Quantstamp

$BOBA Teleportation And Token As A Fee

Executive Summary

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

Type	Layer Two Blockchain / Bridge
Timeline	2023-04-03 through 2023-04-11
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Boba - Unaudited Features - Subset Quantstamp
Source Code	• bobanetwork/boba #aa609fa
Auditors	• Ibrahim Abouzied Auditing Engineer • Valerian Callens Senior Auditing Engineer • Andy Lin Senior Auditing Engineer • Adrian Koegl Auditing Engineer • Pavel Shabarkin Auditing Engineer

Documentation quality	Low
Test quality	Medium
Total Findings	24 Fixed: 3 Acknowledged: 17 Mitigated: 4
High severity findings	0
Medium severity findings	4 Fixed: 1 Acknowledged: 3
Low severity findings	12 Fixed: 1 Acknowledged: 8 Mitigated: 3
Undetermined severity findings	0
Informational findings	8 Fixed: 1 Acknowledged: 6 Mitigated: 1

Summary of Findings

Initial Audit: This Boba audit focused on two contracts: `Teleportation` and `Boba_GasPriceOracle`. The `Teleportation` contract is designed to help users transfer tokens across different L2's. The `Boba_GasPriceOracle` allows users to opt into using BOBA tokens for fee payments, as well as exchange a small amounts of their BOBA to ETH to help perform the meta-transactions without having to bridge their own ETH. Both of these processes rely on centralized off-chain components. The implementation and use of these off-chain components are outside the scope of this audit.

We would like to highlight some of the issues found in this audit. The `Teleportation` cannot retry failed disbursements on alternative L2 chains given the current method of tracking processed messages (BOBA-1). The `Teleportation` contract also has a global rate limit on token transfers. It is possible for one user to DOS the contract for others by consuming this entire limit themselves (BOBA-3). Given how `Boba_GasPriceOracle` uses both hardcoded prices and relies on an Oracle price feed, it is possible that the contract be drained of funds or rendered unusable in the right market conditions (BOBA-2 & BOBA-5).

In terms of project quality, the code was straightforward and exhibited consistent patterns. However, there is room for improvement in test coverage. As for documentation, general information about the project from Boba and Optimism is readily accessible. Nonetheless, documentation for new features could benefit from consolidation, as it is currently dispersed across multiple Google Docs, posing challenges for regular users to access.

Since the audit concentrated on the recent modifications to the roll-up protocol, the active involvement of the Boba team in addressing our inquiries was essential and significantly facilitated the completion of the audit.

Update: The Boba team has fixed some issues, most notably the issues around the inability to retry disbursements. However, many of the findings were left as acknowledged either because they were related to deployment concerns (and the contracts were already safely deployed) or due to the substantial efforts it would take to update the predeployed `Boba_GasPriceOracle.sol` contract. Given these findings, we encourage the Boba team to update their processes and monitor the protocol closely.

ID	DESCRIPTION	SEVERITY	STATUS
BOBA-1	Inability to Retry Disbursements	Medium	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
BOBA-2	Attacker May Drain `Boba_GasPriceOracle` of Value Given Arbitrage Opportunities	Medium	Acknowledged
BOBA-3	A DOS Attack Can Prevent Teleports	Medium	Acknowledged
BOBA-4	Privileged Roles and Ownership	Medium	Acknowledged
BOBA-5	Hardcoded Values May Not Hold in a Fluctuating Market	Low	Acknowledged
BOBA-6	New Min/max Price Ratio Can Break Current Price Ratio	Low	Acknowledged
BOBA-7	The `priceRatio` Can Exceed the `marketPriceRatio`	Low	Acknowledged
BOBA-8	Initial Ownership Is Claimable by Anyone Both Directly and via a Proxy	Low	Acknowledged
BOBA-9	`disburseNativeBOBA()` Locking Tokens with Failed Transfer	Low	Fixed
BOBA-10	A Drift in Supported Chains Can Lead to Locked Funds	Low	Mitigated
BOBA-11	Little Separation of Roles	Low	Acknowledged
BOBA-12	Effective Daily Limit Is Smaller than `maxTransferAmountPerDay`	Low	Acknowledged
BOBA-13	`disburser` and `owner` Are Set to the Same Address	Low	Acknowledged
BOBA-14	Contract Ownership Can Be Lost with `transferOwnership()`	Low	Acknowledged
BOBA-15	Missing Input Validation	Low	Mitigated
BOBA-16	`Teleportation` May Be Reentered	Low	Mitigated
BOBA-17	Implementing Soft Limits to Ensure Pricing Updates	Informational	Acknowledged
BOBA-18	Users Can Switch to the Fee Token They Have Already Configured	Informational	Acknowledged
BOBA-19	`feeWallet` Cannot Be Updated	Informational	Acknowledged
BOBA-20	`WithdrawBOBA` Event Emits the Wrong Address	Informational	Acknowledged
BOBA-21	Unlocked Pragma	Informational	Mitigated
BOBA-22	Transfer Instead of Call	Informational	Fixed
BOBA-23	The Distinction AltI2s / NotaltI2 Results in Large Portions of Unused Code	Informational	Acknowledged
BOBA-24	EOA-only Checks Can Be Bypassed	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**
>
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration 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.

Scope

All related off-chain components are beyond the scope of this audit.

Files Included

- `packages/boba/contracts/contracts/Teleportation.sol`
- `packages/contracts/contracts/L2/predeploys/Boba_GasPriceOracle.sol`

Findings

BOBA-1 Inability to Retry Disbursements

Medium | Fixed

> **Update**
>
> Marked as "Fixed" by the client. Addressed in: `9989649c24e695987acc428693081e032640d8bd` , `eba22769e7c14e130f4be2edba2beea3a29b3121` . The client provided the following explanation: A new method was added that allows retrying disbursements through the contract, and tests were added

> **Update**
>
> Though the update does not pose a security threat, we recommend two changes given the update:
> 1. Delete the `FailedNativeDisbursement` mapping after it's been successfully processed to receive a gas refund.
> 2. Emit an event to indicate a failure if `retryDisburseNativeBOBA()` fails.

File(s) affected: `Teleportation.sol`

Description: According to the discussion with the Boba team, the off-chain teleportation service will have the distributions in a file and retry the distribution if it fails. This mechanism works for the `disburseBOBA()` function, as the whole transaction will revert if it fails. However, the `disburseNativeBOBA()` allows native Boba transfers to fail and continue progressing with the subsequent distribution. In such a case, the

retry mechanism would not work because `totalDisbursements[_sourceChainId]` has increased, and the validation `_depositId == totalDisbursements[_sourceChainId]` will fail on the retry.

Exploit Scenario:

1. Assume a contract tries to send funds to another contract from ETH L2 to AltL2.
2. On the AltL2, the contract is temporarily paused and blocks token receiving.
3. It will fail when the distributor tries to distribute the token.
4. Later, the contract was unpaused. However, it cannot be retried since the `depositId` has been bumped.

Recommendation: The following are some potential solutions to this problem:

1. The team proposed manually sending the funds (without the contract) separately, given that Teleportation is centralized anyway. However, a downside is that it might be hard to prove that the team has processed the specific distribution without emitting an event on the chain.
2. To solve the problem of solution 1, an alternative might be exposing a function in the `Teleportation` contract to retry any `depositId`, and emit a log as proof of the distribution.
3. In the long term, the team might consider using hashed data as the `depositId`. For instance, it can be a hash of `(source_chain_id, target_chain_id, amount, recipient, nonce)`. The `nonce` can be the current incremental `depositId`. With the hash as the ID, the distributor service can retry automatically.

BOBA-2
Attacker May Drain `Boba_GasPriceOracle` of Value Given Arbitrage Opportunities

Medium | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: One reason to maintain the current structure is to allow change of oracles, while we look for ways to add a wrapper that allows aggregation. Since Boba_GasPriceOracle (implementation) is a predeploy and is also stored as a constant in the contract 'Lib_PredeployAddresses', any updates to Boba_GasPriceOracle would demand an update to all contracts that import 'Lib_PredeployAddresses' to be safe. Hence, we are currently avoiding doing changes to the contracts that are not critical/high, but taking a note of them to include in the subsequent bedrock network upgrade

File(s) affected: `Boba_GasPriceOracle.sol`

Description: When swapping BOBA for ETH through `swapBOBAForETHMetaTransaction(..)`, the exchange rate is determined by `marketPriceRatio`. To perform such a swap, the user has to pay an additional `_metaTransactionFee`, currently 3 BOBA.

An attacker could drain `Boba_GasPriceOracle` of value if the `owner` fails to update `marketPriceRatio` in times of high volatility. The Gas Price Oracle service updates the `marketPriceRatio` in intervals of five minutes. If the actual market price of 0.005 ETH is 3 BOBA higher than through `marketPriceRatio`, an attacker is incentivized to drain `Boba_GasPriceOracle` of ETH.

To formalize, an attacker has an incentive to drain the value whenever the following holds true:

```
0.005 ETH * actual_market_price > 0.05 ETH * marketPriceRatio + 3 BOBA
```

As of April 6 2023, 0.005 ETH is worth about 41 BOBA. Therefore, a relative value increase of 8% would suffice to create an incentive for an attacker. Phases of high volatility and/or when the service is unreliable at times might open a window for such an attacker. Please note that a lower required value increase might suffice in the future, depending on price developments.

While `priceRatio` may be stale as well, it might only lead to cheaper transaction fees which do not create a strong incentive for attackers.

The oracle creates another point of vulnerability. Its possible for the oracle to provide an incorrect price where the price of BOBA is overestimated in comparison with the price of ETH.

This could happen in two situations:

- a flash-crash of one of the two tokens ETH and BOBA would take time to be reflected in the contract. The team mentioned that price feeds are updated every 5 minutes.
- price manipulation by oracles: According to the documentation (https://docs.boba.network/other/oracle) the oracle network used by the contract (`Boba_Straw`) allows a minimum of one provider per price feed. It is profitable for oracles to manipulate the price feed for a short period if the number of native tokens to steal from the contract is higher than what they staked for financial sanctions for preventing price manipulation.

Recommendation: We recommend pulling the `marketPriceRatio` whenever `swapBOBAForETHMetaTransaction(..)` is called. This would remove this attack vector of draining the contract of ETH funds. However, If the current design is maintained, make sure to update `metaTransactionFee` to reduce the risk of such incentive windows by updating the required value increase.

Additionally, consider verifying the price feed contracts on the blockchain explorers to allow anyone to verify on-chain the activity of the network of oracles. Also, consider assessing the risks described above. If necessary, update the polling interval to update price feeds and limit the number of tokens owned by the contract.

BOBA-3 A DOS Attack Can Prevent Teleports

Medium | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: Since this is only one way to move Boba tokens around to another chain we are avoiding adding a fee for now. There exists the native way of using the LayerZero bridges to bridge tokens onto the other chain.

File(s) affected: `Teleportation.sol`

Description: Since the `Teleportation` contract does not charge a fee, attackers can exhaust the `maxTransferAmountPerDay` by calling `teleportBOBA()` or `teleportNativeBOBA()` functions to teleport the full value. The attack would be relatively cheap, only costing L2 gas. Once the daily limit is reached, other users can no longer bridge.

Any user owning at least `maxTransferAmountPerDay` BOBA may be incentivized to block the teleportations in the right situation, such as a governance decision impacting them is up for a vote. To do so, they could teleport enough BOBA through the different chains to reach the `maxTransferAmountPerDay` threshold on each supported chain, thereby blocking teleportations for the next 24 hours.

Recommendation: Consider charging a teleportation fee to mitigate DOS attacks.

BOBA-4 Privileged Roles and Ownership

Medium | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: We will approach this by making centralization risks clear on docs

File(s) affected: `Boba_GasPriceOracle.sol`

Description: Once a user chooses to use the Boba as the gas token, the user bears the centralization risk from the Boba team. The `Boba_GasPriceOracle` contract owner can update the Boba token's price ratio quite freely. If a user chooses to use Boba as the fee token, they will be charged based on the price ratio of this contract.

Additionally, the `updateMetaTransactionFee` function lacks a maximum fee limit enforcement. A compromised contract owner could alter the fee, causing users to pay more than anticipated. In instances where the fee becomes excessively high, the `swapBOBAForETHMetaTransaction` function would fail, as users would be unable to cover fee expenses.

Recommendation: Since there is no easy short-term solution, this centralization of power needs to be made clear to the users. In the long-term, the team should consider porting decentralized oracles into the ecosystem. Consider introducing a maximum fee limit for the `updateMetaTransactionFee` function.

BOBA-5
Hardcoded Values May Not Hold in a Fluctuating Market

Low | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: Since Boba_GasPriceOracle (implementation) is a predeploy and is also stored as a constant in the contract 'Lib_PredeployAddresses', any updates to Boba_GasPriceOracle would demand an update to all contracts that import 'Lib_PredeployAddresses' to be safe. Hence, we are currently avoiding doing changes to the contracts that are not critical/high, but taking a note of them to include in the subsequent bedrock network upgrade

File(s) affected: `Boba_GasPriceOracle.sol`

Description: The `Boba_GasPriceOracle` contract has several hardcoded magic constants. If the market price of Boba or ETH heavily fluctuates, these hardcoded prices may become outdated, rendering the contract unusable. The following is the list of the hardcoded values:

1. `MIN_WITHDRAWAL_AMOUNT`: this is used to set the minimum contract balance to allow withdraws of the Boba fee to the L1 fee contract (see the `withdrawBOBA()` function). However, when Boba's price heavily drops or the ETH drastically grows, the hardcoded amount might be less than the gas fee needed in L1 to relay the message. The price change will cause the Boba operation team to lose money when calling the `withdrawBoba()` function. Additionally, `withdrawBOBA()` can be called by anyone. An attacker could also prevent the contract from accumulating and withdrawing more fees by performing an early withdrawal. This would force the owners of `feeWallet` to wait for more fees to accumulate and meet the withdrawal threshold rather than get all of the tokens in one withdrawal.
2. `useBobaAsFeeToken()`: the function uses a hardcoded value `3e18` as the minimum Boba balance required for a user to set the Boba as the fee token. However, when the price of Boba and ETH diverges, this value might be insufficient or excessive, depending on the market situation.
3. `useETHAsFeeToken()`: the function hardcodes the `2e15` as the minimum ETH amount required to set the fee token as ETH. When the gas price largely increases, it might be possible for the threshold to be insufficient.
4. `updateReceivedETHAmount()`: the function hardcodes the `1e15` and `10e15` as the threshold for the `_receivedETHAmount` variable. The threshold might become unsuitable if the gas price increases or decreases too much.

Exploit Scenario: Here is a sample scenario to attack with the hardcoded `MIN_WITHDRAWAL_AMOUNT`:

1. Currently, `MIN_WITHDRAWAL_AMOUNT` is set as `150e18`. As for 2023-04-05, the Boba/ETH price is `0.00011978`. Thus, the value of the minimal amount of Boba tokens equals `0.017967` ETH.
2. Assume that the market price of Boba/ETH drops to `0.000001`.
3. The `MIN_WITHDRAWAL_AMOUNT` amount of the Boba token will be equivalent to `0.00015` ETH. This amount is likely insufficient to perform L1 calls on Ethereum.
4. Since anyone can call the `Boba_GasPriceOracle.withdrawBoba()` function, attackers can call the function to cause the Boba team either lose the Boba or ETH, as the team can only choose to either not to relay the message (loses Boba) or relay the message to L1 (loses ETH).

Recommendation: The following are our recommendations:

1. For point 1 in the description section, consider implementing a way to update the `MIN_WITHDRAWAL_AMOUNT`. Also, add an authorization check for the `withdrawBoba()` function.
2. For the points 2-4 in the description section, consider implementing a way to allow the contract owner to update the magic numbers: `3e18` (minimal Boba), `2e15` (minimal ETH), `1e15` (minimal `_receivedETHAmount`), and `10e15` (maximal `_receivedETHAmount`). However, if a hardcoded value is still preferred, consider using constant variables for better readability.

BOBA-6
New Min/max Price Ratio Can Break Current Price Ratio

• Low ⓘ Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: This is a good validation to have, but due to the stored constant on 'Lib_PredeployAddresses' for the implementation for BobaGasPriceOracle, we are considering including this on the next upgrade

File(s) affected: `Boba_GasPriceOracle.sol`

Description: The `updateMinPriceRatio()` and `updateMaxPriceRatio()` functions do not check whether the new min/max price ratio exceeds the current price ratio. Thus, the contract owner might accidentally update the min/max price ratio that breaks the current price.

Exploit Scenario:

1. The current `priceRatio` is 50, the `maxPriceRatio` is 100, and the `minPriceRatio` is 30.
2. The contract owner accidentally sets the `minPriceRatio` as 70.
3. Subsequent calls of the `updatePriceRatio()` are likely to fail unless the price ratio from the oracle increases over 70.

Recommendation: Consider validating that the updated min/max price ratio cannot break the current price ratio (both `priceRatio` and `marketPriceRatio`) in the `updateMinPriceRatio()` and `updateMaxPriceRatio()` functions.

BOBA-7 The `priceRatio` Can Exceed the `marketPriceRatio`

• Low ⓘ Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: This is a good validation to have, but due to the stored constant on 'Lib_PredeployAddresses' for the implementation for BobaGasPriceOracle (mentioned earlier), we are considering including this on the next upgrade

File(s) affected: `Boba_GasPriceOracle.sol`

Description: The `Boba_GasPriceOracle` tracks the exchange price from BOBA to ETH within the `priceRatio` and `marketPriceRatio` variables. The `priceRatio` represents a discounted exchange rate when using BOBA as a fee token, while the `marketPriceRatio` represents the undiscounted exchange rate. However, it is currently possible for the `priceRatio` to be greater than the `marketPriceRatio`.

Recommendation: In `updatePriceRatio()` validate that `priceRatio <= marketPriceRatio`.

BOBA-8
Initial Ownership Is Claimable by Anyone Both Directly and via a Proxy

• Low ⓘ Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. We note that this risk exists should the client ever perform an upgrade or redeployment. The client provided the following explanation: Since the contracts are already deployed and initialized we didn't create a new contract.

File(s) affected: `Boba_GasPriceOracle.sol`, `Teleportation.sol`

Description: Given that the contract is expected to be run as an implementation contract following the pattern `proxy/implementation`, the `initialize()` function exists to play the role of the constructor. The drawback is that, once deployed, anyone can call the function and acquire ownership of the contract. Note that this function can be called both directly and via a proxy.

Recommendation: Confirm that the function `initialize()` is not called by anyone outside the team. A Factory Contract could be used to ensure that this operation is called within the same transaction as the contract deployment. Consider also calling the function `initialize()` from the constructor. These two actions will make it impossible to call that function once the contract is deployed (both directly and via a proxy).

BOBA-9 `disburseNativeBOBA()` Locking Tokens with Failed Transfer

Low Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `9989649c24e695987acc428693081e032640d8bd`. The client provided the following explanation: The disburser can use the extra method added on QSP-1 to retry failed disbursements with the funds on the contract

File(s) affected: `Teleportation.sol`

Description: The `Teleportation.disburseNativeBOBA()` will continue distributing tokens even when some transfers fail. However, in such cases, those failed tokens will be locked in this contract. The distributor cannot get those out without reaching out to the contract owner for help withdrawing and returning the locked balance.

We expect the Boba team to run both the distributor and the owner, so the impact is low.

Recommendation: Consider refunding the distributor if there is any failed transfer. However, if this is the intended behavior, please document this.

BOBA-10 A Drift in Supported Chains Can Lead to Locked Funds

Low Mitigated

> **Update**
> Marked as "Mitigated" by the client.
>
> The team discovered that the pause mechanism is only applied on the "sender" side, and they anticipate that the centralized teleportation distributor will always bridge the funds once the initiation side has successfully bridged. We agree that the process and risks are clear enough and assess this issue as mitigated.
>
> We encourage the team to update their documentation to make this behavior clear to users.
>
> The client provided the following explanation: Unless contracts are paused, all withdrawals/deposits initiated will be disbursed on the other layer, since this does not include cross-chain messages. The limits are specified on the initiation side, and once a bridge is initiated it should be able to be finalized, courtesy of the centralized actor

File(s) affected: `Teleportation.sol`

Description: For each pair of `chainIds`, the admin can block cross-chain transfers by:

1. Pausing/unpausing the contracts;
2. Updating the allowed amounts (`minDepositAmount`, `maxDepositAmount`, `maxTransferAmountPerDay`);

In practice, all these actions are done separately on different networks. If the actions are not synchronized, it may still be possible to trigger new cross-chain requests on one layer when the other layer is already blocked. This would leave user funds locked in the source contract.

Recommendation: Consider documenting a resolution process for teleportations in unsynchronized contracts:

- Can a new cross-chain request be recorded on one side of the bridge and rejected on the other side?
- If so, what process should be put in place to finalize or reimburse the transaction? Consider doing the same analysis if the contracts are resynchronized.

BOBA-11 Little Separation of Roles

Low Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: the owner role is to be transferred to a multisig, pausing/unpausing, adding/removing supported chains, configuring min/max amount is something we expect to be an occasional task so we are sticking with ownership to be transferred to multisig

File(s) affected: `Teleportation.sol`

Description: The `owner` role is required in potentially daily and emergency activities such as `setMinAmount(..)`, `setMaxAmount(..)`, `pause()`, `unpause()`, `addSupportedChain(..)`, and `removeSupportedChain(..)`. Yet, it has access to sensitive functions as well,

including `transferDisburser(..)`, `transferOwner(..)`, `withdrawNativeBOBABalance(..)`, and `withdrawBOBABalance()`.

Accounts that are involved in frequent activities are more prone to compromisation. Therefore, the risk of a compromised `owner` is increased in this contract. In case an attacker successfully compromises the `owner` account, they can irreversibly acquire control, drain funds, and steal any future teleportations.

Recommendation: We recommend separating roles and clearly distinguishing frequently used functions from sensitive functions. Furthermore, the role with access to sensitive functions should be a multi-signature to prevent fund drainage.

BOBA-12
Effective Daily Limit Is Smaller than `maxTransferAmountPerDay`

• Low Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: The rolling 24-hour period is a security measure to protect the centralized actor providing the service, in order to limit the amount that it disburses in any rolling 24 hour period. In the above case extending the period by one hour is slightly more preferable for this reason

File(s) affected: `Teleportation.sol`

Description: The variable `transferTimestampCheckPoint` is used to track the last `block.timestamp` when `transferredAmount` was reset. However, `transferTimestampCheckPoint` is only updated when either `teleportBOBA(..)` or `teleportNativeBOBA(..)` are called. As a result, if these functions are not called for more than 24 hours, the `transferTimestampCheckPoint` will not be reset during that time.

This means that the practical daily limit could be reduced when the `transferTimestampCheckPoint` is reset after more than 24 hours. For example, if a user transfers tokens 25 hours after the previous reset, they would lose one hour of their daily limit, as the new `transferTimestampCheckPoint` would start counting from the time of the most recent transfer instead of exactly 24 hours ago.

Recommendation: Effectively reset the transferred amount at a fixed time, e.g., 12 AM. You can achieve this by implementing a function to check if the current date has changed. If it did, it will reset the `transferredAmount` and `transferTimestampCheckPoint` to the last fixed time (e.g., 12 AM) to ensure an accurate daily limit.

BOBA-13 `disburser` and `owner` Are Set to the Same Address

• Low Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: The ownership has to be transferred to a multisig, however until then we are considering controlling them by the same address, hence we are going without the requirement of having them explicitly different

File(s) affected: `Teleportation.sol`

Description: According to the documentation, those two roles should have different accounts:

> The relayer account and admin account are two different accounts.

However, they are both set to `msg.sender` in the `initialize()` function. If they are assigned to distinct addresses immediately after deployment, they can still be reassigned to the same address through calls to `transferOwnership()` and `transferDisburser()`.

Recommendation: Assign the roles to different addresses. Update `transferOwnership()` and `transferDisburser()` such that the two cannot be the same address.

BOBA-14
Contract Ownership Can Be Lost with `transferOwnership()`

• Low Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: since the contracts are already deployed we are not updating this - since this is an operational measure and as a last resort updating the contracts can be considered if contract ownership is transferred to a contract that cannot have ownership

File(s) affected: `Teleportation.sol`

Description: The function `transferOwnership()` allows the current owner to transfer their ownership of the contract to the address `newOwner`. However, if `newOwner` isn't an owned address, it would be equivalent to renouncing the contract's ownership and make it impossible to execute functions using the `onlyOwner()` modifier.

Recommendation: If the contract should always have a valid owner, consider adding a two-step mechanism to make sure 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

BOBA-15 Missing Input Validation

Low Mitigated

Update

Marked as "Fixed" by the client. Addressed in: `afc7dc5e607f4da7d0dc5586515a9bf3dd01c110`. The client provided the following explanation: Input validations were added

However, recommended validations for the following functions are still missing:

1. `disburseNativeBOBA()`
2. `disburseBOBA()`
3. `setMaxTransferAmountPerDay()`

File(s) affected: `Teleportation.sol`

Related Issue(s): SWC-123

Description: It is essential 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 we recommend adding more validations:

1. `initialize()`: consider validating that the `_minDepositAmount` and `_maxDepositAmount` should be larger than zero and `_minDepositAmount <= _maxDepositAmount`. Also, the `_maxDepositAmount <= maxTransferAmountPerDay` after setting the value of `maxTransferAmountPerDay`.
2. `disburseNativeBOBA()`: consider checking that `_disbursements[i].amount > 0` inside the loop to calculate `_totalDisbursed`.
3. `disburseBOBA()`: consider checking that `_disbursements[i].amount > 0` inside the loop to calculate `_totalDisbursed`.
4. `setMinAmount()`: consider checking that `_minDepositAmount > 0` and `_minDepositAmount <= _maxDepositAmount`.
5. `setMaxAmount()`: consider checking that `_maxDepositAmount > 0` and `_minDepositAmount <= _maxDepositAmount`.
6. `setMaxTransferAmountPerDay()`: consider checking that `_maxTransferAmountPerDay > 0` and `_maxTransferAmountPerDay >= _maxDepositAmount`.

Recommendation: Add validations as pointed out in the description section.

BOBA-16 `Teleportation` May Be Reentered

Low Mitigated

Update

Marked as "Mitigated" by the client. Addressed in: `a07ee58b6a8bdf0fcc19564b5c6c7cefafaa45f0`. The client provided the following explanation: The method is permissioned where only the disburser can call. Check-effects interaction also used

However, we would like to highlight some functions not having the mitigations as stated:

1. `teleportBOBA()`: the line `totalDeposits[_toChainId] += 1` is placed after the `IERC20(BobaTokenAddress).safeTransferFrom()` call, and this function is not permissioned.
2. `disburseNativeBOBA()`: Although permissioned, the function does not (and likely cannot) follow the CEI pattern, as the `failedNativeDisbursements[_depositId]` mapping is updated after the `_addr.call()` call.
3. `disburseBOBA()`: Although permissioned, the function does not follow the CEI pattern, as the `totalDisbursements[_sourceChainId]` mapping is updated after the `IERC20(BobaTokenAddress).safeTransferFrom()` call before the for-loop.
4. `retryDisburseNativeBOBA()`: Although permissioned, the function does not (and likely cannot) follow the CEI pattern, as the `failedNativeDisbursements[_depositId]` mapping is updated after the `_addr.call()` call.
5. Apart from the aforementioned functions, several other functions have events emitted after the external call, which may pose a risk of re-entrant events depending on the design of off-chain components, although this is outside the scope of this audit.

File(s) affected: `Teleportation.sol`

Description: When disbursing native BOBA through `disburseNativeBOBA(..)`, the `Teleportation` contract may be reentered by recipients of native BOBA. While the gas is limited to `3000`, this might not be sufficient to prevent re-entrancy.

Though we were not able to find a vulnerability, re-entrancy should be prevented as a best practice.

Recommendation: We recommend implementing a re-entrancy guard in addition to limiting the fallback gas usage.

BOBA-17 Implementing Soft Limits to Ensure Pricing Updates

Informational Acknowledged

Update

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: We are considering avoiding a fix here, since this might be counterproductive if its case 1, where an error in the oracle provides strange updates, and the contracts update the `priceRatio` to `threshold` in that case

File(s) affected: `Boba_GasPriceOracle.sol`

Description: The `Boba_GasPriceOracle.updatePriceRatio()` function allows the contract owner to update the price ratio as long as it is between the min/max threshold. There could be two possible reasons for the price to exceed the thresholds:

1. The Oracle experiences some issues and provides strange pricing updates.
2. The Oracle price is indeed the true market price, but the threshold is too restrictive.

For case one, this validation would help prevent updating with unexpected values. However, for case two, this will also block correct price updates. Applying a soft limit might be a better solution. In other words, when the value exceeds the min/max threshold, use the threshold value as the updated result. The soft limitation ensures the price moves in the correct direction instead of refusing to update entirely.

Recommendation: Consider applying a soft cap to the `updatePriceRatio()` instead of reverting when the price is out-of-range. Alternatively, another mitigation is that the gas-price-oracle service can embed the logic to ensure the value used for `updatePriceRatio()` will not exceed the threshold.

BOBA-18
Users Can Switch to the Fee Token They Have Already Configured

Informational | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: Since Boba_GasPriceOracle (implementation) is a predeploy and is also stored as a constant in the contract 'Lib_PredeployAddresses', any updates to Boba_GasPriceOracle would demand an update to all contracts that import 'Lib_PredeployAddresses' to be safe. Hence, we are currently avoiding doing changes to the contracts that are not critical/high, but taking a note of them to include in the subsequent bedrock network upgrade

File(s) affected: `Boba_GasPriceOracle.sol`

Description: Users can call `useETHAsFeeToken()` and `useBobaAsFeeToken()` even if they have already configured ETH or BOBA as their fee token, respectively. This may lead to user confusion.

Recommendation: We recommend adding a requirement to both functions that the fee token was not already configured in the respective way. This will inform the user that the intended token has already been configured.

BOBA-19 `feeWallet` Cannot Be Updated

Informational | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: Since this is an operational measure we are considering avoiding a fix for this currently

File(s) affected: `Boba_GasPriceOracle.sol`

Description: The `Boba_GasPriceOracle` contract cannot change the fee wallet address after deployment. This limitation brings the operational risk that once the private key of the fee wallet is leaked, it will have an irreversible impact.

Also, the address `feeWallet` is used to withdraw native tokens from the contract `Boba_GasPriceOracle.sol`, using the function `withdrawETH()`. However, withdrawals could fail if the account `feeWallet` is a contract that has no `receive()` or `fallback()` function. If the operations team cannot update the `feeWallet`, they will be unable to correct any mistakes made during its setup.

Additionally, it is possible for the same contract address to be controlled by different people in L1 and L2.

Recommendation: Consider adding a function to update the `feeWallet` address, as well as tracking the address of the `feeWallet` through separate variables for L1 and L2.

BOBA-20 `WithdrawBOBA` Event Emits the Wrong Address

Informational | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation: This is a good fix to do, but due to the stored constant on 'Lib_PredeployAddresses' for the implementation for BobaGasPriceOracle (mentioned earlier), we are considering including this on the next upgrade

File(s) affected: `Boba_GasPriceOracle.sol`

Description: Once the `withdrawBOBA()` function is complete, it emits the following event: `WithdrawBOBA(owner(), feeWallet)`. However, the function is callable by anyone and may not necessarily be triggered by the contract owner.

Recommendation: Pass `msg.sender` rather than `owner()` to the event. Alternatively, update the function to use the `onlyOwner` modifier.

BOBA-21 Unlocked Pragma

Informational | Mitigated

Update

Marked as "Fixed" by the client. Addressed in: `5afddb1ea83c1095319cea13b80970e1f2022dcc`. The client provided the following explanation: the pragma version was locked

However, the `Boba_GasPriceOracle` contract remains unchanged.

File(s) affected: `Teleportation.sol`, `Boba_GasPriceOracle.sol`

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".

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

BOBA-22 Transfer Instead of Call

Informational | Fixed

Update

Marked as "Fixed" by the client. Addressed in: `be978d0b4c53340e8ffb881b80efc180941b308c`. The client provided the following explanation: transfer was changed to low level call

File(s) affected: `Teleportation.sol`

Description: The `withdrawNativeBOBABalance` function uses the `transfer` function to execute ETH payments to the owner. The recipient smart contract must have a `fallback` function defined; otherwise, the `transfer` call will throw an error. A gas limit of 2300 gas is sufficient for completing the transfer operation. Although this particular operation does not present any security concerns, it is always advisable to maintain consistency in approaches and employ the low-level call function.

```
function withdrawNativeBOBABalance()
    external
    onlyOwner()
    onlyInitialized()
    onlyAltL2s()
{
    uint256 _balance = address(this).balance;
    payable(owner).transfer(_balance);
    emit NativeBOBABalanceWithdrawn(owner, _balance);
}
```

Recommendation: Change usage of `transfer` to the low-level `call` function.

BOBA-23 The Distinction AltL2s / NotaltL2 Results in Large Portions of Unused Code

Informational | Acknowledged

Update

Marked as "Acknowledged" by the client. The client provided the following explanation: We are considering sticking with the current design for now, since the contracts have been deployed

File(s) affected: `Teleportation.sol`

Description: Three pairs of functions are mutually exclusive depending on the chain on which the contract is deployed:

- `withdrawNativeBOBABalance()` and `withdrawBOBABalance()`;
- `teleportNativeBOBA()` and `teleportBoba()`;
- `disburseNativeBOBA()` and `disburseBoba()`;

The distinction is made using the mutually exclusive modifiers `onlyAltL2s()` and `onlyNotAltL2s()`.

It results in having two mutually exclusive behaviors in the same contract, where the behavior could be determined at deployment time. It ultimately leads to a contract with higher complexity and deployment costs.

Recommendation: Consider whether the contract should be split into two contracts, one for `AltL2s` and one for `NotAltL2s`.

BOBA-24 EOA-only Checks Can Be Bypassed

Informational | Acknowledged

> **Update**
> Marked as "Mitigated" by the client. Addressed in: `a07ee58b6a8bdf0fcc19564b5c6c7cefafaa45f0`. The client provided the following explanation: since the fee is always deducted from the tx.origins account, even if a contract has fee token set - cannot make use of the fee setting. (so there is no impact) Please confirm this assumption
>
> We agree with the analysis and have updated the issue's severity to Informational.

File(s) affected: `Boba_GasPriceOracle.sol`

Description: Several functions use require statements to authorize some actions only for contracts or EOA addresses. However, these checks can be bypassed in the functions `useBobaAsFeeToken()` and `useETHAsFeeToken()` if either of the following conditions is met:

- When a contract is created using the instruction `create2`, it is possible to deterministically calculate its address and send tokens to that address before the deployment of the contract;
- When a contract calls another contract from its constructor, it will be considered an EOA because the size of its code is still 0 for other contracts. During deployment, `Address.isContract()` will return `false` for the contract address.

Exploit Scenario:

- Before a contract's deployment, `3e18` BOBA are transferred to its precalculated address. It can then make a call to function `useBobaAsFeeToken()` from its constructor.
- Contracts by default accept ETH as a fee token. However, it is possible for a contract to accept back ETH as a fee token by following the exploit described above, and adding to its constructor a call to `useETHAsFeeToken()`.

Recommendation: Consider assessing if additional checks are necessary or if the identified impact is acceptable.

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. In the `Boba_GasPriceOracle.withdrawBOBA()` function, it calls the `L2StandardBridge(...).withdrawTo()` function with the `l1Gas` parameter as zero (the fourth input parameter). Consider adding a comment here stating that `l1Gas` is currently not used by the `withdrawTo()` function, and that is why setting it to zero.
2. Remove the irrelevant comment: "...If the disbursement fails, the amount will be kept by the contract rather than reverting to prevent blocking progress on other..." on the `Teleportation.disburseBOBA()` function. The comment is only valid in the `disburseNativeBOBA()` function. The `disburseBOBA()` function expects no failure from `IERC20(BobaTokenAddress).safeTransfer()` call.

Adherence to Best Practices

Boba_GasPriceOracle:

1. The team should check whether they can benefit from adding indexes to the events in the `Boba_GasPriceOracle` contract. Specifically, those with the `address` type might help off-chain components monitor future contract state changes. The following is a list of events that we think might have a good chance to benefit from adding indexes: `TransferOwnership`, `UseBobaAsFeeToken`, `SwapBOBAForETHMetaTransaction`, and `UseETHAsFeeToken`.
2. In the `Boba_GasPriceOracle.swapBOBAForETHMetaTransaction()` function, reuse the `getBOBAForSwap()` function to calculate the `totalCost`.
3. Save gas by not setting `address oldOwner = _owner` in `transferOwnership(..)`. Instead, the event `TransferOwnership` can be emitted before, such that `_owner` does not have to be cached.
4. The library `SafeMath` is imported and used at several locations to prevent integer undetected integer underflows and overflows when performing arithmetic operations. However, this is only valid when contracts were compiled with a `Solidity` compiler version < 0.8.0. As the contract uses `pragma solidity ^0.8.9`, underflows and overflows checks are embedded in the compiled code. As a result, using SafeMath is redundant and could be avoided.
5. The format of import statements is inconsistent: both operators `import { X } from` and `import "Y.sol"` are used in the same contract.
6. In the function `withdrawBOBA()`, the expression `L2GovernanceERC20(l2BobaAddress).balanceOf(address(this))` is called twice. Its content could be saved in a local variable to reduce the number of external calls.
7. At several locations, the function `owner()` is used, instead of directly using the storage variable `_owner`.
8. Some functions like `updateMetaTransactionFee()` use the visibility keyword `public` when the keyword `external` could be used instead to reduce gas costs.
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 modifier `onlyInitialized()`.
10. The utility of the event `WithdrawETH` could be improved by sending the amount transferred.

Teleportation:

1. Consider removing the `onlyNotInitialized()` modifier. The usage of the `Initializable.initializer()` modifier on `Teleportation.initialize()` is sufficient.
2. Consider replacing hardcoded magic numbers with readable constant variable names:
    1. `0x4200000000000000000000000000000000000006`: used in `onlyAltL2s()` and `onlyNotAltL2s()`. The address seems to be the "native token" representative address on L2.
    2. `86400`: used in the `teleportBOBA()` and `teleportNativeBOBA()` functions. Solidity supports the syntax of `1 days`, which is more readable.
3. The `ContextUpgradeable` contract, which is inherited by `PausableUpgradeable`, is initialized within the `__Pausable_init()` function. As a result, invoking this function from the `initialize(..)` method in the `Teleportation` contract is equivalent to calling the unchained initializers of both `Context` and `Pausable`. In line with best practices for delegating responsibility, it is recommended to call only `__Pausable_init()` instead.
4. In the functions `disburseNativeBOBA()` and `disburseNativeBOBA()`, consider adding a max length check for the list `_disbursements` to avoid running out of gas.

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

- `b68...b90 ./Boba_GasPriceOracle.sol`
- `080...eb4 ./Teleportation.sol`

Tests

- `d6c...140 ./Boba_GasPriceOracle.spec.ts`
- `ea8...c87 ./endToEndTests/teleportation.spec.ts`

Toolset

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

Setup

Tool Setup:

- Slither v0.8.3

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/Teleportation.sol --solc-remaps @=node_modules/@ --filter-paths node_modules --checklist > slither.md` for the `Teleportation.sol` contract.
6. Run the Slither command: `slither packages/contracts/contracts/L2/predeploys/Boba_GasPriceOracle.sol --solc-remaps @=node_modules/@ --filter-paths node_modules --checklist > slither.md` for the `Boba_GasPriceOracle.sol` contract.

Automated Analysis

Slither

Slither analyzed 38 contracts using 78 detectors, resulting in 146 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/contracts` for `Boba_GasPriceOracle`
    - `cd packages/boba/contracts` for `Teleportation`
3. Run the coverage command: `yarn test`

```
Boba_GasPriceOracle
    owner
      ✔ should have an owner
      ✔ should transfer ownership
      ✔ should revert if called by someone other than the owner
      ✔ should revert if ownership is transferred to zero address
    initialize
      ✔ should revert if contract has been initialized
    updatePriceRatio
      ✔ should revert if called by someone other than the owner
      ✔ should revert if number is too small or too large
      ✔ should succeed if called by the owner and is equal to `1234`
      ✔ should emit event
    get priceRatio
      ✔ should change when priceRatio is called
      ✔ is the 5th and 10th storage slot
    maxPriceRatio
      ✔ should revert if called by someone other than the owner
      ✔ should revert if maxPriceRatio is smaller than minPriceRatio
      ✔ should succeed if called by the owner
      ✔ should emit event
    get maxPriceRatio
      ✔ should change when maxPriceRatio is called
      ✔ is the 3rd storage slot
    minPriceRatio
      ✔ should revert if called by someone other than the owner
      ✔ should revert if minPriceRatio is larger than maxPriceRatio
      ✔ should succeed if called by the owner
      ✔ should emit event
    get minPriceRatio
      ✔ should change when minPriceRatio is called
      ✔ is the 4th storage slot
    gasPriceOracleAddress
      ✔ should revert if called by someone other than the owner
      ✔ should revert if the new address is address(0)
      ✔ should succeed if called by the owner
      ✔ should emit event
    get gasPriceOracleAddress
      ✔ should revert if caller is not EOA
      ✔ should change when gasPriceOracleAddress is called
```

```
    ✔ is the 6th storage slot
  metaTransactionFee
    ✔ should revert if called by someone other than the owner
    ✔ should revert if the new transaction fee is 0
    ✔ should succeed if called by the owner
    ✔ should emit event
  get metaTransactionFee
    ✔ should revert if caller is not EOA
    ✔ should change when updateMetaTransactionFee is called
    ✔ is the 8th storage slot
  receivedETHAmount
    ✔ should revert if called by someone other than the owner
    ✔ should revert if the new receivedETHAmount is below 0.001 ETH
    ✔ should revert if the new receivedETHAmount is larger than 0.01 ETH
    ✔ should succeed if called by the owner
    ✔ should emit event
  get receivedETHAmount
    ✔ should revert if caller is not owner
    ✔ should change when updateReceivedETHAmount is called
    ✔ is the 9th storage slot
  withdrawBOBA
    ✔ should revert if the balance is not enough
  withdrawETH
    ✔ should revert if called by someone other than the owner
    receive ETH
      ✔ should receive ETH
    getBOBAForSwap
      ✔ should get correct BOBA for swapping BOBA for ETH
    getL1BobaFee
      ✔ case: 0x
      ✔ case: 0x00
      ✔ case: 0x01
      ✔ case: 0x0001
      ✔ case: 0x0101
      ✔ case: 0xffff
      ✔ case: 0x00ff00ff00ff00ff00ff00ff


  56 passing (5s)

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 (41ms)
      ✔ should not teleport BOBA tokens if the amount exceeds the daily limit (42ms)
      ✔ should reset the transferred amount (47ms)
      ✔ should revert if call teleportNativeBOBA function
      ✔ should revert if _toChainId is not supported
      ✔ should disburse BOBA tokens
      ✔ should disburse BOBA tokens and emit events
      ✔ 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
      ✔ 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
      ✔ should not withdraw BOBA balance if caller is not owner
      ✔ should pause contract
      ✔ should unpause contract (47ms)
    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
    ✔ should not teleport BOBA tokens if the amount exceeds the daily limit
    ✔ should reset the transferred amount
    ✔ should revert if call teleportBOBA function
    ✔ should revert if _toChainId is not supported
    ✔ should disburse BOBA tokens
    ✔ 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
    ✔ 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
    ✔ should unpause contract
  Admin tests
    ✔ should transferOwnership
    ✔ should not transferOwnership if caller is not owner
    ✔ should set minimum amount
    ✔ should not set minimum 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


  55 passing (3s)
```

Code Coverage

The coverage for `Boba_GasPriceOracle.sol` was gathered by running:

1. `yarn && yarn build`
2. `cd packages/contracts`
3. `npx hardhat coverage --testfiles "test/contracts/L2/predeploys/Boba_GasPriceOracle.spec.ts"`

The coverage for `Teleportation.sol` was gathered by running:

1. `yarn && yarn build`
2. `cd packages/boba/contracts`
3. `npx hardhat coverage --testfiles "test/endToEndTests/teleportation.spec.ts"`

The tests for `Teleportation.sol` show high coverage. The tests for `Boba_GasPriceOracle.sol` could be improved.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
Boba_GasPriceOracle.sol	64.62	45.45	77.78	65.67	... 307,308,309
Teleportation.sol	99.07	75.81	100	100	

Changelog

- 2023-04-11 - Initial report
- 2023-05-09 - Fix Review

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

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.