B.Protocol – LUSD Chicken Bonds Integration

Smart Contract Security Assessment

21.07.2022

ABSTRACT

Dedaub was commissioned to audit the LUSD Chicken Bonds integration with the B.Protocol. The B.Protocol pools users' funds into a Backstop pool that is used for liquidations happening on an underlying platform, in this case, Liquity. The B.Protocol then uses an AMM (B.AMM) to swap liquidated assets to LUSD. The LUSD Chicken Bonds Protocol makes use of B.Protocol to store funds in the stability pool, whilst generating yield.

SETTING AND CAVEATS

The financial logic of the B.Protocol is described in https://www.bprotocol.org, with an accompanying whitepaper. The Chicken Bonds protocol is described in https://www.chickenbonds.org.

The scope of the audit includes the integration of LUSD Chicken Bonds with the B.Protocol. Specifically, files:

BAMM.sol
GemSeller.sol
GemSellerController.sol
UniV3Twap.sol

One more file, PriceFormula.sol, was not included in the audit scope, as it contains standard stableswap calculations and it has been audited before without modification, as indicated in the notes we were supplied.

This audit report covers commit hash `d219536229884a4ece00d2229fc4e27d77d9b0fc` from the chicken branch. Two auditors and a mathematician worked over the codebase over 0.5 weeks.

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than regular use of the protocol. Functional

correctness (i.e., issues in “regular use“) was a secondary consideration, however intensive efforts were made to check the correct application of the mathematical formulae in the reviewed code. Functional correctness relative to low-level calculations (including units, scaling, quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

This scope of the audit also included crypto-economic considerations. A number of checks have been carried out, however the crypto-economic effectiveness of this specific design is novel. Therefore, the financial viability of this protocol in real market conditions cannot be fully established. In terms of protocol completion, the protocol appears to be ready for staging.

PROTOCOL-LEVEL CONSIDERATIONS

There are relatively few protocol-level concerns. Fundamentally, one can question whether the B.Protocol is truly effective as an AMM, since it has no captive liquidity, as a traditional AMM does. That is, instead of guaranteeing that the protocol can always perform an exchange between two assets, it relies on external agents to step in and buy an asset, enticed by a discount. (In this case, the integration documentation informs us that the discount will be capped at 4%.) However, the B.Protocol has been operating in different settings for some time and its whitepaper makes an argument about its financial viability.

Another protocol-level consideration concerns the exact AMM accounting used in the B.AMM. Per the B.Protocol whitepaper, the two assets are being converted into USD values and then the price impact of a trade is used as a guide to whether to issue a discount: when one of the assets is much more plentiful than the other, it can be swapped more cheaply (again, up to 4%, per the expected deployment setting). In this case, the two assets are ETH and LUSD. However, the USD price of both is computed with the assumption that LUSD is well-pegged, at exactly 1 USD. This is not necessarily a reasonable assumption, especially since the code contains an explicit call to

`compensateForLusdDeviation`, exactly after the stableswap calculation of price impact (which produces the discount). In other words, the possibility for LUSD deviation from USD is acknowledged and considered at the final point in the calculation, in order to offer market-attractive rates to potential buyers of ETH, but is not considered when computing the discount. We do expect that the deviation of LUSD-USD will be small and will matter very little in the price impact calculation, but we have not seen simulations or arguments establishing this.

Furthermore, the Y-asset (ETH) USD value ("Y reserves") calculation is:
*yBalance = lusdBalance + (ethValue * 2)*.
This is justified by the B.Protocol whitepaper (sec.3), but there is an arbitrary assumption baked into this calculation: the reserves are calculated with an inventory component (*inv*) and an imbalance component (*imb*) equal to *ethValue*. This is what produces the reserves quantities given to the stableswap calculation:
X = inv - imb = (lusdValue + ethValue) - ethValue = lusdValue
*Y = inv + imb = (lusdValue + ethValue) + ethValue = lusdValue + ethValue*2*
However, this means, in words, that *the discount is computed by considering the stableswap price impact if a buyer of ETH were to offer (in USD considered as LUSD) an amount equal to the current value of the entire ETH reserves, as priced by an off-chain oracle.*
This is an arbitrary choice, and not encoded in the B.Protocol whitepaper (which stops its analysis at the level of an *inv* and *imb.*). Why "the entire ETH reserves" and not, e.g., half? Why not a constant amount?

The arbitrary choice means that the discount formula is *ad hoc* and its effectiveness is established empirically. We expect that the formula is well-tuned for expected swap values in this particular integration with the Chicken Bonds protocol, but encourage the development team to double-check through simulations.

An even more extreme case (but with low impact) concerns the GemSeller contract, where the "X reserves" (and part of the "Y reserves") input to the stableswap calculator

is an immutable quantity, set at contract construction: `lusdVirtualBalance`. The impact is small, since this is used only to decide the discount offered for LQTY sales, i.e., rewards, not principal. The developers probably have a tight range of rewards in mind when deploying, so that the `lusdVirtualBalance` amount can be fixed once and for all, for effective discount calculations. However, any misconfiguration here (or change in LQTY amounts/value after deployment) may make LQTY rewards hard to sell, due to non-competitive pricing.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues that affect the functionality of the contracts. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: -User or system funds can be lost when third party systems misbehave. -DoS, under specific conditions. -Part of the functionality becomes unusable due to programming error.
LOW	Examples: -Breaking important system invariants, but without apparent

	consequences. -Buggy functionality for trusted users where a workaround exists. -Security issues which may manifest when the system evolves.

Issue resolution includes “dismissed” or “acknowledged” but no action taken, by the client, or “resolved”, per the auditors.

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

[No high severity issues]

MEDIUM SEVERITY:

ID	Description	STATUS
M1	The integration of ChickenBonds with BAMM allows limited financial manipulation (attacker can get maximum discount)	**RESOLVED** (commit a55871ec takes the Curve LUSD, in virtual terms, also into account)
BAMMSP holds not only LUSD, but also ETH from liquidations. Anyone can buy ETH at a discount, which depends on the relative amounts of LUSD and ETH of the BAMMSP. In essence, the larger the amount of ETH compared to LUSD, the larger the discount. An attacker could act as follows: call `shiftLUSDFromSPToCurve` of the Chicken Bond protocol, decrease the amount of LUSD in BAMMSP and then buy ETH at a greater discount. There are no restrictions on who can call the `shiftLUSDFromSPToCurve` function, but the shift of the LUSD amounts takes place only if the LUSD price in Curve is too high. If this condition is satisfied, the attacker can perform the attack in order to buy ETH at a maximum discount at no extra cost. If not, then the attacker should first manipulate the price at Curve using a flashloan. The steps are the following: 1.Increase the price of LUSD in Curve pool - above the threshold which allows shift from the SP to Curve - possibly using a flashloan.		

2. Call `shiftLUSDFromSPToCurve` and move as many LUSD as possible to increase the discount on ETH
3. Buy ETH from BAMMSP at a discount.
4. Repay the flashloan.

This attack is profitable in a specific price range for LUSD, close to the "too high" threshold (otherwise the cost of tilting the pool will likely outweigh any benefit from the discount), and the discount is bounded (at 4%, based on the design documents we were supplied). Hence, we consider this only a medium-severity issue.

A general consideration of the profitability of the attack should consider:
a) that the second step drops the price of LUSD in Curve, resulting in losses for the attacker when he repays the flashloan at the 4th step.
b) However, the amount of ETH the attacker can buy from BAMMSP at discount is independent from the amounts in the Curve pool, therefore under some circumstances the discount may also compensate for the losses making the attack profitable.

M2	Uniswap v3 TWAPs can be manipulated, and this will become much easier post-Merge	**DISMISSED**

A Uniswap v3 TWAP is expected to be used to price LQTY relative to ETH. Uniswap v3 TWAPs can be manipulated, especially for less-used pools. (There have been at least three instances of attacks already, for pools with liquidity in the low millions. The LQTY-ETH pool currently has $780K of liquidity.)

Although currently manipulation for *active* pools is considered rarely profitable, once Ethereum switches to proof-of-stake (colloquially, "after the Merge") such manipulation will be much easier to perform with guaranteed profit.

Specifically, to manipulate a data point used for a Uniswap v3 TWAP, an attacker needs to control two consecutive pool transactions (i.e., transactions over the manipulated pool) that are in separate blocks. (This typically means the last pool transaction of a block and the first of the next block.) Under Ethereum proof-of-stake, validators are known in advance (at the beginning of the epoch) hence an attacker

can know when they are guaranteed to control the validator of the next block. The attack is:

- The attacker places the first transaction as the last pool transaction of the previous block (either by being the validator of both blocks or using flashbots). The first transaction tilts the pool.
- The attacker is guaranteed to not suffer any losses from swaps over the unrealistic price of the tilted pool because the attacker controls the immediately next block, prepending to it a transaction that restores the pool, while affecting a TWAP data point in this way.

The issue is at most medium-severity, because it only concerns selling LQTY, not the principal assets of the contracts.

M3	Values from Chainlink are not checked	**DISMISSED**

The protocol does not check whether the LUSD-USD price is successfully returned from Chainlink in `BAMM::compensateForLusdDeviation` and in `GemSeller::compensateForLusdDeviation`. Since this price is used to adjust the amount of ETH or LQTY returned by a swap in `BAMM` and `GemSeller` respectively, it is important to ensure that the values from Chainlink are accurate and correct.

This is in contrast with how Chainlink ETH-USD prices are retrieved in `BAMM::fetchPrice` and `GemSeller::fetchPrice`, where each call to Chainlink is checked and any failures are reported using a return value of 0.

LOW SEVERITY:

[No low severity issues]

CENTRALIZATION ISSUES:

It is often desirable for DeFi protocols to assume no trust in a central authority, including the protocol's owner. Even if the owner is reputable, users are more likely to engage with

a protocol that guarantees no catastrophic failure even in the case the owner gets hacked/compromised. We list issues of this kind below. (These issues should be considered in the context of usage/deployment, as they are not uncommon. Several high-profile, high-value protocols have centralization threats.)

ID	Description	STATUS
N1	Owner can set parameters with financial impact	**DISMISSED**

The owner of both the BAMM contract and the GemSeller contract can set several parameters with financial impact. None is a major threat: the "principal" deposited by the Chicken Bonds protocol is safe, even if the owner of BAMM is malicious. However, some funds are at threat.

Specifically:

- (BAMM) Owners can set the `chicken` address. The `chicken` address holds considerable control over the protocol, as it is the only address permitted to withdraw or deposit funds in the system. However, since this address can only ever be set once, the risk posed is limited. Once this address is set to the `ChickenBondManager` contract from the LUSD Chicken Bonds Protocol, this will no longer be an issue, as `ChickenBondManager` is itself very decentralized.
- (BAMM) Owners can set the `gemSeller` address. This is a centralization threat because `gemSeller` has infinite approval for `gem`, in this case LQTY. However, this means that a malicious owner can only steal rewards, not principal. Furthermore, the `gemSellerController` makes use of a time lock system. This prevents the owner from immediately changing the address of the `gemSeller`. A new address will first be stored as pending, and can only be set as the new `gemSeller` after a fixed time period has elapsed. Once set, the `gemSeller` has maximum approval for all LQTY held in the B.AMM.
- (BAMM and gemSeller) Owners can set parameters, including `fee` and `A`. The `fee` parameter is a threat, but is bounded by a maximum value (1% in BAMM, 10% in gemSeller). The `A` parameter only affects the discount given to buyers, which is bounded by a maximum, limiting the effect of any changes.

OTHER/ ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

ID	Description	STATUS
A1	Typo in `BAMM::constructor` parameter	**RESOLVED**
Parameter `address _fronEndTag` should be `address _frontEndTag`.		
A2	Misrepresentative function names when converting LUSD	**RESOLVED**
The functions `gemSeller::gemToLUSD` and `gemSeller::LUSDToGem` convert the given quantity of gem tokens to their LUSD value and vice versa. However, the functions both return the USD price of the gem asset, not the LUSD price (more accurately, the GEM-ETH and ETH-USD prices are used together). Although the protocol assumes that 1 LUSD is always equivalent to 1 USD, `gemSeller::gemToUSD` and `gemSeller::USDToGem` would be more accurate function names.		
A3	Compiler bugs	**INFO**
The code is compiled with Solidity 0.6.11. This version of the compiler has some known bugs, which we do not believe to affect the correctness of the contracts.		

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring through Dedaub Watchdog.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.