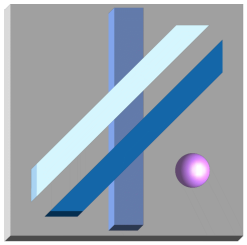

Fixed Point Solutions, LLC

B.AMM Protocol Rari Integration Assessment

2021/12/19
Prepared by: Kurt Barry

1. Scope

Fixed Point Solutions previously reviewed an earlier version of the B.AMM, which integrated with the Liquity protocol:

https://github.com/Fixed-Point-Solutions/published-work/blob/master/SmartContractAudits/FPS_B.AMM_Liquity_Assessment_FINAL.pdf

The previous assessment can be referred to for general considerations about the B.AMM mechanism. This assessment covered changes in functionality to integrate with Rari lending pools, and confirmed that no working functionality (for example, the StableSwap implementation) had been regressed since the previous review.

The following files from https://github.com/backstop-protocol/dev/ were reviewed at commit 5fe7ce3a2ce98bb0822f123c31e0f28b60d131cd:
/packages/contracts/contracts/B.Protocol/BAMM.sol
/packages/contracts/contracts/B.Protocol/PriceFormula.sol
/packages/contracts/contracts/B.Protocol/TokenAdapter.sol

Additionally, the following pull request was reviewed (diff only):
https://github.com/backstop-protocol/compound-protocol/pull/1/

2. Limitations

No assessment can guarantee the absolute safety or security of a software-based system. Further, a system can become unsafe or insecure over time as it and/or its environment evolves. This assessment aimed to discover as many issues and make as many suggestions for improvement as possible within the specified timeframe. Undiscovered issues, even serious ones, may remain. Issues may also exist in components and dependencies not included in the assessment scope.

3. Findings

Findings and recommendations are listed in this section, grouped into broad categories. It is up to the team behind the code to ultimately decide whether the items listed here qualify as issues that need to be fixed, and whether any suggested changes are worth adopting. When a response from the team regarding a finding is available, it is provided.

Findings are given a severity rating based on their likelihood of causing harm in practice and the potential magnitude of their negative impact. Severity is only a rough guideline as to the risk an issue presents, and all issues should be carefully evaluated.

Severity Level Determination		Impact		
		High	Medium	Low
Likelihood	High	**Critical**	**High**	**Medium**
	Medium	**High**	**Medium**	**Low**
	Low	**Medium**	**Low**	**Low**

Issues that do not present any quantifiable risk (as is common for issues in the Code Quality category) are given a severity of **Informational**.

3.1 Security and Correctness

Findings that could lead to harmful outcomes or violate the intentions of the system.

SC.1 Tokens with Call-on-Transfer Hooks Can Be Used to Drain Funds Via Reentrancy in `withdraw()` or to Achieve Better Prices in `swap()`

Severity: Medium

Code Location:

[1]https://github.com/backstop-protocol/dev/blob/5fe7ce3a2ce98bb0822f123c31e0f28b60d131cd/packages/contracts/contracts/B.Protocol/BAMM.sol#L223
[2]https://github.com/backstop-protocol/dev/blob/5fe7ce3a2ce98bb0822f123c31e0f28b60d131cd/packages/contracts/contracts/B.Protocol/BAMM.sol#L269
[3]https://github.com/backstop-protocol/dev/blob/5fe7ce3a2ce98bb0822f123c31e0f28b60d131cd/packages/contracts/contracts/B.Protocol/BAMM.sol#L274

Description: Withdrawals from the B.AMM are assigned a proportional amount of each token held by the pool. By necessity these transfers are processed sequentially. If one of these tokens supports a call-on-transfer hook (e.g. if it implements the ERC-777 standard), it could be used to reenter the `withdraw()` function. When this happens, balances for tokens that were not yet transferred during the initial call will be erroneously high and the attacker will receive a disproportionately large quantity of them in the reentrant call.

In the `swap()` function, the debt token is first transferred from the caller, and the purchased collateral token is then sent to them. If the debt token has a call-on-transfer hook, or if either token has a call-on-transfer hook that is invoked prior to balance updates taking effect, the caller can reenter `swap()` and purchase at a price more favorable than after completion of the first swap (since one or both balances will not yet be updated).

These issues can be mitigated by either excluding tokens with call-on-transfer hooks from being debt or collateral assets, or by preventing reentrancy into `withdraw()` and `swap()` (and possibly `deposit()` as well, although it does not appear to be vulnerable in the same way as the other two functions).

Response: Fixed in commit 7dd446847d00a2a8e661fb6c4c89fcd25a719a44.

3.2 Usability and Incentives

Findings that could lead to suboptimal user experience, hinder integrations, or lead to undesirable behavioral outcomes.

None.

3.3 Gas Optimizations

Findings that could reduce the gas costs of interacting with the protocol, potentially on an amortized or averaged basis.

G.1 `decimals` in `TokenAdapter` Could Be Marked `constant`

Severity: Low

Code Location:
https://github.com/backstop-protocol/dev/blob/rari/packages/contracts/contracts/B.Protocol/TokenAdapter.sol#L13

Description: This value never changes; marking it `constant` would save on gas costs if it is ever accessed on-chain, and during deployment.

Response: Fixed in commit 0d23e935d8abacb1585bf5b33d26d984ecc9ba3f.

G.2 `TokenAdapter` Does Not Support Infinite Approvals

Severity: **Low**

Code Location:
https://github.com/backstop-protocol/dev/blob/5fe7ce3a2ce98bb0822f123c31e0f28b60d131cd/packages/contracts/contracts/B.Protocol/TokenAdapter.sol#L42

Description: It is common in ERC-20 implementations to let an approval for the maximum value of a 256-bit unsigned integer represent an “infinite approval” that is never decreased by transfers. This is primarily a gas optimization for highly trusted contracts, as it prevents an expensive SSTORE operation.

Response: Fixed in commit bb6fd3c7467877339e7ab33b655e1b571fd90618.

3.4 Code Quality

CQ.1 `decimals()` of TokenAdapter Returns `uint256` instead of `uint8`

Severity: **Informational**

Code Location:
https://github.com/backstop-protocol/dev/blob/rari/packages/contracts/contracts/B.Protocol/TokenAdapter.sol#L13

Description: The storage field `decimals` is declared as type `uint`, which is an alias for `uint256`, and this will be the return type of the generated accessor method. However, the ERC-20 standard specifies the return type of `decimals()` as `uint8`.

Response: Fixed in commit 961bcbd11da825bcb595e1361ebdde229fe7717f.

4. Notes

This section contains general considerations for interacting with or maintaining the system and various conclusions reached or discoveries made during the course of the assessment. Whereas findings generally represent things for the team to consider changing, notes are more informational and may be helpful to those who intend to interact with the system.

4.1 ERC-20 `approve` Race Condition

When one address sends a transaction to change the `transferFrom` allowance granted to another address, the other address can frontrun this change and claim both approval amounts. This is a well-known issue with the ERC20 standard (more information: https://swcregistry.io/docs/SWC-114).

This is a widely-known issue with the ERC-20 standard and most integrators know how to deal with it, thus it is not considered a “finding” in the formal sense and no specific mitigation is recommended.

4.2 Token Precision Considerations

When normalizing decimals in total collateral value calculations, it is possible to lose a significant amount of precision or even spuriously arrive at a price of zero when the normalization requires a division:

https://github.com/backstop-protocol/dev/blob/5fe7ce3a2ce98bb0822f123c31e0f28b60d131cd/packages/contracts/contracts/B.Protocol/BAMM.sol#L151

Whether this happens in practice depends on the price of the collateral token, the number of decimals it has, the number of decimals used by the price oracle, and the number of decimals the debt token has. Generally, all tokens added as accepted collateral should be assessed based on their decimals and a conservative price lower bound for this risk.

Addendum 1

In March 2022, a number of revisions were made to the Rari B.AMM integration. Notably:

1) pool reserves are lent out to generate yield (in the same market that is backstopped);
2) flash loan functionality was added to the `swap()` function;
3) a gas-saving withdrawal was added that allows skipping withdrawal of collateral tokens if the amount of all of them would be zero.

The following diff was reviewed:
https://github.com/backstop-protocol/dev/compare/multi_collateral...ctoken_as_backstop

No significant issues were found. Several risks with adequate mitigation were noted:

1) The liquidation process now suffers from a solvency risk–if much of the backstop reserves are borrowed in the market, there may not be enough liquid reserves to complete larger liquidations in a single step. This is mitigated by a) modifying `canLiquidate` to account for market liquidity, and b) liquidators can still liquidate a position piecemeal if necessary. The interest rate model of the underlying market will also raise rates precipitously at around 80% borrow utilization, helping to ensure a minimal level of solvency under non-manipulated conditions.

2) Users attempting to do a gas-efficient withdrawal can protect themselves against the pool acquiring collateral balances after their transaction is sent but prior to its confirmation by setting the `minLusd` parameter appropriately.

3) The flash swap mechanism does not perform any check on the result of the external call similar to what is done in https://eips.ethereum.org/EIPS/eip-3156; however, no such check is necessary here as repayment is drawn from `msg.sender`, not the receiver of the collateral tokens, which means that the sort of attack that motivates such a check cannot be executed.

Minor suggestions:

- Nest the the loop on line 228 within an `if (withdrawCollateral) {}` block as the Solidity compiler is unlikely to optimize away doing the check after every loop iteration. This also makes the logic a bit more clear.