SMART CONTRACT AUDIT

zokyo

August 7th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed security qualifications.

TECHNICAL SUMMARY

This document outlines the overall security of the Boba smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Boba smart contracts codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the audit. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contracts but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Boba team put in place a bug bounty program to encourage further active analysis of the smart contracts.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Boba repository:
https://github.com/bobanetwork/boba/tree/develop/packages/boba/account-abstraction

Last commit - 4f1eb11ed90a4b0f92985cc28d9d2386102d31b1

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- EntryPointWrapper.sol
- BaseAccount.sol
- EntryPoint.sol
- NonceManager.sol
- StakeManager.sol
- BasePaymaster.sol
- Helpers.sol
- SenderCreator.sol
- IAccount.sol
- IEntryPoint.sol
- IPaymaster.sol
- UserOperation.sol
- IAggregator.sol
- INonceManager.sol
- IStakeManager.sol
- BobaDepositPaymaster.sol
- IBobaGasPriceOracle.sol
- VerifyingPaymaster.sol
- BobaVerifyingPaymaster.sol
- IBobaStraw.sol
- SimpleAccount.sol bls
- DepositPaymaster.sol
- Oracle.sol
- SimpleAccountFactory.sol callback
- GPODepositPaymaster.sol
- ManualDepositPaymaster.sol
- TokenPaymaster.sol
- BLSAccount.sol
- BLSHelper.sol
- IBLSAccount.sol
- BLSAccountFactory.sol
- BLSSignatureAggregator.sol lib
- BLSOpen.sol:
- BLS.sol
- BNPairingPrecompileCostEstimator.sol
- ModExp.sol
- TokenCallbackHandler.sol
- EIP4337Fallback.sol
- EIP4337Manager.sol
- GnosisAccountFactory.sol
- Exec.sol
- Teleporation (b9c46f0655428d f141ef4220d8865c44bc15f673)

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Boba smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Thorough manual review of the codebase line by line.

Executive Summary

During the review, no critical issues were identified. However, we did uncover some medium and low severity issues, as well as a few informational issues. For a detailed description of these findings, please refer to the "Complete Analysis" section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Boba team and the Boba team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Using oracle method that could return `0`	Medium	Resolved
2	Method could return `0` based on oracles returned decimals feeds	Low	Resolved
3	Floating Pragma	Informational	Acknowledged
4	Outdated Dependencies	Informational	Resolved
5	`getUserOpHashes()` Can Cause Unexpected Behavior	Informational	Acknowledged
6	Lack of NatSpec in `EntryPointWrapper.sol`	Informational	Resolved
7	Bridge Cannot Handle ERC-20 Fee on Transfer	Informational	Resolved
8	SafeMath Not Needed	Informational	Resolved
9	`sourceChainId` Can Be Converted to Save One Slot	Informational	Resolved

MEDIUM | RESOLVED

Using oracle method that could return 0

In contract BobaDepositPaymaster, function “getTokenValueOfEth” is translating a given value amount of ethereum into the underlying token amount, for that the business logic is using oracles to query for values and different token decimals, oracles will be chosen by the admin of the contract however we expect that most preferred oracles will be BobaStraw, as BobaStraw is a fork of chainlink as it is saying in it’s documentation, we noticed that the used function to query the oracles is the function “latestAnswer” which based on it’s implementation does not revert if the price returned it’s staled or if it can not retrieve the price it will simply return 0 which will break all the logic of the application resulting in a token cost of 0 and and free transactions.

Recommendation:

Use function “latestRoundData" instead of “latestAnswer” and ensure you are receiving fresh data from it by sanity checking updatedAt and answeredInRound returned values.

LOW | RESOLVED

Method could return 0 based on oracles returned decimals feeds

In contract BobaDepositPaymaster, function “getTokenValueOfEth” is translating a given value amount of ethereum into the underlying token amount, for that the business logic is using oracles to query for values and different token decimals, oracles will be chosen by the admin of the contract, based on the function comments there is no check over the variable requiredAmount to ensure it’s not 0 as priceRatio shouldn’t normally exceed ethBought value, that statement is not always true and it is dependent of the values returned by the oracles, as the oracles are chosen by the owner, there is the possibility that one oracles will returned a price feed on 8 decimals and another one on 18 decimals and so on, and that will ruin the assumption that requiredAmount can not be 0.

Recommendation:

Add a sanity check to ensure that the function can not return the 0 value at the end.

INFORMATIONAL | ACKNOWLEDGED

Floating Pragma

Throughout the codebase, the contracts that are unlocked at version ^0.8.12, and they should always be deployed with the same compiler version. By locking the pragma to a specific version, contracts are not accidentally getting deployed by using an outdated version that can introduce unintended consequences.

Recommendation:
Lock the compiler version to a specific. Known bugs are featured here.

Comment: The client acknowledges the finding, but did not make any changes.

INFORMATIONAL | RESOLVED

Outdated Dependencies

The codebase is using an outdated Open Zeppelin version "@openzeppelin/contracts": "^4.2.0". Below are issues tied to outdated OZ versions. This lists errors such as DOS, improper initialization, etc.

INFORMATIONAL | ACKNOWLEDGED

`getUserOpHashes()` Can Cause Unexpected Behavior

In the contract, 'EntryPointWrapper.sol`, `getUserOpHashes()` on line 196 may cause unexpected behavior. If the wrong `entryPoint` address is input to the function parameters, then it will revert. There could be a check to ensure the right EntryPoint is used or removed from the parameters. Even if it is implied for one contract per chain, this might lower risk of accidental error.

Recommendation:

1. Add a simple check to ensure the right entrypoint is used to ensure all hashes are the same. As the return value will hash something else with the wrong `Entrypoint` address. If this check is supposed to be off-chain, then proper documentation should be completed for developers to understand the importance.
2. Remove `IEntryPoint entryPoint` from the function parameter as it is already defined inside the constructor.

Comment: The client acknowledges the finding but did not make any changes as there will only be one contract.

INFORMATIONAL | RESOLVED

Lack of NatSpec in `EntryPointWrapper.sol`

In the contract, `EntryPointWrapper.sol`, the codebase lacks documentation that might be useful to developers. For example: documentation for structs such as `FailOpStatus` might be useful for developers without having to guess the intended functionality.

Recommendation:
We recommend that documentation is added throughout `EntryPointWrapper.sol` to ensure external reviewers and developers understand intended meaning.

INFORMATIONAL | RESOLVED

Bridge Cannot Handle ERC-20 Fee on Transfer

In the contract `Teleportation.sol` is used to transfer funds from L1 to L2. The function `teleportAsset()` sends and emits tokens based on the input of `_amount`. If a token had a fee attached on transfer the contract can expect potential unexpected consequences if the fee is not exempt for this contract.

Recommendation:
We recommend checking the balance received by the contract before updating/or emitting a value that is potentially incorrect. This additional logic will allow fee on transfer tokens to be bridged across without users losing funds.

Comment: The client does not expect this to be an issue as they will not support fee on transfer tokens.

INFORMATIONAL | RESOLVED

SafeMath Not Needed

In the contract, `Teleportation.sol`, the codebase uses SafeMath. This causes unnecessary overflow and underflow checks that are reverted by default. By removing this library and using built in arithmetic, users can save gas for not unnecessary checks.

Recommendation:
We recommend removing SafeMath as the compiler version is above `0.8.0`.

`sourceChainId` Can Be Converted to Save One Slot

In the contract, `Teleportation.sol`, the variable `sourceChainId` can be converted to save gas. `sourceChainId` may benefit from being reduced from uint256 to smaller size such as uint32. By changing `sourceChainId`, the slot size will be lowered by one.

Recommendation:
If converting `sourceChainId` to uint32 is within the protocol specification, then we recommend making this change.

	EntryPointWrapper.sol **BaseAccount.sol** **EntryPoint.sol** **NonceManager.sol** **StakeManager.sol** **BasePaymaster.sol** **Helpers.sol** **SenderCreator.sol** **IAccount.sol** **IEntryPoint.sol**
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	IPaymaster.sol **UserOperation.sol** **IAggregator.sol** **INonceManager.sol** **IStakeManager.sol** **BobaDepositPaymaster.sol** **IBobaGasPriceOracle.sol** **VerifyingPaymaster.sol** **BobaVerifyingPaymaster.sol** **IBobaStraw.sol**
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	SimpleAccount.sol bls **DepositPaymaster.sol** **Oracle.sol** **SimpleAccountFactory.sol callback** **GPODepositPaymaster.sol** **ManualDepositPaymaster.sol** **TokenPaymaster.sol** **BLSAccount.sol** **BLSHelper.sol** **IBLSAccount.sol**
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	BLSAccountFactory.sol **BLSSignatureAggregator.sol lib** **BLSOpen.sol:** **BLS.sol** **BNPairingPrecompileCostEstimator.sol** **ModExp.sol** **TokenCallbackHandler.sol** **EIP4337Fallback.sol** **EIP4337Manager.sol** **GnosisAccountFactory.sol**
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

	Exec.sol Teleporation
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

We are grateful for the opportunity to work with the Boba team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Boba team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo