgrading contracts, setting key parameters, and processing swap requests, which can only be executed by

privileged roles. Misconfigurations of these parameters can significantly impact the contracts' functionality, potentially rendering them unusable. Consequently, if the private keys of these privileged roles were compromised, the entire protocol could be incapacitated, leading to potential centralization risks.

2.3.2 Potential off-chain risks

Introduced by [Version 1](#)

Description Some features in XY Finance contracts, such as the request verification, are implemented off-chain. Since off-chain logic falls outside the scope of this audit, we must assume its design and correctness, which poses potential risks.

2.3.3 Ensure the fund migration when changing a vault address

Introduced by [Version 1](#)

Description In the `YBridgeV3` contract, the `setYBridgeVault` function can add, remove, or replace the vault for any `supportedToken`, as indicated in the code annotation. However, this function does not verify whether there are remaining assets in the old vault when replacing an existing one. It is crucial for the team to ensure that any remaining funds are properly accounted for and transferred during manual migrations.

2.3.4 Avoid dependence on `RebalanceRewardCalculation` for reward calculation

Introduced by [Version 1](#)

Description The `calculate` function of the `RebalanceRewardCalculation` contract does not account for the case where `prevToChainPCV == amountOut`. In this case, `newFromToPCVProduct` is set to the updated PCV value on the source chain (i.e., `prevFromChainPCV + amountIn`) instead of zero. This could potentially lead to an inaccurate reward rate calculation.

As the `calculate` function is not currently used in the codebase, this issue does not pose immediate risks. However, it is important to emphasize that neither the team nor any third parties should rely on the return value of this function.

Feedback from the project The contract `RebalanceRewardCalculation` is obsoleted. It's kept in the `YBridgeVaultSettlementV3` contract just to hold the storage slot. We'll be changing that storage slot to type address instead of type `RebalanceRewardCalculation` and add comments on the variable to avoid future confusion.

2.3.5 Potential DoS due to insufficient check on the `_swapStatus`

Introduced by [Version 1](#)

Description In the `YBridgeVaultSettlementV3` contract of the yBridge project, the `refund` function updates the refund status of provided swap requests. However, swap requests with a `Nonexist` status are not properly validated. This lack of validation could result in incorrect status updates, causing `Nonexist` swap requests to be mistakenly marked as `Completed`. Additionally,

since the swap ID is incremental, invoking the `refund` function with a nonexistent swap ID could hinder the future initialization of swap requests in the `initiateCrossChainSwapWithDexAggregator` function.

```
785 function refund(
786     EVMRefundDescription calldata refundDesc,
787     bytes[] calldata signatures,
788     bytes[] calldata signaturesForRefund
789 ) public onlyRole(ROLE_SETTLEMENT_WORKER) {
790     // Check
791     bytes32 universalSwapId = _getUniversalId(refundDesc.srcChainId, refundDesc.srcChainSwapId)
792     ;
793     require(_swapStatus[universalSwapId] != SwapStatus.Completed, "ERR_INVALID_SWAP_STATUS");
794     _verifySignatures(
795         abi.encodePacked(
796             REFUND_IDENTIFIER, address(this), address(supervisor), refundDesc.srcChainId,
797             refundDesc.srcChainSwapId
798         ),
799         signatures
800     );
801     _verifySignatures(
802         bytes.concat(
803             abi.encodePacked(
804                 LOCK_CLOSE_SWAP_AND_REFUND_IDENTIFIER,
805                 refundDesc.yBridge,
806                 refundDesc.supervisor,
807                 refundDesc.srcChainId,
808                 refundDesc.srcChainSwapId
809             ),
810             abi.encodePacked(
811                 refundDesc.receiver,
812                 refundDesc.gasFeeReceiver,
813                 refundDesc.vaultToken,
814                 refundDesc.refundAmount,
815                 refundDesc.refundGasFee
816             )
817         ),
818         signaturesForRefund
819     );
820     // If has initiateCrossChainSwap, unlock liquidity on destination chain. But this case
821     // shouldn't happen.
822     if (_swapStatus[universalSwapId] == SwapStatus.Initiated) {
823         uint32 dstChainId = _swapInfo[universalSwapId].dstChainId;
824         chainLockedAmount[dstChainId] -= (
825             _swapInfo[universalSwapId].amountOut + _swapInfo[universalSwapId].gasFee
826             + _swapDaoFeeInfo[universalSwapId]
827         );
828     }
829     _swapStatus[universalSwapId] = SwapStatus.Completed;
830     emit Refunded(refundDesc, signaturesForRefund);
831 }
```

Listing 2.15: contracts/settlement-chain/YBridgeVaultSettlementV3.sol

Feedback from the project This is intentional behavior. It allows the refunding of transactions that have been requested but haven't made it to the Settlement chain. Concerns about DoS from sabotaging future swap IDs should be mitigated by the `onlyRole` constraint.

