# xU3LP Smart Contract Audit

**xToken, 11 May 2021**

## 1. Introduction

iosiro was commissioned by xToken to conduct a smart contract audit of the xU3LP implementation. The audit was performed between 29 April and 7 May 2021 by one auditor, consuming seven resource days. A review of changes was performed on 10 May 2021.

This report is organized into the following sections.

- **Section 2 - Executive summary:** A high-level description of the findings of the audit.
- **Section 3 - Audit details:** A description of the scope and methodology of the audit.
- **Section 4 - Design specification:** An outline of the intended functionality of the smart contracts.
- **Section 5 - Detailed findings:** Detailed descriptions of the findings of the audit.

The information in this report should be used to understand the risk exposure of the smart contracts, and as a guide to improving the security posture of the smart contracts by remediating the issues that were identified. The results of this audit are only a reflection of the source code reviewed at the time of the audit and of the source code that was determined to be in-scope.

The purpose of this audit was to achieve the following:

- Ensure that the smart contracts functioned as intended.
- Identify potential security flaws.

Assessing the market effect, economics, game theory, or underlying business model of the platform were strictly beyond the scope of this audit.

Due to the unregulated nature and ease of transfer of cryptocurrencies, operations that store or interact with these assets are considered very high risk with regards to cyber attacks. As such, the highest level of security should be observed when interacting with these assets. This requires a forward-thinking approach, which takes into account the new and experimental nature of blockchain technologies. There are a number of techniques that can help to achieve this, some of which are described below.

- Security should be integrated into the development lifecycle.
- Defensive programming should be employed to account for unforeseen circumstances.
- Current best practices should be followed when possible.

## 2. Executive summary

This report presents the findings of an audit performed by iosiro on the smart contract implementation of xU3LP.

The purpose of xU3LP was to wrap UniswapV3 liquidity provider positions in order to provide a managed service for providing liquidity. Users would only need to deposit either of the two assets into the contract, which would then be managed by an admin account. This solution provides a fungible token that can be traded to represent shares of the underlying liquidity position, simplifying the liquidity-providing process and eliminating the need for higher technical expertise.

Five stable pairs are planned for release, with the exact price ranges for each pair to be determined before launch.

**System description**

To provide liquidity, users provide one of the two assets in the liquidity position and are allocated tokens based on their contribution to the system's Net Asset Value (NAV). When exiting the system, users would burn tokens in exchange for either asset and a portion of the collected fees.

The system was managed by admin roles, comprising of the owner and two manager addresses. The owner was rewarded with fees for various actions in the system, such as mint and burn fees. The admin roles were responsible for claiming the liquidity provider fees, rebalancing the contract's liquidity, and migrating the liquidity position if deemed necessary.

For a comprehensive description of the system, see Section 4 - Design Specification.

**Audit findings**

Overall, the system was found to operate as intended and the implementation was of a high standard.

- One high and one medium risk issue was identified and closed during the audit.
- Several low and informational risk issues were raised and closed, leaving one low risk and one informational open at the conclusion of the audit.

The dependency on the admin role presented a centralized risk, as the absence of the admin could lead to insufficient liquidity for withdrawals. Some precautions were taken to limit this exposure, including a mechanism that allowed users to unstake all provided liquidity and exit the system if the admin was absent for a set period of time.

Another audit was performed on the same scope by Quantstamp. Findings from both the iosiro and Quantstamp teams were communicated to xToken during the course of their respective audits. Findings that were first identified by Quantstamp's audit have not been included in this report, but have independently been confirmed to be appropriately addressed.

**Further precautions**

While the assessment attempted to identify any potential functional issues that may be encountered after deploying to mainnet, given that UniswapV3 is a relatively new product, it is recommended that the project is initially launched and tested on mainnet using limited liquidity before completely opening access to the contract.

# 3. Audit details

## 3.1 Scope

The source code considered in-scope for the assessment is described below. Code from all other files is considered to be out-of-scope. Out-of-scope code that interacts with in-scope code is assumed to function as intended and introduce no functional or security vulnerabilities for the purposes of this audit.

### 3.1.1 xU3LP smart contracts

**Project name:** xU3LP
**Commit:** 262fc9c
**Files:** xU3LPStable.sol, BlockLock.sol, libraries/ABDKMath64x64.sol, libraries/Utils.sol

### 3.1.2 xU3LP smart contracts review

**Project name:** xU3LP
**Commit:** fdaabe9

**Files:** xU3LPStable.sol, BlockLock.sol, libraries/ABDKMath64x64.sol, libraries/Utils.sol

## 3.2 Methodology

A variety of techniques were used in order to perform the audit. These techniques are briefly described below.

### 3.2.1 Code review

The source code was manually inspected to identify potential security flaws. Code review is a useful approach for detecting security flaws, discrepancies between the specification and implementation, design improvements, and high risk areas of the system.

### 3.2.2 Dynamic analysis

The contracts were compiled, deployed, and tested in a Ganache test environment, both manually and through the test suite provided. Manual analysis was used to confirm that the code operated at a functional level, and to verify the exploitability of any potential security issues identified.

### 3.2.3 Automated analysis

Tools were used to automatically detect the presence of several types of security vulnerabilities, including reentrancy, timestamp dependency bugs, and transaction-ordering dependency bugs. The static analysis results were manually analyzed to remove false-positive results. True positive results would be indicated in this report. Static analysis tools commonly used include Slither, Securify, and MythX. Tools such as the Remix IDE, compilation output, and linters are also used to identify potential issues.

## 3.3 Risk ratings

Each issue identified during the audit has been assigned a risk rating. The rating is determined based on the criteria outlined below.

- **High risk** - The issue could result in a loss of funds for the contract owner or system users.
- **Medium risk** - The issue resulted in the code specification being implemented incorrectly.
- **Low risk** - A best practice or design issue that could affect the security of the contract.
- **Informational** - A lapse in best practice or a suboptimal design pattern that has a minimal risk of affecting the security of the contract.
- **Closed** - The issue was identified during the audit and has since been addressed to a satisfactory level to remove the risk that it posed.

# 4. Design specification

The following section outlines the intended functionality of the system at a high level.

## 4.1 xU3LP

The specification given below was derived by iosiro from the code in the target commit hashes. Any discrepancies between this specification and expected behaviour should be brought to the attention of iosiro.

### 4.1.1 xU3LP token

Five xU3LP tokens with different stable pairs are to be launched. Each xU3LP token is an ERC20-compliant token with the following values.

| Field | Value |
|---|---|
| Name | xU3LP |
| Symbol | xU3LP(a/b/c/d/e) |
| Decimals | 18 |

### 4.1.2 User functionality

The following functionality was exposed publicly for users to interact with the xU3LP system.

#### Mint

In order to mint xU3LP, users would need to provide either of the stable assets in the liquidity position. In return, users would receive xU3LP tokens in proportion to their contribution to the contract's NAV. The funds would remain in the contract until a rebalance was performed. Each time a user minted tokens, an admin fee was taken.

#### Burn

Users were able to exit the system by burning their xU3LP tokens in exchange for either underlying asset. If there was not enough of the requested asset in the buffer, some of the remaining asset was swapped into the requested asset. Each time a user burned tokens, an admin fee was taken.

#### Emergency unstake

Due to limited liquid assets to exit the system, users were able to call the `emergencyUnstake` function if the admin had not rebalanced within a set period of time. This would allow users to unstake the liquidity position to unwind the assets and collect the fees. Users would then be able to exit the system by burning their xU3LP tokens.

### 4.1.3 Admin functionality

The following functionality was exposed to the owner and manager roles.

#### Initial mint

The `mintInitial()` function allowed an admin to open a liquidity position with the two allocated assets. An initial amount of xU3LP tokens are minted in this way, setting a baseline for the contract NAV.

#### Rebalance

A rebalance function was made available for admins to reallocate the assets between the contract and liquidity position. The function aimed to allocate 95% of the assets into the position, and aimed to allocate the remaining 5% of the assets into the contract to provide exit liquidity. This function also collected fees from the liquidity position.

#### Migrate

The `migratePosition()` function allowed an admin to migrate the liquidity position to a new position with a different range. During this process, the liquidity was withdrawn, fees were collected, and the underlying NFT was burnt. A new position with different ticks was then created.

#### Helper functions

Some helper functions were exposed to allow the admin to manage the system, including:

- Staking functions for the admin to directly stake and unstake.
- Swap functions for the admin to directly swap between the two assets.
- A function to withdraw fees and any tokens that had been sent to the contract erroneously.

# 5. Detailed findings

The following section includes in-depth descriptions of the findings of the audit.

## 5.1 High risk

No high risk issues were present at the conclusion of the audit.

## 5.2 Medium risk

No medium risk issues were present at the conclusion of the audit.

## 5.3 Low risk

### 5.3.1 `mintInitial()` function vulnerable to front running

*xU3LPStable.sol#L545*

The admin-only function `mintInitial()` was used to open a liquidity position and mint an initial amount of xU3LP tokens. However, a user could mint an initial number of tokens by front running a call to `mintInitial()` with a call to the `mintWithToken()` function. This would allow a user to get tokens at a 1:1 rate to their contribution without calculating their contribution to the NAV.

#### Recommendation

The `mintWithToken()` function should require that the `tokenId` is greater than 0, indicating that the liquidity position has been created already.

#### Developer comment

The xToken team indicated that the `mintInitial()` function is to be called directly after the contract is initialized, so any risk should be minimized.

## 5.4 Informational

### 5.4.1 No fees charged for small amounts

*xU3LPStable.sol#L136*, *xU3LPStable.sol#L152*

Fees can be avoided if the amount of liquidity sent is less than the mint fee (e.g. sending 499 wei of Dai when the mint fee is 500). This is unlikely to be exploited due to gas costs.

#### Recommendation

The minimum amount to be minted or burnt should be at least the fee amount to prevent users from avoiding fees.

#### Developer comment

The xToken team indicated that this issue is risk-accepted.

## 5.5 Closed

### 5.5.1 Incorrect mint calculation (high risk)

*xU3LPStable.sol#L141-L142*, *xU3LPStable.sol#L146-L147*

When minting xU3LP tokens, the number of tokens to mint was calculated as the contribution to the contract's NAV. However, when calculating the number of tokens to mint, the previous NAV included the fee amount taken from the user. This led to the user receiving slightly fewer tokens than intended, with the proportion dependent on the amount of fees taken. Moreover, the last person to burn their xU3LP tokens would be allocated the difference in assets that the user was meant to receive.

#### Recommendation

The withdrawable fees should be incremented prior to calling the `_mintInternal()` function, so that the `previousNav` as calculated in `calculateMintAmount()` is accurate. This should be done when minting with both asset 0 and asset 1.

#### Update

Fixed in 3398a90, 3398a90.

### 5.5.2 Ineffective `notLocked` modifier (medium risk)

*BlockLock.sol#L13*

The `notLocked` modifier was intended to prevent attack vectors that could take place within a short window of blocks. This was done by keeping track of the `msg.sender`'s last interaction with either `mintWithToken()`, `burn()` or `transfer()`, and any further interaction within 6 blocks would revert. However, this mechanism was ineffective as the `transferFrom()` function was not overridden. By using `approve()` and `transferFrom()`, it was possible for a user to transfer their tokens to another contract while they were locked. Since a different contract provides a different `msg.sender`, all functions would be unlocked within the 6 window block.

#### Recommendation

The `transferFrom()` function should be overridden to prevent a user from transferring their tokens while they are locked.

#### Update

Fixed in edf25ea.

### 5.5.3 `checkIfAmountsMatchAndSwap()` reverts during migration if the token ratios are not restored beforehand (low risk)

*xU3LPStable.sol#L432*

The call to `checkIfAmountsMatchAndSwap()` reverts during migration if the token ratios are not restored. This is due to a call to `getPositionLiquidity()` reverting, since a burnt `tokenID` is passed to Uniswap. The associated risk is minimal, and is more of a practical concern than a security one as the admin might be front-run.

#### Recommendation

The tokens should be in the correct ratio before migration to prevent this issue.

**Update**

Fixed in 3398a90, 3398a90.

### 5.5.4 `_setFeeDivisors()` does not match in-line comments (informational)

*xU3LPStable.sol#L627-L629*, *xU3LPStable.sol#L644-L646*

A comment in `setFeeDivisors()` indicated that the mint fee should be `0 or <= 2%`, and the claim fee should be `0 <= 4%`. However, this was not matched in the private function `_setFeeDivisors()`, where the `_mintFeeDivisor` needed to be larger than 100, and the claim fee could not be 0.

**Recommendation**

The fee divisor requirements should be updated to match the in-line comments, or the comment should be changed if the requirements are accurate.

**Update**

The comment was updated in edf25ea and the claim fee was adjusted in edf25ea.

### 5.5.5 Design considerations (informational)

Additional suggestions to enhance the overall system design are outlined below.

**Gas optimizations**

`setFeeDivisors()` should have external visibility.

**Update**

Fixed in 3398a90.