Smart Contract Security Audit Report

Tokenlon Limit Order

Decurity, 2022

Contents

1. General Information

This report contains information about the results of the security audit of the Tokenlon (hereafter referred to as “Customer”) smart contracts, conducted by [Decurity](https://decurity.io) in the period from 12/26/2022 to 01/16/2023.

1.1. 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.

1.2. Scope of Work

The audit scope included the contracts in the following repository: https://github.com/consenlabs/tokenlon-contracts. Initial review was done for the tree audit-v5.3.1-decurity (commit 023a33745c50cda34efad24634813f8fcc0efbab) and the re-testing was done for the commit 99b7c6e809894fa371667571b48bc6e64dedac62.

The following contracts have been included into the scope by the Customer:

- AllowanceTarget.sol
- LimitOrder.sol
- PermanentStorage.sol
- ProxyPermanentStorage.sol
- Spender.sol
- Tokenlon.sol
- UserProxy.sol

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

1.4. 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. Summary

As a result of this work, we have discovered a few low-risk and medium-risk issues which have been fixed and re-tested in the course of the work. The other suggestions included some coding best practices.

The Tokenlon team has given the feedback for the suggested changes and explanation for the underlying code.

2.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of Jan 18, 2023.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Inability to call initialize on implementation contract	UserProxy.sol PermanentStorage.sol	Medium	Acknowledged
Lack of two-step process for contract ownership change	PermanentStorage.sol LimitOrder.sol Spender.sol UserProxy.sol	Medium	Fixed
Operator can set 100% fees	LimitOrder.sol	Medium	Fixed
Dependency contracts do not track upstream changes	UpgradeableProxy.sol Proxy.sol TransparentUpgradeableProxy.sol Multicall.sol	Low	Acknowledged

	LibBytes.sol UniswapV3PathLib.sol		
Missing events	AllowanceTarget.sol Spender.sol	Low	Acknowledged
No check for the zero address	LimitOrder.sol	Low	Fixed
Potential proxy init front-running	PermanentStorage.sol UserProxy.sol	Low	Acknowledged
Insufficient sanity checks	LimitOrder.sol	Low	Fixed
Potential sandwiching by the relayers	LimitOrder.sol	Info	Acknowledged
Variable shadowing	ProxyPermanentStorage.sol Tokenlon.sol	Info	Acknowledged
Unused library functions	LibBytes.sol LibUniswapV3.sol	Info	Acknowledged
Checking bool equality	UserProxy.sol	Info	Fixed
Storage variables should be cached in memory	LimitOrder.sol	Info	Fixed
Non-optimal post-increment/decrement	LimitOrder.sol PermanentStorage.sol Spender.sol Multicall.sol	Info	Fixed
Spender should inherit from ISpender	Spender.sol	Info	Acknowledged
Usage of deprecated	LimitOrder.sol	Info	Acknowledged

safeApprove()			

3. General Recommendations

This section contains general recommendations on how to fix discovered weaknesses and vulnerabilities and how to improve overall security level.

Section 3.1 contains a list of general mitigations against the discovered weaknesses, technical recommendations for each finding can be found in section 4.

Section 3.2 describes a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level.

3.1. Current Findings Remediation

Follow the recommendations in section 4.

3.2. Security Process Improvement

- 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.

4. Findings

4.1. Inability to call initialize on implementation contracts

Risk Level: Medium

Contracts:

- UserProxy.sol
- PermanentStorage.sol

Status: Acknowledged: Currently the initialize() is expected to be called once only (via proxy's upgradeToAndCall function). The version var is deprecated but is left in contract due in order to keep slot ordering of the upgradable proxy.

Description:

The function `initialize()` assumes that the state variable `version` is empty when upgrading to the new version of the contract:

```
function initialize(address _operator) external {
    require(keccak256(abi.encodePacked(version)) ==
keccak256(abi.encodePacked("")), "PermanentStorage: not upgrading from
empty");
    require(_operator != address(0), "PermanentStorage: operator can not be
zero address");
    operator = _operator;

    // Upgrade version
    version = "5.3.0";
}
```

Otherwise `initialize()` would always revert which means that the redeployment of the proxy contract is necessary. The function has been found in the following contracts:

- contracts/UserProxy.sol:49 – initialize
- contracts/PermanentStorage.sol:89 – initialize

`initialize()` should be allowed to be called when upgrading from previous versions, e.g. the `version` of [Tokenlon]() contract is currently set to “5.2.0”.

Remediation:

Consider specifying the exact previous version that the new version is expected to be upgraded from.

```
require(keccak256(abi.encodePacked(version)) ==
keccak256(abi.encodePacked("5.2.0")));
```

4.2. Lack of two-step process for contract ownership changes

Risk Level: Medium

Contracts:

- PermanentStorage.sol
- LimitOrder.sol
- Spender.sol
- UserProxy.sol

Status: Spender cannot be upgraded in this version due to compatibility issues so won't be fixed. Fixed in the following commit: https://github.com/consenlabs/tokenlon-contracts/commit/3c07f7852fb6fea97d4354e25f4e625aaedfbd66.

Description:

The function `transferOwnership` immediately sets the contract's new `operator`.

```
function transferOwnership(address _newOperator) external onlyOperator {
      require(_newOperator != address(0), "UserProxy: operator can not be
zero address");
      operator = _newOperator;

      emit TransferOwnership(_newOperator);
  }
```

Making such a critical change in a single step is error-prone and can lead to irrevocable mistakes. There are the following occurrences:

- contracts/PermanentStorage.sol:67 transferOwnership
- contracts/LimitOrder.sol:81 – transferOwnership
- contracts/Spender.sol:43 – transferOwnership
- contracts/UserProxy.sol:38 – transferOwnership

Remediation:

Although `transferOwnership` can only be called by the current `operator` which is a `DelayedMultiSig`, the actual time lock is set to zero (https://etherscan.io/address/0x9aFc226Dc049B99342Ad6774Eeb08BfA2F874465#readContract).

Consider implementing a two-step process for changing `operator` in which the current operator proposes a new address and then the new address executes a call to accept the role, completing the transfer.

4.3. Operator can set 100% fees

Risk Level: Medium

Contracts:

- LimitOrder.sol

Status: Fixed in the following commit: https://github.com/consenlabs/tokenlon-contracts/commit/99b7c6e809894fa371667571b48bc6e64dedac62.

Description:

The `setFactors()` function that is controlled by the operator sets the contract's fee factors. The max value that can be set is equal to 100 percent. This means that all funds from orders that will be settled through this protocol will be collected by `feeCollector`.

Remediation:

Consider implementing a two-step process for changing fee factors which will allow users to cancel their orders if they are not satisfied with the new fee.

4.4. Dependency contracts do not track upstream changes

Risk Level: Low

Contracts:

- UpgradeableProxy.sol
- Proxy.sol
- TransparentUpgradeableProxy.sol
- Multicall.sol
- LibBytes.sol
- UniswapV3PathLib.sol

Status: Acknowledged: the team will add documentation of this part.

Description:

In the Tokenlon codebase multiple modified third party contracts are used, however it is unclear which exact version of the contract is used and what modifications were made. Using Decurity's contract-diff.xyz tool, the closest similar contracts were identified for the following third party contracts:

Contract	Version	Last modified
upgrade_proxy/UpgradeableProxy.sol	v4.0.0-beta.0	27 Jan 2021
upgrade_proxy/Proxy.sol	v3.3.0-rc.0	14 Nov 2020
upgrade_proxy/TransparentUpgradeableProxy.sol	v4.0.0-beta.0	22 Feb 2021
utils/Multicall.sol	v1.1.1	15 Mar 2021

utils/LibBytes.sol	1cf8ae	14 Oct 2019
utils/UniswapV3PathLib.sol (BytesLib)	v1.1.1	15 Mar 2021

Upstream changes in these contracts are not tracked. Thus, updates and security fixes implemented in the dependencies may not be reliably reflected, as those updates must be manually integrated into the contracts.

Remediation:

Document the source, the version and modifications of each dependency. Include third party sources as submodules or preferably as NPM packages. E.g. use openzeppelin-contracts-upgradeable instead of copies of contracts in the `upgrade_proxy` directory.

4.5. Missing events

Risk Level: Low

Contracts:

- AllowanceTarget.sol
- Spender.sol

Status: Acknowledged: known issue but won't fix since Spender cannot be upgraded in this version.

Description:

No events are emitted when the addresses of core roles in Spender.sol (`operator`, `timelockActivated`, `authorized`, `allowanceTarget`, `tokenBlacklist`) and AllowanceTarget.sol (`newSpender`) contracts are changed.

Remediation:

Consider to add events that could help to track changing in roles in these functions of Spender.sol file:

- transferOwnership()
- activateTimelock()

- authorize()
- deauthorize()
- setAllowanceTarget()
- setNewSpender()
- teardownAllowanceTarget()
- blacklist()

4.6. No check for the zero address

Risk Level: Low

Contracts:

- LimitOrder.sol

Status: Fixed in the following commit:
https://github.com/consenlabs/tokenlon-contracts/commit/9111d79fd38b9e74dfd3452103850c1a45b3b0e9.

Description:

The LimitOrder contract contains several functions that don’t verify input values.

The `fillLimitOrderByTrader()` function does not ensure that `_params.recipient` is not set to the address zero.

The `fillLimitOrderByProtocol()` function does not ensure that `_params.profitRecipient` is not set to the address zero.

Without address verification, funds may be lost during order filling.

Remediation:

Consider adding a check that the input address of the recipient is not equal to zero address.

4.7. Potential proxy init front-running

Risk Level: Info

Contracts:

- PermanentStorage.sol
- UserProxy.sol

Status: Acknowledged: known issue. The current contract cannot be init again so won't fix.

Description:

In PermanentStorage.sol and UserProxy.sol there are functions `initialize()` for contract initialization purposes.

```
function initialize(address _operator) external {
        // setting operator address
        operator = _operator;
        // ...
}
```

However, if the function `upgradeToAndCall()` with initialization calldata is not invoked from the proxy side while upgrading the implementation contract, anyone would be able to front-run the call to `initialize()` with a malicious calldata.

Remediation:

Initialize the contract immediately or in the same transaction.

4.8. Insufficient sanity checks

Risk Level: Low

Contracts:

- LimitOrder.sol

Status: Fixed in the following commit:
https://github.com/consenlabs/tokenlon-contracts/commit/f665d7470e210c2a0e7e243822173b4c76d60b0e.

Description:

There is a `_quoteOrder()` function in the LimitOrder contract that calculates the amount of tokens that should be transferred during settlement.

During calculation of `makerTokenQuota` there is a possibility of loss of precision which can lead to loss of traders funds.

```
uint256 makerTokenQuota =
takerTokenQuota.mul(_order.makerTokenAmount).div(_order.takerTokenAmount);
```

Suppose that `takerTokenQuota` and `_order.makerTokenAmount` are small values but the value of `_order.takerTokenAmount` , on the contrary, is large. So the value of the `makerTokenQuota` can be equal to zero.

In addition, you should check that `_takerTokenAmount` is not equal to zero. If `_takerTokenAmount` is equal to zero then there are no funds that would be transferred during the execution of settlement.

Remediation:

Check that `makerTokenQuota` and `takerTokenQuota` are greater than `0`.

4.9. Potential sandwiching by the relayers

Risk Level: Info

Contracts:

- LimitOrder.sol

Status: Acknowledged.

Description:

The relayers control the slippage value and can sandwich their own trades so that the protocol doesn't get any profit.

Remediation:

Analyze the trades off-chain and prune the relayers who abuse the protocol. Alternatively, set the minimal profit value.

4.10. Variable shadowing

Risk Level: Info

Contracts:

- ProxyPermanentStorage.sol
- Tokenlon.sol

Status: Acknowledge: proxy contract won't be upgraded in this release.

Description:

The variable `_admin` in the `constructor()` function shadows the state variable `_admin` inherited from the TransparentUpgradeableProxy contract.

There are the following occurrences:

- ProxyPermanentStorage.sol#L11
- Tokenlon.sol#L10

Remediation:

To prevent shadowing, consider renaming the variables.

4.11. Unused library functions

Risk Level: Info

Contracts:

- LibBytes.sol
- LibUniswapV3.sol

Status: Acknowledgement.

Description:

There are functions that are never user:

- LibBytes.readAddress(bytes,uint256) (contracts/utils/LibBytes.sol#47-66)
- LibBytes.readBytes2(bytes,uint256) (contracts/utils/LibBytes.sol#111-125)
- LibBytes.readBytes4(bytes,uint256) (contracts/utils/LibBytes.sol#95-109)
- LibUniswapV3.exactInputSingle(address,LibUniswapV3.ExactInputSingleParams)(contracts/utils/LibUniswapV3.sol#28-42)

Remediation:

Consider removing unused functions from the libraries.

4.12. Checking bool equality

Risk Level: Info

Contracts:

- UserProxy.sol

Status: Fixed in the following commit: https://github.com/consenlabs/tokenlon-contracts/commit/3a21076fa5ab7efec03bba16a666245b265b2263.

Description:

Several unnecessary booleans are compared for boolean variables.

```
178:  if (callSucceed == false) {}
197:  if (callSucceed == false) {}
216:  if (callSucceed == false) {}
232:  if (callSucceed == false) {}
```

Boolean constants can be used directly without having to be compared to true or false. Moreover, every comparison costs extra gas.

Remediation:

Consider removing the comparison with the boolean constants.

4.13. Storage variables should be cached in memory

Risk Level: Info

Contracts:

- LimitOrder.sol

Status: Fixed in the following commit:
https://github.com/consenlabs/tokenlon-contracts/commit/ad68baa5e56323d34f1e557254155aab496599d2.

Description:

Storage variables should be cached in the memory rather than rereading them from storage.

```
contracts/LimitOrder.sol:
  272:          spender.spendFromUser(_settlement.trader,
address(_settlement.takerToken), _settlement.takerTokenAmount);
  276:              _settlement.takerToken.safeTransfer(feeCollector,
takerTokenFee);
  280:          spender.spendFromUser(_settlement.maker,
address(_settlement.makerToken), _settlement.makerTokenAmount);
  284:              _settlement.makerToken.safeTransfer(feeCollector,
makerTokenFee);
```

Remediation:

The `spender` variable from `_settleForTrader()` function should be cached to save gas.

The `feeCollector` variable could be cached in the memory if you plan to charge a commission from maker and taker. Therefore, the `makerTokenFee` and `takerTokenFee` variables will not be zero, and the `feeCollector` variable will be reading from storage several times which consumes extra gas.

4.14. Non-optimal post-increment/decrement

Risk Level: Info

Contracts:

- LimitOrder.sol
- PermanentStorage.sol
- Spender.sol
- Multicall.sol

Status: Will fix contracts other than Spender. Fixed in the following commit: https://github.com/consenlabs/tokenlon-contracts/commit/86896b9c3af960629911fd2ac0d88011c0ec74f7.

References:

- https://github.com/byterocket/c4-common-issues/blob/main/0-Gas-Optimizations.md/#g012---use-prefix-increment-instead-of-postfix-increment-if-possible

Description:

The difference between the prefix increment and postfix increment expression lies in the return value of the expression.

In case of post-increments/decrements, the compiler needs to copy the previous value in order to return and then process the expression.

In case of pre-increments/decrements, the compiler simply returns an already incremented/decremented value.

So, if you are not using the value of the expression, there is never a reason to use `i++` instead of `++i` because there is never a reason to copy the value of a variable, increment the variable, and then throw the copy away. That results in a little intrinsic gas optimization which is pretty significant in loops.

There are the following occurrences:

```
contracts/LimitOrder.sol:
  106:  for (uint256 i = 0; i < _tokenList.length; i++) {}
  114:  for (uint256 i = 0; i < _tokenList.length; i++) {}

contracts/PermanentStorage.sol:
```

```
  244:  for (int128 i = 0; i < underlyingCoinsLength; i++) {}
  251:  for (int128 i = 0; i < coinsLength; i++) {}
  292:  for (uint256 i = 0; i < _relayers.length; i++) {}

contracts/Spender.sol:
   97:  for (uint256 i = 0; i < _tokenAddrs.length; i++) {}
  112:  for (uint256 i = 0; i < _pendingAuthorized.length; i++) {}
  118:  for (uint256 i = 0; i < _pendingAuthorized.length; i++) {}
  129:  for (uint256 i = 0; i < numPendingAuthorized; i++) {}
  138:  for (uint256 i = 0; i < _deauthorized.length; i++) {}

contracts/utils/Multicall.sol:
  12:   for (uint256 i = 0; i < data.length; i++) {}
```

Remediation:

Consider using `++i` over `i++` if there is no usage of the copied value.

4.15. Spender should inherit from ISpender

Risk Level: Info

Contracts:

- Spender.sol

Status: Acknowledged.

Description:

Spender contract doesn't inherit from ISpender interface.

Remediation:

Inherit Spender contract from the ISpender interface.

4.16. Usage of deprecated safeApprove()

Risk Level: Info

Contracts:

- LimitOrder.sol

Status: Acknowledged: The major reason for using safeApprove() here is for non-standard ERC20 implementation. In our case the approval should be exhausted completely after the swap. Also, it's expected that no token will be left in the LimitOrder contract. So approval front-running is considered not relevant here.

References:

- https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2219

Description:

In LimitOrder.sol there is a function `fillLimitOrderByProtocol()` that could be called by the relayer in order to fill an open order. During the trade execution, the `_swapByProtocol()` is invoked to perform a swap using a specified router after collecting makerTokens from the maker.

```
contracts/utils/Limitorder.sol:
  383:  // Collect maker token from maker in order to swap through protocol
  384:  spender.spendFromUserTo(_settlement.maker,
address(_settlement.makerToken), address(this), _settlement.makerTokenAmount);
  385:
  386:  uint256 takerTokenOut = _swapByProtocol(_settlement);
```

To make that happen, LimitOrder.sol approves the collected amount of makerTokens to a router of either UniswapV3 or Sushiswap via `SafeERC20.safeApprove()` which doesn't fully resolve the issue related to front-running `ERC20.approve()`.

```
contracts/utils/Limitorder.sol:
  434:  _settlement.makerToken.safeApprove(_settlement.protocolAddress,
_settlement.makerTokenAmount);

  467:  _settlement.makerToken.safeApprove(_settlement.protocolAddress, 0);
```

Therefore, it was deprecated from using according to: https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2219.

Remediation:

Consider using `safeIncreaseAllowance()` and `safeDecreaseAllowance()` to handle approvals.

5. Appendix

5.1. About us

The Decurity (former DeFiSecurity.io) 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.