Smart Contract Security Assessment

May 24, 2024

Project Name:

Azuro V2 Audit

Prepared by:

HYDN

Executive Summary

Client Name: Azuro

Language: Solidity

Timeline: Delivered 24th May 2024

Repository: https://github.com/Azuro-protocol/azuro-v2

Method: Static Automated Analysis + Manual Review

Vulnerability Summary

TBC After Remediation

Contents

Introduction

About HYDN

HYDN is an industry leading Smart Contract Audit firm which has extensive experience working with a multitude of blockchain projects to help them get the most out of their release. HYDN's team are CISSP, CCNP, GCIH, GREM, and GNFA certified and have worked uncovering some of the biggest cybersecurity hacks in history including the Olympic Destroyer hack in 2018.

HYDN was the winner of the EthernautDAO CTF Hacking Challenge 2022, hacking the smart contracts twice in under 5 minutes. HYDN's Lead Auditor also placed as the highest solo competitor in the Paradigm CTF 2023.

HYDN worked closely with Azuro to consider their unique business needs, objectives, and concerns. Our mission is to ensure the blockchain supports secure business operations for all and he has built the team at HYDN from the ground up to meet this very personal objective.

To keep up to date with our latest news and announcements, check out our website https://hydnsec.com/ or follow @hydnsecurity on Twitter.

Methodology

When tasked with conducting a security assessment, HYDN works through multiple phases of security auditing to ensure smart contracts are audited thoroughly. To begin the process

automated tests are carried out, before HYDN then moves onto carrying out a detailed manual review of the smart contracts.

HYDN uses a variety of open-source tools and analyzers as and when they are required and alongside this, HYDN primarily focuses on the following classes of security and reliability issues:

Basic Coding Mistakes

One of the most common causes of critical vulnerabilities is basic coding mistakes. Countless projects have experienced hacks and exploits due to simple, surface level mistakes that could have been flagged and addressed by a simple code review. The HYDN automated audit process which includes model checkers, fuzzers, and theorem provers analyses the smart contract for many of these basic coding mistakes. Once the automated audit has taken place, HYDN then performs a manual review of the code to gain familiarity with the contracts.

Business Logic Risks

HYDN reviews the platform or projects design documents before analysing the code to ensure that the team has a deep understanding of the business logic and goals. Following this, HYDN reviews the smart contracts to ensure that the contract logic is in line with the expected functionality.

HYDN also analyses the code for inconsistencies, flaws, vulnerabilities, or non-technical business risks which could impact business logic such as tokenomics errors, arbitrage opportunities, and share pricing.

Complex Integration Risks

Several high-profile exploits have been the result of not any bug within the contract itself, but rather an unintended consequence of its interaction with the broader DeFi ecosystem.
We perform a meticulous review of all of the contract's possible external interactions, and summarise the associated risks; for example: flash loan attacks, oracle price manipulation, MEV/sandwich attacks, etc.

Code Maturity

HYDN reviews the smart contracts for potential improvements in the codebase. These improvements ensure that the code follows industry best practices and guidelines, or code quality standards. Alongside this, HYDN makes suggestions for code optimization items such as gas optimization, upgradeability weaknesses, centralization risks, and more.

Impact Rankings

Once HYDN has completed the security assessment, each issue is assigned an impact rating. This impact rating is based upon both the severity and likelihood of the issue occurring. Each

security assessment is different and the business needs and goals, such as project timelines, and Azuro threat modelling. are taken into account when the impact rankings are created.

HYDN assigns the following impact rankings: Critical, High, Medium, Low, and Informational (listed by severity).

Disclaimer

This HYDN security assessment does not provide any warranties on finding all possible issues within the scope and the evaluation results do not guarantee the absence of any issues. **HYDN cannot make any guarantees on any further code which is added or altered after the security assessment has taken place**. As a single security assessment can never be considered fully comprehensive, HYDN always recommends multiple independent assessments paired with a bug bounty program. This security assessment report should not be considered as financial or investment advice.

Summary of Findings

Executive Summary

From April 10th 2024 through to May 23rd 2024, HYDN was tasked with performing a Smart Contract Audit on the Azuro V2 Smart Contracts. The scope covered the following repository: https://github.com/Azuro-protocol/azuro-v2.

During the audit process, HYDN first spent time to understand the business logic and goals of the client through discussions and by studying documentation and whitepapers. Following this HYDN carried out a detailed manual analysis of the code attempting various edge case attacks but remained unsuccessful in finding any vulnerabilities. HYDN found the code to be of a very high standard.

HYDN did find some areas of interest to be raised which are covered in more detail in the Detailed Findings section, but as an overview these are to do with the lack of orchestration and delegated responsibility. The protocol is designed with modular components, where each part functions independently but contributes to the overall system meaning that all parts are interconnected, maintaining their states independently.

If we look at the area of Component Creation, users can create everything or just specific parts (e.g., a Vault). This flexibility allows for the potential injection of a malicious component, such as a Vault and potential re-entrancy attempts. Whilst HYDN wasn't able to leverage this vulnerability into anything harmful, we still believe that this is an area worth raising with Azuro.

Project Summary

Delivery Date	May 24th 2024
Language	Solidity
Repository	https://github.com/Azuro-protocol/azuro-v2
Scope	https://github.com/Azuro-protocol/azuro-v2
Initial Commits	b30d964d9ab84a800c0d5195a29d858b1ae4b757
Latest Commit	TBC

Breakdown of Findings

ID	Title	Severity	Status
1	Delegated Responsibility	**Recommendation**	Outstanding
2	A Malicious User Can Create a Pool with a Self-Controlled Vault	**Recommendation**	Outstanding
3	Current Integration Leads To High Gas Costs	**Recommendation**	Outstanding
4	Make SDK More Developer Friendly	**Recommendation**	Outstanding

Detailed Findings

1 Delegated Responsibility [Recommendation]

Description	Broadly speaking, in HYDN's opinion, the weakest part of the Azuro system resides in the integration and design with a lack of orchestration and delegated responsibility. The system is designed with modular components, where each part functions independently but contributes to the overall system. All parts are interconnected, maintaining their states independently meaning there are various entry points for users, each leading to different parts of the workflow. This flexibility allows for the potential injection of a malicious component, such as a vault.
Impact	**Integration Weakness:** ● Vulnerabilities may arise not from smart contract security but from the integration of components. ● Different core versions could lead to inconsistencies if components do not perform the same actions. ● Although the end result appears correct for Azuro, there are potential vulnerabilities in how components integrate, especially if they end up not functioning uniformly.
Recommendation	HYDN would recommend Azuro to Refactor the codebase to do the following:

	Simplify On-Chain Logic: ● Reduce the complexity of the logic to lower the number of internal calls within a single transaction. This will help decrease gas costs and processing costs. ● Break down complex operations into smaller, more efficient functions to make the system more manageable. **Enhance State Management:** State Synchronization: ● Implement a more efficient method for sharing and updating the current state of contracts to prevent potential attack vectors. ● Consider using state channels or other off-chain solutions to manage state synchronization more securely and efficiently. **Delegate Responsibility:** Implement Delegated Responsibility: ● Ensure responsibilities are clearly defined and delegated within the code to improve maintainability and clarity. ● Restrict entry points to specific functions and areas of responsibility to avoid confusion and reduce the potential for errors. **Streamline Workflow:** Reduce Complexity: ● Analyze and streamline the workflow to reduce unnecessary steps and improve overall efficiency. By addressing these recommendations, the system can be made more efficient, secure, and user-friendly, especially for developers who wish to build on top of it using the SDK.

2 A Malicious User Can Create a Pool with a Self-Controlled Vault [Recommendation]

Description	Due to the design of the Azuro protocol covered above, a malicious user can create a pool with a self-controlled vault through factory.createPoolWithDeployedVault(). The SafeOracle contract holds funds from the resolver and disputer. The expected flow of resolution from the oracle entry point may be: 1. safeoracle.applyProposal 2. prematchcore.resolveCondition 3. lp.addReserve 4. vault.addLiquidity. Since all access control requirements can be met by an attacker who creates and controls the LP, Core, and (custom) Vault, at the vault.addLiquidity step, from SafeOracle's viewpoint, and its own storage, applyProposal can be re-entered. Due to Core and LP safety checks, the re-entered call would fail because from the LP and Core perspective, it is already resolved, and leaves the SafeOracle viewpoint back to the 'created' state due to the failure condition. Then the first re-entrant execution flow would set it as resolved in the SafeOracle.

Impact	Whilst users can re-enter a contract, they can only do this to one at a time and the platform relies on multiple contracts, meaning that while one part may be vulnerable alone, they are secure when linked together. At the end of the day, the state is consistent and whilst HYDN did not manage to leverage the issue into anything harmful, this still represents a potential vector of attack to be aware of and one that could potentially be open to malicious actors in the future.
Recommendation	

3 Current Integration Leads To High Gas Costs [Recommendation]

Description	The current design of the Azuro protocol leads to high gas costs. This is due to a probable need for flexibility and modularity, but it comes at the cost of high complexity and processing power required meaning the cost to execute functions is not always as efficient as it could be.
Impact	The current setup is expensive in terms of storage and execution as each module maintains its own state, and a typical flow requires multiple ping-pong calls between contracts resulting in high gas costs.
Recommendation	**Minimize Gas Costs:** Avoid unnecessary computations within smart contracts. **Example**: A full condition creation and lifecycle requires LP (& vault(, core, bet, oracle to store locally the same state, which is de facto duplicate.

4 Make SDK More Developer Friendly [Recommendation]

Description	One of Azuro's goals is for other developers to build dApps on top of the Azuro Protocol using the Azuro SDK. Currently, the SDK is very complex (reflecting the lack of orchestration and delegated responsibility), so whilst it is not necessary for an external developer to understand how all parts of the on-chain system is working, Azuro should provide a clearer explanation so that developers can build on top of this and understand what they are working with.
Impact	The current SDK and documentation is very complex and cannot be easily understood.
Recommendation	HYDN recommends simplifying the SDK and documentation. ● Ensure the SDK abstracts the complex on-chain logic, allowing developers to build without needing to understand the underlying blockchain mechanics. ● Provide clear and comprehensive documentation that explains how to use the SDK effectively, focusing on ease of understanding and practical examples. **Documentation Improvement:** ● Revise the current documentation to make it more accessible and less complex.

	• Include step-by-step guides, tutorials, and example projects that demonstrate common use cases and best practices. • Use visuals such as diagrams and flows to explain complex concepts more clearly.