bebop

Bebop Security Analysis

by Pessimistic

This report is public

November 15, 2023

Abstract

In this report, we consider the security of smart contracts of Bebop project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of Bebop smart contracts. We described the audit process in the section below.

The initial audit showed several issues of medium severity: Function vulnerable to double-entry tokens, Insufficient documentation, Possible DoS of batch settlement and Not measured code coverage. Also, several low-severity issues were found.

The overall quality of the code is good. While there are a few issues of medium and low severity, it is worth mentioning that the protocol relies significantly on signed data. Therefore, it is crucial to ensure that the contents of `JamOrder.Data` are thoroughly validated by the signer to maintain the security and integrity of the protocol. In addition, the project has insufficient documentation.

After the initial audit, the codebase was updated. The developers partially fixed medium severity issue Possible DoS of batch settlement and marked other medium severity issues as acknowledged. Also, several low severity issues were resolved.

Developers marked many problems as acknowledged, but we still consider them parts requiring additional attention. It is important to consider that these unaddressed issues may impact various project parts. Additionally, enhanced documentation and comprehensive code coverage measurement could potentially aid in identifying previously unknown issues before they become problematic.

General recommendations

We recommend fixing the mentioned issues, improving NatSpec coverage, and supplementing the documentation. We also recommend implementing CI to calculate code coverage and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with Bebop project on a private GitHub repository, commit 6a038f1e189cbf5faf4e0307c42eef8f71819a60.

The scope of the audit included:

- **JamSettlement.sol**;
- **JamBalanceManager.sol**;
- base directory;
- libraries directory;
- interfaces directory.

The documentation for the project included `README.md` and the following links: project description and API.

All 49 tests pass successfully. The code coverage is not measured because of warnings.

The total LOC of audited sources is 881.

Codebase update #1

After the initial audit, the codebase was updated. For the recheck, we were provided with commit f3c5fda7250588c80920aeca35085979ada34963.

The scope of the audit has not changed.

This update included fixes for one issue of medium severity, several low severity issues, and one note issue. All 49 tests passed. The coverage issue was not resolved.

Audit process

We started the audit on October 20 and finished on October 31, 2023.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among others, we verified the following properties of the contracts:

- Whether arbitrary calls are safe;
- It is impossible to use approvals and steal someone else's tokens;
- Whether signatures are generated and verified correctly;
- Whether the batch of orders is stable and cannot be reverted;
- Whether different types of tokens are handled correctly;
- etc.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts. Also, we sent the results to the developers in the text file.

We ran tests and tried to calculate the code coverage, but it was not measured due to warnings.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

After the initial audit, we received a new commit with the updated codebase.

We checked if the issues from the initial audit were fixed. We also inspected whether the logic of the updated functionality, which was responsible for managing uncancellable nonces, was implemented correctly.

During the recheck, we mentioned additional ways of batch orders filling denial at M03, removed one false positive issue from L10, and introduced two new low severity issues: L14 and L15.

After the recheck, we have updated the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Function vulnerable to double-entry tokens (commented)

The `calculateNewAmounts` and `hasDuplicate` functions of the **JamTransfer** contract compare addresses of different tokens to identify duplicates at lines 89 and 133. However, this comparison method does not account for double-entry tokens, which are tokens with different addresses but share the same storage.

It can result in incorrect duplicate checks and bonus amount calculations, as the code may not recognize these double-entry tokens as one entity. For instance, if tokens were sent through the first address, sending tokens to the second address may revert because there is not enough balance available.

> *Comment from the developers:* *Acknowledged status.*

M02. Insufficient documentation (commented)

The documentation gives only a brief description of the functionality without main technical details. Code is covered with NatSpecs, but they do not explain the overall project structure. The documentation is required to streamline both development and audit processes. It should explicitly explain the purpose and behavior of the contracts, their interactions, and main design choices. The main concerns are in the project:

- In some cases, the NatSpec comments do not provide complete descriptions of the function's parameters.
- The calculation of additional amounts in the `calculateNewAmounts` function of the **JamTransfer** contract appears to vary for different cases, and there is a lack of additional explanation in the documentation.
- Lack of explanation for the transfer of native tokens for sale inside settling functions.

> *Comment from the developers:* *Acknowledged status.*

M03. **Possible DoS of batch settlement** (commented)

There are several ways to prevent the execution of a batch settlement. A malicious actor can request the execution of an order, which is then included in the batch settlement. Upon observing the batch settlement in the mempool, this actor can front-run the solver's transaction by invoking a function that will lead to the failure of the whole batch execution. For example:

- **(fixed)** `JamSigning.cancelOrder`. This action has the potential to impact the entire batch, resulting in the reverting of other orders at the line 180 in the `invalidateOrderNonce` function of the **JamSigning** contract. The path is the following: `JamSettlement.settleBatch` -> `JamSigning.validateBatchOrders` -> `JamSigning.validateOrder` -> `JamSigning.invalidateOrderNonce`;
- Removing approved amount for the `ERC20` token included in the order;
- Invalidating permit signature for the token included in the order.

Issues have been partially fixed. Developers introduced uncancellable orders, preventing cancelation of them using `JamSigning.cancelOrder`. Other DoS mechanics were not introduced in the initial report. Developers marked them as acknowledged.

M04. **Not measured code coverage** (commented)

The code coverage is not measured, failing with `Stack too deep` with/without `ir`. We always note the availability of tests as well as code coverage and whether it is sufficient.

> *Comment from the developers: Acknowledged status.*

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Change memory to calldata (fixed)

The location of the `permitSignature` parameter in the `permitToken` function can be changed to `calldata` at line 183 of the **JamBalanceManager** contract.

The issue has been fixed and is not present in the latest version of the code.

L02. Constants

Consider declaring `10000` literals as constants all over the **BMath** library. It can improve code readability, help avoid "magic numbers" and promote reusability.

The issue has been partially fixed.

L03. Dependency management (commented)

The used version of OpenZeppelin is not specified.

> *Comment from the developers: Acknowledged status.*

L04. EIP712 type hash lacks transparency (commented)

In the **JamSigning** contract, the hash of hooks is signed (lines 63–96) within the `JamOrder` type hash (lines 25–27). However, it is crucial for users to be able to understand and evaluate the hooks during the signing process.

In the current implementation, only the hash of hooks is considered, and it may not be easily validated by users. It may be worth considering the use of a complex type hash with nested structures for hooks using the `Def` structure of the **JamHooks** library. This would allow for a more transparent and user-friendly way to validate hooks during the signing process.

> *Comment from the developers: Acknowledged status.*

L05. Redundant checks (commented)

When the condition `if order.receiver == address(this)` in the `JamSettlement._settle` is met, redundant checks are performed at lines 180-186 to protect the order creator from possible loss of funds. However, it is important to note that the data of the order, including hooks, should be signed by the order taker, and they validate it.

The gas consumption of a call directly impacts the profits of the solver. Consider avoiding conducting extensive checks on-chain, as these checks can increase gas costs and reduce the overall profitability of the solver.

Additionally, the `hasDuplicate` check is susceptible to double-entry tokens.

> *Comment from the developers:* *Acknowledged status.*

L06. Hardcoded chainId (commented)

The `chainId` of the Polygon network can be altered at a later time, potentially disrupting the integration of `DAI` on the Polygon network in the `permitToken` function of the **JamBalanceManager** contract. It is important to note that the contract is not upgradeable and as a result, there will be no option to modify it in the future.

> *Comment from the developers:* *Acknowledged status.*

L07. Incorrect nonce validation (fixed)

The `isNonceValid` function of **JamSigning** contract currently compares the entire slot of the mapping to validate if a nonce is correct. However, for proper validation, it should check a specific bit within the obtained slot.

The issue has been fixed and is not present in the latest version of the code.

L08. Memory-safe (commented)

Consider making the `assembly` block in the `validateSignature` function of the **JamSigning** contract and the `getRsv` function of the **Signature** contract `memory-safe`. This ensures safe handling of memory, reducing the risk of vulnerabilities related to incorrect memory access or manipulation.

> *Comment from the developers:* *Acknowledged status.*

L09. No validation of consumed NFTs (fixed)

In the **JamBalanceManager** contract, there is a check at line 85 that validates that all NFT IDs were consumed. Consider adding a similar check when transferring tokens through the `JamTransfer.transferTokensFromContract` function to ensure that all necessary conditions are met and that the transfer is performed correctly.

The issue has been fixed and is not present in the latest version of the code.

L10. Pure and view functions (fixed)

Consider changing the mutability of the following functions as long as they do not change or read the storage:

- `getPercentage` and `getInvertedPercentage` of the **BMath** library to pure;
- `hashHooks` to pure in the **JamSigning** contract.

The issue has been fixed and is not present in the latest version of the code. Also, the part of the issue has been removed as a false positive.

L11. Reduce memory access as gas optimization (commented)

In the `transferTokens` and `transferTokensWithPermits` functions of the **JamBalanceManager** contract, the `batchTransferDetails` is allocated and updated to perform batch transfer later. The memory is preallocated, and its size is reduced at lines 82 and 150 on every iteration when the transfer type is not `permit2` transfer. However, this code can be optimized by setting the new size only once after the for-loop, with the value of the `batchLen` variable.

> *Comment from the developers:* *Acknowledged status.*

L12. Unused imports (fixed)

In the **JamSettlement** contract, it appears that there are several files (**IWETH** and **BMath** contracts) imported at lines 9 and 14. They are not utilized or referenced anywhere else in the code. This could potentially be unnecessary and can be removed to improve code cleanliness and efficiency.

The issues have been fixed and are not present in the latest version of the code.

L13. Unused type (commented)

The `Type` enum value `NONE` is not utilized in the **Signature** contract.

> *Comment from the developers:* *Acknowledged status.*

L14. Ambiguous settlement events (commented)

`Settlement` events emitted at lines 165, 187, and 203 contain information only about the nonce, and it can emit the same event for two different takers. Consider adding `order.taker` as a parameter for the event.

> *Comment from the developers:* *Acknowledged.*

L15. **Pure and view functions - part 2** (commented)

Consider changing the mutability of the following functions as long as they do not change or read the storage:

- `validateIncreasedAmounts` to view in the **JamSigning** contract;
- `calculateNewAmounts` of the **JamTransfer** contract to pure.

> *Comment from the developers:* *Acknowledged status.*

Notes

N01. **Infinite approval** (commented)

JamBalanceManager stores all approvals. Also, it has the `permitToken` function that gives infinite approvals from an order's taker to that contract. This poses a risk as any vulnerability in the contract could potentially enable an attacker to utilize these approvals and acquire users' tokens.

> *Comment from the developers:* *Acknowledged status.*

N02. **Unintuitive dividing excess** (fixed)

If we enter the condition `if (fullAmount == curOrder.buyAmounts[i] && tokenBalance >= curOrder.buyAmounts[i])` of the `JamTransfer.calculateNewAmounts` function, then the `newAmount` value will be equal to the contract balance in that specific token. However, in `JamSettlement.settleBatch`, the receiver will only receive a percentage of that balance. According to the logic, they should receive the full amount since it is the excess that is being distributed among all orders.

In the case of the `settleBatch` function, if there are two identical tokens, the first receiver of the token will receive the amount without multiplication by percentage, and the second receiver of the same token will get the amount multiplied by the percentage (because we will get into the `if` branch that does not modify `newAmount` with `invertedPercent` function).

The issue has been fixed and is not present in the latest version of the code.

This analysis was performed by Pessimistic:

Daria Korepanova, Senior Security Engineer
Oleg Bobrov, Junior Security Engineer
Irina Vikhareva, Project Manager
Konstantin Zherebtsov, Business Development Lead

November 15, 2023