Quantstamp Security Assessment Certificate

February 3rd 2023 — Quantstamp Verified

LandX Finance

This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	Tokenized Commodities and Perpetual Vaults
Auditors	Ibrahim Abouzied, Auditing Engineer Marius Guggenmos, Senior Research Engineer Jonathan Mevs, Auditing Engineer
Timeline	2022-10-31 through 2022-11-10
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	LandX Smart Contract Documentation LandX Whitepaper
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
LandXit/land-x-smart-contracts	a04e915 initial audit
LandXit/land-x-smart-contracts	a0afdd1 fixes

Total Issues	**25** (24 Resolved)
High Risk Issues	**5** (5 Resolved)
Medium Risk Issues	**4** (3 Resolved)
Low Risk Issues	**6** (6 Resolved)
Informational Risk Issues	**8** (8 Resolved)
Undetermined Risk Issues	**2** (2 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).
Fixed	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

LandX is a protocol that offers tokenized commodities and perpetual vaults for agricultural products. Farmers raise capital by selling ownership shares of their crops sales to investors. Investors purchase the tokenized commodity in hopes of having an inflation hedged return. Farmers are held accountable through legal contracts secured on the underlying farmland. We found several high-severity issues, mostly surrounding incorrect accounting in the reward mechanism, lack of support of multiple crops, and vulnerability to sandwich attacks. We encourage the LandX team to address these issues before launch.
The test suite showed high coverage for the contracts under test. However, for some contracts, no tests exist at all and the existing tests were not diverse enough to catch issues like the hardcoded crop type. We highly recommend adding tests for all contracts and improving the existing ones to catch the issues found in this report.
Though the LandX protocol is built on a decentralized platform, the protocol does grant special privileges to the contract owners. While this is by design and may be necessary given the protocol's business requirements, we would like to note to users that the LandX protocol is not an entirely trustless system. Contract owners may be able to redirect user funds or manipulate trusted addresses. We encourage the LandX team to follow security best practices to keep any privileged addresses from being compromised. Additionally, parts of the protocol will need to be enforced offline and some risk management is also done off-chain.
Update: The team has addressed all of the issues.

ID	Description	Severity	Status
QSP-1	Sandwich Attacks	⇈ High	Fixed
QSP-2	Missing Support for Multiple Crops	⇈ High	Fixed
QSP-3	`xBasket` Not Minted to Receiver	⇈ High	Fixed
QSP-4	Incorrect Rewards Accounting for Vested Tokens	⇈ High	Fixed
QSP-5	`rewardSharesPerToken` Is Not Always Updated	⇈ High	Fixed
QSP-6	Identical Token Symbols for cTokens	^ Medium	Fixed
QSP-7	Maximum Allowable Crop Share Not Enforced	^ Medium	Fixed
QSP-8	Farmers Not Guaranteed NFT Return	^ Medium	Acknowledged
QSP-9	Unclear Precision Used	^ Medium	Fixed
QSP-10	Hardcoded Price Used for xToken	⌄ Low	Fixed
QSP-11	Ownership Can Be Renounced	⌄ Low	Fixed
QSP-12	Missing Input Validation	⌄ Low	Fixed
QSP-13	Require Message Deviates From Checked Condition	⌄ Low	Fixed
QSP-14	Inconsistent Return Values for `calculateGrantClaim()`	⌄ Low	Fixed
QSP-15	Missing Address Validation	⌄ Low	Mitigated
QSP-16	`totalAssets` Does Not Account for Potential Yield	○ Informational	Fixed
QSP-17	No Error Handling for Failed Grain Price Assignment	○ Informational	Fixed
QSP-18	Untraceable Contract Events	○ Informational	Fixed
QSP-19	Significant Centralization	○ Informational	Mitigated
QSP-20	Unlocked Pragma	○ Informational	Fixed
QSP-21	Unnecessary Use of `SafeMath` in Solidity 0.8.x	○ Informational	Fixed
QSP-22	Inherited Contract Unused	○ Informational	Fixed
QSP-23	Incorrect Version of Solidity	○ Informational	Fixed
QSP-24	`xBasket` Auto Compound Potentially Vulnerable to Sandwich Attack	? Undetermined	Fixed
QSP-25	`unstakePreview()` Returns Inaccurate Values	? Undetermined	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.

DISCLAIMER:
The audit was performed on the following files only:

- `contracts/*`

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 v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Sandwich Attacks

Severity: *High Risk*

Status: Fixed

File(s) affected: `xToken.sol`

Description: A common attack in DeFi is the sandwich attack. Upon observing a trade of asset X for asset Y, an attacker front-runs the victim trade by also buying asset Y, lets the victim execute the trade, and then executes another trade after the victim by trading back the amount gained in the first trade. Intuitively, one uses the knowledge that someone is going to buy an asset, and that this trade will increase its price, to make a profit. The attacker's plan is to buy this asset cheaply, let the victim buy at an increased price, and then sell the received amount again at a higher price afterward.
The `getShards()` function of `xToken` converts both the shards mint fee and the `xTokensAnnualRent` from `xToken` to USDC through a Uniswap router. The `amountOutMinimum` and `sqrtPriceLimitX96` variables specified in the `ExactInputSingleParams` required for the Uniswap swap are set to 1, and 0 respectively, which will execute the swap even if there is one USDC returned to the user or a price of zero for the `xToken`. A savvy user could front-run these swaps to minimize the USDC returned to the `xToken` contract following the swap and profit off of

the price change. This issue is also present in `xBasket.autoCompoundRewards()`, except they would be front-running USDC to `xToken` swaps.

Exploit Scenario:

1. Attacker sees a transaction containing `getShards()` in the pending transaction pool.
2. Attacker swaps a large amount of the corresponding `xToken` for USDC in the target pool, significantly decreasing the price of the `xToken`.
3. `convertToUsdc()` in the `getShards()` function receives a lot less USDC from the corresponding `xToken` swap. Additionally, this swap further decreases the price of `xToken`.
4. Attacker swaps USDC for `xToken` and receives more `xTokens` than they begin with due to the price decrease.

Recommendation: Sandwich attacks are hard to prevent reliably without a user-defined minimum output amount or an oracle-computed price. It's recommended to keep the trade order size low relative to the pool's liquidity to make such attacks economically less attractive, or only use pools for highly liquid tokens.
Limit `amountOutMinimum` and `sqrtPriceLimitX96` to a reasonable value to minimize slippage during the Uniswap swaps. Additionally, consider using a TWAP pricing mechanism instead of spot price.
Consider adding a parameter to `getShards()` that denotes a fair price for xTokens, e.g. 99% of the current market value. This value should then be passed to the `convertToUsdc()` function and used to compute a slippage-protected token amount out.

Update: The team now calculates a time-weighted average price across the previous hour. We suggest that the `twapInterval` be made configurable should a different interval be required in the future.

QSP-2 Missing Support for Multiple Crops

Severity: *High Risk*

Status: Fixed

File(s) affected: `xToken.sol`, `xBasket.sol`

Description: The `crop` state variable in `xToken` is set to the constant value, "CORN". The `xToken` constructor is designed to receive a value for the string of the crop being created, however, this is just used for the name of the ERC20 token, and not for the assignment of the `crop` state variable. This protocol is intended to work around multiple crops (e.g. xSOY, xWHEAT) however this constant state variable does not allow this as any reading of this state will read "CORN". Further, `xBasket` only considers four predetermined crops. While it is understood that these are the four crops that the `xBasket` will be beginning with, the team should have the functionality to add or remove crops used in the `xBasket` as needed.

Recommendation: The `xToken.crop` state variable should be assigned during construction so that a variety of crops can be supported as the platform intends. Additionally, the team should have the functionality to modify the crops used in `xBasket`.

Update: The `crop` variable is now assigned in the constructor. The team has decided to restrict `xBasket` to four crops and said that there are currently no plans to add additional crops to the protocol.

QSP-3 `xBasket` Not Minted to Receiver

Severity: *High Risk*

Status: Fixed

File(s) affected: `xBasket.sol`

Description: The specified receiver of the `_deposit()` function in `xBasket` never receives the minted xBasket tokens as intended. Additionally, the `deposit()` and `mint()` functions check whether the receiver is able to mint/deposit the requested amount whereas the funds are taken from the `msg.sender`.

Recommendation: `xBasket` should be minted to the address specified as the `receiver` when `_deposit()` is called, not `_msgSender()`. Update the require statements in `mint()` and `deposit()` to check the `msg.sender`.

Update: Tokens are now minted to the `receiver`.

QSP-4 Incorrect Rewards Accounting for Vested Tokens

Severity: *High Risk*

Status: Fixed

File(s) affected: `LNDX.sol`

Description: The `LNDX` contract uses the global variables `rewardSharesPerToken` and `feeSharesPerToken` to track the amount of rewards and fees to distribute on a per token basis. To track how many rewards each user should receive, the contract stores the amount the user is not entitled to in the mappings `rewardsPerGrant` and `feePerGrant`. The simplified computation for the rewards of a user is then `amountOfTokens * rewardSharesPerToken - rewardsPerGrant`. Whenever a user claims rewards, `rewardsPerGrant` will therefore need to be updated to exclude the rewards that have just been claimed by adding them to `rewardsPerGrant`. The code however subtracts the claimed rewards from `rewardsPerGrant` (L198), which means the user will be able to claim even more tokens with each claim. Similarly, the code subtracts the fee that has been distributed from `feePerGrant` instead of adding it (L192). Additionally, the amount used is likely wrong since all tokens - including unvested ones - are used when computing the rewards.

Exploit Scenario:

1. Alice is granted 10 `LNDX` through the `grantLNDX()` function over four years. She therefore receives 10 `veLNDX`. The `rewardSharesPerToken` is equal to 100 at the time of the grant, which means `rewardsPerGrant` is initialized with 100 * 10 = 1000 (ignoring the precision for simplicity).
2. After the first tokens vested, Alice calls `claimVestedTokens()`. Time has passed to vest just 1 token. We assume `rewardsToDistribute()` updates `rewardsPerShare` from 100 to 200. The amount of `totalRewards` for Alice is therefore `veTokens` * `rewardSharesPerToken` - `rewardsPerGrant`, i.e. 10 * 200 - 1000 = 1000. The rewards are then 1 * 1000 / 10 = 100, which is then subtracted from `rewardsPerGrant`.
3. After another token vested, Alice calls `claimVestedTokens()` again. Assuming `rewardsToDistribute()` updates `rewardsPerShare` from 200 to 300, the amount of `totalRewards` for Alice is now 10 * 300 - 900 = 2100, even though it should be 1000 again.

Recommendation: Fix the code by adding the claimed rewards to `rewardsPerGrant` and `feePerGrant` instead of subtracting them. Additionally, `rewards` should be a subset of `totalRewards` proportional to the percentage of the grant claimed.

Update: `claimVestedTokens()` now correctly calculates a user's rewards and guarantees them their full set of rewards by the time the grant concludes.

QSP-5 `rewardSharesPerToken` Is Not Always Updated

Severity: *High Risk*

Status: Fixed

File(s) affected: `LNDX.sol`

Description: When `rewardsToDistribute()` is called, it mints the vested tokens and increments the `rewardSharesPerToken`. However, the `rewardSharesPerToken` is not incremented if the vesting has fully completed at the time of the function call, as seen in the following code segment on L245:

```
if (elapsedDays >= rewardVestingDuration) {
    amountVested = MAX_REWARD_AMOUNT.sub(rewardVested.amountVested);
    _mint(address(this), amountVested);
    rewardVested.amountVested += amountVested;
    rewardVested.lastVestedAt = block.timestamp;
}
```

Recommendation: Update the code segment to properly update the `rewardSharesPerToken`.

Update: `rewardSharesPerToken` is now updated in all cases.

QSP-6 Identical Token Symbols for cTokens

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `cToken.sol`

Description: The `cToken` constructor instantiates the ERC20 symbol of all cTokens created as "cCorn". This creates cTokens with identical symbols.

Recommendation: Use the `_crop` value passed to the constructor in the ERC20 constructor.

Update: The ERC20 symbol is now based on the `_crop` variable.

QSP-7 Maximum Allowable Crop Share Not Enforced

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `nft.sol`

Description: The maximum allowable crop share value stored in `KeyProtocolVariables` is unused when minting a new LandXNFT. With nothing limiting the assignable value of the crop shares, LandXNFTs can be minted with crop share values that exceed the `maxAllowableCropShare` for the given crop as specified in `KeyProtocolVariables`.

Recommendation: When minting a new LandXNFT, ensure that the crop share does not exceed the corresponding `maxAllowableCropShare` value stored in `KeyProtocolVariables`. If the team no longer needs to enforce this maximum share, then documentation should be updated accordingly.

Update: The variable `maxAllowableCropShare` has been removed.

QSP-8 Farmers Not Guaranteed NFT Return

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `xToken.sol`

Description: When Farmers first deposited the LandXNFT, they received less shards than the NFT was valued at in xTokens, based on the deduction of the fees and yearly rent. Farmers who wish to get their NFT returned are required to have an xToken balance of at least the original value of calculated shards. It is understood that that shard amount represents the underlying value of the NFT, however, in a situation where a lot of farmers are looking to buy back xTokens to meet the balance requirement of getting their NFT returned, there could be more xTokens locked in protocol accounts than are available for sale to Farmers.

Recommendation: Provide alternative methods for farmers to retrieve a deposited NFT in case of insufficient xToken liquidity.

Update: The team has mitigated the issue by reducing the number of `xTokens` a farmer needs to supply themselves to return an NFT by burning the farmer's security deposit and using their surplus rent to buy and burn `xTokens`.
They also stated the following: "It is not possible to provide an alternative method to retrieve a deposited NFT because this would lead to a yield bearing asset in circulating supply without any backing. xTokens taken as fees are provided to the Uniswap pool where they are made available at market value. In the example of limited supply the price would appreciate causing an incentive for users to make more xTokens available in accordance with market principles."

QSP-9 Unclear Precision Used

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `rentFoundation.sol`, `LNDX.sol`, `OraclePrices.sol`, `xBasket.sol`, `xToken.sol`

Description: The contracts use different precision values throughout the code base without properly documenting them. Constants such as `10 ** 3`, `1e6`, `10 ** 7` and `10**9` are just some of the constants that can be found in the code. Without documentation, it is impossible to verify that the correct precision is used everywhere.

Recommendation: Clearly document assumptions about expected precisions with comments in the code.

Update: Precision values are now documented by inline comments.

QSP-10 Hardcoded Price Used for xToken

Severity: *Low Risk*

Status: Fixed

File(s) affected: `OraclePrices.sol`

Description: While it may be acceptable to use hardcoded prices pre-launch, these prices should not be used following the LandX launch. If the Uniswap pool for the queried xToken does not exist, these hardcoded values are used.

Recommendation: Following the launch, only prices from the Uniswap pool should be used.

Update: `getXTokenPrice()` now reverts if no pool is found after the protocol has launched.

QSP-11 Ownership Can Be Renounced

Severity: *Low Risk*

Status: Fixed

File(s) affected: `cToken.sol`, `KeyProtocolVariables.sol`, `LNDX.sol`, `OraclePrices.sol`, `rentFoundation.sol`, `xBasket.sol`, `xToken.sol`, `xTokenRouter.sol`

Description: If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be able to be executed.

Recommendation: Confirm whether or not this is the desired functionality. If not, disable `renounceOwnership()` so that the contract always has an owner.

Update: The `renounceOwnership()` function has been overridden to revert if called, maintaining the ownership role.

QSP-12 Missing Input Validation

Severity: *Low Risk*

Status: Fixed

File(s) affected: `KeyProtocolVariables.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. The functions `updateXTokenMintFee()`, `updateCTokensSellFee()`, `updatePayRentFee()`, `updateMaxAllowableCropShare()`, `updateHedgeFundAllocation()`, `updateSecurityDepositMonths()` in `KeyProtocolVariables` do not impose bounds on the values being set. This allows for fees over 100% to be assigned which would result in the loss of user funds.
A non-exhaustive list of functions missing validation includes:

- `KeyProtocolVariables`: Confirm that the following functions update variables to reasonable values (Ex. confirm that percentages cannot be above 100%).
  - `updateXTokenMintFee()`
  - `updateCTokenSellFee()`
  - `updatePayRentFee()`
  - `updateHedgeFundAllocation()`
- `nft.setBaseURI()`: Validate that `newuri` is not an empty string.
- `OraclePrices.setGrainPrice()`: Confirm that the grain exists.
- `OraclePrices.setXTokenPrice()`: Confirm that the xToken exists.
- `xToken.preview()`: Perform the same require statements performed in `getShards()`.

Recommendation: We recommend adding the relevant checks. Impose reasonable bounds on the values being assigned in these functions.

Update: The input validation is now present for all cases except for `OraclePrices`. The client provided the following explanation: "`OraclePrices.setGrainPrice()`, `OraclePrices.setXTokenPrice()` - no reason to add check to confirm grain exists because these methods can be used for adding prices for new grains/`xTokens` (example `COFFEE`/`xCOFFEE`)."

QSP-13 Require Message Deviates From Checked Condition

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LNDX.sol`

Description: The function `grantLNDX()` checks that the `cliffInMonths` argument is less than five years when the require message states that it should be less than one year.

Recommendation: Clarify whether the bound is one or five years and either adjust the condition or the message accordingly.

Update: The message has been fixed to align with the requirement.

QSP-14 Inconsistent Return Values for `calculateGrantClaim()`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LNDX.sol`

Description: `calculateGrantClaim()` currently behaves inconsistently. If the entire vesting duration has elapsed, it returns a tuple of the full vesting duration and any unclaimed rewards. If only part of the vesting duration has elapsed, it returns a tuple of the number of vested but unclaimed days, and the number of unclaimed rewards. An excerpt of the function is included below:

```
// Check cliff was reached
uint256 elapsedDays = block.timestamp.sub(grant.startTime - 1 days).div(
    1 days
);

// If over vesting duration, all tokens vested
if (elapsedDays >= grant.vestingDuration) {
    uint256 remainingGrant = grant.amount.sub(grant.totalClaimed);
    return (grant.vestingDuration, remainingGrant); // <- Returns full vesting duration (of which only a subset may be unclaimed)
} else {
    uint16 daysVested = uint16(elapsedDays.sub(grant.daysClaimed));
    uint256 amountVestedPerDay = grant.amount.div(
        uint256(grant.vestingDuration)
```

```
    );
    uint256 amountVested = uint256(daysVested.mul(amountVestedPerDay));
    return (daysVested, amountVested); // <- returns only the unclaimed vested days
}
```

Recommendation: Update `calculateGrantClaim()` to either always return the number of unclaimed but vested days or the number of vested days.

Update: `calculateGrantClaim()` now always returns the number of unclaimed but vested days.

QSP-15 Missing Address Validation

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `cToken.sol`, `KeyProtocolVariables.sol`, `LNDX.sol`, `LNDXGovernor.sol`, `OracleMulti.sol`, `OraclePrices.sol`, `rentFoundation.sol`, `xBasket.sol`, `xToken.sol`, `xTokenRouter.sol`

Description: Some functions do not validate their input addresses to be non-zero, which can result in unexpected behavior by the contracts. A non-exhaustive list includes:

- `cToken.constructor()`
- `cToken.setRentFoundation()`
- `cToken.setXTokenRouter()`
- `KeyProtocolVariables.constructor()`
- `KeyProtocolVariables.updateHedgeFundWallet()`
- `KeyProtocolVariables.updateLandxOperationalWallet()`
- `KeyProtocolVariables.updateLandxChoiceWallet()`
- `KeyProtocolVariables.updateXTokensSecurityWallet()`
- `KeyProtocolVariables.updateValidatorCommisionWallet()`
- `LNDX.constructor()`
- `LNDXGovernor.constructor()`
- `nft.constructor()`
- `nft.setXTokenRouter()`
- `OracleMulti.updateOraclePrices()`
- `OraclePrices.constructor()`
- `RentFoundation.constructor()`
- `RentFoundation.setXTokenRouter()`
- `RentFoundation.setGrainPrices()`
- `RentFoundation.changeLandXNFTAddress()`
- `xBasket.constructor()`
- `xToken.constructor()`
- `xToken.changeLandXNFTAddress()`
- `xToken.changeXBasketAddress()`
- `xToken.setRentFoundation()`
- `xToken.setOraclePrices()`
- `xToken.setXTokenRouter()`
- `xTokenRouter.setToken()`

Recommendation: Add the missing address validation.

Update: Most addresses are now checked, but a few remain unchecked:

- `_dao` in `KeyProtocolVariables.constructor()`
- `_xTokenRouter` and `_keyProtocolValues` in `LandXNFT.constructor()`.
- `_quoter` in `xBasket.constructor()`.
- `_oraclePrices` in `xToken.constructor()`.
- `_quoter` and `_uniswapRouter`in `xToken.updateUniswapContracts()`.

QSP-16 `totalAssets` Does Not Account for Potential Yield

Severity: *Informational*

Status: Fixed

File(s) affected: `xBasket.sol`

Description: EIP-4626 specifies the following requirement for `totalAssets()`:

> SHOULD include any compounding that occurs from yield.

However, the current implementation of `totalAssets()` does not account for the potential yield.

Recommendation: Update `totalAssets()` to account for the yield.

Update: `totalAssets()` now calls `calculateTVL()` which accounts for yield.

QSP-17 No Error Handling for Failed Grain Price Assignment

Severity: *Informational*

Status: Fixed

File(s) affected: `OracleMulti.sol`

Description: There is no error handling in the case that `oraclePrices.setGrainPrice()` fails in the Chainlink fulfill functions found in `OracleMulti`. Setting the grain price will fail if an invalid price was returned from the Chainlink request and the contract should react accordingly.

Recommendation: Emit an event for listeners to see whether the price assignment failed or succeeded.

Update: An event signaling a failed price assignment is now emitted.

QSP-18 Untraceable Contract Events

Severity: *Informational*

Status: Fixed

File(s) affected: `xToken.sol`, `xBasket.sol`

Description: Staking and unstaking xTokens, as well as auto-compounding xBasket rewards, does not emit events and is therefore untraceable. This inhibits off-chain monitoring of these events.

Recommendation: Emit events in the `xToken.stake()`, `xToken.unstake()`, `xToken.claim()`, and `xBasket.autoCompoundRewards()` functions.

Update: The events have been included as suggested.

QSP-19 Significant Centralization

Severity: *Informational*

Status: Mitigated

Description: The owner of the `xToken` contract has substantial power by being able to modify the addresses of the `LandX`, `xBasket`, `RentFoundation`, and `oraclePrices` contracts being used for the functionality of `xToken`. Substantial trust in the owner is required for this contract to function as intended because the protocol depends on accurate real-world commodity data.

Recommendation: This centralization of power needs to be made clearer to users. Additionally, we recommend the team move towards a more decentralized model for modifying key contracts of the protocol and use a multi-sig wallet if one is not currently being used.

Update: The team has removed all setters except for the `xBasket` address, mentioning: "`xBasket` address should be changeable because it is deployed after `xTokens`. We have plans that we will use multi-sig wallet as owner of protocol contracts."
Though `xBasket` will be deployed following `xTokens`, this restriction can be circumvented by predetermining the address.

QSP-20 Unlocked 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 (^)0.8.*`. 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".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

Update: The contracts are locked to use version `0.8.16`.

QSP-21 Unnecessary Use of `SafeMath` in Solidity 0.8.x

Severity: *Informational*

Status: Fixed

File(s) affected: `LNDX.sol`

Description: Solidity 0.8.x has a built-in mechanism for dealing with overflows and underflows. There is no need to use the `SafeMath` library (it only increases gas usage).

Recommendation: We recommend against using `SafeMath` in Solidity 0.8.x.

Update: The `SafeMath` library has been removed.

QSP-22 Inherited Contract Unused

Severity: *Informational*

Status: Fixed

File(s) affected: `OraclePrices.sol`

Description: The `OraclePrices` contract inherits from both `Ownable` and `AccessControlEnumerable`. `Ownable` is unused and provides only a subset of the `AccessControlEnumerable` functionality and can thus be removed.

Recommendation: Remove `Ownable` from the `OraclePrices` contract.

Update: The unused contracts have been removed.

QSP-23 Incorrect Version of Solidity

Severity: *Informational*

Status: Fixed

Description: The contracts currently use solidity version `0.8.6`. Due to a medium severity bug in the solidity compiler since version `0.5.8`, this version is not in the list of recommended versions.

Recommendation: Consider using solidity version `0.8.16` instead and refer to the list of recommended versions for up to date suggestions.

Update: The contracts now use version `0.8.16`.

QSP-24 `xBasket` Auto Compound Potentially Vulnerable to Sandwich Attack

Severity: *Undetermined*

Status: Fixed

File(s) affected: `xBasket.sol`

Description: Similar to the `xToken` sandwich attack issue, the `xBasket` also converts tokens using the UniswapV3 router without specifying a minimum amount out. Since sandwich attacks are only profitable for larger trades, it is unclear whether the yields from auto compounding are large enough to be at risk.

Recommendation: Create a model for how large the rewards will likely be in practice when auto compounding to determine whether sandwich attacks are a real threat. If it turns out that attacks are feasible, consider adding an array parameter to `autoCompundRewards()`. The array should contain a fair price for each `xToken` that is being bought. Use these prices to compute a minimum amount of tokens out in the `convertToXToken()` function.

Update: The team now calculates a time-weighted average price across the previous hour. We suggest that the `twapInterval` be made configurable should a different interval be required in the future.

QSP-25 `unstakePreview()` Returns Inaccurate Values

Severity: *Undetermined*

Status: Fixed

File(s) affected: `LNDX.sol`

Description: `unstakePreview()` returns a tuple of the staked amount, rewards, and fees for a given `stakeID` if it were to be unstaked. However, the rewards value is not guaranteed to be accurate because `computeStakeReward()` is dependent on `rewardSharesPerToken`, which can be modified by the `rewardsToDistribute()` call in `unstake()`. The `previewUnstake()` function does not take into account possible changes to `rewardSharesPerToken` when `rewardsToDistribute()` is called.

Recommendation: Update `unstakePreview()` to account for possible changes to `rewardSharesPerToken`.

Update: `unstakePreview()` has been updated to account for changes in staking rewards that have taken place over time.

Automated Analyses

Slither

```
LNDX.claimVestedTokens() (contracts/LNDX.sol#174-203) ignores return value by IERC20(usdc).transfer(msg.sender,fee) (contracts/LNDX.sol#200)
LNDX.unstake(uint256) (contracts/LNDX.sol#354-375) ignores return value by IERC20(usdc).transfer(msg.sender,fee) (contracts/LNDX.sol#373)
RentFoundation.payRent(uint256,uint256) (contracts/rentFoundation.sol#94-117) ignores return value by usdc.transfer(keyProtocolValues.hedgeFundWallet(),((amount - platformFee - validatorFee) *
keyProtocolValues.hedgeFundAllocation()) / 10000) (contracts/rentFoundation.sol#103-107)
RentFoundation.payRent(uint256,uint256) (contracts/rentFoundation.sol#94-117) ignores return value by usdc.transfer(keyProtocolValues.validatorCommisionWallet(),validatorFee) (contracts/rentFoundation.sol#108-111)
RentFoundation.sellCToken(address,uint256) (contracts/rentFoundation.sol#146-154) ignores return value by usdc.transfer(account,usdcAmount - cellTokenFee) (contracts/rentFoundation.sol#152)
RentFoundation.feeDistributor(uint256) (contracts/rentFoundation.sol#156-171) ignores return value by usdc.transfer(lndx,lndxFee) (contracts/rentFoundation.sol#161)
RentFoundation.feeDistributor(uint256) (contracts/rentFoundation.sol#156-171) ignores return value by usdc.transfer(keyProtocolValues.landxOperationalWallet(),operationalFee) (contracts/rentFoundation.sol#163-166)
RentFoundation.feeDistributor(uint256) (contracts/rentFoundation.sol#156-171) ignores return value by usdc.transfer(keyProtocolValues.landxChoiceWallet(),_fee - lndxFee - operationalFee) (contracts/rentFoundation.sol#167-
170)
xBasket._withdraw(address,address,address,uint256,uint256) (contracts/xBasket.sol#313-335) ignores return value by IXToken(xWheat).transfer(receiver,assets) (contracts/xBasket.sol#330)
xBasket._withdraw(address,address,address,uint256,uint256) (contracts/xBasket.sol#313-335) ignores return value by IXToken(xSoy).transfer(receiver,assets) (contracts/xBasket.sol#331)
xBasket._withdraw(address,address,address,uint256,uint256) (contracts/xBasket.sol#313-335) ignores return value by IXToken(xRice).transfer(receiver,assets) (contracts/xBasket.sol#332)
xBasket._withdraw(address,address,address,uint256,uint256) (contracts/xBasket.sol#313-335) ignores return value by IXToken(xCorn).transfer(receiver,assets) (contracts/xBasket.sol#333)
XToken.getShards(uint256) (contracts/xToken.sol#143-229) ignores return value by ERC20(usdc).transfer(keyProtocolValues.hedgeFundWallet(),toHedgeFund) (contracts/xToken.sol#207-210)
XToken.getShards(uint256) (contracts/xToken.sol#143-229) ignores return value by ERC20(usdc).transfer(address(rentFoundation),usdcAnnualRent - toHedgeFund) (contracts/xToken.sol#211-214)
XToken.feeDistributor(uint256) (contracts/xToken.sol#283-298) ignores return value by ERC20(usdc).transfer(lndx,lndxFee) (contracts/xToken.sol#288)
XToken.feeDistributor(uint256) (contracts/xToken.sol#283-298) ignores return value by ERC20(usdc).transfer(keyProtocolValues.landxOperationalWallet(),operationalFee) (contracts/xToken.sol#290-293)
XToken.feeDistributor(uint256) (contracts/xToken.sol#283-298) ignores return value by ERC20(usdc).transfer(keyProtocolValues.landxChoiceWallet(),_fee - lndxFee - operationalFee) (contracts/xToken.sol#294-297)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

LNDX.grantLNDX(address,uint256,uint16,uint16) (contracts/LNDX.sol#116-172) performs a multiplication on the result of a division:
    -veLNDXAmount = (amount * coefficient) / 100 (contracts/LNDX.sol#146)
    -rewardsPerGrant[recipient] += (rewardSharesPerToken * veLNDXAmount) / 1e6 (contracts/LNDX.sol#151-153)
LNDX.grantLNDX(address,uint256,uint16,uint16) (contracts/LNDX.sol#116-172) performs a multiplication on the result of a division:
    -veLNDXAmount = (amount * coefficient) / 100 (contracts/LNDX.sol#146)
    -feePerGrant[recipient] += (feeSharesPerToken * veLNDXAmount) / 1e6 (contracts/LNDX.sol#154)
LNDX.claimVestedTokens() (contracts/LNDX.sol#174-203) performs a multiplication on the result of a division:
    -veLNDXAmount = (amountVested * grant.veLndxClaimed) / grant.amount (contracts/LNDX.sol#186-187)
    -fee = (veLNDXAmount * totalFee) / grant.veLndxClaimed (contracts/LNDX.sol#193)
LNDX.claimVestedTokens() (contracts/LNDX.sol#174-203) performs a multiplication on the result of a division:
    -veLNDXAmount = (amountVested * grant.veLndxClaimed) / grant.amount (contracts/LNDX.sol#186-187)
    -rewards = (veLNDXAmount * totalRewards) / grant.veLndxClaimed (contracts/LNDX.sol#197)
LNDX.rewardsToDistribute() (contracts/LNDX.sol#225-274) performs a multiplication on the result of a division:
    -amountVestedPerDay = MAX_REWARD_AMOUNT.div(uint256(rewardVestingDuration)) (contracts/LNDX.sol#255-257)
    -amountVested = uint256(daysVested.mul(amountVestedPerDay)) (contracts/LNDX.sol#258)
LNDX.calculateGrantClaim(address) (contracts/LNDX.sol#276-307) performs a multiplication on the result of a division:
    -amountVestedPerDay = grant.amount.div(uint256(grant.vestingDuration)) (contracts/LNDX.sol#301-303)
    -amountVested = uint256(daysVested.mul(amountVestedPerDay)) (contracts/LNDX.sol#304)
LNDX.stakeLNDX(uint256,LNDX.StakePeriods) (contracts/LNDX.sol#325-352) performs a multiplication on the result of a division:
    -mintAmount = (amount * coefficients[period]) / 100 (contracts/LNDX.sol#328)
    -feePerStake[stakesCount] += (feeSharesPerToken * mintAmount) / 1e6 (contracts/LNDX.sol#347)
LNDX.stakeLNDX(uint256,LNDX.StakePeriods) (contracts/LNDX.sol#325-352) performs a multiplication on the result of a division:
    -mintAmount = (amount * coefficients[period]) / 100 (contracts/LNDX.sol#328)
    -rewardsPerStake[stakesCount] += (rewardSharesPerToken * mintAmount) / 1e6 (contracts/LNDX.sol#348-350)
LandXNFT.uint2str(uint256) (contracts/nft.sol#95-119) performs a multiplication on the result of a division:
    -temp = (48 + uint8(_i - (_i / 10) * 10)) (contracts/nft.sol#113)
RentFoundation.getDepositBalance(uint256) (contracts/rentFoundation.sol#136-144) performs a multiplication on the result of a division:
    -rentPerSecond = (landXNFT.cropShare(tokenID) * landXNFT.tillableArea(tokenID) * 10 ** 3) / delimeter (contracts/rentFoundation.sol#139-140)
    -int256(deposits[tokenID].amount) - int256(rentPerSecond * elapsedSeconds / 10 ** 7) (contracts/rentFoundation.sol#141-143)
RentFoundation.sellCToken(address,uint256) (contracts/rentFoundation.sol#146-154) performs a multiplication on the result of a division:
    -usdcAmount = (amount * grainPrices.prices(crop)) / (10 ** 9) (contracts/rentFoundation.sol#149)
    -cellTokenFee = (usdcAmount * keyProtocolValues.cTokenSellFee()) / 10000 (contracts/rentFoundation.sol#150-151)
XToken.getShards(uint256) (contracts/xToken.sol#143-229) performs a multiplication on the result of a division:
    -xTokensAnnualRent = ((annualRent * oraclePrices.prices(crop)) / oraclePrices.getXTokenPrice(xTokenRouter.getXToken(crop))) * 1e3 (contracts/xToken.sol#185-186)
XToken.getShards(uint256) (contracts/xToken.sol#143-229) performs a multiplication on the result of a division:
    -toSecurityDepositsAmount = (xTokensAnnualRent / 12) * keyProtocolValues.securityDepositMonths() (contracts/xToken.sol#187-188)
XToken.preview(uint256) (contracts/xToken.sol#385-420) performs a multiplication on the result of a division:
    -xTokensAnnualRent = ((annualRent * oraclePrices.prices(crop)) / oraclePrices.getXTokenPrice(xTokenRouter.getXToken(crop))) * 1e3 (contracts/xToken.sol#405-406)
XToken.preview(uint256) (contracts/xToken.sol#385-420) performs a multiplication on the result of a division:
    -toSecurityDepositsAmount = (xTokensAnnualRent / 12) * keyProtocolValues.securityDepositMonths() (contracts/xToken.sol#407-408)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

Code Documentation

1. Litepaper contains an extra period in the second paragraph of the "For Investors" section.

2. The comment above `xToken.getTheNFT()` suggests that shards are being deposited to this contract, rather than being burned.
3. The comment next to the `validator` state variable in `LandXNFT` suggests that validators or landowners can be stored as a validator for a given token, although documentation suggests a validator is a separate, distinct party in the protocol. This ambiguity should not be present in a code comment.
4. The `RentFoundation` portion of the smart contract diagram included in the Technology section of the Whitepaper is very inaccurate as to the functionality found in that contract. For example, there is no minting or burning of xTokens present in this contract.
5. Whitepaper says that the validator commission begins at 1% however the `validatorCommission` state variable in `KeyProtocolVariables` begins at 0.25%. These values should be consistent.
6. Consider using the NatSpec format to document all functions. The current code contains very few comments and would greatly benefit from better documentation.

Adherence to Best Practices

1. All internal functions should begin with an underscore. This is inconsistent throughout the code and the following functions should be changed:
    - `xToken.calculateYield()`
    - `xToken.calculateTotalYield()`
    - `xToken.feeDistributor()`
    - `rentFoundation.feeDistributor()`
    - `xBasket.convertToXToken`
    - `rentFoundation.feeDistributor()`
    - `LNDX.rewardsToDistribute()`
    - `LNDX.calculateEndDate()`
    - `LNDX.calculateGrantClaim()`
2. `rentFoundation.sol` and `nft.sol` should follow the same camel-casing used in other file names. Additionally, interfaces should follow this same naming convention.
3. `nft.sol` should have a more descriptive name as it is the LandXNFT.
4. Both the `landXOpertationsPercentage` and `lndxHoldersPercentage` variable names in `KeyProtocolVariables` contain typos.
5. Contracts should adhere to the following order when declaring members. This is inconsistently followed throughout the codebase:
    1. Type declarations
    2. State variables
    3. Events
    4. Modifiers
    5. Functions
6. The `_keyProtokolValues` parameter in the constructor for `RentFoundation` contains a typo and should be renamed to `_keyProtocolValues`.
7. L150 in `rentFoundation.sol` should rename the instance variable from `cellTokenFee` to `sellTokenFee`.
8. `LNDX.stakesCount:` Assigning the state variable to 0 when it is declared is redundant, as the default value will be 0.
9. There is no need to assign the `crop` state variable in `cToken` as it is assigned in the constructor. As it can only be assigned in the constructor, it should be marked as `immutable`.
10. Events are improperly emitted throughout the contracts. For example, events should be emitted throughout the setters in `KeyProtocolVariables`.
11. The constructor in `cToken` contains a typo. `_rentForndation` should read `_rentFoundation`.
12. `LandXNFT.totalSupply` is redundant since it is only ever 0 or 1 and used to check whether an ID has already been initialized. The `initialOwner` mapping can be used for checking that as well.
13. `LandXNFT.setDetailsAndMint()` calls `_mint(..., "0x0000")` with the last argument `data` set to `"0x0000"`. Since this appears to be unused, it can be replaced by an empty string `""` to save some gas.
14. `OracleMulti.fee` and `OracleMulti.jobId` should be declared as constants.
15. In the `OracleMulti` contract, consider using an immutable storage variable for the request URI and other hardcoded addresses to allow using the same code for tests and deployments.
16. In the `OracleMulti` contract, consider refactoring the `request*` functions into a single function that accepts a string and a function selector.
17. Consider renaming the parameter `_address` of the `OraclePrices.constructor()` to something more specific such as `admin`.
18. The variable `OraclePrices.usdc` uses a hardcoded address with a comment "to be changed for mainnet". Consider using an immutable variable instead to allow using the same code for tests and deployments.
19. `xToken.getShards()#L186` calls `xTokenRouter.getXToken(crop)`. This should be replaced by `address(this)`, since an earlier require statement enforces that the return value must be equal to the current contract's address.
20. Consider replacing the four division operations in the function `xBasket.calculateCollateral()` by a single one to save gas.
21. In the `RentFoundation` contract, consider declaring the variable `usdc` and `lndx` as immutable.
22. Create interfaces in their own files and import them wherever needed.
23. `KeyProtocolVariables.sol`: Set the DAO as the owner and use the `onlyOwner` modifier. The contract is currently `Ownable` but the `onlyOwner` modifier is never used.
24. `RentFoundation.sol`: Set `grainPrices`, `landXNFT`, and `xTokenRouter` in the constructor to avoid making calls before the addresses are set.
25. In the function `LNDX.grantLNDX()`, the assignment of the `coefficient` variable could be optimized by replacing it with the following code:

```
uint8 coefficient;
if (totalPeriod >= 48) {
    coefficient = coefficients[StakePeriods.MONTHS_48];
}
else if (totalPeriod >= 12) {
    coefficient = coefficients[StakePeriods.MONTHS_12];
} else {
    coefficient = coefficients[StakePeriods.MONTHS_3];
}
```

Test Results

Test Suite Results

The test suite was executed by running `npm run test`.

```
  cToken

     ✓ check symbol value

     ✓ owner can't renounceOwnership


mint 1000000 cCORN for 0xBC7B3321581341c91548286CE26B840b0cA55110
     ✓ minting  works (101ms)


try to mint 1000000 cCORN for 0xBC7B3321581341c91548286CE26B840b0cA55110 when caller is not xCORN contract
     ✓ minting doesn't  work (79ms)


get decimal parameter for cToken, it should be equal 6
     ✓ get decimals  works


updates xTokenRouter contract by contract owner
     ✓ set XTokenRouter (67ms)


try to update xTokenRouter contract by NOT contract owner
     ✓ it is not possible to set XTokenRouter (not owner contract) (51ms)

     ✓ it is not possible to set XTokenRouter (zero address)


updates RentFoundationcontract by contract owner
     ✓ set RentFoundation contract (60ms)


try to update RentFoundation contract by NOT contract owner
     ✓ it is not possible to set RentFoundation contract (not owner contract) (52ms)

     ✓ it is not possible to set RentFoundation contract (zero address)


0xBC7B3321581341c91548286CE26B840b0cA55110has 1000000 cCORN and burns 500000, final his balance should be equal 50000
     ✓ burn works (205ms)


0xBC7B3321581341c91548286CE26B840b0cA55110has 1000000 cCORN and burns 500000, for some reason they can't be converted to USDC, final his balance should be equal 1000000
     ✓ burn doesn't work impossible to exchange CTokens (98ms)


0xBC7B3321581341c91548286CE26B840b0cA55110has 1000000 cCORN and burns 2000000, so transaction should be reverted
     ✓ burn doesn't work impossible to exchange CTokens (not enaugh balance) (98ms)

  Key Protocol Values


updates xToken mint fee
     ✓ updateXTokenMintFee works


try to update xToken mint fee when it is not allowed
     ✓ updateXTokenMintFee doesn't work (not dao)

     ✓ updateXTokenMintFee doesn't work (unsuitable value)


updates cToken sell fee
     ✓ updateCTokenSellFee works


try to update cToken sell fee when it is not allowed
     ✓ updateCTokenSellFee doesn't work (not dao)

     ✓ updateCTokenSellFee doesn't work (unsuitable value)


updates pay rent fee
     ✓ updatePayRentFee works


try to update pay rent fee when it is not allowed
     ✓ updatePayRentFee doesn't work (not dao)

     ✓ updatePayRentFee doesn't work (unsuitable value)


updates pay rent fee
     ✓ updateSellXTokenSlippage works


try to update pay rent fee when it is not allowed
     ✓ updateSellXTokenSlippage doesn't work (not dao)

     ✓ updateSellXTokenSlippage doesn't work (unsuitable value)


updates pay rent fee
     ✓ updateBuyXTokenSlippage works


try to update pay rent fee when it is not allowed
     ✓ updateBuyXTokenSlippage doesn't work (not dao)

     ✓ updateBuyXTokenSlippage doesn't work (unsuitable value)


updates hedge fund allocation percentage to 10%
     ✓ updateHedgeFundAllocation works


try to update  hedge fund allocation percentage when it is not allowed
     ✓ updateHedgeFundAllocation doesn't work (not dao)

     ✓ updateHedgeFundAllocation doesn't work (unsuitable value)


updates size of security deposit
     ✓ updateSecurityDepositMonths works


try to update size of security deposit when it is not allowed
     ✓ updateSecurityDepositMonths doesn't work (not dao)


updates fee distribution precentages
     ✓ updateFeeDistributionPercentage works (47ms)


try to update fee distribution precentages when it is not allowed
     ✓ updateFeeDistributionPercentage doesn't work (not dao)


try to update fee distribution precentages when their sum is greater then 100%
     ✓ updateFeeDistributionPercentage doesn't work (inconsistent values)


updates hedge fund wallet
     ✓ updateHedgeFundWallet works


try to update hedge fund wallet when it is not allowed
```

```
      ✓ updateHedgeFundWallet doesn't work (not dao)


      ✓ updateHedgeFundWallet doesn't work (zero address)


updates landx operational wallet
      ✓ updateLandxOperationalWallet works


try to update landx operational wallet when it is not allowed
      ✓ updateLandxOperationalWallet doesn't work (not dao)


      ✓ updateLandxOperationalWallet doesn't work (zero address)


updates landx choice wallet
      ✓ updateLandxChoiceWallet works


try to update landx choice wallet when it is not allowed
      ✓ updateLandxChoiceWallet doesn't work (not dao)


      ✓ updateLandxChoiceWallet doesn't work (zero address)


updates xToken security wallet
      ✓ updateXTokensSecurityWallet works


try to update xToken security wallet when it is not allowed
      ✓ updateXTokensSecurityWallet doesn't work (not dao)


      ✓ updateXTokensSecurityWallet doesn't work (zero address)


updates validator's commission wallet
      ✓ updateValidatorCommisionWallet works


try to update validator's commission wallet when it is not allowed
      ✓ updateValidatorCommisionWallet doesn't work (not dao)


      ✓ updateValidatorCommisionWallet doesn't work (zero address)


updates max validator fee
      ✓ updateMaxValidatorFee works


try to updates max validator fee when it is not allowed
      ✓ updateMaxValidatorFee doesn't work (not dao)


      ✓ updateMaxValidatorFee doesn't work (unsuitable value)


update validator commision fee
      ✓ update validator commision fee works


try to update validator commision fee when it is not allowed
      ✓ update validator commision fee doesn't work (not dao)


      ✓ update validator commision fee doesn't work (unsuitable value)


disable pre launch mode
      ✓ launch works


try to disable pre launch mode when it is not allowed
      ✓ launch doesn't work (not dao)

  LNDX
     ✓ Grant LNDX with cliff=0 and vesting=0 (38ms)
     ✓ Impossible create second grant for the same address (45ms)
     ✓ Impossible create grant with cliff more then 60 months
     ✓ Impossible create grant with vesting duration more then 60 months
     ✓ Impossible create grant with vesting duration more then 60 months
     ✓ Impossible to mint more than 64400000 LNDX
     ✓ Only Owner can create grant
     ✓ Grant LNDX with cliff=2 and vesting=5, total lock < 12 months (92ms)
     ✓ Grant LNDX with cliff=1 and vesting=24, 12 months <= total lock < 48 months (91ms)
     ✓ Grant LNDX with cliff=12 and vesting=36, total lock >= 48 months (93ms)
     ✓ Grant LNDX with cliff=12 and vesting=36, total lock >= 48 months (91ms)
     ✓ Reward distribution first call (69ms)
     ✓ Reward distribution first call, veLNDX totalSupply = 0 (111ms)
     ✓ Reward distribution second call after two days (123ms)
     ✓ Impossible to mint rewards more then 15 600 000 (169ms)
     ✓ Impossible to mint rewards when all rewards already minted (161ms)
     ✓ impossible claim when cliff is not over (120ms)
     ✓ Partial Claim grant (197ms)
     ✓ Full Claim grant (282ms)
     ✓ Try to claim totally claimed grant (256ms)
     ✓ stake, wrong period (51ms)
     ✓ stake (180ms)
     ✓ Unstake impossible, stake period is not finished (182ms)
     ✓ Unstake impossible, caller is not staker (187ms)
     ✓ Unstake (390ms)
     ✓ Unstake (362ms)
     ✓ Unstake Preview (388ms)
     ✓ Unstake Preview (332ms)
     ✓ Unstake impossible, already unstaked (389ms)
     ✓ Unable to distribute fee, caller has no role granted (85ms)
     ✓ owner can't renounceOwnership
     ✓ get decimals  works

  NFT


Minting nft for 0xBC7B3321581341c91548286CE26B840b0cA55110 with parameters
      ✓ minting  works (151ms)


try to mint NFT with id=1, but token with ID=1 already exists
      ✓ minting  doesn't work because token exists (114ms)


try to mint NFT with id=1, but token with ID=1 already exists
      ✓ minting  doesn't work because two high validator's fee (124ms)


try to mint NFT but there is no xToken contract for provided crop type
      ✓ minting  doesn't work because xToken is not exists(not set) (137ms)


NFT owner burns his NFT
      ✓ burning NFT works for NFT owner (149ms)


not NFT owner burns the NFT of other owner if he has allowance
      ✓ burning NFT works for not owner when it approved (171ms)


not NFT owner try to burn the NFT, transaction should be reverted
      ✓ burning NFT doesn't work for not owner (113ms)


set token URI by contract owner
      ✓ set base token URI works


set token URI by not contract owner, transaction should be reverted
      ✓ set base token URI doesn't work


      ✓ set base token URI doesn't work, empty string


updates xTokenRouter contract by contract owner
      ✓ set xTokenRouter works


try to update xTokenRouter contract by NOT contract owner
```

```
     ✓ set xTokenRouter doesn't work


     ✓ set xTokenRouter doesn't work, zero address is not allowed


returns uri for NFT token by token ID
     ✓ get uri (38ms)


try to get uri for NFT token by token ID that has no valid type(35h), should be reverted
     ✓ get uri doesn't work

  Oracle Prices


set price=500000000 (USDC per megatone) for SOY
     ✓ setGrainPrice works  (40ms)


try to set price for SOY by address with no PRICE_SETTER role
     ✓ setGrainPrice doesn't work (has no role)


try to set TOO HIGH price for SOY
     ✓ setGrainPrice doesn't work (invalid value)


set price=6000000 (USDC per megatone) for xToken
     ✓ setXTokenPrice works


try to set price for xToken by address with no PRICE_SETTER role
     ✓ setXTokenPricee doesn't work (has no role)


try to set TOO HIGH price for xToken
     ✓ setXTokenPrice doesn't work (invalid value)


     ✓ setXTokenPrice doesn't work, zero address (54ms)


get xToken price when prelaunch mode is enabled
     ✓ get xToken price (prelauch is true) (52ms)


try to get xToken price when pre launch mode is disabled and uniswap pool is not found; in this case transaction is reverted
     ✓ get xToken price (prelauch is false, pool not found) (264ms)


get xToken price when prelaunch mode is disabled, uniswap pool exists and token0 is USDC
     ✓ get xToken price (prelauch is false, pool exists, token0 is usdc) (280ms)


get xToken price when prelaunch mode is disabled, uniswap pool exists and token0 is xToken
     ✓ get xToken price (prelauch is false, pool exists, token0 is xToken) (281ms)


returns xToken/USDC uniswap pool address
     ✓ get xToken pool (151ms)

  RentFoundation


     ✓ owner can't renounceOwnership


updates NFT contract by contract owner
     ✓ set NFT Contract (103ms)


try to update NFT contract by NOT contract owner
     ✓ impossible set NFT Contract (55ms)


     ✓ impossible set NFT Contract, zero address


updates xTokenRouter contract by contract owner
     ✓ set XTokenRouter (59ms)


try to update xTokenRouter contract by NOT contract owner
     ✓ impossible to set XTokenRouter (52ms)


     ✓ impossible to set xTokenRouter, zero address


updates OraclePrice contract by contract owner
     ✓ set OraclePrices contract (60ms)


try to update OraclePrice contract by NOT contract owner
     ✓ it is not possible to set OraclePrices contract (not owner contract) (53ms)


     ✓ it is not possible to set OraclePrices contract, zero address


pay initial rent for NFT with ID=1
     ✓ pay initial rent (128ms)


try to pay initial rent for NFT with ID=1 by not initial payer, only xToken contract can pay initial rent
     ✓ pay initial rent, not initial payer (114ms)


try to pay initial rent for NFT with ID=1 but initial rent was already applied, initial rent can be paid once
     ✓ pay initial rent, initial rent is applied before (364ms)


try to pay rent for NFT with ID=1 but initial rent was not applied, it is impossible to pay rent for NFT that was not converted to xTokens
     ✓ pay rent, initial rent was not applied


try to pay 1000000 USDC of rent for NFT with ID=1 but account has not enaough of USDC
     ✓ pay rent, not enough USDC to transfer (276ms)


pay 1000000 USDC of rent for NFT with ID=1
     ✓ pay rent (1050ms)


pay 1000000 USDC of rent for NFT with ID=1
     ✓ pay rent (1050ms)


     ✓ pay rent, security deposit (1154ms)


     ✓ impossible pay rent, security deposit (1137ms)


     ✓ Buy Out reverts (1206ms)


     ✓ Buy Out, reverts not initial payer (163ms)


     ✓ Buy Out, reverts not initial rent applied (68ms)


     ✓ Buy Out (1253ms)


     ✓ Buy Out Preview, reverts not initial payer (143ms)


     ✓ Buy Out, reverts not initial rent applied (64ms)


     ✓ Buy Out Preview (1181ms)


     ✓ Buy Out Preview (1217ms)


get deposit amount for NFT with ID=1:, it become less each second: start deposit is 81000, after 100000 seconds it becomes 80744
     ✓ get deposit balance (221ms)


get deposit amount for NFT with ID=1:, it become less each second: can be negative, start deposit is 81000, after 32336000 seconds it becomes -2051
```

```
    ✓ get deposit balance, negative value if there wasn't payments (257ms)


    ✓ sell cTokens (833ms)


    ✓ can't sell cTokens, no valid cToken (224ms)

  xBasket
    ✓ owner can't renounceOwnership
    ✓ get decimals  works
    ✓ Asset
    ✓ convert to shares, total supply 0 (50ms)
    ✓ convert to assets, total supply 0 (51ms)
    ✓ deposit, total supply 0 (1010ms)
    ✓ deposit, total supply > 0 (1108ms)
    ✓ deposit, total supply > 0, not enough assets (1020ms)
    ✓ mint, total supply 0 (1050ms)
    ✓ mint, total supply > 0 (2033ms)
    ✓ mint, total supply > 0, not enough fund (1202ms)
    ✓ withdraw (4224ms)
    ✓ get xBasket price (1913ms)
    ✓ redeem (4212ms)
    ✓ can't redeem, ERC4626: redeem more than max (4002ms)
    ✓ get total assets (1617ms)

  xToken
    ✓ check symbol value
    ✓ owner can't renounceOwnership
    ✓ Get Shards works (preLaunch mode is enabled) (751ms)
    ✓ Get Shards works (preLaunch mode is disabled) (1769ms)
    ✓ impossible to get Shards (not initial owner) (1141ms)
    ✓ impossible to get Shards (nft has no land area) (75ms)
    ✓ impossible to get Shards (nft has no rent) (67ms)
    ✓ impossible to get Shards (unsupported grain) (154ms)
    ✓ impossible to get Shards (wrong xToken contract) (110ms)
    ✓ impossible to get Shards (not token owner) (259ms)
    ✓ impossible to get Shards (rent already applied) (1095ms)
    ✓ Get NFT back (1204ms)
    ✓ Get NFT back, security deposit is applied (1134ms)
    ✓ Get NFT back, remaining rent is 0 (1108ms)
    ✓ Cant' get NFT back, there is a debt (990ms)
    ✓ Get NFT back Preview (946ms)
    ✓ Get NFT back Preview, security deposit applied (905ms)
    ✓ Get NFT back Preview, there is a debt (802ms)
    ✓ Can't get NFT back (not owner) (797ms)
    ✓ Stake works (787ms)
    ✓ Preview not distributed yield (808ms)
    ✓ Get not distributed yield reverted
    ✓ Additional Stake works (821ms)
    ✓ Stake doesn't work (not enough funds) (88ms)
    ✓ Unstake works (996ms)
    ✓ get decimals  works
    ✓ set XTokenRouter (81ms)
    ✓ impossible to set XTokenRouter (75ms)
    ✓ impossible to set XTokenRouter, zero address
    ✓ set XBasketAddress (79ms)
    ✓ impossible to set XBasketAddress (75ms)
    ✓ impossible to set XBasketAddresst, zero address
    ✓ preview (425ms)
    ✓ preview reverted (55ms)
    ✓ preview reverted (54ms)
    ✓ preview reverted (140ms)
    ✓ preview reverted (101ms)
    ✓ XBasket Transfer (720ms)
    ✓ XBasket Transfer not allowed (752ms)

  xTokenRouter
    ✓ owner can't renounceOwnership
    ✓ Set Tokens works (39ms)
    ✓ not owner can't set Tokens
    ✓ owner can't set Tokens with zero adress
    ✓ owner can't set Tokens with zero adress
    ✓ get cToken


  210 passing (3m)
```

Code Coverage

The code coverage was gathered by running `npm run test`. The code coverage is strong overall, with room for improvement for `RentFoundation.sol`.

The following files were not instrumented for coverage, and thus their coverage cannot be assessed:

- `LNDXGovernor.sol`
- `OracleMulti.sol`
- `usdc.sol`
- `veLNDX.sol`

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	95.61	83.96	95.49	95.19	
KeyProtocolVariables.sol	100	92.86	100	100	
LNDX.sol	97.56	84.62	100	94.94	… 397,398,457
LandXNFT.sol	91.67	95	85.71	96.3	100
OraclePrices.sol	100	86.36	100	100	
RentFoundation.sol	79.45	75	70.59	80.41	… 232,236,240
cToken.sol	100	81.25	100	100	
xBasket.sol	98.7	67.86	100	98.33	283,315,409
xToken.sol	97.1	84.72	100	97.16	… 482,483,484
xTokenRouter.sol	100	87.5	100	100	
All files	**95.61**	**83.96**	**95.49**	**95.19**	

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

```
13715e36515dcf52b695a58d8ccd74605288337c2d59f035873748abf18ec2b7 ./contracts/xBasket.sol
850b5d6fd6b5fa551b48174bab7b100ae0c984260fef07fcf2a9d5b165d3019e ./contracts/LNDX.sol
5eca6f74c6458875b6d6435f246e7f4185cc1d6e610a59ea9fe8e65d7587d6bf ./contracts/veLNDX.sol
91429a7bb02ddc64d6300b8d67df724c30ed5e66883bebb7e2ec22f21a9e15a7 ./contracts/OraclePrices.sol
7ba9a2496fc63c4f3a135464d7830b8f4e222f947fc0c29c6c5117e786c995a7 ./contracts/cToken.sol
74db4948c161fadec8f4b2a5b12fb3eabad3c0dc744886e258df9c4519d9a5ef ./contracts/rentFoundation.sol
b1bb00a7df4c67bde61eab411ce89fbe2955b0f38852634efe37a8724b438059 ./contracts/usdc.sol
c26a9022d7b218c8f6f3c84ffb59085722e764b8c947db27c61efee9c5f1b57d ./contracts/OracleMulti.sol
cb78d2e2d81fcc181fd44478e1cd76454b20818bcd9e42afc6d597a4adab96e7 ./contracts/nft.sol
e6cd6db01e2ef414289713e21d53dea84e4ae11dfb7a131c3bcde15f715acf05 ./contracts/KeyProtocolVariables.sol
94b7ae16a90a194dc082a0b2289e36f5dd9919e55ecae5dbe50afcfad2323041 ./contracts/LNDXGovernor.sol
030375d67fcbe2cb01d33ca5a398c9cfe8d403e0282531f21384b47d950a0b0e ./contracts/xToken.sol
b4fd960deede8265fda239a8e80c8ea806015f600f9bb7d45a382b19ff2cf7df ./contracts/xTokenRouter.sol
```

Tests

```
8afed033b44f5d3f76b891d5071c20d122492598f6cdf9f3dab5657e47b0230d ./test/xtoken_test.js
965da23fae35a517c5b7eaf1daa81f4548fac7b042105cc4639b1e852e950e3a ./test/xBasket_test.js
abfc188691e86275e5b7e184bd551ec5cb5145a48c27d17919338cdd40a45052 ./test/oraclePrices_test.js
7836cbb71ef3d8ef93125c7dc9555d50ad1287d663dc465f6f1180d1748ddc8b ./test/nft_test.js
edffedba6e4cbfccfc3d472c9cb18cdd2528d1a5d7a4dca40edd1b70aabb2cc2 ./test/xTokenRouter_test.js
492441d6e6fed129cc0003c7a7f4fa6dee6c8f8ac92534d7158c79de99ed318e ./test/ctoken_test.js
87b15678f65ebe1bd6a17150693d13baae1e6e0b7727ea7b99178737ad87ef21 ./test/keyProtocolValues_test.js
da53e9b232cf1bf0f16265426fe65ff805f4660b26e2876a15e451d91596b872 ./test/rentFoundation_test.js
```

Changelog

- 2022-11-10 - Initial report
- 2023-01-20 - Fix Review

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Aave, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.