Smart Contract Security Audit Report

Bebop

1. Contents

2. General Information

This report contains information about the results of the security audit of the Bebop (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 11/09/2023 to 11/23/2023.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: https://github.com/bebop-dex/bebop-jam-contracts. Initial review was done for the commit f3c5fda7250588c80920aeca35085979ada34963 and the re-testing was done for the commit 70c30886b773eb84e55216014fc6f3073999e66f.

The following contracts have been tested:

- src/JamSettlement.sol
- src/JamBalanceManager.sol
- src/base/*.sol (all files)
- src/libraries/*.sol (all files)

2.3. Threat Model

The assessment presumes the actions of an intruder who might have the capabilities of any role (an external user, token owner, token service owner, or a contract). The risks of centralization were not taken into account at the Customer's request.

The main possible threat actors are:

- Taker (EOA and contracts),
- Solver (EOA and contracts),
- Protocol owner,
- 3rd parties (tokens).

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, and the impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises the best effort to perform its contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using limited resources.

3. Summary

As a result of this work, we have discovered a few potential security issues, which has been fixed and re-tested in the course of the work.

The other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement).

The Bebop team has given feedback for the suggested changes and an explanation for the underlying code.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of November 28, 2023.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Taker can lose ERC1155/ERC721/Native tokens	/src/JamBalanceManager.sol	**Medium**	Fixed
Standard orders cannot be cancelled	/src/base/JamSigning.sol	**Low**	Acknowledged
Solvers cannot distinguish legitimate calls from `JamSettlement`	/src/JamSettlement.sol /libraries/JamInteraction.sol	**Low**	Acknowledged
String error messages are used instead of custom errors	JamBalanceManager.sol, JamSettlement.sol, JamSigning.sol, JamTransfer.sol, Signature.sol	**Info**	Acknowledged
Percentage value hardcoded in `settleBatch`	src/JamSettlement.sol	**Info**	Fixed
Signature malleability	src/libraries/Signature.sol:	**Info**	Acknowledged

Issue	Contract	Risk Level	Status
Approval amounts not controllable by takers	/src/JamBalanceManager.sol	Info	Acknowledged

4. General Recommendations

This section contains general recommendations on how to improve the overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Taker can lose ERC1155/ERC721/Native tokens

Risk Level: Medium

Status: Fixed in the commit 70c30886.

Contracts:

- /src/JamBalanceManager.sol

Location: Lines: 75,77,141,143. Function: `transferTokens, transferTokensWithPermits`.

Description:

Takers are allowed to set `minFillPercent` for their orders. However, in case a taker wants to sell ERC721/ERC1155/Native tokens they may partly lose the value of their tokens if `minFillPercent != HUNDRED_PERCENT`. This happens because in `transferTokens` and `transferTokensWithPermits` functions transfers for ERC721/ERC1155 are always performed for the full amount on lines 75, 77, 141, and 143. The solver will be able to fill e.g., only 70% of the buyTokens amount and get ERC721 back.

In the case of native tokens, takers have to send the full amount in `hooks.beforeSettle`, because they don't know `curFillPercent` if `minFillPercent != HUNDRED_PERCENT`. If they don't perform necessary checks in `hooks.afterSettle`, the solver will be able to sweep their native tokens from the contract via calling `transferNativeFromContract`

Remediation:

In case ERC721/ERC1155/Native tokens are traded we would recommend checking that `minFillPercent == HUNDRED_PERCENT` so takers will not lose their tokens by mistake.

5.2. Standard orders cannot be cancelled

Risk Level: Low

Status: Acknowledged

Contracts:

- /src/base/JamSigning.sol

Description:

JamSigning contract implements canceling functionality only for limited orders. In case a user has created an incorrect standard order, they won't be able to cancel it, since it's only invalidated by expiration.

Remediation:

Consider implementing functionality for manual standard order canceling.

5.3. Solvers cannot distinguish legitimate calls from `JamSettlement`

Risk Level: Low

Status: Acknowledged

Contracts:

- /src/JamSettlement.sol
- /libraries/JamInteraction.sol

Description:

There's no straightforward way for a solver to distinguish whether a call came from a solver-initiated interaction or from a taker-initiated hook.

This could potentially lead to security vulnerabilities in the solver contracts, such as the sample `JamSolver.sol` where any call coming from `JamSettlement` within a settlement transaction can sweep any tokens from the contract.

While this is the solvers' responsibility to create transactional contracts that don't hold value or don't have such functions that allow sweeping them, such design makes it more likely for these attack vectors to exist.

Remediation:

One of the possible solutions is to create a deny-listing option for the interactions in `JamSettlement`: let solvers prohibit more addresses besides `JamBalanceManager` from calling but also do pass this option only for the hooks.

5.4. String error messages are used instead of custom errors

Risk Level: Info

Status: Acknowledged

Contracts:

- JamBalanceManager.sol,
- JamSettlement.sol,
- JamSigning.sol,
- JamTransfer.sol,
- Signature.sol

Description:

The contracts make use of the `require()` to emit an error. While this is a perfectly valid way to handle errors in Solidity, it is not always the most efficient.

```
JamBalanceManager.sol:
    39: require(account == operator, "INVALID_CALLER");
    67: require(data.tokens[i] == JamOrder.NATIVE_TOKEN,
"INVALID_NATIVE_TOKEN_ADDRESS");
    74: require(data.amounts[i] == 1, "INVALID_ERC721_AMOUNT");
    79: revert("INVALID_TRANSFER_TYPE");
    85: require(nftsInd == data.nftIds.length, "INVALID_NFT_IDS_LENGTH");
    86: require(batchLen == batchTransferDetails.length,
"INVALID_BATCH_PERMIT2_LENGTH");
    133: require(data.tokens[i] == JamOrder.NATIVE_TOKEN,
"INVALID_NATIVE_TOKEN_ADDRESS");
    140: require(data.amounts[i] == 1, "INVALID_ERC721_AMOUNT");
    145: revert("INVALID_TRANSFER_TYPE");
    153: require(indices.batchToApproveInd == batchToApprove.length,
"INVALID_NUMBER_OF_TOKENS_TO_APPROVE");
    154: require(indices.batchLen == batchTransferDetails.length,
"INVALID_BATCH_PERMIT2_LENGTH");
    155: require(indices.permitSignaturesInd ==
takerPermitsInfo.permitSignatures.length,
"INVALID_NUMBER_OF_PERMIT_SIGNATURES");
    156: require(indices.nftsInd == data.nftIds.length,
"INVALID_NFT_IDS_LENGTH");
JamSettlement.sol:
    54: require(runInteractions(hooks.beforeSettle),
"BEFORE_SETTLE_HOOKS_FAILED");
    74: require(runInteractions(hooks.beforeSettle),
"BEFORE_SETTLE_HOOKS_FAILED");
```

```
    92: require(runInteractions(hooks.beforeSettle),
"BEFORE_SETTLE_HOOKS_FAILED");
    111: require(runInteractions(hooks.beforeSettle),
"BEFORE_SETTLE_HOOKS_FAILED");
    136: require(runInteractions(hooks[i].beforeSettle),
"BEFORE_SETTLE_HOOKS_FAILED");
    154: require(runInteractions(interactions), "INTERACTIONS_FAILED");
    163: require(runInteractions(hooks[i].afterSettle),
"AFTER_SETTLE_HOOKS_FAILED");
    175: require(runInteractions(interactions), "INTERACTIONS_FAILED");
    180: require(!hasDuplicate(order.buyTokens, order.buyNFTIds,
order.buyTokenTransfers), "DUPLICATE_TOKENS");
    181: require(hooks.afterSettle.length > 0, "AFTER_SETTLE_HOOKS_REQUIRED");
    183: require(hooks.afterSettle[i].result, "POTENTIAL_TOKENS_LOSS");
    186: require(runInteractions(hooks.afterSettle),
"AFTER_SETTLE_HOOKS_FAILED");
    202: require(runInteractions(hooks.afterSettle),
"AFTER_SETTLE_HOOKS_FAILED");
base/JamSigning.sol:
    109: require(signer != address(0), "Invalid signer");
    111: revert("Invalid EIP712 order signature");
    114: require(IERC1271(validationAddress).isValidSignature(hash,
signature.signatureBytes) == EIP1271_MAGICVALUE, "Invalid EIP1271 order
signature");
    127: require(signer != address(0), "Invalid signer");
    129: revert("Invalid ETHSIGH order signature");
    132: revert("Invalid Signature Type");
    150: require(order.executor == msg.sender || order.executor == address(0),
"INVALID_EXECUTOR");
    151: require(order.buyTokens.length == order.buyAmounts.length,
"INVALID_BUY_TOKENS_LENGTH");
    152: require(order.buyTokens.length == order.buyTokenTransfers.length,
"INVALID_BUY_TRANSFERS_LENGTH");
    153: require(order.sellTokens.length == order.sellAmounts.length,
"INVALID_SELL_TOKENS_LENGTH");
    154: require(order.sellTokens.length == order.sellTokenTransfers.length,
"INVALID_SELL_TRANSFERS_LENGTH");
    155: require(curFillPercent >= order.minFillPercent,
"INVALID_FILL_PERCENT");
    157: require(block.timestamp < order.expiry, "ORDER_EXPIRED");
    181: require(nonce != 0, "ZERO_NONCE");
    186: require(invalidator & invalidatorBit != invalidatorBit,
"INVALID_NONCE");
    200: require(increasedAmounts.length == initialAmounts.length,
"INVALID_INCREASED_AMOUNTS_LENGTH");
    202: require(increasedAmounts[i] >= initialAmounts[i],
"INVALID_INCREASED_AMOUNTS");
    219: require(orders.length == signatures.length,
"INVALID_SIGNATURES_LENGTH");
```

```
    220: require(orders.length == takersPermitsUsage.length ||
allTakersWithoutPermits, "INVALID_TAKERS_PERMITS_USAGE_LENGTH");
    221: require(orders.length == hooks.length || noHooks,
"INVALID_HOOKS_LENGTH");
    222: require(orders.length == curFillPercents.length || isMaxFill,
"INVALID_FILL_PERCENTS_LENGTH");
    225: require(orders[i].receiver != address(this),
"INVALID_RECEIVER_FOR_BATCH_SETTLE");
    234: require(takersPermitsInfo.length == takersWithPermits,
"INVALID_TAKERS_PERMITS_LENGTH");
base/JamTransfer.sol:
    38: require(tokenBalance >= partialFillAmount,
"INVALID_OUTPUT_TOKEN_BALANCE");
    41: require(tokens[i] == JamOrder.NATIVE_TOKEN, "INVALID_NATIVE_TOKEN");
    44: require(tokenBalance >= partialFillAmount,
"INVALID_OUTPUT_NATIVE_BALANCE");
    46: require(sent, "FAILED_TO_SEND_ETH");
    49: require(amounts[i] == 1 && tokenBalance >= 1,
"INVALID_OUTPUT_ERC721_AMOUNT");
    53: require(tokenBalance >= amounts[i], "INVALID_OUTPUT_ERC1155_BALANCE");
    58: revert("INVALID_TRANSFER_TYPE");
    61: require(nftInd == nftIds.length, "INVALID_BUY_NFT_IDS_LENGTH");
    69: require(sent, "FAILED_TO_SEND_ETH");
    92: require(fillPercents.length == 0 || curFillPercent == fillPercents[j],
"DIFF_FILL_PERCENT_FOR_SAME_TOKEN");
libraries/Signature.sol
    26: require(sig.length == 65, "Invalid signature length");
    36: require(uint256(s) <=
0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0, "Invalid
sig value S");
    37: require(v == 27 || v == 28, "Invalid sig value V");
```

Remediation:

Consider using custom errors as they are more gas-efficient while allowing developers to describe the error in detail using NatSpec.

References:

- https://blog.soliditylang.org/2021/04/21/custom-errors/

5.5. Percentage value hardcoded in `settleBatch`

Risk Level: Info

Status: Fixed in the commit 6ac048d4.

Contracts:

- src/JamSettlement.sol

Location: Lines: 142. Function: `settleBatch`.

Description:

The `JamSettlement` contract uses a hardcoded value of `10000` for calculating the fill size, rather than using the `BMath.HUNDRED_PERCENT` variable specifically designed for this purpose. This inconsistency not only complicates the code, but could also lead to inaccurate calculations if the variable `BMath.HUNDRED_PERCENT` changes in future updates and the hardcoded value remains unchanged.

Remediation:

Replace hardcoded value with `BMath.HUNDRED_PERCENT` variable.

5.6. Signature malleability

Risk Level: Info

Status: Acknowledged

Contracts:

- src/libraries/Signature.sol:

Location: Lines: 35. Function: `getRsv`.

Description:

The code snippet provided shows that the value of 'v' in the signature is incremented by 27 if it's less than 27. This can lead to signature malleability, where a valid signature can have multiple valid representations, potentially leading to security vulnerabilities.

```
src/libraries/Signature.sol:
  35:         if (v < 27) v += 27;
```

Remediation:

To mitigate this, it's recommended to enforce a strict check on the 'v' value in the signature. This means that only values of 27 or 28 should be accepted, preventing the signature from having multiple valid representations.

5.7. Approval amounts not controllable by takers

Risk Level: Info

Status: Acknowledged

Contracts:

- /src/JamBalanceManager.sol

Location: Lines: `115,198`. Function: `transferTokensWithPermits, permitToken`.

Description:

Approve amounts in `permit` / `permit2` calls are not controllable by the user and are set as `type(uint160).max` for `permit2` at line 115 and `type(uint).max` for a `permit` at line 198. This fact doesn't allow users to control their approvals to the `JamBalanceManager` contract fully.

Remediation:

Consider allowing users to choose the approval amount.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.