ChickenBonds Updates

Smart Contract Audit

coinspect

Liquity

ChickenBond

Zap and Artwork updates

Smart Contract Audit

V221021 Prepared for Liquity • October 2022

1. Executive Summary

In **October 2022, Liquity** engaged Coinspect to perform a source code review of changes introduced to the **ChickenBond** smart contracts. The objective of the project was to evaluate the security of a specific set of new smart contracts that implement the bLUSD Curve Zap and the dynamic NFTs artwork features.

This review focused on the ChickenBonds NFT Artwork's implementation in terms of how its rarity is calculated and how the bond's information is preserved during the NFT life cycle. Also, Coinspect reviewed the implementation of the bLUSD Curve Zap to be used as the main ChickenBond's AMM.

Liquity's team provided Coinspect with extensive documentation regarding the new features, including specs and tests scripts that aided the audit.

The following issues were identified during the initial assessment:

High Risk	Medium Risk	Low Risk
Open **0**	Open **0**	Open **2**
Fixed 0	Fixed 0	Fixed 0
Reported 0	Reported 0	Reported 2

Coinspect found one low-risk issue regarding how marketplaces could show outdated token's metadata which could deceive traders making them purchase an NFT that already changed its form. Also, Coinspect found one potential issue related to the NFT artwork DNA which depends on the block timestamp, which could be elected to obtain rare items. This possibility was acknowledged by the dev team which considered it unlikely to be exploited as detailed below in the report.

2. Assessment and Scope

The audit started on **October 13, 2022** and was conducted on the **main** branch of the git repository at https://github.com/liquity/ChickenBond/ as of commit **128e52f14ca1874daa1e31e522c2a309cde05c7d** of **October 17, 2022**.

Coinspect reviewed the changes introduced to support two new features.

The audited pull requests are the following:

1. **BLUSD LP ZAP:**
    a. https://github.com/liquity/ChickenBond/pull/244
    b. https://github.com/liquity/ChickenBond/pull/245
    c. https://github.com/liquity/ChickenBond/pull/251
2. **ChickenBonds NFT Artwork:**
    a. https://github.com/liquity/ChickenBond/pull/246

BLUSD LP ZAP

The `BLUSDLPZAP` contract handles user's swaps between the ChickenBonds internal AMM and a Curve pair of `LUSD-3CRV` for `bLUSD` which allows users to stake, withdraw funds, provide liquidity and perform token bLUSD/LUSD swaps.

It was an easy to read and understandable contract with clear comments across the codebase explaining each step performed inside functions. However, it lacks complete documentation and NatSpec for the contract, public variables, and external functions.

A non implemented permit feature is under a pending "`TODO`" comment. It is worth noting that its implementation may require further revisions as it is not part of this audit. Also, a commented-forge console import is currently in the codebase which could be removed before going to the production phase in order to make the code more tidy.

The mentioned contract has four external functions that allow users to add liquidity, stake, remove liquidity both balanced and in LUSD which write the state of the EVM, three read only functions and one internal function.

The contract interacts with the following third party contracts that were not part part of this audit:

1) LUSD Token: 0x5f98805A4E8be255a32880FDeC7F6728C6568bA0
2) BLUSD Token: 0xB9D7DdDca9a4AC480991865EfEf82E01273F79C3
3) LUSD-3CRV Pool: 0xEd279fDD11cA84bEef15AF5D39BB4d4bEE23F0cA
4) BLUSD-3CRV Pool: 0x74ED5d42203806c8CDCf2F04Ca5F60DC777b901c
5) BLUSD-3CRV LP Token: 0x5ca0313D44551e32e0d7a298EC024321c4BC59B4
6) BLUSD-3CRV Gauge: 0xdA0DD1798BE66E17d5aB1Dc476302b56689C2DB4

The ZAP performs transfers without checking their return value as only known tokens are being used. Also approvals for liquidity tokens towards the gauge are performed.

While removing liquidity, the contract checks its balance to ensure that the amount transferred back to the user is correct. Coinspect observed there is no mechanism to recover transfers mistakenly sent to this contract.

ChickenBonds NFT Artwork

We reviewed the contracts in charge of handling and calculating token's properties across their life cycle (i.e., from egg to hatched chickens). The codebase was tidy and easy to understand, however some parts miss from NatSpec and documentation which is advisable to include.

Each NFT has dynamic traits that depend on several factors related to the bond that backs them and its properties such as the user's trove size, the total permanent LUSD, the time being hatched among others.

The tokens start being eggs that represent a user's active bonding position (bond-backed NFT). Users can decide to wait until the bonding time passes getting the accrued bLUSD (performing a "Chicken-in") or recover their LUSD without

getting accrued interests at any time (“Chicken-out”). Eggs can only mutate to one type of chicken. Once an egg mutates to a chicken, the backed bond position is extinguished and it only has collectible purposes. Because of this behavior, the NFT has two main rarity stages (while being egg and chicken) that change dynamically. Each rarity is calculated when a bond is created and extinguished respectively by reading parts of the “DNA” (a pseudo-random generated number that represents the uniqueness of that token) and comparing its value against a probability scale. As the DNA is generated in a pseudo-random way that depends on the current timestamp, there are some edge scenarios that could enable its manipulation in order to improve the overall rarity.

Also, because token images are on chain SVG generated animations, it is worth noting that depending on the browser used to display NFTs, some animations may appear to be broken or not working as intended. In addition, the dynamism of the metadata could not be detected immediately by marketplaces showing outdated NFT animations, which may lead users to purchase eggs that hatched before.

Regarding how contracts communicate between each other, the artwork address is required by the BondNFT mainly to retrieve key information such as the URI. It is advised to perform an off chain input validation before calling `setArtworkAddress` as the owner renounces the ownership at the end of the call.

3. Summary of Findings

Id	Title	Total Risk	Fixed
LCB-11	Dynamic traits can be outdated on marketplaces	Low	✘
LCB-12	NFT DNA could be manipulated by timestamp selection	Low	✓
LCB-13	Missing NatSpec documentation & open TODOs	Info	✘

4. Detailed Findings

LCB-11	Dynamic traits can be outdated on marketplaces

Total Risk	Impact	Location
Low	Low	`BondNFT.sol`
Fixed	Likelihood	
✘	Low	

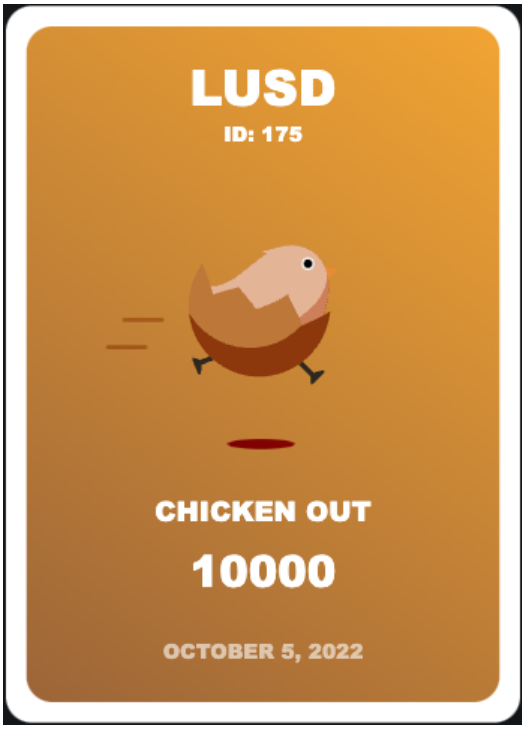
Description

The NFTs' metadata on marketplaces could be outdated and users may purchase an already hatched chicken without being aware.

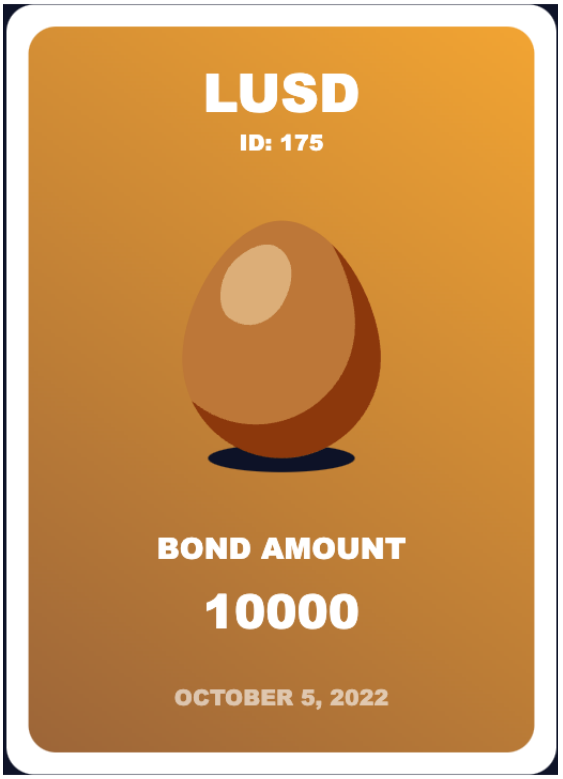
Tokens shown on marketplaces will require periodic metadata refreshes to ensure that the dynamic parameters are properly shown. If the metadata has not been refreshed for more than one day (24 hour cool down feature), offers made to eggs can be fulfilled while the token appears to be still an egg which has already Chicken out/in.

By the time of this audit, this issue was found with the token #175 which was chickened-out 13 days ago, however it is worth remarking that the chicken's SVGs were generated while conducting this revision.

On Looksrare it is shown as chickened-out:



Whereas the same token in X2Y2 is shown as an unhatched egg:



If the token link is opened in the future, its image might be changed because during this audit the metadata for that token was enqueued for a refresh.

Recommendation

This issue depends on each specific market implementation and how quick they react to the NFT metadata updates. Even though the currently used 24 hour cool down period seems appropriate, there is no guarantee that some mistake in the market fails to update the image shown to users.

Clearly document this potential scenario to traders in the token description encouraging them to refresh the metadata and check the bond status before pursuing any actions.

Status

Open.

LCB-12	NFT DNA could be manipulated by timestamp selection		
Total Risk **Low**	Impact Medium	Location `BondNFT.sol`	
Fixed ✓	Likelihood Low		

Description

The DNA of each Chicken could be manipulated by pre-calculating a valuable timestamp within a desired chicken in/out time range in order to maximize the output of `getHalfDna` and increase the rarity of the token.

There are three main known strategies on the platform: trading bLUSD, liquidity providing and collecting NFTs. Secondary market price of NFTs has a strong correlation with their rarity. A similar issue has been exploited previously in The Wolf Game.

The DNA is synthesized via the following path once a user decides to chicken in or out:

`ChickenBondManager.chickenOut() - L319:`

```
uint80 newDna = bondNFT.setFinalExtraData(msg.sender, _bondID, permanentLUSD / NFT_RANDOMNESS_DIVISOR);
```

`ChickenBondManager.chickenIn() - L419:`

```
uint80 newDna = bondNFT.setFinalExtraData(msg.sender, _bondID, permanentLUSD / NFT_RANDOMNESS_DIVISOR);
```

`BondNFT.setFinalExtraData() - L118:`

```
uint80 newDna = getHalfDna(_tokenID, _permanentSeed);
```

`BondNFT.getHalfDna() - L143:`

```
   function getHalfDna(uint256 _tokenID, uint256 _permanentSeed) internal view returns (uint80) {
        return uint80(uint256(keccak256(abi.encode(_tokenID, block.timestamp, _permanentSeed))));
    }
```

Later on, the rarity for each trait is constructed by cutting their ending DNA in subsequent slices of 16 bits:

`ChickenInArtwork._calcChickenInData()` - L55 - L62:

```
       uint80 dna = _commonData.finalHalfDna;
        uint256 troveFactor = uint256(_commonData.troveSize) * 1e18 / MAX_TROVE_SIZE;

            _chickenInData.chickenColor  = _getChickenColor(_cutDNA(dna,   0, 16), _commonData.shellColor,
troveFactor);
        _chickenInData.comb =         _getChickenComb (_cutDNA(dna, 16, 16), troveFactor);
        _chickenInData.beak =         _getChickenBeak (_cutDNA(dna, 32, 16), troveFactor);
        _chickenInData.tail =         _getChickenTail (_cutDNA(dna, 48, 16), troveFactor);
       _chickenInData.wing =         _getChickenWing (_cutDNA(dna, 64, 16), troveFactor);
```

`_cutDNA()`:

```
function _cutDNA(uint256 dna, uint8 startBit, uint8 numBits) pure returns (uint256) {
    uint256 ceil = 1 << numBits;
    uint256 bits = (dna >> startBit) & (ceil - 1);

    return bits * 1e18 / ceil; // scaled to [0,1) range
}
```

Also, the rarity calculation for each trait increases as the cut portion, for example:

```
   function _getChickenBeak(uint256 rand, uint256 troveFactor) internal pure returns (uint8) {
        uint256 tier1Probability = (1e18 + 2 * ALPHA4) / 4 - 2 * R4 * troveFactor / 1e18;
        uint256 tier2Probability = (1e18 + ALPHA4) / 4 - R4 * troveFactor / 1e18;
        uint256 tier3Probability = (1e18 - ALPHA4) / 4 + R4 * troveFactor / 1e18;

        uint256 needle = tier1Probability;
        if (rand < needle) { return 1; }
        needle += tier2Probability;
        if (rand < needle) { return 2; }
        needle += tier3Probability;
        if (rand < needle) { return 3; }
        return 4;
    }
```

As the `tokenID`, `permanentLUSD`, `NFT_RANDOMNESS_DIVISOR` are previously known parameters which can be directly read from the contract and `keccak256` hash is deterministic, users will be able to find useful timestamps to increase the overall rarity.

It is worth noting that validators can control the block timestamp they are validating.

Because of the nuances mentioned before, this issue is considered informational as it is unlikely to be exploited.

Recommendation

Consider using a VRF while generating the half DNA.

Status

Risk acknowledged.

The client was consulted about this scenario and agreed with its consequences but stated that the scope and probabilities to exploit this in a profitable way are limited. Users have to wait to send the transaction putting at risk their bonds and the transaction might not be mined at that exact timestamp.

Liquity dev team stated that they had considered using Chainlink's VRF but decided against to avoid the costs.

Coinspect did not perform a full analysis of scenarios where it might make sense to exploit this issue because of time constraints.

LCB-13	Missing NatSpec documentation & open TODOs

Total Risk	Impact	Location
Info	-	-

Fixed	Likelihood
✘	-

Description

There are several public and external functions, and variables with missing or incomplete NatSpec documentation. Moreover, the codebase has open TODOs.

Providing clear and descriptive comments on each public variable helps devs and users to understand better their meaning and context of usage, among others. Regarding functions, including complete NatsSpec documentation explaining what they are meant to do, return values and input parameters in case of having. Last, contracts, interfaces and libraries should also be commented explaining their purpose and intended behavior.

Recommendation

Add clear and complete NatSpec documentation and resolve the pending TODOs.

Status

Open.

5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.