Zellic

# Token Paymaster

## Smart Contract Security Assessment

**June 2, 2023**

*Prepared for:*

**Ahmed Al-Balaghi**

Biconomy Labs

*Prepared by:*

**Syed Faraz Abrar and Yuhang Wu**

Zellic Inc.

# Contents

# About Zellic

Zellic was founded in 2020 by a team of blockchain specialists with more than a decade of combined industry experience. We are leading experts in smart contracts and Web3 development, cryptography, web security, and reverse engineering. Before Zellic, we founded perfect blue, the top competitive hacking team in the world. Since then, our team has won countless cybersecurity contests and blockchain security events.

Zellic aims to treat clients on a case-by-case basis and to consider their individual, unique concerns and business needs. Our goal is to see the long-term success of our partners rather than simply provide a list of present security issues. Similarly, we strive to adapt to our partners' timelines and to be as available as possible. To keep up with our latest endeavors and research, check out our website zellic.io or follow @zellic_io on Twitter. If you are interested in partnering with Zellic, please contact us at hello@zellic.io.

# 1 Executive Summary

Zellic conducted a security assessment for Biconomy Labs from May 22nd to May 25th, 2022. During this engagement, Zellic reviewed Token Paymaster's code for security vulnerabilities, design issues, and general weaknesses in security posture.

## 1.1 Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Could Paymaster signatures be replayed to steal funds or grief users?
- Could the oracles be exploited to return malformed exchange rates?
- Could an attacker bypass the `UserOperation` validation?

## 1.2 Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

During this assessment, the three-day time frame prevented us from doing a complete review of the test suite for the contracts.

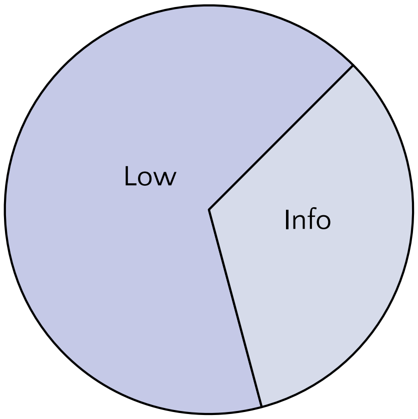
## 1.3 Results

During our assessment on the scoped Token Paymaster contracts, we discovered three findings. No critical issues were found. Of the three findings, two were of low impact, and the remaining finding was informational in nature.

During the remediation phase of the audit, Biconomy Labs implemented a price markup feature in commit 0effa2bd. We conducted a review of this addition and did not identify any security vulnerabilities.

## Breakdown of Finding Impacts

| Impact Level | Count |
|---|---|
| Critical | 0 |
| High | 0 |
| Medium | 0 |
| Low | 2 |
| Informational | 1 |



# 2 Introduction

## 2.1 About Token Paymaster

Token Paymaster is a token-based paymaster that allows user to pay gas fees in ERC20 tokens. Paymasters can sponsor transaction fees for contract accounts. The ERC4337 Entrypoint verifies whether the Paymaster has a sufficient deposit or if the contract account holds enough funds to cover gas fees. During execution, if a Paymaster is involved, it can implement custom fee logic (which in this case is withdrawing ERC20 tokens as gas fees).

## 2.2 Methodology

During a security assessment, Zellic works through standard phases of security auditing including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

**Basic coding mistakes.** Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

**Business logic errors.** Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

**Integration risks.** Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review the contracts' external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

**Code maturity.** We look for potential improvements in the code base in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradeability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

## 2.3 Scope

The engagement involved a review of the following targets:

### Token Paymaster Contracts

| | |
|---|---|
| **Repository** | https://github.com/bcnmy/biconomy-paymasters |
| **Version** | biconomy-paymasters: `2b65188fbdf5bb9d82aec3c723a2c15bb56ac94e` |
| **Programs** | • BiconomyTokenPaymaster<br>• USDCPriceFeedPolygon<br>• ChainlinkOracleAggregator |
| **Type** | Solidity |
| **Platform** | EVM-compatible |

## 2.4 Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of six person-days. The assessment was conducted over the course of three calendar days.

### Contact Information

The following project manager was associated with the engagement:

**Chad McDonald**, Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

**Syed Faraz Abrar**, Engineer
faith@zellic.io

**Yuhang Wu**, Engineer
yuhang@zellic.io

## 2.5 Project Timeline

The key dates of the engagement are detailed below.

| | |
|---|---|
| **May 22, 2023** | Start of primary review period |
| **May 25, 2023** | End of primary review period |
| **May 30, 2023** | Closing call |

# 3 Detailed Findings

## 3.1 Emergency withdraw functions are missing zero address checks

- **Target**: BiconomyTokenPaymaster
- **Category**: Coding Mistakes
- **Severity**: Medium
- **Likelihood**: Low
- **Impact**: Low

### Description

The withdrawERC20(), withdrawERC20Full(), withdrawMultipleERC20(), and withdrawMultipleERC20Full() are emergency withdrawal functions that can be called by the owner to withdraw ERC20 tokens that were mistakenly sent to the Paymaster contract. These tokens are withdrawn to a specified target address.

### Impact

The emergency withdraw functions are missing zero address checks for the target address that the tokens will be withdrawn to. If the owner attempts to withdraw a substantial amount of tokens and accidentally sets target to address(0), the tokens will be lost forever.

### Recommendations

Consider adding in checks to ensure that target is not equal to address(0). This has already been done in the withdrawAllNative() function.

### Remediation

Biconomy Labs implemented a fix for this issue in commit a88357ef2.

## 3.2 Paymaster data is parsed without performing a length check

- **Target**: BiconomyTokenPaymaster
- **Category**: Coding Mistakes
- **Likelihood**: Medium
- **Severity**: Low
- **Impact**: Low

### Description

The `parsePaymasterAndData()` function is used to parse the `UserOperation` structure's `paymasterAndData` field. The `paymasterAndData` field can contain data in any format, with the format being defined by the Paymaster itself.

### Impact

The function does not perform any length checks on the `paymasterAndData` field before attempting to parse it.

```
function parsePaymasterAndData(
    bytes calldata paymasterAndData
)
public pure returns (/* ... */)
{
    // [ ... ]

    (/* ... */) = abi.decode(
        paymasterAndData[VALID_PND_OFFSET:SIGNATURE_OFFSET],
        (uint48, uint48, address, address, uint256, uint256)
    );
    signature = paymasterAndData[SIGNATURE_OFFSET:];
}
```

In the above case, `VALID_PND_OFFSET` is `21`, while `SIGNATURE_OFFSET` is `213`. If the `paymasterAndData` structure does not contain at least that many bytes in it, then the function will revert.

As this field is fully controllable by a user through the `UserOperation` structure, and the parsing is done prior to the signature check in `_validatePaymasterUserOp()`, this would allow a user to trigger reverts, which would cause the Entrypoint contract that's calling into `_validatePaymasterUserOp()` to waste gas.

## Recommendations

Consider adding a check to ensure that the `paymasterAndData` structure has the correct length. If it does not, consider returning an error to allow the Entrypoint contract to ignore this `UserOperation` and continue.

## Remediation

Biconomy Labs implemented a fix for this issue in commit 6787a366.

## 3.3 Function `_getTokenPrice()` could return unexpected value

- **Target**: ChainlinkOracleAggregator.sol
- **Category**: Coding Mistakes
- **Likelihood**: N/A
- **Severity**: Informational
- **Impact**: Informational

### Description

The `_getTokenPrice()` function in the `ChainlinkOracleAggregator` contract performs an external `staticcall` to fetch the price of the specified `token`.

```
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));
        }
    }
```

### Impact

The return value `success` of the `staticcall` is not checked, which leads to the possibility that when `success == false`, the function return value `tokenPriceUnadjusted` could be zero. This could cause the caller function `getTokenValueOfOneNativeToken` to calculate the `exchangeRate` incorrectly, which would ultimately affect the result of `exchangePrice`.

This could potentially lead to unexpected bugs in the future.

### Recommendations

Consider checking the value of `success`, or check the return value at the caller's side.

### Remediation

Biconomy Labs implemented a fix for this issue in commit ca06c2a4.

# 4 Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the smart contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

## 4.1 Module: BiconomyTokenPaymaster.sol

### Function: `parsePaymasterAndData(byte[] paymasterAndData)`

This function is used to parse the `paymasterAndData` field of the `UserOperation` struct.

### Inputs

- `paymasterAndData`
  - **Control**: Fully controlled by user.
  - **Constraints**: It must be data in a valid format, where the data from the `VALID_PND_OFFSET`-1 to the `VALID_PND_OFFSET` should represent the `priceSource`. The data from the `VALID_PND_OFFSET` to the `SIGNATURE_OFFSET` is the ABI-encoded `validUntil`, `validAfter`, `feeToken`, `oracleAggregator`, `exchangeRate`, and `fee`. Data following the `SIGNATURE_OFFSET` position are a valid signature.
  - **Impact**: This is the structural data to be parsed.

### Branches and code coverage (including function calls)

**Intended branches**

- Succeeds with parsing data properly.
  ☑ Test coverage

**Negative behavior**

- Invalid `paymasterAndData` causes revert.
  ☐ Negative test

### Function: `_postOp(PostOpMode mode, bytes calldata context, uint256 actualGasCost)`

This function executes the Paymaster's payment conditions.

### Inputs

- `mode`
  - **Control**: Not controlled by user.
  - **Constraints**: Must be one of these: `opSucceeded`, `opReverted`, or `postOpReverted`.
  - **Impact**: Used to determine the state of the operation.
- `context`
  - **Control**: Not controlled by user.
  - **Constraints**: N/A.
  - **Impact**: This contains the payment conditions signed by the Paymaster.
- `actualGasCost`
  - **Control**: Not controlled by user.
  - **Constraints**: N/A.
  - **Impact**: This is the amount to be paid back to the Entrypoint.

### Branches and code coverage (including function calls)

**Intended branches**

- Succeeds with mode `opSucceeded` or `opReverted`.
  - ☑ Test coverage
- Succeeds with mode `postOpReverted`.
  - ☑ Test coverage
- Oracle aggregator's exchange rate is used.
  - □ Test coverage
- `UserOp`'s exchange rate is used.
  - □ Test coverage

**Negative behavior**

- Failed `transferFrom()` leads to event being emitted.
  - ☑ Negative test

### Function: `_validatePaymasterUserOp(UserOperation calldata userOp, bytes32 userOpHash, uint256 requiredPreFund)`

This function is used to verify that the `UserOperation`'s Paymaster data were signed by the external signer.

### Inputs

- `userOp`
  - **Control**: Fully controlled by user.
  - **Constraints**: All fields are used in signature validation and thus must be valid.
  - **Impact**: This is the `UserOperation` being validated.
- `userOpHash`
  - **Control**: Not controlled by user.
  - **Constraints**: N/A.
  - **Impact**: This is returned as part of the context structure.
- `requiredPreFund`
  - **Control**: Not controlled by user.
  - **Constraints**: N/A.
  - **Impact**: This is the required amount of prefunding for the paymaster.

### Branches and code coverage (including function calls)

**Intended branches**

- Succeeds with valid gas limit, `userOp`, and `requiredPrefund`.
  ☑ Test coverage

**Negative behavior**

- Invalid signature causes error to be returned.
  ☑ Negative test
- Insufficient `requiredPrefund` revert.
  □ Negative test
- Parsing the Paymaster data causes revert.
  □ Negative test

## 4.2 Module: ChainlinkOracleAggregator.sol

### Function: `getTokenValueOfOneNativeToken(address token)`

This function is used to get the value of one native token in terms of the given token.

### Inputs

- `token`
  - **Control**: Fully controlled by user.
  - **Constraints**: Should be a valid ERC20 token address.
  - **Impact**: This is the token for which the price is to be queried.

### Branches and code coverage (including function calls)

**Intended branches**

- Check price result of a single token.
  ☑ Test coverage

**Negative behavior**

- Tokens with zero `tokenPriceUnadjusted` cause revert.
  □ Negative test

## 4.3 Module: USDCPriceFeedPolygon.sol

### Function: `getThePrice()`

This function is used to get the latest price.

### Branches and code coverage (including function calls)

**Intended branches**

- Successfully obtained the latest prices.
  ☑ Test coverage

# 5 Audit Results

At the time of our audit, the audited code was not deployed to mainnet Ethereum.

During our assessment on the scoped Token Paymaster contracts, we discovered three findings. No critical issues were found. Two were of low impact and the remaining finding was informational in nature.

## 5.1 Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the audit version of our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.