decurity

Smart Contract Security Audit Report

Tokenlon RFQv2

1. Contents

2. General Information

This report contains information about the results of the security audit of the TokenLon (hereafter referred to as "Customer") protocol RFQv2 smart contract, conducted by Decurity in the period from 05/15/2023 to 05/20/2023.

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 contract: https://github.com/consenlabs/tokenlon-contracts/blob/v5.3.2-audit/contracts/RFQv2.sol. Initial review was done for the commit 41cb6eeefb6d63af68270a4e2e11e8dd01f22006, the re-testing was done for the commit 692e937dcdbc9637427e60f8458a371a16663af0.

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (taker, maker, DEX, privileged roles). The centralization risks have not been considered upon the request of the Customer.

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 have discovered a single medium risk security issue 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 Tokenlon team has given the feedback for the suggested changes and explanation for the underlying code.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of May 26, 2023.

Issue	Contract	Risk Level	Status
Lack of `chainID` validation	contracts/utils/BaseLibEIP712.sol	**Medium**	Fixed
DoS via permit front running	contracts/utils/TokenCollector.sol	**Low**	Acknowledged
Missing 0x0 address check	contracts/RFQv2.sol	**Low**	Fixed
Taker may lose ETH	contracts/RFQv2.sol	**Low**	Fixed
Suboptimal comparison	contracts/RFQv2.sol	**Info**	Fixed
Revert string is longer than 32 bytes	contracts/utils/TokenCollector.sol	**Info**	Acknowledged
Suboptimal `memory` arguments	contracts/RFQv2.sol contracts/utils/Offer.sol contracts/utils/RFQOrder.sol	**Info**	Fixed
Typo in the function name	contracts/utils/TokenCollector.sol	**Info**	Fixed
Unused imports	contracts/RFQv2.sol contracts/utils/Asset.sol	**Info**	Fixed
Unused library function	contracts/utils/Asset.sol	**Info**	Fixed

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. Lack of `chainID` validation

Risk Level: **Medium**

Status: Fixed in the commit 9b5208.

Contracts:

- contracts/utils/BaseLibEIP712.sol

Description:

The `chainID` parameter of the network, which is fixed at deployment time, is used by the `BaseLibEIP712` contract to compute the domain separator during deployment. In the case of a post-deployment chain fork, `chainID` cannot be altered, but the signatures may be replayed across both chains.

The `isValidSignature()` function is used to check the validity of a signature of a given `rfqOrderHash/offerHash` hash.

Hash is created by calling `getEIP712Hash()` function that concatenates passed to its hash with `EIP712_DOMAIN_SEPARATOR` that was defined at deployment time.

Thus, in case of a hard fork, a hacker might utilize previous signatures to obtain user payment on both chains. If a change in `chainID` is detected, it is possible to cache and renew the domain separator to mitigate this risk. Alternately, `chainID` can be included in the schema of the signature supplied to the order hash in place of completely recreating the domain separator.

Remediation:

The `CHAINID` opcode should be included in the signature schema to guard against post-deployment forks affecting signatures.

References:

- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/EIP712.sol

5.2. 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.

5.3. Missing 0x0 address check

Risk Level: Low

Status: Fixed in the commit 47141c.

Contracts:

- contracts/RFQv2.sol

Location: Lines: 121.

Description:

There is no check that `_rfqOrder.recipient` address is not equal to zero.

```
contracts/RFQv2.sol:
  121:         makerToken.transferTo(_rfqOrder.recipient, makerTokenToTaker);
```

Remediation:

Consider checking that the input address is not zero to prevent users from losing their funds.

5.4. Taker may lose ETH

Risk Level: Low

Status: Fixed in the commit f233fc.

Contracts:

- contracts/RFQv2.sol

Location: Function: `fillRFQ`.

Description:

In case when a taker calls the `fillRFQ()` function of `RFQv2` and sends some amount of ETH but `takerToken` is not `ETH` taker will lose them.

Remediation:

Consider adding additional checks to make sure that if the taker asset is not `ETH` then `msg.value` should be equal to zero.

5.5. Suboptimal comparison

Risk Level: Info

Status: Fixed in the commit e5bf05.

Contracts:

- contracts/RFQv2.sol

Description:

When comparing integers, it is cheaper to use strict `>` and `<` operators instead of `>=` and `<=` operators:

```
contracts/RFQv2.sol:
  74:    require(_offer.expiry > block.timestamp, "offer expired");
  75:    require(_rfqOrder.feeFactor <= LibConstant.BPS_MAX, "invalid fee
factor");
```

Remediation:

Changing the operators does not significantly affect the logic of these `require` statements but it can save gas.

5.6. Revert string is longer than 32 bytes

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/utils/TokenCollector.sol

Description:

There are several places where the revert string is longer than 32 bytes:

```
contracts/utils/TokenCollector.sol:
  92:    require(amount < uint256(type(uint160).max), "TokenCollector: permit2
amount too large");
 112:    require(data.length > 0, "TokenCollector: permit2 data cannot be
empty");
```

Remediation:

Shortening revert strings to fit in 32 bytes will decrease gas costs for deployment and gas costs when the revert condition has been met.

References:

- https://github.com/byterocket/c4-common-issues/blob/main/0-Gas-Optimizations.md/#g007—long-revert-strings

5.7. Suboptimal `memory` arguments

Risk Level: Info

Status: Fixed in the commit 692e93.

Contracts:

- contracts/RFQv2.sol
- contracts/utils/Offer.sol
- contracts/utils/RFQOrder.sol

Description:

All variables that are passed to the `_fillRFQ()` function are not modified during the execution so their location can be changed to `calldata`.

The `_offer` variable is never changed during the `_fillRFQ()` function execution so its location can be changed to `calldata` as well.

The same applies to the `getOfferHash()`, `_emitFilledRFQEvent()`, and `getRFQOrderHash()` functions.

```
contracts/RFQv2.sol:
   66:         RFQOrder memory _rfqOrder,
   67:         bytes memory _makerSignature,
   68:         bytes memory _makerTokenPermit,
   69:         bytes memory _takerSignature,
   70:         bytes memory _takerTokenPermit
   72:         Offer memory _offer = _rfqOrder.offer;
  128:         RFQOrder memory _rfqOrder,

contracts/utils/Offer.sol:
  21: function getOfferHash(Offer memory offer) pure returns (bytes32) {

contracts/utils/RFQOrder.sol:
  18: function getRFQOrderHash(RFQOrder memory rfqOrder) pure returns (bytes32
offerHash, bytes32 orderHash) {
```

Remediation:

Consider changing the location of the variables to `calldata` instead of `memory` to save gas.

5.8. Typo in the function name

Risk Level: Info

Status: Fixed in the commit b8064d.

Contracts:

- contracts/utils/TokenCollector.sol

Location: Lines: 85. Function: `_collectByPermit2AllownaceTransfer`.

Description:

The name of the function `_collectByPermit2AllownaceTransfer()` contains a typo.

```
contracts/utils/TokenCollector.sol:
  85:     function _collectByPermit2AllownaceTransfer(
```

Remediation:

Rename the function to `_collectByPermit2AllowanceTransfer`.

5.9. Unused imports

Risk Level: Info

Status: Fixed in the commit 8e33aa.

Contracts:

- contracts/RFQv2.sol
- contracts/utils/Asset.sol

Description:

Imported `getOfferHash` function in the `RFQv2` contract and `Address` in the `Asset` library are never used:

```
contracts/RFQv2.sol:
  11: import { Offer, getOfferHash } from "./utils/Offer.sol";

contracts/utils/Asset.sol:
  6: import { Address } from "@openzeppelin/contracts/utils/Address.sol";
```

Remediation:

Consider removing the unused imports.

5.10. Unused library function

Risk Level: Info

Status: Fixed in the commit 76c08a.

Contracts:

- contracts/utils/Asset.sol

Location: Lines: 17. Function: `getBalance`.

Description:

The function `getBalance` is never used:

```
contracts/utils/Asset.sol:
  17:     function getBalance(address asset, address owner) internal view 
returns (uint256) {
  18:         if (isETH(asset)) {
  19:             return owner.balance;
  20:         } else {
```

```
                return IERC20(asset).balanceOf(owner);
            }
        }
```

Remediation:

Consider removing unused function from the library.

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.