KAWACH

Security Assessment Report

Biconomy

Version: Final

Date: 27 Jun 2023

Table of Contents

License

Disclaimer

This audit report is based on the findings of the audit conducted by our team, which focused on analyzing the Solidity smart contracts of Biconomy's Token Paymaster contracts. While we have taken all reasonable steps to ensure the accuracy and completeness of the report, we cannot guarantee that our findings are free from errors or omissions. The report should be used for informational purposes only and should not be construed as providing legal or investment advice. We have categorized our findings by level of severity, and provided recommendations to address the security issues we identified. It is our hope that this report will help inform decision making for the development team, and ultimately result in an improved level of security for the contract.

It should be noted that the audit was completely focused on the solidity smart contracts. The project includes some off-chain components which we have assumed to work as per the documentation provided to us.

Introduction

Purpose of this report

Kawach has been engaged by **Biconomy** to perform a security audit of its contracts for the Token Paymaster contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behaviour.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebases Submitted for the Audit

The audit has been performed on the following GitHub repositories:

Repository : https://github.com/bcnmy/biconomy-paymasters

Date	Version	Commit hash
22/05/2023	Initial Codebase : V1	2b65188fbdf5bb9d82aec3c723a2c15bb56ac94e
14/06/2023	Fixed Issues : V2	22be36832bc8333e93767cb3b156bab924516e58

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service. This does not require a lot of effort in execution.
High	An attacker can successfully execute an attack that clearly results in operational issues for the service. This also includes any value loss of unclaimed funds permanently or temporary.
Medium	The service may be susceptible to an attacker carrying out an unintentional action, which could potentially disrupt its operation. Nonetheless, certain limitations exist that make it difficult for the attack to be successful.
Low	The service may be vulnerable to an attacker executing an unintended action, but the impact of the action is negligible or the likelihood of the attack succeeding is very low and there is no loss of value.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Overview

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The contract under the scope is a paymaster contract designed to sponsor fee payments for the accounts using EIP 4337 Account Abstraction. It supports fee payments in ERC20 tokens, according to an exchange rate either by an oracle or signed by the authorized signatory.

The contract's primary functionality can be summarized as follows:

➔ Validate Signatures and account's balance

When the user operation is passed to the Biconomy's API to be signed, it returns the paymasterAndData which includes the signatures. This signature and data is then verified on _validatePaymasterUserOp function. This function also checks the balance of the account to see if it can pay the required fee amount.

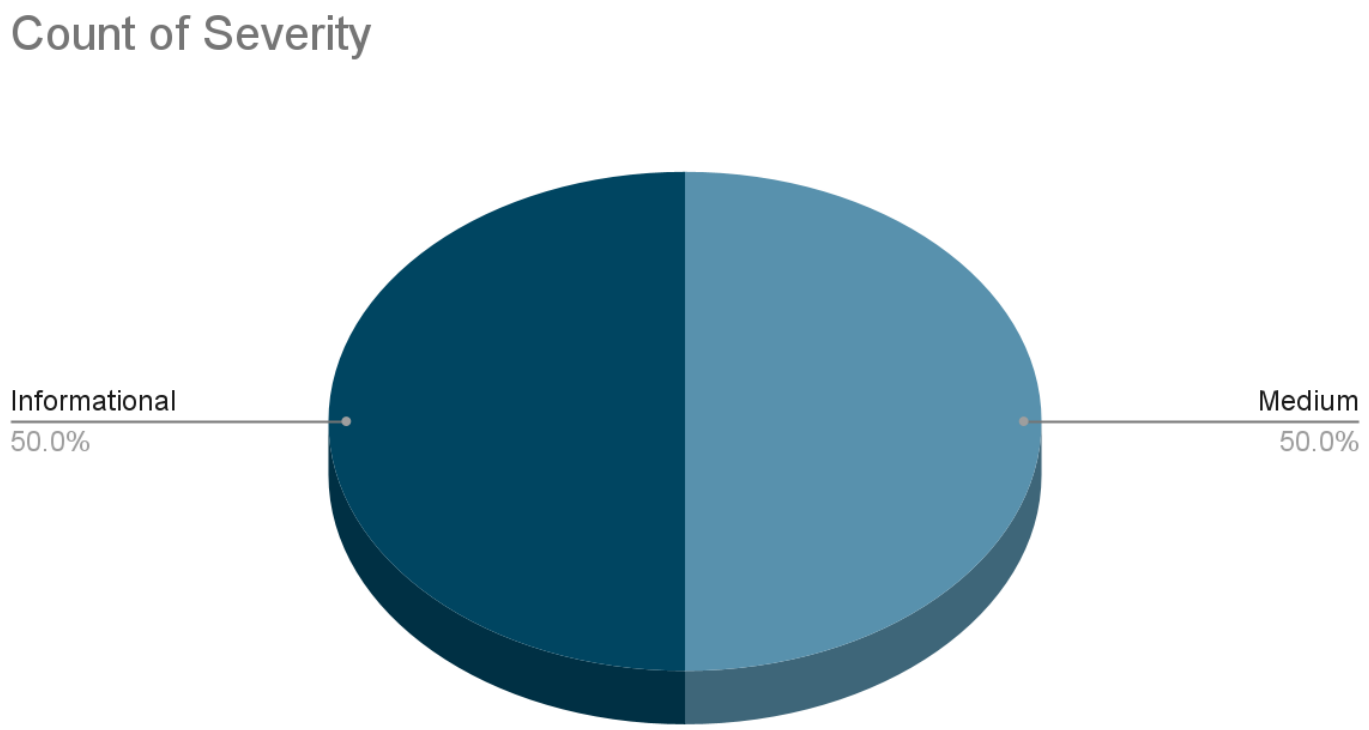
➔ Deduct the fee payment from the accounts

Paymaster deducts the fee amount from the account. It also fetches the ERC20 token to native price, and uses it to calculate the fee amount in ERC20 tokens.

➔ Deposit and Stake in the entry point

There are functions in the contracts to manage the Paymaster's deposits and stakes in the entry point contract.

Summary of Findings



Sr. No.	Description		Severity	Status
1	Centralization Risks	swapTokenForNativeAndDeposit()	Medium	Resolved
		setTokenOracle()		Acknowledged
2	No check for fee tokens that don't revert on transfer		Medium	Acknowledged
3	getTokenPrice() can revert if call to price feed reverts		Informational	Resolved
4	Contract allows single step ownership transfer		Informational	Acknowledged

Detailed Findings

1. Centralization Risks

Severity : **Medium**

Description

Our review of the contracts has revealed a centralization issue, granting the contract owner certain actions that can have affect security of the system.

➔ swapTokenForNativeAndDeposit()

Owner can use this function to make arbitrary calls in the context of the paymaster contract. This can be used to transfer funds from the accounts who had provided allowance to the paymaster contract.

Status : Resolved

The Biconomy team removed the swapTokenForNativeAndDeposit() function from their contract.

➔ setTokenOracle()

This function allows the owner to set the price feed address and calladata to use for fetching the price in Paymaster's `_postOp`. The owner of `ChainlinkOracleAggregator` contract can front-run the bundler's transaction and manipulate the `callAddress` and `calldata` to fetch the wrong price which will charge the account more than he agreed for.

Status : Acknowledged

Recommendation

The owner should not be allowed to make arbitrary calls in the paymaster's context. The call should only be made to whitelisted DEX adapters.

2. No check for fee tokens that don't revert on transfer

Severity : **Medium**

Description

In the `postOp` function, specifically when the mode is set to postOpReverted, the Paymaster makes an attempt to retrieve the fee amount from the account by utilizing the transferFrom function of the feeToken. This action is performed via a low-level call. To verify the success of the transfer, the Paymaster checks whether the boolean value returned by the low-level call is true. However, there exists a vulnerability where the validation process fails to account for fee tokens that do not trigger a revert on transfer.

Consequently, if such a token is utilized for fee payment, an attacker can carry out multiple transactions in which the account retains the balance but lacks the required allowance. These transactions will execute without emitting the `TokenPaymentDue` event, and the token balance of the account will remain unaffected. This can be used to drain the Paymaster's deposit.

BiconomyTokenPaymaster.sol

```
    // In case transferFrom failed in first handlePostOp call, attempt to charge the
tokens again
    bytes memory _data = abi.encodeWithSelector(
        feeToken.transferFrom.selector,
        account,
        feeReceiver,
        charge
    );
    (bool success, bytes memory returndata) = address(feeToken).call(
        _data
    );
    if (!success) {
        // In case above transferFrom failed, pay with deposit / notify at least
        // Sender could be banned indefinitely or for certain period
        emit TokenPaymentDue(address(feeToken), account, charge);
        // Do nothing else here to not revert the whole bundle and harm reputation
    }
```

Recommendation

Implement the same logic as safeTransferFrom and instead of reverting on failure emit the 'TokenPaymentDue' event.

Status

Acknowledged

Comments from Biconomy : 'User's not paying the fees' cannot be avoided in the current design.A backend monitoring of such events and behavior can be implemented to blacklist such users.

3. getTokenPrice() can revert if call to price feed reverts

Severity : **Informational**

Description

If the call to token price feed contract reverts,i.e. `success` is false, the return bytes data `ret` will be empty and abi.decode will revert.

ChainlinkOracleAggregator.sol

```
function _getTokenPrice(
    address token
) internal view returns (uint256 tokenPriceUnadjusted) {
    (bool success, bytes memory ret) = tokensInfo[token]
        .callAddress
        .staticcall(tokensInfo[token].callData);
    if (tokensInfo[token].dataSigned) {
        tokenPriceUnadjusted = uint256(abi.decode(ret, (int256)));
    } else {
        tokenPriceUnadjusted = abi.decode(ret, (uint256));
    }
}
```

Recommendation

Check if the call was successful before decoding the returned data.

Status

Resolved

4. Contract allows single step ownership transfer

Severity: Informational

Description

The contract exhibits a single step ownership change flaw, where the owner can transfer ownership to another address with a single transaction. If an incorrect address is set as the owner, then it may lead to the contract being unusable or funds being locked up in this contract.

The impact of this issue is very low as it does not affect any of the user's funds and the verifying signer can stop providing signatures for this paymaster contract, if the owner loses access to this contract. But the owner might lose any funds in the balance of the contract.

Recommendation

Change ownership process to a two step process.

Status

Acknowledged

Conclusion

In conclusion, the report highlights two medium issues and two informational issues.

Biconomy has either fixed or acknowledged all the mentioned issues with the right remediations.

On account of these results it can be commented that Biconomy's code is high quality and robust.