decurity

# Smart Contract Security Audit Report

Tokenlon

# 1. Contents

# 2. General Information

This report contains information about the results of the security audit of the Tokenlon (hereafter referred to as “Customer”) RFQv2 smart contract, conducted by Decurity in the period from 08/15/2023 to 08/18/2023.

## 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 following smart contract: https://github.com/consenlabs/tokenlon-contracts/blob/v5.3.3-Decurity-audit/contracts/RFQv2.sol. The review was done for the commit e1e6da301a4e6d52087967d0d19a88fe20f604e0.

## 2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract). The centralization risks have not been considered upon the request of the Customer.

The main possible threat actors are:

- User,
- Protocol owner,
- Liquidity Token owner/contract.

The table below contains sample attacks that malicious attackers might carry out.

*Table. Theoretically possible attacks*

| Attack | Actor |
|---|---|
| Contract code or data hijacking<br>*Deploying a malicious contract or submitting malicious data* | Contract owner<br>Token owner |
| Financial fraud<br>*A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack* | Anyone |
| Attacks on implementation<br>*Exploiting the weaknesses in the compiler or the runtime of the smart contracts* | Anyone |

## 2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an 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 best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

# 3. Summary

As a result of this work, we haven't discovered any exploitable security issues. There were no major changes since the last audit iteration.

## 3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of August 22, 2023.

*Table. Discovered weaknesses*

| Issue | Contract | Risk Level | Status |
|---|---|---|---|
| DoS via permit front running | contracts/utils/TokenCollector.sol | **Low** | Acknowledged |
| String error messages used instead of custom errors | contracts/RFQv2.sol | **Info** | Acknowledged |

# 4. General Recommendations

This section contains general recommendations on how to improve 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. DoS via permit front running

**Risk Level**: Low

**Status**: Acknowledged

**Contracts**:

- contracts/utils/TokenCollector.sol

**Description:**

When filling an order using the `fillRFQ()` function, there are several ways to transfer funds. If the protocol should make permit calls it can be done via the `_collectByToken()` or `_collectByPermit2AllownaceTransfer()` functions.

There is a possibility to front-run the filling of the exact order and execute the permit before the original order. This will revert that exact order filling, but the order can still be executed without a permit.

**Remediation:**

Consider wrapping the permit call in a try/catch block.

## 5.2. String error messages used instead of custom errors

**Risk Level**: Info

**Status**: Acknowledged: the v5 is still written in solidity v0.7.6 so no custom error feature yet.

**Contracts**:

- contracts/RFQv2.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.

```
contracts/RFQv2.sol:
    48: require(_newFeeCollector != address(0), "zero address");
    83: require(_offer.expiry > block.timestamp, "offer expired");
    84: require(_offer.feeFactor < LibConstant.BPS_MAX, "invalid fee factor");
    85: require(order.recipient != address(0), "zero recipient");
    94: require(isValidSignature(_offer.maker, getEIP712Hash(offerHash),
bytes(""), makerSignature), "invalid signature");
    98: require(isValidSignature(_offer.taker, getEIP712Hash(rfqOrderHash),
bytes(""), takerSignature), "invalid signature");
    103: require(msg.value == _offer.takerTokenAmount, "invalid msg value");
    107: require(msg.value == 0, "invalid msg value");
```

**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/

# 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.