Quantstamp

Boba 1 (Bridges And LP Floating Fee)

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Layer two blockchain / Bridge
Timeline	2023-03-24 through 2023-03-24
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Diff/Fork information	Boba is a fork of Optimism. At the time of this audit, Boba is using the pre-bedrock optimism.
Source Code	• bobanetwork/boba #aa609fa
Auditors	• Andy Lin Senior Auditing Engineer • Ibrahim Abouzied Auditing Engineer • Pavel Shabarkin Auditing Engineer • Valerian Callens Senior Auditing Engineer • Adrian Koegl Auditing Engineer

Documentation quality	Medium
Test quality	Low
Total Findings	33 Fixed: 13 Acknowledged: 15 Mitigated: 5
High severity findings ⓘ	4 Fixed: 3 Acknowledged: 1
Medium severity findings ⓘ	5 Fixed: 2 Acknowledged: 1 Mitigated: 2
Low severity findings ⓘ	10 Fixed: 4 Acknowledged: 4 Mitigated: 2
Undetermined severity findings ⓘ	0
Informational findings ⓘ	14 Fixed: 4 Acknowledged: 9 Mitigated: 1

Summary of Findings

Initial Audit: This Boba audit focused on LP contracts, ERC721 Bridges, and ERC1155 Bridges. Since this is a rollup solidity project, we reviewed several general concerns, including the potential overflow risk due to the contract version, the risk of reorgs, and the risk of the same contract address appearing in both L1 and L2, controlled by different owners.

LP contracts are intended for users to withdraw funds from L2 to L1 quickly, and the audit discovered some issues with them. The LP contracts underwent changes that added xBoba minting and burning and implemented the use of `L2BillingContract` for exit fees. However, the audit team found that one of the fixes for the previous audit's issues was not applied to the new features, and additional issues were also discovered.

The ERC721 and ERC1155 bridges have similar structures and flows, and we identified some issues in the audit. The most severe issue was using the wrong selector, causing the flow to fail. These issues should be addressed to ensure the security and integrity of the contracts, protect user funds, and prevent potential security breaches.

Regarding the project quality, the audit found that the code was easy to follow and had consistent patterns. However, the test quality and coverage were low, and there were no tests for liquidity pools. Regarding documentation, users could find general information about the project from Boba and Optimism, which was positive. However, documentation for new features was scattered across various Google Docs, making it difficult for regular users to access.

During the audit, our team frequently interacted with the Boba team to clarify code and expected behavior outside of the codebase within the scope of the audit. As the audit focused on the new changes based on the rollup protocol, the Boba team's active engagement in answering our questions was crucial and greatly assisted in completing the audit.

Fix Review: The Boba team fixed or acknowledged all issues within two weeks. They squash merged most of the fixes from PR #727 back to the `develop` branch. The commit hashes in the update sections are mostly the commits of the PR.

After our initial review of the fix, we double-checked the statuses of BOB1-3, 5, and 30 with the team. The Boba team confirmed or provided further fixes regarding those. We have also updated the status of each issue accordingly.

ID	DESCRIPTION	SEVERITY	STATUS
BOB1-1	Unsafe `replyNeeded` Messaging Mechanism	● High	Acknowledged
BOB1-2	Using the Wrong Selector Causes Token Loss During ERC1155 Bridging	● High	Fixed
BOB1-3	Native Tokens From the L1 Liquidity Pools Could Be Drained Depending on the Compiler Version Used	● High	Fixed
BOB1-4	`DiscretionaryExitFee` Can Be Drained	● High	Fixed
BOB1-5	Stealing Liquidity Pool Fund with Reorg	● Medium	Fixed
BOB1-6	Unable to Sync Liquidity Pool Fee From L2	● Medium	Fixed
BOB1-7	Contract Ownership Can Be Claimed Back by Anyone Once Renounced	● Medium	Mitigated
BOB1-8	Attacker Can Steal Tokens by Deploying Malicious Contracts	● Medium	Acknowledged
BOB1-9	Risk of Registering Invalid NFT/token Pairs	● Medium	Mitigated
BOB1-10	Locking NFTs/tokens by Sending to a Non-Supporting Contract	● Low	Acknowledged
BOB1-11	Overflow Risk	● Low	Fixed
BOB1-12	Missing Input Validations	● Low	Mitigated
BOB1-13	Bridge NFTs/tokens to Contracts with Different Ownership Across L1 and L2	● Low	Mitigated
BOB1-14	Contract Ownership Can Be Renounced or Lost with `transferOwnership()`	● Low	Fixed
BOB1-15	Initial Ownership Claimable by Anyone Both Directly and via a Proxy	● Low	Acknowledged
BOB1-16	Risk of User Overcharging Fee	● Low	Fixed
BOB1-17	NFT/Token Can be Finalized without Registration on Target Chain	● Low	Acknowledged
BOB1-18	Reentrancy Risk	● Low	Fixed
BOB1-19	`withdrawRewards()` Not Reflecting the Latest Reward Changes	● Low	Acknowledged
BOB1-20	Application Monitoring Can Be Improved by Emitting More Events	● Informational	Fixed
BOB1-21	Inheriting Non-Upgradeable Contracts	● Informational	Acknowledged
BOB1-22	Partially Succeeded Batch Transfer	● Informational	Fixed
BOB1-23	Proxy Admins Could Update the Admin or the Implementation of the Proxy by Mistake	● Informational	Acknowledged
BOB1-24	Cross-Chain Transfers Could Be Blocked by Unsynchronized Admin Actions	● Informational	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
BOB1-25	Missing Interface Extension in Contracts	● Informational ⓘ	Acknowledged
BOB1-26	Storage Slot Collision Between Proxy and Implementation Contracts	● Informational ⓘ	Acknowledged
BOB1-27	Gas Costs at the Network Level Could Block some Cross-Chain Operations	● Informational ⓘ	Acknowledged
BOB1-28	Ambiguity of `from` and `msg.sender` in the Bridges	● Informational ⓘ	Fixed
BOB1-29	Unlocked Pragma	● Informational ⓘ	Mitigated
BOB1-30	`deposits` Is Not Updated upon NFT Withdrawal	● Informational ⓘ	Fixed
BOB1-31	Centralization Risk of the Liquidity Pool	● Informational ⓘ	Acknowledged
BOB1-32	Improvements to Incentive Model	● Informational ⓘ	Acknowledged
BOB1-33	Inconsistencies in Event Emitting Pattern	● Informational ⓘ	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

The initial audit only covers the following paths:

1. `LP/*`
2. `ERC1155Bridges/*`
3. `ERC721Bridges/*`
4. `standards/*` (exclude - `L2GovernanceERC20.sol`, `xL2GovernanceERC20.sol`)

The `LP/*` folder will undergo a differential audit between commits `426234d` and `aa609fa5`. On the other hand, the remaining files will be subjected to a full audit using the initial audit commit `aa609fa5`. It is worth noting that Quantstamp has already audited commit `426234d` in the past that covers the `LP/*` folder.

It is possible to include out-of-scope findings in this report. However, it should be noted that the related files should not be considered part of the audit scope.

Finally, if the final commit hash provided by the client includes features that are outside the audit scope or not related to the fix review, those features will not be considered in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following

    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Findings

BOB1-1 Unsafe `replyNeeded` Messaging Mechanism

● High ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation:

> Since a fix for this change would include some significant change to the already deployed contracts (with funds at stake), we are avoiding adding a fix for this edge case. Moreover, the condition of such a case could be difficult to attain in practice with multiple depositors on the pool and we have protections on the UI in place that prevents users from bridging when the other side of the pool does not have enough funds. We could also make people aware of this certain edge condition and possibility before bridging.

Users should be aware of the following risks if no changes are to apply. Also, we suggest a few possible mitigations from the frontend:

1. Boba team can rebalance the liquidity and cause issues. The Boba team should consider announcing the rebalance events in advance and warn the users in the frontend before and during the rebalance period.
2. The users should check how many liquidity providers there are and how many liquidity providers it takes to cause the lock of the fund. The Boba team can also consider providing a simulation of the risk in the frontend to the users.

File(s) affected: `LP/L2LiquidityPool.sol` , `LP/L1LiquidityPool.sol`

Description: The codebase incorporates the compensation transaction pattern in multiple locations to enable the reversal of cross-layer message failures. Specifically, the code verifies whether an action can be executed without errors and utilizes the `replyNeeded` variable to determine whether the contract should transmit a compensation transaction message to the originating layer to undo the effect. However, the code does not secure the resources required to undo the action, which can result in the compensation message failing if the resource has been utilized elsewhere. Furthermore, the message cannot be reattempted for an undo message from L1 to L2, leaving the contract in an anomalous state if it fails.

The risk can be a generalized issue whenever there are multiple actions that can manipulate resources. This is particularly relevant in the case of liquidity pools, where resources can be utilized not only through direct bridging but also through pool rebalancing and liquidity withdrawal. These actions increase the attack surface for liquidity pool contracts, making them more vulnerable to security threats. We analyzed the contracts within this audit's scope:

For the LP contracts:

1. `L1LiquidityPool.clientPayL1()` : the function can send a `L2LiquidityPool.clientPayL2Settlement` message back to L2 if the L1 liquidity pool lacks enough balance. However, since the process is async, and new transactions can happen on L2, the L2 liquidity pool might become out of balance when it attempts to revert with the L2LiquidityPool clientPayL2Settlement()` function. Once it fails, the user will lose his/her fund and cannot retry the relay message.
2. `L2LiquidityPool._initiateClientPayL2()` : when the L2 liquidity pool lacks enough balance, it will send the `L1LiquidityPool.clientPayL1Settlement` message back. The risk is lower here as L1 relay messages can be retried until they succeed. So the user can retry later once the liquidity pool is rebalanced.

The design pattern of the ERC721 and ERC1155 bridges is similar, but unlike liquidity pool contracts, the resources (NFT/tokens) can only be transferred by redeeming them from the bridge - there are no other available methods. As a result, the bridge contracts are not susceptible to the aforementioned concern and are considered safe.

Exploit Scenario:

1. Alice calls `L2LiquidityPool.clientDepositL2()` , trying to fast withdraw from L2 to L1. This will relay an `L1LiquidityPool.clientPayL1` message to L1.
2. At the time the message `L1LiquidityPool.clientPayL1` is relayed, the L1 liquidity pool lacks enough balance to pay. As a result, it tries to relay an `L2LiquidityPool.clientPayL2Settlement` message to L2 as the compensation message.
3. Somehow, other users calls `L2LiquidityPool.withdrawLiquidity()` to remove their liquidity on the L2 liquidity pool. The L2 balance is no longer enough to run the compensation message `L2LiquidityPool.clientPayL2Settlement` .
4. When the `L2LiquidityPool.clientPayL2Settlement` message is relayed, it fails due to a lack of balance. However, Alice lost her funds since the L2 relay message could not be retried.

Recommendation: One mitigation is to implement a retry mechanism for the L2 contract. For instance, the `L2LiquidityPool` contract can accept the relayed message, and if it fails, it stored the failed message in a mapping and allows the failed message to be retried again until succeeded.

An alternatively solution, which will solve this entirely, is to follow similar steps of two-phase locking. The steps for a cross-layer message will be like the following (assuming L2 → L1):

1. L2 contract first locks the resource (e.g., fund) necessary to revert later.
2. The message is relayed from L2 to L1.
3. If the L1 relayed message can run the action successfully, it should relay a confirmation message back to L2.
4. Otherwise, the L1 contract should follow the same steps to return the compensation message to L2.
5. The L2 should remove the locking of the resource if getting a confirmation message or undo/revert if receiving a compensation message.

Similar steps apply for L1 → L2 messages, though those are less concerning as the compensation message for L1 can be retried.

BOB1-2
Using the Wrong Selector Causes Token Loss During ERC1155 Bridging

● High ⓘ Fixed

Update
The team fixed the issue as recommended in the commit `535c64c9`.

Description: The `L1ERC1155Bridge.finalizeWithdrawalBatch()` function encodes the wrong selector when `replyNeeded` is true. The line `bytes memory message = abi.encodeWithSelector(...)` uses `iL2ERC1155Bridge.finalizeDeposit.selector` as the selector where the correct one should be `iL2ERC1155Bridge.finalizeDepositBatch.selector` instead. The message will fail because it will not fit the interface of the `iL2ERC1155Bridge.finalizeDeposit()` function. Once failed, the user will lose his/her tokens.

Exploit Scenario: Alice tries to bridge a ERC1155 token where the base network is L2. Somehow, the function fails to batch mint the tokens and tries to send a revert message. However, due to the bug, the revert message would not work and Alice will lose her tokens.

Recommendation: We suggest the following fixes:
1. Use the `iL2ERC1155Bridge.finalizeDepositBatch.selector` instead.
2. This indicates a lack of integration / end-to-end tests. We strongly recommend the Boba team adding the tests and also check if other tests should be added to cover all expected use cases.

BOB1-3
Native Tokens From the L1 Liquidity Pools Could Be Drained Depending on the Compiler Version Used

● High ⓘ Fixed

Update
The Boba team has confirmed that they lean towards making minimal changes to the contracts since they currently hold funds. Therefore, they have decided not to remove the SafeMath dependency at this moment.

Update
The team fixed the issue by ensuring the solidity version is `>=0.8.0` in the commit `c6e85b4`. In such a case, we suggest removing the `SafeMath` dependency as it is no longer necessary to prevent overflow.

File(s) affected: `LP/L1LiquidityPool.sol`, `LP/L1LiquidityPoolAltL1.sol`

Description: An issue in the function `clientDepositL1Batch()` of both `L1LiquidityPool` and `L1LiquidityPoolAltL1` contracts could let an attacker drain all native tokens from the liquidity pool contract of layer 1. However, only contracts compiled with a `Solidity` compiler version `< 0.8.0` are vulnerable. When the issue was discovered, Quantstamp directly shared their concerns with the Boba team who made sure that none of the contracts deployed on Mainnets were vulnerable. Their answer was: `right now, all deployments are compiled with 0.8.9`. The root cause of that issue is how the library SafeMath is used in the contract, which is described in a dedicated issue later in the contract.

Exploit Scenario: If the contract has been compiled in such a case that it is vulnerable to overflows (i.e. compiled with a `Solidity` version `< 0.8.0`):
1. Attacker calls `clientDepositL1Batch()` with a `msg.value` of `X` and the parameter `_tokens` being a sequence `S` of amounts for the native token, such as `sum of all amounts == X [modulo type(uint256).maxValue]`.
2. A batch message will be sent to L2 because the `require` statement will be tricked by an overflow provoked at the line `ETHAmount = ETHAmount + token.amount`;
3. In `L2LiquidityPool.clientPayL2Batch()`, each amount will be treated separately by `_initiateClientPayL2()`. If there are not enough tokens to distribute in the contract, then a reply message will be sent back to the L1 for each amount of `S` (`else` block).

4. In `L1LiquidityPool.clientPayL1Settlement()`, the function will try to transfer to the attacker each amount of `S` independently, in different transactions.
5. If an amount in `S` is close to the current balance of the contract `L1LiquidityPool` when the reply is received back on L1, but lower (to be valid), then almost all native tokens owned by the pool could be transferred to the attacker.

Recommendation: Consider following the recommendations of the dedicated issue related to SafeMath. Also, consider modifying the pragma version of the contracts.

BOB1-4 `DiscretionaryExitFee` Can Be Drained

● High Fixed

Update

The team added a new validation `require(msg.value != 0 || _l2Token != Lib_PredeployAddresses.OVM_ETH, ...)` in the commit `b5a7c572` that fixed the issue. Note that this commit is in a different PR: https://github.com/bobanetwork/boba/pull/716 rather than the main PR for fixing issues on this report.

File(s) affected: `DiscretionaryExitFee.sol`

Description: When calling `payAndWithdraw()` in `DiscretionaryExitFee`, the user is supposed to bridge either native ETH or any ERC20 tokens. Due to wrong requirement checks, `DiscretionaryExitFee` can be drained of funds if it possesses ERC20 tokens.

The user can call `payAndWithdraw()` with `msg.value==0` and `_l2Token == Lib_PredeployAddresses.OVM_ETH`. This will lead to the requirement in L50 to pass, which it presumably should not.

Subsequently, neither the if body in L52 nor L59 is entered, leading to the user not sending any funds. Finally, in L64, the ERC20 tokens will be bridged to the standard bridge from `DiscretionaryExitFee`'s funds, if any.

Recommendation: We recommend to adjust the `require` statement in L50 and set it to the same expression as L52 in `DiscretionaryExitBurn` and other contracts. They were likely intended to perform the same checks, but they are not equivalent.

BOB1-5 Stealing Liquidity Pool Fund with Reorg

● Medium Fixed

Update

Regarding the confirmation time, we discussed with the team and they confirmed that they will update the confirmation time to 64 blocks in the future. However, they have had to delay the change because some UI/UX components are associated with it, and it might take some time to implement the change in practice.

Update

The team added the `_updateDepositHash()` call in the `clientDepositL1Batch()` functions in both `L1LiquidityPool` and `L1LiquidityPoolAltL1` contracts in the commit `d578b81f`.

File(s) affected: `LP/L1LiquidityPool.sol`, `LP/L2LiquidityPool.sol`

Description: The `_updateDepositHash()` is used to update the deposit-related data hashes to protect from relaying reorg-ed messages (see: `L1CrossDomainMessengerFast.relay()` function). The function is introduced as the fix for the QSP-1 in the previous audit (see: report). However, the new `L1LiquidityPool.clientDepositL1Batch()` function does not call the `_updateDepositHash()` function. This means it will not update the deposit hashes, and the reorg protection will be bypassed.

The reorg risk lowered as Ethereum moved to POS, and the network has a better-defined finality. With the current consensus, the chain is considered "finalized" after two epochs, which is 64 blocks. However, if the liquidity pool of Boba exceeds the stake for the consensus on Ethereum, it can still be beneficial for the attacker to conduct a 51% attack.

Exploit Scenario: Here is a potential scenario:

1. Alice starts a 51% attack on L1. Let's say the canonical chain should be `chain1`, and Alice forked to `chain2` with a short period of 51% attack.
2. Alice bridges ETH and Boba from L1 to L2 with `tx1` on `chain2` using `L1LiquidityPool.clientDepositL1Batch()` function.
3. Alice immediately bridges Boba from L2 to L1, using the liquidity pool to withdraw quickly.
4. The `L1LiquidityPool.clientPayL1()` transaction pays Alice the withdrawal fund in `tx2`.
5. Alice cancels her 51% attack, and the longest chain becomes `chain1` again. However, `tx1` is no longer existing in `chain1`. However, `tx2` will still eventually be relayed. The validation in the `L1CrossDomainMessengerFast._verifyDepositHashes()` will not fail because the `tx1` does not change the deposit hash.

Recommendation: Call `_updateDepositHash()` once as part of the `L1LiquidityPool.clientDepositL1Batch()` function. Also, from the discussion with the Boba team, they only wait for 8 blocks of confirmation. We recommend increasing it to a minimum of 64 blocks to wait for the Ethereum finality.

BOB1-6 Unable to Sync Liquidity Pool Fee From L2

• Medium ⓘ Fixed

Update

The team confirmed there is no need to call the function through L2. Thus, the team deleted the `L2LiquidityPoolAltL1.configureFeeExits()` function and updated the misleading NatSpecs of `L1LiquidityPoolAltL1.configureFee()` in the commit `bf92c74a`. The team stated the following reason regarding why it is not needed to be called from L2:

> `L2LiquidityPoolAltL1.configureFeeExits()` method has been changed to be directly called by the owner, since the DAO does not exist on alt -L1s, there is no requirement to initiate the change from L2

File(s) affected: `LP/L1LiquidityPoolAltL1.sol`, `LP/L2LiquidityPoolAltL1.sol`

Description: The `L1LiquidityPoolAltL1.configureFee()` has the modifier `onlyOwner()`, which only allows the contract owner to call the function. However, the NatSpecs states: "Configure fee of this contract. called from L2". If the function is supposed to be called by L2, the modifier should be `onlyFromCrossDomainAccount()` instead of `onlyOwner()`. The `L2LiquidityPoolAltL1.configureFeeExits()` function will also send the `L1LiquidityPool.configureFee` message. So it seems like a mis-implementation.

Exploit Scenario: The DAO calls the `L2LiquidityPoolAltL1.configureFeeExits()`, but it will not update the config in L1.

Recommendation: The team should clarify the expected behavior. Change the modifier if the `L1LiquidityPoolAltL1.configureFee()` function should be called from L2. Otherwise, consider deleting the `L2LiquidityPoolAltL1.configureFeeExits()` function and update the misleading NatSpecs of `L1LiquidityPoolAltL1.configureFee()`.

BOB1-7 Contract Ownership Can Be Claimed Back by Anyone Once Renounced

• Medium ⓘ Mitigated

Update

The team stated that since the contracts have already been deployed, changing the initialization at this point would not be helpful. Therefore, the team will keep the existing code regarding the initialization. Meanwhile, the team added a check that `transferOwnership()` cannot transfer the ownership to the zero address in the commit `a20c6ca`. The mitigation solves the risk in the future. However, we flag this issue as mitigated because if a new contract is deployed, the privileged functions can still be called before the `initialized()` function sets the owner.

File(s) affected: `ERC721Bridges/L2NFTBridge.sol`, `ERC721Bridges/L2NFTBridgeAltL1.sol`, `ERC1155Bridge/L2ERC1155Bridge.sol`, `ERC1155Bridge/L2ERC1155BridgeAltL1.sol`, `LP/L1LiquidityPool.sol`, `LP/L1LiquidityPoolAltL1.sol`, `LP/L2LiquidityPool.sol`, `LP/L2LiquidityPoolAltL1.sol`

Description: Several functions in the system are protected by the modifier `onlyOwner()` which makes sure that only the owner of the contract is allowed to execute them. However, an additional conditional statement `|| owner == address(0)` has been added to the standard body of that modifier in several contracts. As a result, if the current owner of the contract decides to renounce his ownership (meaning, calling the function `transferOwnership()` with the parameter `_newOwner == address(0)`), every protected function from that contract will become executable by anyone. On top of that, as a consequence, anyone could call again the function `transferOwnership()` with the parameter `_newOwner == his address` to claim ownership for himself.

Recommendation: To fix this issue, both of the following should be applied:
1. Consider removing the additional conditional statement `|| owner == address(0)`.
2. Remove the `onlyOwner()` modifier from all of the `initialize()` functions.

BOB1-8 Attacker Can Steal Tokens by Deploying Malicious Contracts

• Medium ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation: Before registering new pairs, the native NFT contracts are to be manually evaluated as per their compatibility and security (along with some automated test cases that are yet to be developed). Most NFTs to be registered as a pair on the NFTBridge should be projects with recognition. For the representation tokens on the non-native layer (L*StandardERC721,L*StandardERC1155), the Boba team could deploy these contracts that work with their own centralized bridge. The NFTs however could have completely different other representation tokens deployed that could work with other bridges.

File(s) affected: `ERC1155Bridges/*.sol`, `ERC721Bridges/*.sol`, `standards/L1StandardERC721.sol`, `standards/L2StandardERC721.sol`, `standards/L1StandardERC1155.sol`, `standards/L2StandardERC1155.sol`

Description: A native ERC721 and ERC1155 token requires a complementing contract on the other chain in order to be bridgeable. In its current design, a user is responsible for deploying a `L1StandardERC721`, `L2StandardERC721`, `L1StandardERC1155`, or `L2StandardERC1155` contract. Subsequently, a user has to request registration of this contract on both chains.

A malicious user, however, could deploy a malicious contract in order to obtain the bridged token on the native chain. There are many possible ways for an attacker to hide malicious logic in such a deployed contract.

The severity of this issue arises through the fact that a configured token contract pair cannot be modified. If Boba admins fail to recognize malicious logic in a contract and register it on one or both chains, it cannot be reverted. This has three main consequences:

- The affected native token contract will never be bridgeable through these contracts anymore (without loosing those tokens).
- Users might accidentally use the bridge to a registered malicious contract and loose their tokens.
- There is no implemented functionality to pause a specific pair.

Recommendation: Elaborate on the process and checklist used to register new contracts. Consider bytecode verification to avoid unnoticed malicious logic of such deployed contracts.

BOB1-9 Risk of Registering Invalid NFT/token Pairs

● Medium ⓘ Mitigated

Update

1. For the first concern, the team fixed the issue as recommended in the commit `5fb99de`.
2. The team is concerned about the gas limit for the second concern. If the gas is insufficient, it might cause the message not able to be executed on the other layer and causes the pair to be non-useable.

Here is the original statement from the team:

> First concern was solved by adding the requirement on the registration step. For the second suggestion, automatic L1>L2 registerPair sync needs a configured or supplied gas limit. If somehow the gas limit is not enough - this would result in L1 contract having the registration while not on L2. This would make further make the bridge unusable

File(s) affected: `ERC1155Bridges/*.sol`, `ERC721Bridges/*.sol`

Description: We noticed some risks in the NFT or token "pair" registration process:

First, contract linkage is not checked during the registration process. When depositing an L2-native NFT on L1, a validation in L351 of `L1NFTBridge._initiateNFTDeposit()` ensures that the `L1StandardERC721` points to the same `l2Contract` as `pairNFTInfo`. However, this necessary check is not performed when calling `registerNFTPair`. This opens the possibility of mistakenly registering an invalid `L1StandardERC721`. Maintaining invalid mappings in `pairNFTInfo` should be avoided, particularly because they cannot be modified. This would lead to pairs and their native token contract being unable to be used anymore for this bridge.

The same reasoning applies to `L2NFTBridge._initiateWithdrawal()` in L378. Moreover, this is equivalently true for `pairTokenInfo` in the ERC1155 contracts. The checks are performed for `L1ERC1155Bridge` in L340 & L444, and for `L2ERC1155Bridge` in L367 & L474.

Secondly, the functions in different layers are registered separately by the contract owner (`onlyOwner()` modifier on the registration functions). If there is a mismatch between the configurations on the layer 1 and layer 2 chains, the token will be unusable by the bridge. There is no way to update a token's entry. Since it is easy to send messages across layers, it is less risky if one side of the contract updates its configuration by accepting the relayed message. So the contract owner only needs to call the function in a single layer once, and the other layer will get synchronized automatically.

The severity of this issue is escalated to medium because once the pair is registered, the incorrect pair cannot be updated/fixed.

Recommendation: To solve the first concern, in the `L1NFTBridge` contract, we recommend moving the following check from `_initiateNFTDeposit()` to `registerNFTPair()`. The change ensures that only `L1StandardERC721` are registered, and the same check no longer has to be performed in `_initiateNFTDeposit(..)`.

```
address l2Contract = IL1StandardERC721(_l1Contract).l2Contract();
require(pairNFT.l2Contract == l2Contract, "L2 NFT Contract Address Error");
```

Equivalently, please move those checks for the above lines in `L2NFTBridge(AltL1)`, `L1ERC1155Bridge`, and `L2ERC1155Bridge(AltL1)` into their respective initiation function.

As for the second concern, we recommend automatically syncing the configuring message from L1 → L2. In this case:

1. The registration functions in L1 should send a message to L2. The following is the list of the impacted functions: `L1NFTBridge.registerNFTPair()` and `L1ERC1155Bridge.registerPair()`.
2. The L2 registration function should be updated with the `onlyFromCrossDomainAccount()` modifier and set accordingly. The following is the list of the impacted functions: `L2NFTBridge.registerNFTPair()`, `L2NFTBridgeAltL1.registerNFTPair()`, `L2ERC1155Bridge.registerPair()`, and `L2ERC1155BridgeAltL1.registerPair()`.

BOB1-10 Locking NFTs/tokens by Sending to a Non-Supporting Contract

• Low Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation: multiple reply needed for edge cases increases complexity. We will approach this with increased information on the UI side to avoid user mistakes

File(s) affected: `ERC721Bridges/L1NFTBridge.sol`, `ERC721Bridges/L2NFTBridge.sol`, `ERC1155Bridges/L1ERC1155Bridge.sol`, `ERC1155Bridges/L2ERC1155Bridge.sol`

Description: In both the ERC721 and ERC1155 bridges, their "finalize" functions will send the reverting message (when `replyNeeded` is true) only when the bridged NFTs/tokens' base network is on the other layer. For instance, when bridging a layer one native NFT back to layer one, even if the `IERC721(_l1Contract).safeTransferFrom(...)` fails, no reverting message will be sent back to layer two to release the locked NFT. However, ERC721 and ERC1155 check the `on721Received()` and `onERC1155Received()` callbacks if the receiver of the NFT/token is a contract. Once sent to an unexpected contract, the bridge transaction will fail but not revert to the source layer.

Exploit Scenario: Alice accidentally bridges her L1-based NFT from L2 to a contract that cannot handle ERC721. Alice will fail to finalize the L1 transaction, and her NFT will be stuck on the bridge.

Recommendation: Consider applying the try-catch pattern and set `replyNeeded` if the transfer fails regardless of the base network. However, naively applying the change will cause an endless loop as the functions might bounce the revert message back and forth between the same "finalize" functions. The fix might need to have a new function for the revert message.

BOB1-11 Overflow Risk

• Low Fixed

Update

The team locked contracts as listed the version to `0.8.9` in the commit `cf17a6f`. However, not all contracts in this repository have changed, and only those mentioned in this issue have changed.

File(s) affected: `LP/L2LiquidityPool.sol`, `ERC1155Bridges/L1ERC1155Bridge.sol`, `ERC1155Bridges/L2ERC1155Bridge.sol`, `ERC1155Bridges/L2ERC1155BridgeAltL1.sol`

Description: Before version 0.8, Solidity did not have built-in overflow/underflow checks, which left contracts vulnerable to arithmetic issues. The Boba codebase currently uses the `pragma solidity >0.7.5` version, which poses a risk. During this audit, we identified a list of places that may have overflow/underflow issues:

LP contracts:

1. `L2LiquidityPool.getUserRewardFeeRate()`: the calculation of `userRewardMinFeeRate * poolLiquidity / poolBalance` does not use the `SafeMath` library. The risk is rather low as `userRewardMinFeeRate` is limited to the configured value, and it needs a large `poolLiquidity` to trigger an overflow. Most tokens do not have such a large total supply.

`L1ERC1155Bridge` contract:

1. `_initiateDeposit()`: the line `deposits[_l1Contract][_tokenId] += _amount` can overflow.
2. `_initiateDepositBatch()`: the line `deposits[_l1Contract][_tokenIds[i]] += _amounts[i]` can overflow.
3. `finalizeWithdrawal()`: the line `deposits[_l1Contract][_tokenId] -= _amount` can overflow.
4. `finalizeWithdrawalBatch()`: the line `deposits[_l1Contract][_tokenIds[i]] -= _amounts[i]` can overflow.

`L2ERC1155Bridge` and `L2ERC1155BridgeAltL1` contracts:

1. `_initiateWithdrawal()`: the line `exits[_l2Contract][_tokenId] += _amount` can overflow.
2. `_initiateWithdrawalBatch()`: the line `exits[_l2Contract][_tokenIds[i]] += _amounts[i]` can overflow.
3. `finalizeDeposit()`: the line `exits[_l2Contract][_tokenId] -= _amount` can overflow.
4. `finalizeDepositBatch()`: the line `exits[_l2Contract][_tokenIds[i]] -= _amounts[i]` can overflow.

Recommendation: From the discussion with the Boba team, the implementation contracts are compiled and deployed with version `0.8.9`, so they should be safe (only the proxy contracts on Ethereum are with version `0.7.6`). However, we still recommend guarding it against the contract codebase itself and consider using `SafeMath` for the places listed or locking the contracts to a version later than `0.8`.

BOB1-12 Missing Input Validations

• Low Mitigated

Update

The team added several validations as recommended in the commit `a20c6ca`. The following is the list of validations still missing after the fix:

`L1NFTBridge` contract:

> - `configureGas()` : the range of `_depositL2Gas` still lacks validation.
> - `_initiateNFTDeposit()` : validations for `_l2Gas` and `_data` are still missing.
>
> `L2NFTBridge` and `L2NFTBridgeAltL1` contracts:
>
> - `configureGas()` : the range check of the `_exitL1Gas` is still missing.
> - `_initiateWithdrawal()` : the min value check for `_l1Gas` is still missing, and the `_data` size is not checked.
>
> `L1ERC1155Bridge` contract:
>
> - `configureGas()` : the range check of the `_depositL2Gas` is still missing.
> - `_initiateDeposit()` : the min value check for the `_l2Gas` is still missing, and the `_data` size is not checked.
> - `_initiateDepositBatch()` : the min value check for the `_l2Gas` is still missing, and the `_data` size is not checked.
>
> `L2ERC1155Bridge` and `L2ERC1155BridgeAltL1` contracts:
>
> - `configureGas()` : the range check of the `_exitL1Gas` is still missing.
> - `_initiateWithdrawal()` : the min value check for the `_l1Gas` is still missing, and the `_data` size is not checked.
> - `_initiateWithdrawalBatch()` : the min value check for the `_l1Gas` is still missing, and the `_data` size is not checked.

File(s) affected: `ERC721Bridges/L1NFTBridge.sol` , `ERC721Bridges/L2NFTBridge.sol` , `ERC721Bridges/L2NFTBridgeAltL1.sol` , `ERC1155Bridges/L1ERC1155Bridge.sol` , `ERC1155Bridges/L2ERC1155Bridge.sol` , `ERC1155Bridges/L2ERC1155BridgeAltL1.sol` , `LP/*.sol`

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. For instance, functions arguments of type `address` may be initialized with value `0x0` . The following is a list of places where we recommend adding more validations:

`L1NFTBridge` contract:

1. `transferOwnership()` : consider validating that the `_newOwner` is not `address(0)` to avoid revoking the ownership. However, if it is intended to have the ability to revoke the ownership, please document this.
2. `configureGas()` : consider having a minimal and maximal value that the `_depositL2Gas` should be in the range. Otherwise, if it is accidentally set to an impractical number, all cross-domain messages will fail.
3. `_initiateNFTDeposit()` :
   1. This function should have validation that the `_from == msg.sender` . The reason is that in some parts of the function, it collects the NFT with `IERC721.safeTransferFrom(_from, ...)` while in the other part of the function, it checks the authorization of the NFT with `msg.sender` in the line `require(msg.sender == NFTOwner, ...)` before burning the NFT. The mix of the two makes the expected behavior unclear. Note that currently all functions calling `_initiateNFTDeposit()` passes the `msg.sender` as the `_from` input.
   2. The input `_l2Gas` can potentially be insufficient for the message to relay. Consider validating against a minimum value otherwise, the message might fail.
   3. Consider checking `_data` input against a max size limit. Otherwise, it can be spammed or even not able to process large data sizes.
4. `registerNFTPair()` :
   1. Please validate that the `_l2Contract != address(0)` . Several places in this contract check that `pairNFT.l2Contract != address(0)` after the registration. If accidentally registered with a zero address, it can have unexpected consequences.
   2. We also recommend validating that `_l1Contract != address(0)` to avoid registering value for the empty key of the `pairNFTInfo[]` mapping.
   3. Validate that the non-base contract points to its correct counterpart (e.g., `require(IL1StandardERC721(_l1Contract).l2Contract() == _l2contract)`)

`L2NFTBridge` and `L2NFTBridgeAltL1` contracts:

1. `transferOwnership()` : consider validating that the `_newOwner` is not `address(0)` to avoid revoking the ownership. However, if it is intended to have the ability to revoke the ownership, please document this.
2. `configureGas()` : consider having a minimal and maximal value that the `_exitL1Gas` should be in the range. Otherwise, if it is accidentally set to an impractical number, all cross-domain messages will fail.
3. `_initiateWithdrawal()` :
   1. This function should have validation that the `_from == msg.sender` . The reason is that in some parts of the function, it collects the NFT with `IERC721.safeTransferFrom(_from, ...)` while in the other part of the function, it checks the authorization of the NFT with `msg.sender` in the line `require(msg.sender == NFTOwner, ...)` before burning the NFT. The mix of the two makes the expected behavior unclear. Note that currently, all functions calling `_initiateWithdrawal()` pass the `msg.sender` as the `_from` input.
   2. The input `_l1Gas` can potentially be insufficient for the message to relay. Consider validating against a minimum value otherwise, the message might fail.
   3. Consider checking `_data` input against a max size limit. Otherwise, it can be spammed or even not able to process large data sizes.
4. `registerNFTPair()` :
   1. Please validate that the `_l1Contract != address(0)` . Several places in this contract check that `pairNFT.l1Contract != address(0)` after the registration. If accidentally registered with a zero address, it can have unexpected consequences.
   2. We also recommend validating that `_l2Contract != address(0)` to avoid registering value for the empty key of the `pairNFTInfo[]` mapping.
   3. Validate that the non-base contract points to its correct counterpart (e.g., `require(IL1StandardERC721(_l1Contract).l2Contract() == _l2contract)`)

`L1ERC1155Bridge` contract:

1. `transferOwnership()` : consider validating that the `_newOwner` is not `address(0)` to avoid revoking the ownership. However, if it is intended to have the ability to revoke the ownership, please document this.

2. `configureGas()` : consider having a minimal and maximal value that the `_depositL2Gas` should be in the range. Otherwise, if it is accidentally set to an impractical number, all cross-domain messages will fail.
3. `registerPair()` :
    1. Please validate that the `_l2Contract != address(0)` . Several places in this contract check that `l2Contract != address(0)` after the registration. If accidentally registered with a zero address, it can have unexpected consequences.
    2. We also recommend validating that `_l1Contract != address(0)` to avoid registering value for the empty key of the `pairTokenInfo[]` mapping.
4. `_initiateDeposit()` :
    1. This function should have validation that the `_from == msg.sender` . The reason is that in some parts of the function, it collects the tokens with `IERC1155.safeTransferFrom(_from, ...)` while in the other part of the function, it checks the balance of the tokens with `msg.sender` in the line `IL1StandardERC1155(_l1Contract).balanceOf(msg.sender,...)` before burning the tokens of the `msg.sender` . The mix of the two makes the expected behavior unclear. Note that currently, all functions calling `_initiateDeposit()` pass the `msg.sender` as the `_from` input.
    2. The input `_l2Gas` can potentially be insufficient for the message to relay. Consider validating against a minimum value otherwise, the message might fail.
    3. Consider checking `_data` input against a max size limit. Otherwise, it can be spammed or even not able to process large data sizes.
5. `_initiateDepositBatch()` :
    1. Same as `_initiateDeposit()` , this function should check that `_from == msg.sender` , `_l2Gas` is larger than a minimum size, and the `_data` input should be capped within a maximum size.
    2. The function should also check that `_tokenIds.length == _amounts.length` in case the two arrays do not match. Currently, the function will only revert if `_tokenIds.length > _amounts.length` but the inverse situation will remain undetected and trigger a cross-chain action.

`L2ERC1155Bridge` and `L2ERC1155BridgeAltL1` contracts:

1. `transferOwnership()` : consider validating that the `_newOwner` is not `address(0)` to avoid revoking the ownership. However, if it is intended to have the ability to revoke the ownership, please document this.
2. `configureGas()` : consider having a minimal and maximal value that the `_exitL1Gas` should be in the range. Otherwise, if it is accidentally set to an impractical number, all cross-domain messages will fail.
3. `registerPair()` :
    1. Please validate that the `_l1Contract != address(0)` . Several places in this contract check that `l1Contract != address(0)` after the registration. If accidentally registered with a zero address, it can have unexpected consequences.
    2. We also recommend validating that `_l2Contract != address(0)` to avoid registering value for the empty key of the `pairTokenInfo[]` mapping.
    3. Validate that the non-base contract points to its correct counterpart (e.g., `require(IL1StandardERC721(_l1Contract).l2Contract() == _l2contract)`)
4. `_initiateWithdrawal()` :
    1. This function should have validation that the `_from == msg.sender` . The reason is that in some parts of the function, it collects the tokens with `IERC1155.safeTransferFrom(_from, ...)` while in the other part of the function, it checks the balance of the tokens with `msg.sender` in the line `IL1StandardERC1155(_l1Contract).balanceOf(msg.sender,...)` before burning the tokens of the `msg.sender` . The mix of the two makes the expected behavior unclear. Note that currently, all functions calling `_initiateWithdrawal()` pass the `msg.sender` as the `_from` input.
    2. The input `_l1Gas` can potentially be insufficient for the message to relay. Consider validating against a minimum value otherwise, the message might fail.
    3. Consider checking `_data` input against a max size limit. Otherwise, it can be spammed or even not able to process large data sizes.
5. `_initiateWithdrawalBatch()` :
    1. Same as `_initiateWithdrawal()` , this function should check that `_from == msg.sender` , `_l1Gas` is larger than a minimum size, and the `_data` input should be capped within a maximum size.
    2. The function should also check that `_tokenIds.length == _amounts.length` if the two arrays do not match.

`L1LiquidityPool` , `L2LiquidityPool` , `L1LiquidityPoolAltL1` , and `L2LiquidityPoolAltL1` contracts:

1. `withdrawLiquidity()` : consider adding a check that the amount is greater than zero. While this observation does not pose a direct security threat, it highlights a potential gap in the function's business logic. Ensuring that the withdrawal amount is greater than zero could help prevent unexpected behavior and improve the overall functionality of the smart contract.

Recommendation: Consider adding validations as pointed out in the description section.

BOB1-13
Bridge NFTs/tokens to Contracts with Different Ownership Across L1 and L2

• Low ⓘ Mitigated

Update

The team fixed as recommended for the `NFT bridges` in the commit `3af1450` and decided to leave the LP contracts as it is. The following is their original statement:

> the contract check to prevent user side errors were moved one level up as per suggestion. Validation were however not added to LP contracts to avoid adding new methods

File(s) affected: `ERC721Bridges/*`, `ERC1155Bridges/*`, `LP/*`

Description: Because of the behavior of the `CREATE` opcode, a user can create a contract on L1 and L2 that share the same address but have different bytecode. By default, several functions in their bridge or liquidity pool send the tokens or NFTs to the same address as where it is from. Without the validation, it is likely for a contract to send a token from L1 to L2 while the L2 address is a different contract. From the discussion with the Boba team, they added the validation `require(!Address.isContract(msg.sender), "Account not EOA")` in both the `ERC721Bridges/*` and `ERC1155Bridges/*` to mitigate this. However, we noticed a few potential issues with the current implementation:

1. **Inconsistent check on the NFT bridges**: the validation might only appear in part of the code flow in the function. For instance, in the `L1NFTBridge._initiateNFTDeposit()` function, the validation only exists when `pairNFT.baseNetwork == Network.L1`. The same validation does not apply when the base network is L2. However, a user can bridge an L2 native token to L1 and then transfer it to a contract and try to bridge back to L2 from the contract. The following is the list of functions we identified with a similar issue: `L1NFTBridge._initiateNFTDeposit()`, `L2NFTBridge._initiateWithdrawal()`, `L2NFTBridgeAltL1._initiateWithdrawal()`, `L1ERC1155Bridge._initiateDeposit()`, `L1ERC1155Bridge._initiateDepositBatch()`, `L2EC1155Bridge._initiateWithdrawal()`, `L2EC1155Bridge._initiateWithdrawalBatch()`, `L2EC1155BridgeAltL1._initiateWithdrawal()`, `L2EC1155BridgeAltL1._initiateWithdrawalBatch()`.
2. **Lacking validation in LP contracts**: the LP contracts do not have the same validation. However, the `L1LiquidityPool.clientDepositL1()` and `L2Liquidity.clientDepositL2()` function by default bridge to the same address on the different layers. A similar issue applies to `L1LiquidityPoolAltL1` and `L2LiquidityPoolAltL1` contracts.

Exploit Scenario: Here is a sample scenario:

1. Alice has a L2 native NFT token `n1`.
2. Alice bridges the `n1` to L1.
3. In L1, Alice transfers the `n1` to a multisig contract: `c1` which is owned by Alice and Bob.
4. Later, Alice and Bob want to bridge the `n1` back to L2 from `c1`. They trigger the `L1NFTBridge.depositNFT()` function, and the `n1` is now transferred to the `c1` address in L2.
5. Unfortunately, Alice and Bob do not own the `c1` address in L2. They lost their NFT.

Recommendation: We suggest the following:

1. A easier fix would be to implement consistent checks, or validate it at the beginning of the functions pointed out in the description section. However, As per the discussion with the team, some functions that allow the user to specify the `to` parameter, (e.g., `L1NFTBridge.depositNFTTo()`) might be able to lift the validation on checking EOA account. Judging from the intention, only functions that automatically set both `from` and `to` as the `msg.sender` need stricter validations to avoid user error. We recommend to move the validation one layer up from the internal function to the specific public/external functions: `L1NFTBridge.depositNFT()`, `L1NFTBridge.depositNFTWithExtraData()`, `L2NFTBridge.withdraw()`, `L2NFTBridge.withdrawWithExtraData()`, `L2NFTBridgeAltL1.withdraw()`, `L2NFTBridgeAltL1.withdrawWithExtraData()`, `L1ERC1155Bridge.deposit()`, `L1ERC1155Bridge.depositBatch()`, `L2ERC1155Bridge.withdraw()`, `L2ERC1155Bridge.withdrawBatch()`, `L2ERC1155BridgeAltL1.withdraw()`, `L2ERC1155BridgeAltL1.withdrawBatch()`.
2. Consider adding validation to the LP contracts.

BOB1-14
Contract Ownership Can Be Renounced or Lost with `transferOwnership()`

● Low ⓘ Fixed

Update

The team added the validation to block the transfer of the ownership to zero address in the commit `a20c6ca`. However, the risk of transferring to an address no one owns still exists. However, since that is a minor risk, we flagged this issue as fixed.

File(s) affected: `ERC721Bridge/L1NFTBridge.sol`, `ERC721Bridge/L2NFTBridge.sol`, `ERC1155Bridge/L1ERC1155Bridge.sol`, `ERC1155Bridge/L2ERC1155Bridge.sol`

Description: The function `transferOwnership()` lets the current owner update the current owner of the contract with an address `newOwner` sent as a parameter. Two issues are possible:

1. `newOwner == 0`: that case is equivalent to renouncing the contract's ownership. It would have the consequence of locking all functions with the modifier `onlyOwner()`.
2. `newOwner == an address controlled by no one`: that case would also be equivalent to renouncing the contract's ownership and would have the same consequence as above.

In `L1NFTBridge` and `L1ERC1155Bridge`, this would make it impossible to call functions with the `onlyOwner` modifier, such as `registerPair()` and `configureGas()`. For the other affected contracts, any user could call these privileged functions, as the `onlyOwner` implementation in these contracts considers the zero address to be an owner.

Recommendation:

1. If the contract should always have a valid owner, consider adding a check to ensure that `newOwner != 0`.
2. Consider adding a two-step mechanism to ensure that only addresses controlled by someone can become an owner. An option could be to use `Ownable2Step` designed by OpenZeppelin: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol.

BOB1-15 Initial Ownership Claimable by Anyone Both Directly and via a Proxy

• Low ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation: Since the contracts are already deployed and initialized we didn't create a new contract

File(s) affected: `ERC721/L1NFTBridge.sol` , `ERC721/L2NFTBridge.sol` , `ERC1155Bridge/L1ERC1155Bridge.sol` , `ERC1155Bridge/L2ERC1155Bridge.sol` , `ERC1155Bridge/L2ERC1155BridgeAltL1.sol` , `ERC721/L2NFTBridgeAltL1.sol`

Description: The `L1NFTBridge` contract is intended to be executed as an implementation contract using the `proxy/implementation` pattern. To serve as the constructor, the contract includes an `initialize()` function. However, a potential drawback of this approach is that anyone can call the `initialize()` function and gain ownership of the contract once it is deployed. It is important to note that this function can be invoked directly or via a proxy.

It is worth noting that this issue applies to all contracts that use an `initialize()` function.

Recommendation: Consider making sure that the function `initialize()` is not called by anyone else than the team. A `FactoryContract` could be used to ensure this operation is called within the same transaction as the contract deployment. Consider also calling the function `initialize()` from the constructor with empty or dummy values. These two actions will make it impossible to call that function once the contract is deployed (both directly and via a proxy).

Note that almost all contracts in this audit scope have the same issue.

BOB1-16 Risk of User Overcharging Fee

• Low ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `568d23b2a04ef95b4bd4e0738dc9feb59487eead` . The client provided the following explanation: the operator was replaced as per suggestion

File(s) affected: `ERC721Bridge/L2NFTBridgeAltL1.sol`

Description: Based on the first requirement of the function `L2NFTBridgeAltL1._initiateWithdrawal()` , which states `require(msg.value >= billingContract.exitFee(), "Insufficient Boba amount");` , the user may accidentally send more funds than necessary for the exit fee. Once the excess fee is provided, the user has no way to claim it back.

Recommendation: One solution is to replace the operator `>=` with the operator `==` (like what is done in the contract `L2ERC1155BridgeAltL1.sol`). Alternatively, the contract can return the extra fund collect back to the user.

BOB1-17 NFT/Token Can be Finalized without Registration on Target Chain

• Low ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation: We will approach with providing Information on token registrations on the UI

File(s) affected: `ERC721Bridges/*.sol` , `ERC1155Bridges/*.sol`

Description: When finalizing an ERC721 or ERC1155 on either L1 or L2, the pair will be accepted without registration on the target chain. However, to bridge it back to its native chain, it must be registered. If a user decides to bridge an NFT/Token without approved registration on the target chain, the user trusts the `owner` to register the pair. Otherwise, the asset cannot be bridged back.

Exploit Scenario:

1. For some reason, the operation team only registered an NFT pair on L1 and forgot to register the pair information on L2.
2. Alice bridges the NFT from L1 to L2, and it successfully finalized on L2.
3. When Alice wants to bridge the token back to L1, it will fail because the pair is not registered in L2.

Recommendation: The user should be explicitly informed that their pair should be registered on both chains before bridging in order to make sure that the reverse direction will be possible as well.

BOB1-18 Reentrancy Risk

• Low ⓘ Fixed

Update

The team ensures that the `L2LiquidityPool.withdrawReward()` function follows checks-effects-interactions pattern by moving the `burnXBOBA()` call above. Note that other functions remain unchanged, but they follow the checks-effects-interactions pattern.

Here is the original statement from the team:

> `withdrawLiquidity` on L2LP was changed to check-effects-interaction, `burnxBoba()` was moved before the external call

File(s) affected: `LP/*.sol`

Description: Different mechanisms are applied to prevent reentrancy attacks. However, they are inconsistently applied, and setting an upper boundary for gas is not always guaranteed prevention of reentrancy attacks.

The likelihood of reentrancy is lower when the function has access controls. However, the following functions use the low-level instruction `call()` without access controls might be at more risk:

1. `L1LiquidityPool(AltL1).withdrawLiquidity()`
2. `L1LiquidityPool(AltL1).withdrawReward()`
3. `L2LiquidityPool(AltL1).withdrawReward()`
4. `L2LiquidityPool(AltL1).withdrawLiquidity()`.

Recommendation: We do not find any obvious attacks during this audit. Still, double-checking the likelihood of reentrancy attacks and their impact is recommended. Also, consider implementing the `ReentrancyGuard` for more sensitive functions using the `nonReentrant` modifier.

BOB1-19 `withdrawRewards()` Not Reflecting the Latest Reward Changes

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation: Since this is not a compulsion, we avoid calling this as a trade off for saving gas. Users can be provided the option to update to latest rewards before withdrawing in order to withdraw all latest rewards altogether

File(s) affected: `LP/*.sol`

Description: In LP contracts, `accUserRewardPerShare` should change its value whenever a user pays fees for bridging. However, it is only updated on `addLiquidity()` and `withdrawLiquidity()`.

As a result, the rewards paid out when calling `withdrawReward()` are not up to date if fees were paid since the last call to `updateUserRewardPerShare()`.

These rewards are not lost and can be withdrawn later. However, the currently allocated rewards may not be up to date and not withdrawable.

Exploit Scenario:

1. Assuming from a clean state. Alice deposits 100, and there is 100 shares.
2. Now the contract collects 1000 rewards. However, the `accUserRewardPerShare` is not updated during reward collection (e.g., `L1LiquidityPool.clientPayL1()` function), so the value is still zero.
3. If Alice now calls the `withdrawReward()` function, since the `accUserRewardPerShare` is zero, she cannot withdraw anything. However, Alice can call `updateUserRewardPerShare()` before `withdrawReward()` to get around this issue.

Recommendation: Consider calling `updateUserRewardPerShare(..)` in `withdrawRewards()`. However, there is a trade-off for inflicted gas costs.

BOB1-20 Application Monitoring Can Be Improved by Emitting More Events

• Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `2cda117cac935bdb291ed5e80ec841cefd3613c9`, `5fb99de4f980db1bcce8696761e4c9a92cf95d8b`. The client provided the following explanation: more events were added as per suggestion

File(s) affected: `ERC721Bridges/L1NFTBridge.sol`, `ERC721Bridges/L2NFTBridge.sol`, `ERC721Bridges/L2NFTBridgeAltL1.sol`, `ERC1155Bridges/L1ERC1155Bridge.sol`, `ERC1155Bridges/L2ERC1155Bridge.sol`, `ERC1155Bridges/L2ERC1155BridgeAltL1.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs or hacks. Below we present a non-exhaustive list of events that could be emitted to improve application management:

`L1NFTBridge` contract:

1. `transferOwnership()`
2. `configureGas()`
3. `registerNFTPair()`

`L2NFTBridge` and `L2NFTBridgeAltL1` contracts:

1. `transferOwnership()`
2. `configureGas()`
3. `registerNFTPair()`

`L1ERC1155Bridge` contract:

1. `transferOwnership()`
2. `configureGas()`
3. `registerPair()`

`L2ERC1155Bridge` and `L2ERC1155BridgeAltL1` contracts:

1. `transferOwnership()`
2. `configureGas()`
3. `configureBillingContractAddress()`
4. `registerPair()`

Recommendation: Consider adding events to the places pointed out in the description section.

BOB1-21 Inheriting Non-Upgradeable Contracts

● Informational ⓘ Acknowledged

Update

We flagged the issue as acknowledged since the contracts are deployed, as mentioned in the recommendation, it cannot be fixed at this moment. The team did add the code comments as recommended in the commit `5984883`.

File(s) affected: `LP/L1LiquidityPool.sol`, `LP/L2LiquidityPool.sol`, `LP/L1LiquidityPoolAltL1.sol`, `LP/L2LiquidityPoolAltL1.sol`, `ERC721Bridges/L1NFTBridge.sol`, `ERC721Bridges/L2NFTBridge.sol`, `ERC721Bridges/L2NFTBridgeAltL1.sol`, `ERC1155Bridges/L1ERC1155Bridge.sol`, `ERC1155Bridges/L2ERC1155Bridge.sol`, `ERC1155Bridges/L2ERC1155BridgeAltL1.sol`

Description: Most contracts in the scope of this audit are upgradeable contracts. However, they either inherits the `CrossDomainEnabledFast` or the `CrossDomainEnabled` contract. Both `CrossDomainEnabled` and `CrossDomainEnabledFast` contracts does not have the `__gap` storage field that most upgradeable contracts have. The `__gap` is set in case future code changes requires using out new storage. Without the `__gap` field, the `CrossDomainEnabled` and `CrossDomainEnabledFast` are less flexible for future changes.

Recommendation: In general, we recommend to use `CrossDomainEnabledFastUpgradeable` and `CrossDomainEnabledUpgradeable` contracts with the `__gap` storages (need new implementation, though). However, since the contracts are already deployed, adding `__gap` storages is impractical now. We do not suggest changing the inherited contract now but adding comments to document those limits in the contracts. If there is a plan to upgrade or redeploy the contracts, we recommend swapping them for upgradeable ones.

BOB1-22 Partially Succeeded Batch Transfer

● Informational ⓘ Fixed

Update

Marked as "Fixed" by the client. Addressed in: `f47635e2bf36b215dfe6d2b1fce09c3074238318`. The client provided the following explanation: The intended behaviour was documented on the contracts

File(s) affected: `LP/L2LiquidityPool.sol`, `ERC1155Bridges/L1ERC1155Bridge.sol`, `ERC1155Bridges/L2ERC1155Bridge.sol`, `ERC1155Bridges/L2ERC1155BridgeAltL1.sol`

Description: The `L2LiquidityPool.clientPayL2Batch()` function loops through the tokens to call the `_initiateClientPayL2()` function to transfer funds to the user on L2. However, the `_initiateClientPayL2()` might not transfer the token to the user if the liquidity pool lacks the funds for the token and send a revert message back to L1 for the specific token. This makes it possible for some tokens to be bridged while others fail.

A similar pattern shows in the ERC1155 bridge as well. All the `L1ERC1155Bridge.finalizeWithdrawalBatch()`, `L2ERC1155Bridge.finalizeDepositBatch()`, and `L2ERC1155BridgeAltL1.finalizeDepositBatch()` functions might send the revert message for some of the tokens in the batch when `replyNeeded` is true.

The risk is not high as the user can try to bridge the failed token again later.

Exploit Scenario: Alice tries to bridge both ETH and Boba in the same batch. However, the L2 liquidity pool lacks funds for ETH. Alice will successfully bridge the Boba token but not the ETH. Without ETH on layer two, Alice cannot send L2 transactions to pay the gas.

Recommendation: Please clarify if this is the intended behavior. If this is the intended behavior, please document the risk to the user and add code documents in the corresponding functions. Alternatively, consider revising the implementation to ensure the whole batch succeeds or revert.

BOB1-23
Proxy Admins Could Update the Admin or the Implementation of the Proxy by Mistake

● Informational ⓘ Acknowledged

Update
The team checked and confirmed there is no selector collusion with the proxy at this moment. However, we would like to warn that future contract upgrades should be cautious on this as well, and the team might consider having some automation such as embedding the check into their CI to help prevent collision in the future.

File(s) affected: `libraries/Lib_ResolvedDelegateProxy.sol`

Description: Multiple contracts of the system are used following a proxy/implementation pattern, to let the team update their behavior, when necessary. The proxy contract used is the following: `Lib_ResolvedDelegateProxy.sol` . It has two main functions that will let the admin of the contract modify any of the storage variables:

1. the implementation contract can be changed with the function `setTargetContract()` ;
2. the `owner` of the proxy can be changed with the function `transferProxyOwnership()` ;

If another address than the owner calls the contract with a given function selector, he will be redirected to the function fallback that will execute that call on the implementation contract using a delegateCall.

However, as a consequence, it is possible in certain conditions for the admin to mistakenly change one of the two storage variables of the proxy. These conditions are the following:

- Let the function selector of the function `setTargetContract(address)` be `S1` ;
- Let the function selector of the function `transferProxyOwnership(address)` be `S2` ;
- Let the function selector of a function in the contract implementation taking one fixed-type parameter as an argument (preferably an `address`) be `S3` ; If the admin of the proxy wants to call the function `S3` with a value `A1` that can be considered by the EVM as equivalent to an `address` , and we have `S3 == S1` or `S3 == S2` , then the fallback function will not be executed as expected, because the function with the same selector of the proxy will be executed in priority. Such a scenario would result in `A1` being the new value of the associated storage variable in the proxy contract.

Recommendation: Even if that kind of scenario is unlikely, consider double-checking for each function of the implementation contract their selector and the parameters they accept. If you identify a selector collision, consider renaming the function identified. Also, avoid calling any function of the implementation contract with the admin of the proxy.

BOB1-24
Cross-Chain Transfers Could Be Blocked by Unsynchronized Admin Actions

● Informational ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation: These are indeed method that can interrupt cross-chain messages and the NFT Bridges and Liquidity Pools have centralized aspects in their operation

File(s) affected: `ERC721Bridges/L1NFTBridge.sol` , `ERC721Bridges/L2NFTBridge.sol` , `ERC721Bridges/L2NFTBridgeAltL1.sol` , `ERC1155Bridges/L1ERC1155Bridge.sol` , `ERC1155Bridges/L2ERC1155Bridge.sol` , `ERC1155Bridges/L1ERC1155BridgeAltL1.sol` , `LP/LiquidityPool.sol` , `LP/L1LiquidityPoolAltL1.sol` , `LP/L2LiquidityPool.sol` , `LP/L2LiquidityPoolAltL1.sol`

Description: For each cross-chain pair of contracts (L1NFTBridge/L2NFTBridge(AltL1), L1ERC1155Bridge/L2ERC1155Bridge(AltL1), L1LiquidityPool(AltL1)/L2LiquidityPool(AltL1)), the admin can block cross-chain transfers by modifying the following parameters:

1. adding/removing a pair;
2. pausing/unpausing the contracts;
3. updating the gas amount to use on the other layer;

In practice, all these actions are done separately because they are done on a different network. As a result, if actions for the first two items are not synchronized, it may still be possible to trigger new cross-chain requests on one layer when the other layer is already blocked. The same issue could happen when unpausing the contracts.

Recommendation: Consider documenting what can happen if the process is partially or totally blocked using the actions listed above:

- can a new cross-chain request be recorded on one side of the bridge and rejected on the other side?
- if yes, what scenario should be implemented to finalize or reimburse it? Consider doing the same analysis if the process is partially or totally put back into activity.

BOB1-25 Missing Interface Extension in Contracts

● Informational ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation: Since the contracts are already deployed and under a proxy, interfaces were not added and imported

File(s) affected: `LP/*`

Description: It is generally recommended for Solidity contracts to inherit interfaces to define an abstract behavior for the extending contract. This helps to ensure that the contract implements the required functions and adheres to the specified behavior. None of the interfaces declared for the contracts in the folder `LP/` is explicitly extended by the associated contract.

Recommendation: Consider using explicit interface extensions for the contracts listed above.

BOB1-26 Storage Slot Collision Between Proxy and Implementation Contracts

● Informational ⓘ Acknowledged

Update
We consider this issue as acknowledged, as we cannot verify the changes. However, the team states that the current contracts have been aware of this issue, and they documented this concern in their internal documents.

File(s) affected: `libraries/Lib_ResolvedDelegateProxy.sol`

Description: Multiple contracts of the system are used following a proxy/implementation pattern, to let the team update their behavior, when necessary. The proxy contract used is the following: `Lib_ResolvedDelegateProxy.sol`.

Its only storage variable, the mapping `addressManager` is stored at the first storage slot of the contract. As a result, there is a storage slot collision with the storage variable of the implementation contract stored at the first slot. For example, that variable is the variable `address messenger` for the contract `CrossDomainEnabled`.

However, the storage slot of a mapping is never used so there is no impact, only the storage location where the value of its keys is stored.

Still, we can imagine an issue if in a later contract implementation used by the same proxy, the code either:

- adds a mapping at the first storage slot and it is possible to give any value for any key;
- adds an array at the first storage slot, with a possible underflow/overflow and a way to give any value to any id of that array;
- adds assembly code where it is possible to write to any storage slot id of the contract;

In that case, it could be possible for unauthorized users to update the current value of `addressManager["proxyOwner"]` and `addressManager["proxyTarget"]` from the implementation contract.

Recommendation: Consider keeping in mind the existence of that storage slot collision when a new implementation contract is added.

BOB1-27 Gas Costs at the Network Level Could Block some Cross-Chain Operations

● Informational ⓘ Acknowledged

Update
The team states that they plan to make changes to the relayer contracts soon, which might impact the need for the Billing Contract. The pointed-out places, specifically the `_initiateWithdrawal()`, call the billing contract with hardcoded gas. Here is the original statement:

> the call on the mentioned contracts are used to send funds to the BillingContract. We have avoided making the change for now - since we intend to modify the relayers in the near future and that will impact the need for a billing contract

File(s) affected: `ERC721Bridges/L1NFTBridge.sol`, `ERC721Bridges/L2NFTBridge.sol`, `ERC721Bridges/L2NFTBridgeAltL1.sol`, `ERC1155Bridges/L2ERC1155BridgeAltL1.sol`

Description: Even if it is rare, the gas of some EVM opcodes can be modified for a particular network. To mitigate that, a function `configureGas()` lets the owner of the impact contract modify the gas to be used for the cross-chain operation. However, if the update is not made directly when the network update is released, some cross-chain operations could be negatively impacted. It is also the case in the function `_initiateWithdrawal()` of the contracts `L2NFTBridgeAltL1.sol` and `L2ERC1155BridgeAltL1.sol` where the low-level instruction `call()` is used with a hardcoded gas amount of `3000`.

Recommendation: Consider avoiding hardcoding gas amounts in the contract and putting in place internally a monitoring and alert mechanism to identify any plan of gas cost update on the networks where the system is deployed.

BOB1-28 Ambiguity of `from` and `msg.sender` in the Bridges

● Informational ⓘ Fixed

Update
Marked as "Fixed" by the client. Addressed in: `a20c6cae92aa75f11320957a3246b991d4aaae7a`. The client provided the following explanation: a from == msg.sender requirement was set for the methods

File(s) affected: `ERC721Bridges/*.sol`, `ERC1155Bridges/*.sol`

Description: Several functions can have the `msg.sender` or the `_from` as the token/NFT owner. For instance, in the `L1NFTBridge._initiateNFTDeposit()` function, it collects the NFT with `IERC721.safeTransferFrom(_from, ...)` while in the other part of the function, it checks the authorization of the NFT with `msg.sender` in the line `require(msg.sender == NFTOwner, ...)` before burning the NFT. The mix of the two makes the expected behavior unclear. This is currently safe because all functions calling `_initiateNFTDeposit()` passes the `msg.sender` as the `_from` input.

The same pattern can also be found in `L2NFTBridge(AltL1)._initiateWithdrawal()`, `L1ERC1155Bridge._initiateDeposit()`, `L1ERC1155Bridge._initiateDepositBatch()`, `L2ERC1155Bridge(AltL1)._initiateWithdrawal()`, and `L2ERC1155Bridge(AltL1)._initiateWithdrawalBatch()` functions.

Exploit Scenario: `NFTWithdrawalFinalized` and `DepositFinalized` emit the addresses the token was moved `from` and `to`. However, if an operator were to trigger a withdrawal/deposit, the event would incorrectly list the operator as the `from` address rather than the token owner.

Recommendation:
1. Set the `from` address to the token owner rather than `msg.sender`.
2. If `msg.sender` should always be the same as `from`, add the validation to ensure that.

BOB1-29 Unlocked Pragma

● Informational ⓘ Mitigated

Update
The team locked the version to `0.8.9` for some contracts in the commit `cf17a6fe`. However, within the scope of this audit, contracts under `LP/`, `ERC721Bridges/`, and `standards` still have unlocked pragma.

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Also, since Solidity 0.8.0, the ABI Encoder V2 has been stabilized and is now the default encoder. If the team locks the Solidity to version `>= 0.8.0`, the `pragma experimental ABIEncoderV2` statements can also be removed.

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend removing the caret to lock the file onto a specific Solidity version.

Also, once the pragma is locked to a version >=0.8.0, consider removing the `pragma experimental ABIEncoderV2` from all contracts.

BOB1-30 `deposits` Is Not Updated upon NFT Withdrawal

● Informational ⓘ Fixed

Update
After discussing with the Boba team about a potential misunderstanding regarding the recommendation, the team resolved the issue in commit `7feda887` (PR#773).

Update
The team acknowledged the issue stating that the contract is deployed and would like to avoid deleting the storage value to keep the layout for the proxy contract. However, they might need clarification on our intention as we recommend deleting the value on certain conditions and not removing the storage.

File(s) affected: `ERC721Bridges/*.sol`

Description: The `deposit` variable in `L1NFTBridge` holds the same information as `pairNFTInfo`. We assume its intended usage is to hold the `l2Contract` value for a token while it is deposited. This assumption derives from the fact that `deposits` is used in the ERC1155 bridge contracts to keep track of the deposited token balances. Therefore, we would assume that it should be used in ERC721 to represent the binary state of the tokenID: "deposited" or "undeposited".

However, when the token is withdrawn, the value in `deposits` is not deleted to reflect the state change. It should be noted, that this inconsistency does not lead to any vulnerabilities.

The same holds true for the `exists` variable in `L2NFTBridge`.

Recommendation: We recommend deleting the `deposit` and `exists` values upon withdrawal/deposit for consistency.

BOB1-31 Centralization Risk of the Liquidity Pool

● Informational ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation: Centralization risks to be informed to the actors on the LP UI

File(s) affected: `LP/*.sol`

Description: To enable fast-exit L2 tokens, a centralized `Proposer` is operated by Boba, which is responsible for submitting the state of L2 to L1. A malicious `Proposer` could submit the wrong states to withdraw tokens from an L1 liquidity pool.

This cannot be reversed as the token is already withdrawn from the pool when the error is detected and disputed.

Recommendation: This issue exists by design, but the risk should be brought to the attention of liquidity providers.

BOB1-32 Improvements to Incentive Model

● Informational ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation: Since the contracts hold significant funds and are already live, we are avoiding making a change to the fee logic at the moment

File(s) affected: `LP/*.sol`

Description: Currently, liquidity providers are more incentivized to provide liquidity the higher the demand of the pool is. More specifically, the rewards are higher if the provided balance is higher than the liquidity available.

While this metric is valuable in incentivizing liquidity provision, there should also be an incentive to provide liquidity to smaller pools. This would help to get liquidity pools starting.

Recommendation: To improve the incentive model, consider marking a threshold of pool balance up to which the rewarded fees decrease. This can be combined with increasing incentive for higher balance than available liquidity.

BOB1-33 Inconsistencies in Event Emitting Pattern

● Informational ⓘ Acknowledged

Update
Marked as "Acknowledged" by the client. The client provided the following explanation: We are avoiding changing the order of variables of the events, because these contracts have been live for some time and there might be third party services that are currently watching the events

File(s) affected: `LP/L1LiquidityPool.sol`, `LP/L1LiquidityPoolAltL1.sol`

Description: In `L1LiquidityPool` and `L1LiquidityPoolAltL1`, the parameter ordering in the two events `ClientDepositL1` and `ClientDepostL1Batch` are inconsistent. This might lead to difficulties for developers trying to read those events.

In the `ClientDepositL1` event, the parameter sequence is "address, amount, tokenAddress". However, for each `ClientPayToken` in `ClientDepositL1Batch`, the order is "address, tokenAddress, amount".

Recommendation: Consider changing the parameter ordering in the `ClientPayToken` struct or `ClientDepositL1` event for consistency.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

1. Some functions are described with a NatSpec comment, but it is not the case in all files.
2. **(fixed)** Consider adding a comment to the NatSpecs of the `L2LiquidityPool.clientDepositL2()` function that user should approve the `billingContract.feeTokenAddress()` token with at least the `billingContract.exitFee()` amount before calling this function.
3. **(fixed)** Consider updating the NatSpecs of the `L2LiquidityPool.clientDepositL2()` function from "Client deposit ERC20 from their account to this contract..." to "Client deposit either ERC20 or ETH from their account to this contract, ..." as the function supports both ERC20 and ETH.
4. **(fixed)** In `L1LiquidityPool.clientDepositL1Batch()` function, the comment `// Construct calldata for L1LiquidityPool.clientPayL2Batch(ClientPayToken)` is wrong. The comment should be `... for L2LiquidityPool.clientPayL2Batch...` instead (L1 → L2). Similarly in the later comment `// Send calldata into L1` where it should be L2 instead.
5. **(fixed)** Consider adding a `@dev` NatSpecs to the `L2LiquidityPool.registerPool()` function that unlike the `_l2TokenAddress`, the `_l1TokenAddress` can be `adddress(0)`, and is used to represent the native L1 token.
6. **(fixed)** Consider updating the NatSpecs for the `L1ERC1155Bridge._initiateDepositBatch()` function to emphasize this is for "batch" transfer. "Performs the logic for deposits by informing the L2 Deposited Token" → ".... Deposited Token in batch".
7. **(fixed)** Please fix the wrong comment for `L2ERC1155Bridge._initiateWithdrawal()` function. "Performs the logic for withdrawals by burning the token and informing the L1 ERC721 Gateway ..." should be "L1 ERC1155" instead of "ERC721".
8. Some functions use NatSpec inheritance to use the content of the interface as function documentation. However, in some cases like in `L1NFTBridge.depositNFTWithExtraData()`, the associated interface has no NatSpec.
9. **(fixed)** The parameter `_baseTokenURI` is missing in the NatSpec of the constructor of the contract `L1StandardERC721`.
10. **(fixed)** There is a typo in the inline comment `// Construct calldata for _l2Contract.finalizeDeposit(_to, _amount)` of the function `L1ERC1155Bridge._initiateDepositBatch()`. The same remark is valid for the contracts `L2ERC1155Bridge.sol` and `L2ERC1155BridgeAltL1`.
11. **(fixed)** There is a duplicated inline comment in `L2ERC1155Bridge._initiateWithdrawal()`.
12. **(fixed)** `L1NFTBridge._initiateNFTDeposit()#L319 - L324`: Code comments are contradictory. There is probably a typo in L323: "so this will fail if _from is an EOA or address(0)". It probably should be "is not an EOA".

Adherence to Best Practices

1. Check if the team can benefit by adding an index to the `address` type field in the events in `L2LiquidityPool.sol`: `RebalanceLP`, `OwnershipTransferred`, and `DaoRoleTransferred`.
2. Check if the team can benefit by adding an index to the `address` type field in the `L1LiquidityPool.OwnershipTransferred` event.
3. **(fixed)** Consider removing the unused storage variable `extraGasRelay` in the `L2LiquidityPool` contract. If it is for upgrading reason to keep the storage layout, please add a comment indicate this. **Update:** comment is added.
4. Fix the error message for the `L1LiquidityPool.onlyL1StandardBridge()` modifier. The `require(address(L1StandardBridgeAddress) == msg.sender, ...)` validation does not match the error message "Can't receive ETH".
5. In the `L1NFTBridge.finalizeNFTWithdrawal()` function, the error message of the validation `require(deposits[_l1Contract][_tokenId] == _l2Contract, "Incorrect Burn")` seems misleading. The code block here does not burn the token.

6. It is recommended to name interfaces starting with a capital letter. It is not the case for all interfaces in the folders `ERC721Brides/`, `ERC1155Bridges/` and `LP/`.
7. Some common portions of code like functions or events are shared by different interfaces. Consolidating redundant code into a single interface could be considered wherever considered relevant.
8. **(fixed)** It is recommended to use the type notation `uint256` for `uint256` variables instead of the similar `uint` to make the code more explicit. That situation can be found for instance in the contract `L1NFTBridge`: `using SafeMath for uint;`.
9. At several locations, the operator `address()` is used for variables that already have the type `address`. In such a situation, the operator `address()` is redundant and can be removed. That situation can be found for instance in the contract `L1NFTBridge`, modifier `onlyInitialized()`.
10. **(fixed)** There is a typo confirmed by the team at several locations (including in `L1NFTBridge._initiateNFTDeposit()`) where an inline comment states that `safeTransferFrom() will fail if _from is an EOA or address(0)`. The correct statement should be `safeTransferFrom would fail if it’s address(0). And the call fails if it’s not an EOA`.
11. At several locations (including in `L1NFTBridge._initiateNFTDeposit()`), the local variable `message` is declared in assigned to a value in two different steps. It is recommended to merge these two steps into one. Note that this remark is also valid in the contract `L2NftBridge.sol`.
12. The event emitted `L1NFTBridge.NFTDepositInitiated` does not store the information of whether the deposited token is native or not. If that information is important, consider adding that information as a boolean field of the event. Note that this remark is also valid for any event emitted when a token is deposited or withdrawn.
13. At several locations, like in `IL1StandardERC721` or `IL1StandardERC1155`, it is recommended to place the events before the functions. Consider following the order of layout recommended in the style guide of Solidity (https://docs.soliditylang.org/en/v0.8.17/style-guide.html#order-of-layout).
14. The hardcoded value `1400000` could be passed as a parameter to the function `initialize()` in the contract `L1NFTBridge.sol` to make the contract code more flexible. The same remark is valid for each contract where a hardcoded value is used in the function `initialize()`.
15. As the local variables l1 and l2 get a deterministic value in the function `L1NFTBridge.registerNFTPair()`, they could be stored as constant variables within the contract instead of being recalculated each time the function is executed. It is also the case in `L2NFTBridge`.
16. The format of import statements is inconsistent: both operators `import { X } from` and `import "Y.sol"` can be used in the same contract, like in the contract `L2NFTBridge.sol`.
17. In some contracts, items are imported but not used (ex: `OVM_GasPriceOracle.sol` in `L2NFTBridge.sol`).
18. In some contracts, items are imported twice (ex: `SafeERC20.sol` in `L2NFTBridge.sol`).
19. **(fixed)** The storage variable `extraGasRelay` is declared in the contracts `L2NFTBridge.sol`, `L2NFTBridgeAltL1.sol`, `L2LiquidityPool.sol` and `L2LiquidityPoolAltL1.sol` but never used. **Update:** comment is added to clarify this variable is no longer used.
20. Some functions like `L2NFTBridge.configureBillingContractAddress()` use the visibility keyword `public` when the keyword `external` could be used instead to reduce gas costs.
21. At several locations, when the variable `array.length` is used in a for loop, it could be stored before the loop in a local variable. It would reduce the execution gas.
22. Whenever necessary, consider adding the keyword `indexed` to the fields of the events declared in the interfaces `iL1LiquidityPool.sol` and `iL2LiquidityPool.sol` to simplify the monitoring of on-chain activities by external actors.
23. Because the `ABIEncoderV2` is not considered experimental anymore, it can be selected via `pragma abicoder v2` (please see above) since Solidity 0.7.4: https://docs.soliditylang.org/en/v0.8.2/layout-of-source-files.html?highlight=experimental#abiencoderv2
24. Add the `initializer` modifier to the constructors to disable users being able to call `initialize()` on the implementation contract.
25. Extract the `interfaceId` for the standard contracts to a constant.
26. `DiscretionaryExitBurn.burnAndWithdraw()`, L52: "Amount Incorrect" is not meaningful in terms of the requirement that either a `msg.value` or a ERC20 token address other than OVM_ETH has to be provided.
27. `DiscretionaryExitFee.payAndWithdraw()`, L50: "Amount Incorrect" is not meaningful in terms of the requirement that either a `msg.value` or a ERC20 token address other than OVM_ETH has to be provided.
28. `L1NFTBridge.finalizeNFTWithdrawal()`, L407: "Incorrect burn" does not provide any information on why the error occurred for the validation: `require(deposits[_l1Contract][_tokenId] == _l2Contract, ...)`.
29. `DiscretionaryExitFee.payAndWithdraw()`, L48: Transferring exit fees is performed by looking up necessary parameters in the `billingContract`. However, `billingContract` has a function performing the exact same operations as L48, which is `collectFee()`. Consider calling `L2BillingContract.collectFee()` in L48 instead.
30. `L1LiqudityPool.addLiquidity()`: the validations regarding the `msg.value` and `_tokenAddress` on L407 and L409 can be combined into one `require` statement to save on gas. Furthermore, there is the XOR operator `^` in Solidity.
31. In `L1NFTBridge.sol#L168` the hash for `L1` and `L2` are computed with every call of `registerNFTPair()`. It is advisable to store those hashes in a variable instead of computing them every time to save gas.
32. L332 - L347 and L367 - L382 are duplicated sections of code in the `L1NFTBridge._initiateNFTDeposit()` function. To save on deployment gas costs, it is advisable to reuse and move the message creation to the end of the function.
33. In the `L1NFTBridge` contract, `depositNFTWithExtraData()#L248-257` and `depositNFTWithExtraDataTo()#L279-288` are duplicated sections of code. To save on deployment gas cost, we recommend implementing another function that carries this code and is called from both, `depositNFTWithExtraData` and `depositNFTWithExtraDataTo` to get the `extraData`. By contrast, this was implemented in the ERC1155 bridge contracts.
34. On `L2NFTBridge.sol#L194`, the hash for `L1` and `L2` are computed with every call of `registerNFTPair(..)`. It is advisable to store those hashes in a variable instead of computing them every time to save gas.
35. In the `L2NFTBridge` contract, `withdrawWithExtraData()#L288-297` and `withdrawWithExtraDataTo()#L326-335` are duplicated sections of code. To save on deployment gas cost, we recommend implementing another function that carries this code and is called from both `withdrawWithExtraData` and `withdrawWithExtraDataTo` to get the `extraData`. By contrast, this was implemented in the ERC1155 bridge contracts.
36. In `L1ERC1155Bridge.sol#L167`, the hash for `L1` and `L2` are computed with every call of `registerPair(..)`. It is advisable to store those hashes in a variable instead of computing them every time to save gas.
37. **(fixed)** Fix the typo in `L1LiquidityPool.clientDepositL1Batch()#L531`: "Invaild Token" → "Invalid Token".
38. There are several values introduced in the code that lack supporting documentation, and therefore we cannot easily verify the correctness:

1. In `L1NFTBridge.registerNFTPair()`: `0xec88b5ce`;
2. In `L1ERC1155Bridge.registerPair()`: `0xc8a973c4`;
3. In `L2ERC1155Bridge.registerPair()`: `0x945d1710`;

Adherence to Specification

1. The schema of the documentation `https://docs.boba.network/other/lp` is unclear: separate flows are drawn on the same chart and it is hard to identify where each flow starts.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `b16...a88 ./contracts/standards/IL2StandardERC1155.sol`
- `233...838 ./contracts/standards/L2CustomERC721.sol`
- `bf2...6a9 ./contracts/standards/IL1StandardERC1155.sol`
- `31f...de7 ./contracts/standards/IL1StandardERC721.sol`
- `7bf...204 ./contracts/standards/IL2StandardERC721.sol`
- `82c...da6 ./contracts/standards/L1StandardERC721.sol`
- `1e9...523 ./contracts/standards/L2StandardERC1155.sol`
- `217...a37 ./contracts/standards/L1StandardERC1155.sol`
- `941...bb9 ./contracts/standards/L1CustomERC721.sol`
- `19c...c43 ./contracts/standards/L2StandardERC721.sol`
- `79a...6b2 ./contracts/ERC721Bridges/L2NFTBridgeAltL1.sol`
- `0d1...73c ./contracts/ERC721Bridges/L1NFTBridge.sol`
- `79d...532 ./contracts/ERC721Bridges/L2NFTBridge.sol`
- `668...bf8 ./contracts/ERC721Bridges/interfaces/iL1NFTBridge.sol`
- `105...7d4 ./contracts/ERC721Bridges/interfaces/iL2NFTBridge.sol`
- `14b...b43 ./contracts/ERC721Bridges/interfaces/iSupportBridgeExtraData.sol`
- `8a0...e90 ./contracts/ERC721Bridges/interfaces/iL2NFTBridgeAltL1.sol`
- `8ba...196 ./contracts/ERC1155Bridges/L2ERC1155Bridge.sol`
- `afc...526 ./contracts/ERC1155Bridges/L1ERC1155Bridge.sol`
- `c83...cc0 ./contracts/ERC1155Bridges/L2ERC1155BridgeAltL1.sol`
- `410...45a ./contracts/ERC1155Bridges/interfaces/iL2ERC1155Bridge.sol`
- `1bb...95f ./contracts/ERC1155Bridges/interfaces/iL2ERC1155BridgeAltL1.sol`
- `ad0...558 ./contracts/ERC1155Bridges/interfaces/iL1ERC1155Bridge.sol`
- `2b3...93f ./contracts/LP/L2LiquidityPool.sol`
- `f04...804 ./contracts/LP/L1LiquidityPoolAltL1.sol`
- `d51...ad7 ./contracts/LP/L2LiquidityPoolAltL1.sol`
- `4d3...cdd ./contracts/LP/L1LiquidityPool.sol`
- `269...161 ./contracts/LP/interfaces/iL2LiquidityPool.sol`
- `830...bb4 ./contracts/LP/interfaces/iL1LiquidityPool.sol`

Tests

- `dc2...ab9 ./test/setup.ts`
- `ea8...c87 ./test/endToEndTests/teleportation.spec.ts`
- `238...52a ./test/endToEndTests/BobaFixedSavings.spec.ts`
- `ef1...81e ./test/endToEndTests/L2BillingContract.spec.ts`
- `1c9...0da ./test/endToEndTests/L2BillingContractAltL1.spec.ts`
- `c87...05c ./test/contracts/standards/l1standarderc721.spec.ts`
- `5a9...1fe ./test/contracts/standards/l2standarderc721.spec.ts`

- `b55...007 ./test/contracts/ERC721Bridges/L2NFTBridgeAltL1.spec.ts`
- `f29...ff2 ./test/contracts/ERC721Bridges/l1nftbridge.spec.ts`
- `6ec...f50 ./test/contracts/ERC721Bridges/l2nftbridge.spec.ts`
- `6da...fa3 ./test/contracts/ERC1155Bridges/l1ERC1155Bridges.spec.ts`
- `87f...138 ./test/contracts/ERC1155Bridges/L2ERC1155BridgesAltL1.spec.ts`
- `58f...a29 ./test/contracts/ERC1155Bridges/l2ERC1155Bridges.spec.ts`
- `68f...605 ./test/contracts/oracle/aggregator.spec.ts`
- `e7c...50b ./test/contracts/oracle/subscription.spec.ts`
- `872...6e3 ./test/contracts/oracle/aggregatorHC.spec.ts`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.9.3

Steps taken to run the tools:

1. In the root folder, runs `yarn && yarn build` to install the dependencies and build the project.
2. Install the Slither tool: `pip3 install slither-analyzer`.
3. Install `solc-select` by: `pip3 install solc-select`.
4. Change the `solc` version to be `0.8.9`: `solc-select install 0.8.9 && solc-select use 0.8.9`.
5. Run the Slither command: `slither packages/boba/contracts/contracts --solc-remaps @=node_modules/@ --filter-paths node_modules --checklist > slither.md`

Automated Analysis

Slither

Slither analyzed 143 contracts using 85 detectors, resulting in 634 findings. However, the majority of these findings were false positives or out-of-scope, and only valid ones were included in the report.

Test Suite Results

Steps to run the test:

1. In the root directory: `yarn && yarn build`.
2. Get into the contract folder: `cd packages/boba/contracts`
3. Run the coverage command: `yarn test`

Note the test result might include those files that are out-of-scope of this audit.

Update: After the fix-review, we ran coverage on commit `7feda887`, which was the latest commit of their main branch (`develop`) for the fix of BOB1-30.

```
yarn run v1.22.17
$ yarn test:contracts
$ hardhat test --show-stack-traces


  L2ERC1155BridgeAltL1 Tests
    L2ERC1155Bridge ownership
      ✔ should NOT be able to change the owner (147ms)
      ✔ changing gas reverts on not initialized (137ms)
    L2ERC1155Bridge tests initialized
      ✔ should be able to initialize and change the gas (152ms)
      ✔ should not be able to init twice (87ms)
      ✔ should not be able to init with zero address messenger
      ✔ should not be able to init with zero address L2ERC1155Bridge
    cover registerPair
      ✔ can register a token with L1 creation (59ms)
      ✔ can not register a token twice (64ms)
      ✔ cant register a NFT with incorrect settings (124ms)
      ✔ can register a token with L2 creation (38ms)
      ✔ cant register a token with faulty base network
      ✔ cant register if not owner

  L1ERC1155Bridge Tests
    L1ERC1155Bridge ownership
      ✔ should NOT be able to change the owner
      ✔ changing gas reverts on not initialized
    L1ERC1155Bridge tests initialized
      ✔ should be able to initialize and change the gas (112ms)
      ✔ should not be able to init twice (66ms)
      ✔ should not be able to init with zero address messenger
      ✔ should not be able to init with zero address L2ERC1155Bridge
    cover registerPair
      ✔ can register a token pair with L1 creation (46ms)
      ✔ can not register a NFT twice (67ms)
      ✔ can register a token with L2 creation (49ms)
      ✔ cant register a token with faulty base network
      ✔ cant register a NFT with incorrect settings (111ms)
      ✔ cant register if not owner

  L2ERC1155Bridge Tests
    L2ERC1155Bridge ownership
      ✔ should NOT be able to change the owner (50ms)
      ✔ changing gas reverts on not initialized
    L2ERC1155Bridge tests initialized
      ✔ should be able to initialize and change the gas (60ms)
      ✔ should not be able to init twice (127ms)
      ✔ should not be able to init with zero address messenger
      ✔ should not be able to init with zero address L2ERC1155Bridge
    cover registerPair
      ✔ can register a token with L1 creation
      ✔ can not register a token twice (53ms)
      ✔ cant register a NFT with incorrect settings (109ms)
      ✔ can register a token with L2 creation
      ✔ cant register a token with faulty base network
      ✔ cant register if not owner

  L2NFTBridgeAltL1 Tests
    L2NFTBridge ownership
      ✔ should NOT be able to change the owner
      ✔ changing gas reverts on not initialized
    L2NFTBridge tests initialized
      ✔ should be able to initialize and change the gas (55ms)
      ✔ should not be able to init twice
      ✔ should not be able to init with zero address messenger
      ✔ should not be able to init with zero address l2NFTbridge
    cover registerNFTPair
      ✔ can register a NFT with L1 creation
      ✔ can not register a NFT twice (52ms)
      ✔ cant register a NFT with incorrect settings (118ms)
      ✔ can register a NFT with L2 creation
      ✔ cant register a NFT with faulty base network
      ✔ cant register if not owner

  L1NFTBridge Tests
    L1NFTBridge ownership
      ✔ should NOT be able to change the owner
      ✔ changing gas reverts on not initialized
    L1NFTBridge tests initialized
      ✔ should be able to initialize and change the gas (47ms)
      ✔ should not be able to init twice
      ✔ should not be able to init with zero address messenger
      ✔ should not be able to init with zero address l2NFTbridge
    cover registerNFTPair
      ✔ can register a NFT with L1 creation
      ✔ can not register a NFT twice
      ✔ can register a NFT with L2 creation
      ✔ cant register a NFT with faulty base network
      ✔ cant register a NFT with incorrect settings (139ms)
      ✔ cant register if not owner
```

```
L2NFTBridge Tests
  L2NFTBridge ownership
    ✔ should NOT be able to change the owner
    ✔ changing gas reverts on not initialized
  L2NFTBridge tests initialized
    ✔ should be able to initialize and change the gas (51ms)
    ✔ should not be able to init twice
    ✔ should not be able to init with zero address messenger
    ✔ should not be able to init with zero address l2NFTbridge
  cover registerNFTPair
    ✔ can register a NFT with L1 creation
    ✔ can not register a NFT twice (43ms)
    ✔ cant register a NFT with incorrect settings (95ms)
    ✔ can register a NFT with L2 creation
    ✔ cant register a NFT with faulty base network
    ✔ cant register if not owner

Oracle Flux Aggregator Tests
  Optional payments for oracle submission
    ✔ should not allow submissions when funds depleted and voluntary submissions stopped (40ms)
    ✔ should allow submissions when funds depleted and voluntary submissions allowed (79ms)
  Aggregator available funds status
    ✔ should update available funds data on using method (42ms)
    ✔ should be able to withdraw directly transferred funds (52ms)
  Allow rounds between min-max submissions
    ✔ rounds with less than min submissions is timedOut (71ms)
    ✔ rounds with at least min submissions is not timedOut (89ms)

Oracle Flux Aggregator Tests
  owner tests
    ✔ should not intialize again
    ✔ should be able to transfer ownership
    ✔ should not be able to transfer ownership if not owner
    ✔ should update turing url
    ✔ should not update turing url if not owner
    ✔ should update turing address
    ✔ should not update turing address if not owner
    ✔ should update ChainLink's contract address
    ✔ should not update ChainLink's contract address if not owner
  Oracle admin tests
    ✔ should add an oracle
    ✔ should not add an oracle again
    ✔ should transfer oracle admin
    ✔ should not transfer oracle admin if not admin (38ms)
    ✔ should get oracle address
  Data submission tests
    ✔ should submit data for round 1001 (48ms)
    ✔ should not submit data twice for the same round id
    ✔ should not submit data for wrong chainLinkLatestRoundId (41ms)
    ✔ should continously submit data (54ms)

Oracle Subscription Tests
  Payment Parameters
    ✔ boba token cannot be zero address
    ✔ should set initial parameters (49ms)
    ✔ should not allow setting zero minimum subscription (49ms)
    ✔ should not allow non-owner to update payment parameters (74ms)
    ✔ should allow owner to updating payment parameters (61ms)
  Local Access
    ✔ should not allow access for period lower than min
    ✔ should not allow access if token transfer fails
    ✔ should be able to subscribe for local access (204ms)
  Local Access - Access Expiry
    ✔ should set expiry time correctly for fresh subscriber (46ms)
    ✔ should set expiry time correctly for active subscriber (74ms)
    ✔ should set expiry time correctly for inactive past subscriber (93ms)
    ✔ should prevent access after expiry (113ms)
  Global Access
    ✔ should not allow access for period lower than min
    ✔ should not allow access if token transfer fails
    ✔ should be able to subscribe for global access (91ms)
  Global Access - Access Expiry
```

```
      ✔ should set expiry time correctly for fresh subscriber
      ✔ should set expiry time correctly for active subscriber (75ms)
      ✔ should set expiry time correctly for inactive past subscriber (86ms)
      ✔ should prevent access after expiry (106ms)
    Multiple Access
      ✔ Global acccess should override Local acess (146ms)
    Owner Permissions
      ✔ should not allow non owner to increase expiry time
      ✔ should allow only the owner to increase expiry time (193ms)
      ✔ should not allow the owner to decrease expiry time (72ms)
      ✔ should allow owner to withdraw contract funds (364ms)
      ✔ should not allow non owner to withdraw contract funds (93ms)

  L1StandardERC721 Tests
    ✔ should deploy and check supportsInterface (83ms)

  L2StandardERC721 Tests
    ✔ should deploy and check supportsInterface (75ms)

  BobaFixedSavings Tests
    Initialization
      ✔ should have correct L2BOBA address set
      ✔ should have correct xBOBA address set
      ✔ should have correct owner address set
      ✔ totalStakeCount should be zero
      ✔ should not be able to initialize again
    Staking
      ✔ should fail staking zero amount
      ✔ should fail staking without enough BOBA balance (152ms)
      ✔ should fail staking with incorrect staking amount (49ms)
      ✔ should fail staking with incorrect amount approved (56ms)
      When user has enough BOBA approved
        ✔ should successfully stake amount with correct parameters (61ms)
        ✔ should increase total stake count
        ✔ should store the correct stake data (38ms)
        ✔ should mint xBOBA for user
        ✔ should increment stakeId for next stakes (319ms)
    Unstaking
      ✔ should not be able to unstake non owned stake
      ✔ should not be able to unstake stake before lock (61ms)
      when in unstaking period after lock
        ✔ should be able to unstake (60ms)
        ✔ should update the stake data
        ✔ should burn xBOBA for user (53ms)
        ✔ should not be able to unstake again (91ms)
      when unstaking after multiple periods
        ✔ should not allow unstaking at lock periods (174ms)
        ✔ should be able to unstake in the unstake period after second lock (215ms)
      when unstaking after contract interest is closed
        ✔ should not allow non owner to stop the interest bearing contract
        ✔ should allow owner to stop the interest bearing contract (55ms)
        ✔ should not allow to restop interest bearing contract
        ✔ should not allow new stakes after stopping contract
        ✔ should give out rewards until the contract was stopped (85ms)

  L2BillingContract Tests
    Initialization
      ✔ should revert when initialize with invalid params (46ms)
      ✔ should have correct address
      ✔ should have correct treasury address
      ✔ should have correct owner address set
      ✔ should have correct exit fee
      ✔ should not initialize twice
    Collect fee
      ✔ should revert when having insufficient balance
      ✔ should collect fee successfully (62ms)
      ✔ should not withdaw fee if balance is too low
      ✔ should withdraw fee successfully (60ms)

  L2BillingContractAltL1 Tests
    Initialization
      ✔ should revert when initialize with invalid params (308ms)
    ✔ should have correct address
    ✔ should have correct treasury address
    ✔ should have correct owner address set
    ✔ should have correct exit fee
    ✔ should not initialize twice
  Collect fee
    ✔ should revert when sending insufficient balance
    ✔ should collect fee successfully (99ms)
    ✔ should not withdaw fee if balance is too low
    ✔ should withdraw fee successfully (65ms)

BOBA Teleportation Tests
  Ethereum L2 - BOBA is not the native token
    ✔ should revert when initialize again
    ✔ should add the supported chain
    ✔ should not add the supported chain if it is added
    ✔ should not add the supported chain if caller is not owner
    ✔ should remove the supported chain
    ✔ should not remove if it is already not supported
    ✔ should not remove the supported chain if caller is not owner
    ✔ should teleport BOBA tokens and emit event (85ms)
    ✔ should not teleport BOBA tokens if the amount exceeds the daily limit (55ms)
    ✔ should reset the transferred amount (69ms)
    ✔ should revert if call teleportNativeBOBA function
    ✔ should revert if _toChainId is not supported
    ✔ should disburse BOBA tokens (75ms)
    ✔ should disburse BOBA tokens and emit events (41ms)
    ✔ should not disburse BOBA tokens if the depositId is wrong
    ✔ should not disburse tokens if it is not approved
    ✔ should not disburse tokens if caller is not disburser (39ms)
    ✔ should revert if disburse the native BOBA token
    ✔ should transfer disburser to another wallet
    ✔ should not transfer disburser to another wallet if caller is not owner
    ✔ should withdraw BOBA balance (51ms)
    ✔ should not withdraw BOBA balance if caller is not owner
    ✔ should pause contract (79ms)
    ✔ should unpause contract (118ms)
  Alt L2 - BOBA is the native token
    ✔ should revert when initialize again
    ✔ should add the supported chain
    ✔ should not add the supported chain if it is added
    ✔ should not add the supported chain if caller is not owner
    ✔ should remove the supported chain
    ✔ should not remove if it is already not supported
    ✔ should not remove the supported chain if caller is not owner
    ✔ should teleport BOBA tokens and emit event (50ms)
    ✔ should not teleport BOBA tokens if the amount exceeds the daily limit (65ms)
    ✔ should reset the transferred amount (68ms)
    ✔ should revert if call teleportBOBA function
    ✔ should revert if _toChainId is not supported
    ✔ should disburse BOBA tokens (70ms)
    ✔ should disburse BOBA tokens and emit events
    ✔ should not disburse BOBA tokens if the depositId is wrong
    ✔ should not disburse tokens if msg.value is wrong
    ✔ should not disburse tokens if caller is not disburser
    ✔ should transfer disburser to another wallet (43ms)
    ✔ should not transfer disburser to another wallet if caller is not owner
    ✔ should withdraw BOBA balance
    ✔ should not withdraw BOBA balance if caller is not owner
    ✔ should pause contract (57ms)
    ✔ should unpause contract (112ms)
  Alt L2 - Failed Disbursements
    ✔ should emit events when disbursement of BOBA tokens fail (102ms)
    ✔ should not be able to retry incorrect Disbursements
    ✔ should not be able to call from non disburser
    ✔ should be able to retry failed Disbursements
    ✔ should not be able to retry an already retried Disbursements
    ✔ should not be able to retry passed Disbursements
    ✔ should be able to retry a batch of failed disbursements (142ms)
  Admin tests
    ✔ should transferOwnership (44ms)
    ✔ should not transferOwnership if caller is not owner
```

```
      ✔ should not set minimum amount if caller is not owner
      ✔ should set minimum amount
      ✔ should not set minimum amount if caller is not owner
      ✔ should not set maximum amount if caller is not owner
      ✔ should set maximum amount
      ✔ should not set maximum amount if caller is not owner
      ✔ should set daily limit
      ✔ should not set daily limit if caller is not owner
    Init tests
      ✔ should not be able to set incorrect init values


  235 passing (39s)

Done in 60.53s.
```

Code Coverage

Steps to run the coverage:

1. In the root directory: `yarn && yarn build`.
2. Get into the contract folder: `cd packages/boba/contracts`
3. Run the coverage command: `yarn test:coverage`

The following table is the result of the coverage. We ran the coverage against all files in the `contracts` folder but exclude the result in the table. Note that the last row, "All files", are still the numbers of all the original files, including those we excluded from the table.

Update: After the fix-review, we ran coverage on commit `7feda887`, which was the latest commit of their main branch (`develop`) for the fix of BOB1-30.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/ERC1155Bridges/	26.65	25.77	32.73	27.78	
L1ERC1155Bridge.sol	27.88	27.59	35.29	29.25	... 628,641,648
L2ERC1155Bridge.sol	26.55	26.56	31.58	27.59	... 729,741,748
L2ERC1155BridgeAltL1.sol	25.64	23.61	31.58	26.67	... 738,750,757
contracts/ERC1155Bridges/interfaces/	100	100	100	100	
iL1ERC1155Bridge.sol	100	100	100	100	
iL2ERC1155Bridge.sol	100	100	100	100	
iL2ERC1155BridgeAltL1.sol	100	100	100	100	
contracts/ERC721Bridges/	34.63	30.12	36.73	35.85	
L1NFTBridge.sol	35.8	32	40	37.35	... 476,489,496
L2NFTBridge.sol	34.48	30.36	35.29	35.56	... 550,562,569
L2NFTBridgeAltL1.sol	33.71	28.33	35.29	34.78	... 556,568,575
contracts/ERC721Bridges/interfaces/	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
iL1NFTBridge.sol	100	100	100	100	
iL2NFTBridge.sol	100	100	100	100	
iL2NFTBridgeAltL1.sol	100	100	100	100	
iSupportBridgeExtraData.sol	100	100	100	100	
contracts/LP/	0	0	0	0	
L1LiquidityPool.sol	0	0	0	0	... 887,907,908
L1LiquidityPoolAltL1.sol	0	0	0	0	... 887,907,908
L2LiquidityPool.sol	0	0	0	0	... 6,1029,1030
L2LiquidityPoolAltL1.sol	0	0	0	0	... 910,913,914
contracts/LP/interfaces/	100	100	100	100	
iL1LiquidityPool.sol	100	100	100	100	
iL2LiquidityPool.sol	100	100	100	100	
contracts/standards/	33.67	18.18	28.79	33.64	
IL1StandardERC1155.sol	100	100	100	100	
IL1StandardERC721.sol	100	100	100	100	
IL2StandardERC1155.sol	100	100	100	100	
IL2StandardERC721.sol	100	100	100	100	
L1CustomERC721.sol	0	100	0	0	26,36
L1StandardERC1155.sol	30.77	0	28.57	28.57	... 55,57,61,63
L1StandardERC721.sol	45.45	0	33.33	41.67	... 48,52,54,58
L2CustomERC721.sol	0	100	0	0	27,37
L2GovernanceERC20.sol	0	0	0	0	... 66,70,78,82
L2StandardERC1155.sol	30.77	0	28.57	28.57	... 54,56,60,62
L2StandardERC721.sol	45.45	0	33.33	41.67	... 49,53,55,59
xL2GovernanceERC20.sol	50	33.33	45.83	52.94	... 172,176,180
All files	23.98	18.25	25.42	24.59	

Changelog

- 2023-04-05 - Initial report
- 2023-05-03 - Fix review report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT