Ajna Security Review

This report was produced for the Ajna Protocol by Prototech Labs

Mail: info@prototechlabs.dev
Twitter: @Prototech_Labs
Web: www.prototechlabs.dev

Contents

1. Executive Summary

This smart contract security review was prepared for the Ajna Protocol by Prototech Labs, a smart contract consultancy providing auditing services, code reviews, blockchain and DAO solutions. Prototech Labs would like to thank the Ajna team for giving us the opportunity to review the current state of their protocol in the lead-up to its relaunch in Q4 2023.

This document outlines the findings, limitations, and methodology of our review, which is broken down by issue and categorized by severity. It is our hope that this review provides valuable findings and insight into the current implementation. Prototech Labs is happy to receive questions and feedback to improve our services and look forward to working with you in future project endeavors.

2. Project Overview

Category	**Description**
Type	Protocol audit and code security review
Security Researcher(s)	Chris Smith Brian McMichael Christopher Mooney Kurt Barry Derek Flossman
Timeline	2023-09-27 to 2023-10-13
Method(s)	Game theory, invariant analysis and limited code review for the discovered griefing attack and other changes introduced by the Ajna team.
Code Repository	https://github.com/ajna-finance/contracts
Commit Referenced	n/a
Security Report	https://github.com/Prototech-Labs/published-work

3. Findings Overview

Prototech Labs found one medium and several low and informational findings for which recommendations have been provided.

Below is a numerical overview of the identified findings, split up by their severity, illustrating their status:

Medium Severity Findings	**[1]**
7.1 htp could exceed MAX_NEUTRAL_PRICE	Fixed

Low Severity Findings	**[5]**
8.1 Additional critical tests - Borrower Liquidation -> Settle -> Retrieves collateral	Acknowledged
8.2 Additional critical tests - Borrower Liquidation -> Settle -> Borrow again	Acknowledged
8.3 PRBMathUD60x18.exp revert after long periods without accumulation can lock pool	Fixed
8.4 pool.updateInterest() and pool.debtInfo() revert with Arithmetic over/underflow for large interest rates over reasonable timeframes	Fixed
8.5 t0DebtPenalty becomes unused and can be removed	Fixed

Informational Findings	**[7]**
9.1 Use upgrade opportunity to emit pool subset hash on pool creation	Acknowledged
9.2 Make invariant tests more robust	Acknowledged
9..3 Risks to Ajna users on protocols with scheduled or predictable downtime	Acknowledged
9.4 Significant changes to Whitepaper	Fixed
9.5 Synchronize and reorder state updates of pledgedCollateral between ERC20Pool and ERC721Pool	Acknowledged
9.6 PoolState.Collateral is unnecessary	Acknowledged
9.7 Informational: Code Quality	Acknowledged

4. Introduction

Ajna's protocol is a non-custodial, peer-to-peer, permissionless lending, borrowing and trading system that requires no governance or external price feeds to function. The protocol consists of pools with lenders and borrowers.

Security Review Scope and Future Recommendation

This security review focused on the game theory and included a limited code review for the griefing attack as well as other significant changes that the Ajna team made to the protocol. As with all security reviews, no assessment can guarantee the absolute safety or security of a system.

Due to the shifting nature of the solutions and code (no final commit hash was delivered during the audit engagement time), our findings may be limited in scope and coverage and undiscovered issues, even serious ones may remain. We would recommend engaging for additional security reviews once the solution/code has been finalized and a firm commit hash is available. Further, we would recommend ensuring an updated whitepaper or other write up of the changes and the problems they are meant to solve is provided to the security reviewers.

5. Limitations and Report Use

Disclaimer: 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.

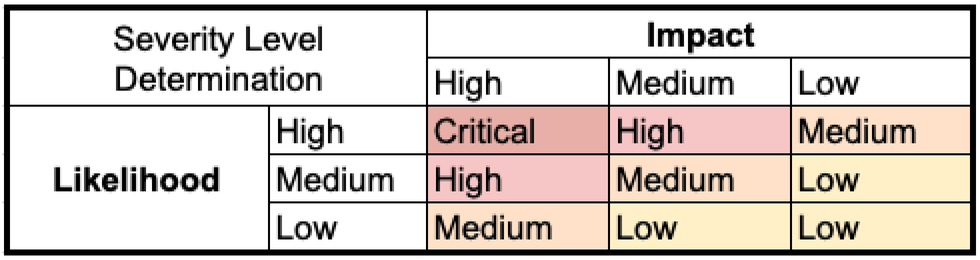
The software systems herein are emergent technologies and carry with them high levels of technical risk and uncertainty. This report and related analysis of projects to not constitute statements, representations or warranties of Prototech Labs in any respect, including regarding the security of the project, utility of the project, suitability of the project's business model, a project's regulatory or legal status or any other statements, representations or warranties about fitness of the project, including those related to its bug free status. You may not rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Our complete terms of service can be reviewed here.

Specifically, for the avoidance of doubt, any report published by Prototech Labs does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of any client or project, and is not a guarantee as to the absolute security of any project. Prototech Labs does not owe you any duty by virtue of publishing these reports.

6. 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 are given a severity of Informational.

7. Medium Risks

7.1 htp could exceed MAX_NEUTRAL_PRICE

Severity: Medium

Context: Kicker Actions Contract

Description: A potential overflow was identified when basing the reference price based on max of htp or neutral price.

Recommendation: Limit reference price to MAX_NEUTRAL_PRICE.

Ajna: Fixed in this commit

8. Low Risks

8.1 Additional critical tests - Borrower Liquidation -> Settle -> Retrieves collateral

Severity: Low

Description: Our review of the code and tests available during the audit highlighted a possible untested path. There did not seem to be a test that showed an over collateralized borrower would be able to retrieve their "extra" collateral after liquidation. Our review of our code indicates that this should be possible, but testing would prove this is the case and ensure no future regressions.

Recommendation: Add the appropriate tests.

8.2 Additional critical tests - Borrower Liquidation -> Settle -> Borrow again

Severity: Low

Description: An untested flow was identified while reviewing the new changes during the audit timeline. There does not seem to be a test that shows a borrower that has been liquidated can open a new position. Our review of our code indicates that this should be possible, but testing would prove this is the case and ensure no future regressions.

This path is especially important to test with the current changes because borrowers can no longer save their position once it is liquidated. Ensuring that a user can open a fresh position at the same borrowerAddress after their previous position has been liquidated and settled is important.

Recommendation: Add appropriate testing.

8.3 PRBMathUD60x18.exp revert after long periods without accumulation can lock pool

Severity: Low

Context:

- PoolCommons.sol#L228
- PoolCommons.sol#L390
- PoolCommons.sol#L407

Description: The pool interest factor utilizes the PRBMathUD60x18.exp(x) function, which can fail when x is greater than 133_084258667509499441.

If a pool has a high rate and the interest has not been accumulated for some time, this operation will fail, resulting in a revert in both the pendingInterestFactor function, the pendingInflator function, and in the accrueInterest function, the latter of which is called in many operations throughout the protocol, effectively locking them from being used and preventing withdrawal of funds.

Failures occur approximately at the following rates when pool interest has not happened for a period of time. This may be less common for popular pools but long-tail pools could be neglected and locked if not accumulated regularly.

Accumulated Rate	Time elapsed at lock
48600.00%	100 Days
40500.00%	120 Days
27000.00%	180 Days
13400.00%	365 Days
6700.00%	2 Years

3400.00%	4 Years
1400.00%	10 Years
300.00%	50 Years
200.00%	100 Years

Recommendation:

- De-duplicate this operation into an internal function to prevent code re-use and for easier management.
- Include tests of functions that explore the range of expected values. In this case, PoolCommonsTest only explores a max 10% for 365 days.
- Investigate other areas where external math libraries may cause reversions of expected values (use of PRBMathSD59x18, for example)
- Provide a clean path to extract funds if the accumulation process fails, as this issue will effectively lock user funds from extraction.
- Several possible mitigations:
    - Consider lowering the maximum interest rate for pools to buy time between accumulations, which will also alleviate #3.
    - Consider a math library which allows for a higher .exp(x) value within expected ranges.
    - Consider that when a pool has a high rate but hasn't been accumulated for some amount of time, you could sacrifice the accumulation of some gains to prevent a revert and continue operation going forward:

```
Unset

//  This will reduce the amount of accumulated interest in the pool when over the max,

//   but it will prevent a lock of the system.

uint256 elapsedRate = Maths.min((poolState_.rate * elapsed_) / 365 days, 133_084258667509499440);

uint256 pendingFactor = PRBMathUD60x18.exp(elapsedRate);
```

Ajna: Max rate is considered too high and will cap.

8.4 pool.updateInterest() and pool.debtInfo() revert with Arithmetic over/underflow for large interest rates over reasonable timeframes

Severity: Low

Context: PoolCommons.sol#L231 and PoolCommons.sol#L405-L408

Description: Ajna's pool.updateInterest() and pool.debtInfo() revert with Arithmetic over/underflow as a result of accumulator multiplication. As a result, many pool operations fail, ultimately bricking the pool and resulting in locked funds. The Arithmetic over/underflow comes from a wmul() when either updating the existing accumulator, or fetching the pendingInflator() value. The terminal error occurs under somewhat reasonable conditions:

1. where the existing inflator has accumulated to a large value in the past, as a result of a high interest rate over time, and is then multiplied by a reasonable interest rate over time.
2. where a small or modest inflator is multiplied by a a high interest rate over time.

The initial problem was surfaced by aggressive invariant testing parameters. The regressions can be found at the bottom of PrototechRegressions.t.sol in the PrototechRegressionTestERC20PoolRewardsInvariantsWith50Buckets contract at the following gist.

To our knowledge, manipulating a pool to high accumulations of interest would require manipulating the lenders and borrowers to parity every 12 hours. If this can be done with small deltas of debt, such that the origination fee is negligible, it can occur in as little as 47 days (until the max interest rate of 50,000% is reached). After that, it takes roughly 58 more days of accumulating interest at this rate to trigger the overflow and lock the pool. There are many factors that may prevent this ramping up of the accumulator, such as market conditions on a token incentivizing more lenders to the pool, and thus making the delta for manipulation larger. For large TVL pools, the interest rate will almost certainly accumulate so much debt as to push most positions to liquidation.

One of the more compelling attacks we can think of is to manipulate a pool's rate up, wait some time, and then manipulate it back down, all before a pool enjoys enough popularity for market actors to participate. Since the inflator is an accumulator with no way to reset, it will have already accumulated close to the top of the range where the wmul() overflow occurs, and then had its rate reset to normal. Now, when the market begins using the pool, it would take a trivial amount of time or rate manipulation to push it into the bricked state.

Recommendation: Ultimately, there are almost too many market conditions and human responses to consider. For this reason, we suggest lowering the max interest rate to a value that would make even extreme cases of interest accumulation impossible to brick the pool over the protocol's expected lifespan. This would completely remove the incentive to manipulate a pool over all unforeseeable market conditions.

Ajna: Acknowledged and fixed in this commit

8.5 t0DebtPenalty becomes unused and can be removed

Severity: Low

Context: File.sol#L123

Description: The t0DebtPenalty is unused after this change and can be removed to save gas and state.

Recommendation: Remove t0DebtPenalty.

Ajna: Fixed in this commit

9. Informational Findings

9.1 Use upgrade opportunity to emit pool subset hash on pool creation

Severity: Informational

Context: ERC721PoolFactory.sol#L58

Description: When a new ERC721 Subset pool is created, the subset hash is calculated internally and is not provided to the user. This can make it difficult for future integrators, who must recalculate the subset hash from the parameter data or inspect the internal transaction operations. This hash is used in the PositionManager and elsewhere to determine whether the pool is a valid Ajna pool and should be made more readily available.

Recommendation: Emit an event in pool creation that announces the subset pool hash.

9.2 Make invariant tests more robust

Severity: Informational

Context: BaseHandler.sol#L208-L217 and BasicERC20PoolHandler.sol#L117-L121

Description: Invariant tests set actor to msg.sender, but then only ever attempt to manipulate the actor's positions. Given many of the protocol's functions can be called to manipulate other user's positions this testing is too limited and may not surface invariant violations or unintended reverts given these more complicated interactions.

This testing originally explored chasing our a deployment bug with an uninitialized pool, where noOfLoans() reverted. This revert was the result of not calling init() to add an address(0) loan to the array. There are so many potential interactions with the protocol, it was difficult to see if this address(0) loan could break a calculation or potentially even be allocated debt and liquidated. The testing involved adding an address(0) actor to the tests to surface all these potential interaction points when we discovered this limitation in the invariant tests.

Recommendation: Add support for a target in invariant tests such that it's also randomly rolled across the set of existing actors and address(0). This will ensure that the tests will sometimes have an actor that is the same as target and sometimes target will be different or address(0). This may cause too many interactions where actor != target, which can be mitigated by only forcing this difference some low percentage of the time, or running the tests with sufficient depth to account for the additional entropy.

Prototech Labs started adding the plumbing to support an actor sending the transaction and a target; however, this means that another targetId parameter needs to be added to functions like repayDebt() and drawDebt() and so most of the invariant regression tests break. What's more, once support is added the change touches so many lower level calls that assume actor == target that it's infeasible to maintain this as a separate version of invariant tests as merging in changes will take too long each time. Ultimately, the recommendation is to develop this particular functionality further.

Ajna: Acknowledged

9.3 Risks to Ajna users on protocols with scheduled or predictable downtime

Severity: Informational

Description: There is risk to kickers, borrowers, lenders, and reserves if a network's downtime is known in advance. Since so much of the protocol's operation depends on market participants taking actions within optimally slotted times, an attacker may be able to gain advantage by timing protocol actions to coincide with known network downtime windows. This could be as simple as timing kicks such that auctions bypass their market prices during a scheduled outage window, front-running a published pause transaction, or cultivating a vulnerability in a networks nodes, sequencers, or other infrastructure.

Recommendation: Note that this risk could impact some assets with pause functions too, and to that end, the most basic mitigation of this risk is to not deploy to any network that can have predictable downtime.

As mentioned in the last prototech audit, however, for some deploys of the protocol on certain networks, using a hybrid function that checks more than block.timestamp for auction pricing may be strongly advised.

9.4 Significant changes to Whitepaper

Severity: Informational

Context: Whitepaper

Description: There are significant changes to the Ajna protocol for this re-release. These include (but aren't limited to):

- Liquidated borrows no longer have a grace period
- Once in liquidation, borrowers can no longer save their position
- The price curve for auctions has been redone
- There is no Reserve Auction Taker reward
- Penalties to borrowers for being liquidated have changed as have the penalties/rewards to kickers
- MOMP is no longer used in the protocol
- How the LUP is used is different

Recommendation: It is encouraged that the Ajna team catalog all changes and ensure that the Whitepaper (and any other protocol documentation) is updated to reflect the new state of the system.

Ajna: Fixed

9.5 Synchronize and reorder state updates of pledgedCollateral between ERC20Pool and ERC721Pool

Severity: Informational

Context:

- ERC20Pool.sol#L183-L197
- ERC20Pool.sol#L262-L274
- ERC721Pool.sol#L195-L212
- ERC721Pool.sol#L279-L292

Description: The ERC20 and ERC721 pools have different orders of operations around token transfers. These contracts should follow similar patterns and probably adhere to checks-effects-interactions patterns and update all state prior to sending tokens to prevent read-side reentrancy vulnerabilities for integrators, especially if the protocol is going to be used as an oracle. (See Curve LP Manipulation). The functions affected are currently write-protected `nonReentrant`, however this doesn't prevent unexpected reads from the protocol during this function execution.

For example, in ERC20Pool.sol `repayDebt()`, the quote token is transferred to the `msg.sender` prior to the update of `poolState.collateral`. A third-party calling `PoolInfoUtils.borrowerInfo()` in the middle of this transfer would read an invalid value for `collateral` prior to the completion of the call.

Recommendation:

- Expose pool reentrancy mutex
- Perform all state updates for the pool prior to performing any external token interactions.
- Try to mimic operations between the pools.

9.6 PoolState.Collateral is unnecessary

Severity: Informational

Context:

- ERC20Pool.sol#L161
- ERC20Pool.sol#L231
- ERC721Pool.sol#L173
- ERC721Pool.sol#L244
- IPoolState.sol#L333

Description: This is considered unnecessary.

Recommendation: PoolState.collateral is never used after being set except to set `poolBalances.pledgedCollateral = poolState.collateral;` which could just be rewritten as `poolBalances.pledgedCollateral = result.poolCollateral;` saving complexity and gas

9.7 Informational: Code Quality

Severity: Informational

Context: [ERC20Pool.sol#L211-L213]

Description: Highlighting minor code tidiness

Recommendation: Recommend adding a space between these lines and removing the alignment padding for readability. This was a little confusing at first due to the if statement.

```
Unset
// ensure accounting is performed using the appropriate token scale

if (maxQuoteTokenAmountToRepay_ != type(uint256).max)

    maxQuoteTokenAmountToRepay_ =
_roundToScale(maxQuoteTokenAmountToRepay_,
poolState.quoteTokenScale);

collateralAmountToPull_        =
_roundToScale(collateralAmountToPull_,
_getArgUint256(COLLATERAL_SCALE));
```

to

```
Unset
// ensure accounting is performed using the appropriate
token scale

if (maxQuoteTokenAmountToRepay_ != type(uint256).max)

    maxQuoteTokenAmountToRepay_ =
_roundToScale(maxQuoteTokenAmountToRepay_,
poolState.quoteTokenScale);


collateralAmountToPull_ =
_roundToScale(collateralAmountToPull_,
_getArgUint256(COLLATERAL_SCALE));
```