Quantstamp Security Assessment Certificate

September 26th 2020 — Quantstamp Verified

Biconomy (Selected Files Only)

This security assessment was prepared by Quantstamp, the leader in blockchain security

Executive Summary

Type	Transaction Standard
Auditors	Jake Goh Si Yuan, Research Engineer Poming Lee, Research Engineer Kevin Feng, Blockchain Researcher
Timeline	2020-08-31 through 2020-09-03
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
metatx-standard	40a7bb3
None	f0e64aa
None	248183d
None	6516441
None	03630e2

Total Issues	**3** (3 Resolved)
High Risk Issues	**1** (1 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	0 (0 Resolved)
Informational Risk Issues	**2** (2 Resolved)
Undetermined Risk Issues	0 (0 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

The code assessed were well-formed, structured and concised, and adheres to the specification of `EIP712` and the meta transaction standard. Whilst the in-code documentation was precise and necessary, and the general documentation was informative, we found it particularly lacking when it came to highlighting the differences between `BasicMetaTransaction` and the `EIP712MetaTransaction`. Additionally, there was also a complete lack of automated tests. We have remarked these issues, along with some others, in the report below. There were one high-key, two informational issues found, and two best practices recommendations made. We highly recommend addressing these findings.

ID	Description	Severity	Status
QSP-1	Lack of tests	High	Fixed
QSP-2	Unlocked and inconsistent pragma	Informational	Fixed
QSP-3	Incorrect/Missing Visibility	Informational	Fixed

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.6.6
- Mythril v0.22.8

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither ./path/to/contract`
3. Installed the Mythril tool from Pypi: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth a path/to/contract`

Findings

QSP-1 Lack of tests

Severity: *High Risk*

Status: Fixed

Description: There are no tests in the repositories audited, besides a `TestContract.sol` which corresponds to a demonstration effort to showcase a manual example of the meta transaction. We highly recommend implementing some basic tests to achieve full test coverage. Given the brevity of the codebase, this should not be too difficult to achieve, and would go a long way in bringing confidence and assurance to the functionality, consistency and security of the code.

Recommendation: Implement some basic tests to achieve full code coverage.

QSP-2 Unlocked and inconsistent pragma

Severity: *Informational*

Status: Fixed

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked." For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.
At the same time, files from the same repository had different pragma versions, which is not recommended from a security perspective.

Recommendation: Lock the pragma.

QSP-3 Incorrect/Missing Visibility

Severity: *Informational*

Status: Fixed

File(s) affected: `EIP712MetaTransaction.sol`, `BasicMetaTransaction.sol`

Description: The visibility of a function or field changes determines how it can be accessed by others. Using the right visibility ensures optimal gas costs and reduces possibilities of attacks. The four types of visibility are: * `external` - can be called by any other contract (but not within the contract itself) * Useful for optimizing gas costs * `public` - can be called anywhere * `internal` - can only be called within contract, and inherited contracts will inherit this functionality * Useful for creating functions which should not be called by anyone else * `private` - can only be called within contract, cannot be inherited
In particular, we find that `mapping(address=>uint256) nonces` to have this issue in both contracts referred.

Recommendation: Be explicit about the visibility required with `nonces`.

Automated Analyses

Slither

Compilation warnings/errors on ./BasicMetaTransaction.sol: Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: " to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information. --> ./BasicMetaTransaction.sol

Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: " to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information. --> ./lib/SafeMath.sol

INFO:Detectors: Contract locking ether found in : Contract BasicMetaTransaction (BasicMetaTransaction.sol#6-76) has payable functions: - BasicMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (BasicMetaTransaction.sol#32-44) But does not have a function to withdraw the ether Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether INFO:Detectors: Reentrancy in BasicMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (BasicMetaTransaction.sol#32-44): External calls: - (success,returnData) = address(this).call(abi.encodePacked(functionSignature,userAddress)) (BasicMetaTransaction.sol#39) Event emitted after the call(s): - MetaTransactionExecuted(userAddress,msg.sender,functionSignature) (BasicMetaTransaction.sol#42) Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3 INFO:Detectors: BasicMetaTransaction.getChainID() (BasicMetaTransaction.sol#13-19) uses assembly - INLINE ASM None (BasicMetaTransaction.sol#15-17) BasicMetaTransaction.msgSender() (BasicMetaTransaction.sol#64-75) uses assembly - INLINE ASM None (BasicMetaTransaction.sol#68-71) Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage INFO:Detectors: Pragma version^0.6.0 (BasicMetaTransaction.sol#1) necessitates versions too recent to be trusted. Consider deploying with 0.5.11 Pragma version^0.6.0 (lib/SafeMath.sol#1) necessitates versions too recent to be trusted. Consider deploying with 0.5.11 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity INFO:Detectors: Low level call in BasicMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (BasicMetaTransaction.sol#32-44): - (success,returnData) = address(this).call(abi.encodePacked(functionSignature,userAddress)) (BasicMetaTransaction.sol#39) Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls INFO:Detectors: executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) should be declared external: - BasicMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (BasicMetaTransaction.sol#32-44) getNonce(address) should be declared external: - BasicMetaTransaction.getNonce(address) (BasicMetaTransaction.sol#46-48) Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-as-external INFO:Slither:./BasicMetaTransaction.sol analyzed (2 contracts with 46 detectors), 9 result(s) found

Warning: Function state mutability can be restricted to pure --> ./EIP712MetaTransaction.sol:45:5: | 45 | function hashMetaTransaction(MetaTransaction memory metaTx) internal view returns (bytes32) { | ^ (Relevant source part starts here and spans across multiple lines).

INFO:Detectors: Contract locking ether found in : Contract EIP712MetaTransaction (EIP712MetaTransaction.sol#7-77) has payable functions: - EIP712MetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (EIP712MetaTransaction.sol#27-43) But does not have a function to withdraw the ether Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether INFO:Detectors: Reentrancy in EIP712MetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (EIP712MetaTransaction.sol#27-43): External calls: - (success,returnData) = address(this).call(abi.encodePacked(functionSignature,userAddress)) (EIP712MetaTransaction.sol#38) Event emitted after the call(s): - MetaTransactionExecuted(userAddress,msg.sender,functionSignature) (EIP712MetaTransaction.sol#41) Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3 INFO:Detectors: EIP712MetaTransaction.msgSender() (EIP712MetaTransaction.sol#64-76) uses assembly - INLINE ASM None (EIP712MetaTransaction.sol#68-71) EIP712Base.getChainID() (lib/EIP712Base.sol#26-30) uses assembly - INLINE ASM None (lib/EIP712Base.sol#27-29) Reference:

https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage INFO:Detectors: Pragma version0.6.9 (EIP712MetaTransaction.sol#1) necessitates versions too recent to be trusted. Consider deploying with 0.5.11 Pragma version0.6.9 (lib/EIP712Base.sol#1) necessitates versions too recent to be trusted. Consider deploying with 0.5.11 Pragma version0.6.9 (lib/SafeMath.sol#1) necessitates versions too recent to be trusted. Consider deploying with 0.5.11 Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity INFO:Detectors: Low level call in EIP712MetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (EIP712MetaTransaction.sol#27-43): - (success,returnData) = address(this).call(abi.encodePacked(functionSignature,userAddress)) (EIP712MetaTransaction.sol#38) Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls INFO:Detectors: executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) should be declared external: - EIP712MetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (EIP712MetaTransaction.sol#27-43) getNonce(address) should be declared external: - EIP712MetaTransaction.getNonce(address) (EIP712MetaTransaction.sol#54-56) Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-as-external INFO:Slither:./EIP712MetaTransaction.sol analyzed (3 contracts with 46 detectors), 10 result(s) found

Mythril

The analysis was completed successfully. No issues were detected.

Adherence to Specification

The codebase adheres to the implementation of `EIP712` and the meta transaction standard.

Code Documentation

In-code documentation and general documentation was present and informative. However, there could be more done to highlight the difference, or lack thereof, between `BasicMetaTransaction` and `EIP712MetaTransaction`.

Adherence to Best Practices

- In `EIP712MetaTransaction.sol::L40` of `f0e64aaf`, `Function call not successfull` is a typo. It should be `Function call not successful`.
- In `BasicMetaTransaction.sol::L41` of `40a7bb35`, `Function call not successfull` is a typo. It should be `Function call not successful`.
- In `EIP712MetaTransaction.sol::L45` of `f0e64aaf`, the function `hashMetaTransaction` can be set to `pure` visibility.

Test Results

Test Suite Results

The Biconomy team has improved on their test suite as a result of this audit, and the tests are relevant and work fine. More work should be made to improve on test setup, as currently it involves taking steps to manually encode the chain id into the test file, and having to use a hardcoded private key.

```
Contract: EIP712MetaTransaction
  Check Methods
     ✓ Should be able to send transaction successfully (188ms)
     ✓ Call the contract method directly (106ms)
     ✓ Should fail when replay transaction (161ms)
     ✓ Should fail when user address is Zero (89ms)
     ✓ Should be failed - Signer and Signature do not match (96ms)


5 passing (890ms)
```

Code Coverage

The Biconomy team has massively improved on their code coverage as a fix for their audit, and it has shown with almost 100% code coverage for relevant contracts. There is still some room to get to full coverage, and it is recommended that the Biconomy team continues to try to cover all possible branches.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	75	100	100	
EIP712MetaTransaction.sol	100	75	100	100	
TestContract.sol	100	100	100	100	
contracts/lib/	100	100	100	100	
EIP712Base.sol	100	100	100	100	
All files	**100**	**75**	**100**	**100**	

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
1d3bfa469433e2b794ecfed746286838c4e567114bd6a44ed56bb3d0d9d7fce4 ./biconomy/03630e205b57fa4f33923d326456b57fadc73742/metatx-standard/src/contracts/lib/EIP712Base.sol

caf0a2e35ff316e41854b772111ab48ee9ab5cc51b449879e0b081cfc0432a34 ./biconomy/03630e205b57fa4f33923d326456b57fadc73742/metatx-standard/src/contracts/EIP712MetaTransaction.sol

ce48756440892311367e30829109e5f4d938321995bb73777b258a3d7b163fee ./biconomy/40a7bb35d5b4bfb7be8136d770b91b04c4f05cea/metatx-standard/src/contracts/BasicMetaTransaction.sol

90dd02d8edda178566141fdd350d95508c5b2cdade470b3f7cefd93c2b6557d2 ./biconomy/f0e64aaf84502d5ef660d9568e7673e6ac780cc8/metatx-standard/src/contracts/EIP712MetaTransaction.sol

e77ff4d8f4c45b0b0236fd5370bad7866321bbed973bea7df4d26c9d33aa6920 ./biconomy/f0e64aaf84502d5ef660d9568e7673e6ac780cc8/metatx-standard/src/contracts/EIP712Base.sol
```

Tests

```
ea93e854ace739eda0239a1e8f395344e148317ab9f3d740a2b740f81496d32e ./biconomy/03630e205b57fa4f33923d326456b57fadc73742/metatx-standard/src/test/EIP712MetaTransaction.test.js
```

Changelog

- 2020-09-02 - Initial report

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.