Azuro V2 LiveCore Security Analysis

by Pessimistic

This report is public

January 25, 2024

Abstract

In this report, we consider the security of smart contracts of Azuro V2 project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of Azuro V2 smart contracts. We described the audit process in the section below.

The audit showed several critical issues: Logic error in binary search, Possible betting on already resolved conditions, Signature replay attack. The audit also revealed several issues of medium severity: Data for condition creation is not validated or signed, Insufficient documentation, New odds are not written, Overpowered role, Possible block of condition resolution, Possible out of gas falling. Moreover, several low-severity issues were found. All the tests passed.

During the audit, the developers independently discovered the critical No oracle validation issue.

After the initial audit, the codebase was updated.

The developers fixed the critical issues: Possible betting on already resolved conditions, Signature replay attack. They also resolved the issues of medium severity: Non-updated odds and Possible out of gas falling, partially fixed Logic error in binary search and fixed most of the low severity issues.

During the recheck, we updated the description of the Overpowered role issue of medium severity (added the point about relayers), and the developers provided the comment for the previous part of the issue. The number of tests and the code coverage increased.

The documentation has been updated, but is still insufficient.

Also, we discovered the new medium severity issue for the new contract of the scope: Possible DoS of the betting batch.

After the recheck #1, the developers provided comments for the medium severity issue Possible DoS of the betting batch and for the low severity issue.

For the recheck #2, the developers updated the codebase.

They fixed the critical Logic error in binary search issue and the medium severity issue Data for condition creation is not validated or signed. They also fixed several low severity issues. The number of tests and the code coverage increased.

At the end of December 2023, we separately checked the Possible block of condition resolution medium severity issue as it was highly dependent on the other part of the codebase. After the review, we have decided that the problem still persists in the code.

During the recheck, we discovered one new medium severity issue The conditionId of client and oracle can be different, several low severity issues, and one note.

After the recheck #2, the codebase was updated again.

The developers fixed the following medium severity issues: Possible block of condition resolution and The conditionId of client and oracle can be different. Also, they fixed all issues of low severity. New issues were not found. The number of tests increased.

The code coverage was unmeasured.

We are concerned about the complex interactions of contracts. It also needs to be clarified why the condition information is duplicated in two contracts: **ClientCoreBase** and **HostCore**. And an essential part of the functionality is off-chain. The project depends on the data that Oracle signs off-chain, which can lead to new issues.

General recommendations

We recommend improving documentation. We also recommend implementing CI to calculate code coverage, and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with Azuro V2 project on a private GitHub repository, commit 4916d2591507533c4ed4ab466975b0afc6b9c78b.

The scope of the audit included:

- **ClientCoreBase.sol**;
- **HostCore.sol**;
- **LiveCore.sol**;
- **IClientCondition.sol**;
- **IClientCoreBase.sol**;
- **IConditionState.sol**;
- **IHostCondition.sol**.

However, the new functionality also interacts with the rest of the project, the report for which (**"Azuro V2 Security Analysis by Pessimistic", October 19, 2023**) we did earlier and together with this report.

The documentation for the project included the following links: off-chain betting and the scheme of the project.

All 7 tests pass successfully. The code coverage is 85.29%.

The total LOC of audited sources is 862.

Codebase update #1

After the initial audit, the codebase was updated, and we were provided with commit [3c473bdd8f9bd31af46869f539f5c4e2e376135c](3c473bdd8f9bd31af46869f539f5c4e2e376135c).

The recheck included previous scope and the new **Relayer.sol** contract.

The developers have addressed several medium severity and most critical issues, along with adding comments or resolving all low severity issues.

Moreover, during our review, we encountered one new medium severity issue and several low severity issues.

All out of 10 tests passed. The code coverage was 89.69%.

Codebase update #2

After the first recheck, the codebase was updated, and we were provided with commit 200ebfec987201cfdfe3fc3e8672ba78d3e58c7e.

The developers fixed critical issue and most of the medium severity issues. The new issue of medium severity was found. All out of 15 tests passed. The code coverage increased to 92.8%

Codebase update #3

After the recheck #2, the codebase was updated, and we were provided with commit [f541648c1eb786c8a451fe32c230f24be32930ba](f541648c1eb786c8a451fe32c230f24be32930ba). The recheck included previous scope and did not include the new `Factory.plugIndieController` function.

The developers fixed two medium severity issues. All out of 18 tests passed. The code coverage was unmeasured.

Audit process

We started the audit on October 9 and finished on October 19, 2023.

This scope interacts with the rest of the project, the report for which (**"Azuro V2 Security Analysis by Pessimistic", October 19, 2023**) we did earlier and together with this report.

Firstly, we contacted the developers for an introduction to the project. Then we inspected the materials provided for the audit and started the review.

During the work, we stayed in touch with the developers and discussed confusing or suspicious parts of the code.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- Whether messages are signed correctly by the oracle;
- Whether the storage in the contracts does not break when adding/removing variables;
- Whether the accounting of the liquidity locking works correctly;
- Whether bets cannot be placed when the outcome of the event is known;
- Whether the code matches the documentation;
- It is not possible to transfer some data bypassing oracle.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

From November 14 to November 16, 2023, we performed the first recheck. This involved a validation of fixes within the prior scope. We also reviewed the **Relayer** contract to determine if the execution of a betting batch could not be reverted.

We ran the tests, calculated the code coverage, re-scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- Semgrep rules for smart contracts (it was launched for **Relayer** as for a new contract). We also sent the results to the developers in the text file.

As a final step, the report was updated.

On November 28, we updated the report again, adding comments from developers to issues M07, L08.

We made the second recheck from December 5 to December 6, 2023. We checked whether the developers fixed the mentioned issues. Additionally, we reviewed M05 during the second part of the recheck that took place from December 25 to December 27, 2023, as it is highly dependent on the other parts of the project. However, we do not consider the issue as fixed.

We ran the tests, calculated the code coverage, re-scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;

At the end, we updated the report again.

During January 22-24, we made the third recheck. We checked whether the developers fixed the remain issues. We ran tests, calculated the code coverage and updated the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

C01. Logic error in binary search (fixed)

The implementation of the `_binarySearch` function of the **LiveCore** contract attempts to find the first snapshot that satisfies the equation `timestamp >= settledAt`. However, in several cases, the result is incorrect. For instance, when a `settledAt` value is provided that is less than any entry in the array, it will not return the first time from the array.

The example of such input. Assume that `settledAt = 50` and `timestamps = [100, 200]`. In this case, the implementation returns `200`, which counts bets made at `block.timestamp == 100` as valid. However, the correct response should be `100`.

This issue can lead to inaccurate reserve calculation at line 198 and incorrect locked liquidity. This will have the effect that the liquidity withdrawal throughput will be reduced in the **LP** contract. Since the value of `topNodeAmount - lockedLiquidity` code will be smaller than it actually is at line 326. Also, the difference between the value of the `lockedLiquidity` variable and the real value will be stuck in the **LP** contract.

Also, profits will not be calculated correctly in the `ClientCoreBase._resolveCondition` at line 299. And at the end of the condition, there will be a discrepancy in how much lp providers and bettors have earned or lost, because all bets after `settledAt` timestamp will be returned at line 204 of the **ClientCoreBase** contract.

The issue has been fixed and is not present in the latest version of the code.

C02. No oracle validation (fixed)

In the existing implementation, the `oracle` parameter is provided by the caller, and there is no validation of the data. This allows anyone to sign arbitrary data and place bets with any `odds` on any `outcome`, potentially resulting in losses for the protocol at line 50 in `putBet` function of the **LiveCore** contract.

> *This issue has been found and fixed by the developers during the audit.*

C03. Possible betting on already resolved conditions (fixed)

The logic of the **LiveCore** and **ClientCoreBase** contracts considers bets made at the `settledAt` timestamp as valid. Since only `bet.timestamp > condition.settledAt` results are paid with the initial bet at line 204 of the **ClientCoreBase** contract. However, `LiveCore.putBet` does not validate that the condition is already settled through `settledAt`. This can lead to a scenario like the following:

1. Assume `settledAt == block.timestamp`, signifying that the condition was settled at the current `block.timestamp`;
2. The oracle sends a `resolveCondition` transaction to resolve the condition;
3. An exploiter spots the transaction in the mempool and places a bet on the already known winning outcome (they obtained oracle signatures for all outcomes a bit earlier, and the signatures are still valid);
4. The bet is considered valid because its timestamp matches `block.timestamp`, which equals `settledAt`;
5. The exploiter profits from the known result and claims their winnings by calling `LP.withdrawPayout`.

The issue has been fixed and is not present in the latest version of the code.

C04. Signature replay attack (fixed)

The message does not specify and does not validate the `chainId` in the `_verifySignature` function of the **ClientCoreBase** contract. If the same oracle signs messages for different networks, it is possible to use the same signature across multiple networks, for instance, with the most favorable odds in the `putBet` function of the **LiveCore** contract.

The issue has been fixed and is not present in the latest version of the code.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Data for condition creation is not validated or signed (fixed)

The initial user who initiates a bet in the **LiveCore** contract provides data `data.conditionData` for condition creation, but this data is not validated or signed by the oracle. This oversight allows for the manipulation of parameters such as setting `winningOutcomes` to zero or toggling the `isExpressForbidden` parameter of the condition as needed.

Such actions can disrupt the betting functionality for other users, necessitating the recreation of conditions, which is particularly undesirable for live betting scenarios.

Moreover, it allows to bypass check at line 103 as you can provide any `payoutLimit` and `winningOutcomes` parameters at line 79 in the `putBet` function.

The issue has been fixed and is not present in the latest version of the code.

M02. Insufficient documentation

The project has the documentation. However, the descriptions of the **LiveCore**, **HostCore** contracts are brief and for the **ClientCoreBase** there is no description at all. There are obscure points in the code that the documentation could help with:

1. It is not clear how the **ClientCoreBase** uses `PAUSED` state of condition and `stopCondition` function.
2. In the description of the **HostCore** it is said that there is an event type management: pre-match, live. But it is not present in the code.
3. Why the **HostCore** contract updates the game start without checking with `block.timestamp`;
4. etc.

In general, some issues may have arisen due to confusion about how the project works.

The codebase is covered with NatSpec comments. It also contains useful regular comments. However, they provide only a brief description of functionality and do not explain the overall project structure.

Proper documentation should explicitly explain the purpose and behavior of the contracts, their interactions, and the main design choices.

The documentation has been updated but is still insufficient.

M03. Non-updated odds (fixed)

The `changeOdds` of the **HostCore** contract does not add new odds to the storage, it only checks to 0. As a result, all bets will be made at the initial odds.

The issue has been fixed and is not present in the latest version of the code.

M04. Overpowered role (commented)

In the current implementation, the system depends heavily on the off-chain oracle and other on-chain roles. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if admin's private keys become compromised or if the oracle provides and signes incorrect data.

- The on-chain roles can:
    - **(commented)** Cancel/resolve/stop condition or update the beginning of the game in the **HostCore** contract at any time;
    - **(commented)** Change odds in the **HostCore** contract to more favorable ones (However, now it is not possible because of Non-updated odds issue);
    - **(commented)** Create and resolve condition with 0 `winningOutcomes` inside `HostCore` contract;
    - **(commented)** Re-create the condition in the **HostCore** contract at any time.

  The entire project depends on the correctness of the information in this contract. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.
- **(commented)**The off-chain oracle can sign the wrong information for betting and creating a condition. For example, there may be inflated odds.
- The `core` and `lp` parameters are initially defined by the owner within the **Relayer** contract. If another core accepts `IClientCoreBase.ClientBetData`, the relayer has the ability to alter the core to a more advantageous option.

  For instance, the relayer has the capability to input the preferred core by adjusting the odds, subsequently placing bets with more favorable odds on the opposite outcome.

> *Comment from the developers: The oracle's role in shaping and managing real-world events (sports matches) remains the same in this approach (**HostCore**). Several access groups (**Access**) are issued to oracles that broadcast real world events to the blockchain.*

M05. Possible block of condition resolution (fixed)

When the `_rollback` function of the **LiveCore** contract is invoked, it updates locked liquidity at line 204 to match the most recent valid snapshot. However, these calculations might necessitate an increase in locked liquidity.

It is possible that there may not be enough liquidity for different conditions within the same core (since `reinforcementAbility` is set for the whole core), and as a result, the condition resolving will be reverted.

Consider improving the management of condition resolution and maintaining sufficient liquidity to cover possible bettor costs.

The issue has been fixed and is not present in the latest version of the code.

M06. Possible out of gas falling (fixed)

The length of the `snapshotTimes[conditionId]` array in the **LiveCore** contract can be increased with each block by making a new bet. You can significantly expand the array, making its deletion expensive or potentially impossible to fit within a block. Deletion of an array at line 209 in the `_rollback` function involves clearing both its length and all its individual elements, which is why it can be cost ineffective.

The issue has been fixed and is not present in the latest version of the code.

M07. Possible DoS of the betting batch (commented)

The batch execution of bets in the **Relayer** contract can be halted due to the cancellation of approval for a particular bet within the batch. Consequently, other bets may expire. This situation may create a profitable opportunity for someone who first submits a fake transaction that will cancel the batch and then sends their transaction with a bet on a favorable outcome with desired odds.

> *Comment from the developers:*
>
> *No. Every bet order pass throw oracle approvement, motivated to approve successfull bet and not to pass incorrect or expired.*
>
> *Relayer can execute bet orders as he will by one or by batch. There are no "race condition" in executing bet order. Every bet order can be executed before expiration time or not. Relayer rewarded only by execution, so he is very motivated to execute bet in any order as soon as possible. Order execution of approved bet does not matter.*

> *Pessimistic's comment:*
> *There remains an issue where a bettor can front-run the `Relayer.betFor` function and cancel their approval. Nevertheless, a relayer can execute one order after another sequentially.*

M08. The conditionId of client and oracle can be different (fixed)

The bettor signs the data, including the `conditionId` to call the `putBet` function of the **LiveCore** contract. However, at line 153 (where the bet data is written to the storage), the `conditionId` is taken from the `conditionData` variable, which is signed by the oracle. These identifiers are not checked anywhere to see if they are equal. Oracle can specify the `conditionData` it wants. This will cause the user to bid on the wrong condition.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Overcomplicated modifier (fixed)

In the `resolveConditionBase` modifier of the **ClientCoreBase** contract, there is complex logic which calculates the reserves, updates the storage.

Consider simplifying the logic, this will avoid unnecessary errors and improve the readability of the code.

> *Comment from the developers: The modifier is needed as a common design element for **CoreBase**-based Betting Engines.*

The modifier has been removed from the code.

L02. Duplicate check (fixed)

The `cancelCondition` function of the **HostCore** contract verifies that condition is neither canceled nor resolved. This check at the 41 line duplicates the check at lines 264-265 inside the same transaction.

Additionally, the check at lines 264-265 can be replaced with `_isConditionCanceled` and `_isConditionResolved` functions of the **HostCore** contract.

The issue has been fixed and is not present in the latest version of the code.

L03. No disable initializer (fixed)

According to OpenZeppelin NatSpec comments, it is advisable to call the `_disableInitializer` function inside the constructor. Consider following it in the **HostCore** and **LiveCore** contracts.

The issue has been fixed and is not present in the latest version of the code.

L04. Unused PAUSED state (fixed)

Consider excluding the `stopCondition` function since the `PAUSED` state is not used in the **ClientCoreBase** and **LiveCore** contracts. The same functionality is available in the **HostCore** contract.

The issue has been fixed and is not present in the latest version of the code.

L05. No validation for existence of condition (fixed)

The `createCondition` and `stopCondition` functions of the **HostCore** contract do not verify the existence of the condition. This opens the possibility, to overwrite existing condition or stop not existing one.

In this case it does not lead to anything serious, since the functions can only be called by certain roles.

The issue has been fixed and is not present in the latest version of the code.

L06. Possible setting of invalid data (commented)

In the `updateGame` function of the **HostCore** contract it is possible to establish a new `startsAt` value that is earlier than the current `block.timestamp`, which is an unexpected behavior.

In this case it does not lead to anything serious, since the functions can only be called by certain roles.

> *Comment from the developers: The data providers beginning the match follow the current practice of sending data with a delay.*

L07. No validation for existence of condition - part 2 (fixed)

The `cancelConditions` and `resolveConditions` functions of the **HostCore** contract do not verify the existence of the condition. This opens the possibility, to cancel or resolve not existing condition.

In this case it does not lead to anything serious, since the functions can only be called by certain roles.

The issues have been fixed and are not present in the latest version of the code.

L08. Identical parameters (commented)

The `minOdds` parameter from `IBet.BetData` duplicates a similar field from the **ClientBetData** structure of the **IClientCoreBase** interface. However, only `minOdds` value from `ClientBetData` is used in the **LiveCore** contract.

> *Comment from the developers: `IBet.BetData` is common protocol calldata structure (used in `lp._bet()`), used only as calldata for passing core specific data for different protocol cores. For Live core used `IBet.BetData` - `bytes data; // core-specific customized bet data`, and `IBet.BetData.minOdds` not used.*

L09. Gas consumption (fixed)

`snapshotTimes[conditionId]` is read multiple times from the storage in the `_rollback` function of the **LiveCore** contract. Consider reading it once to a local variable to reduce gas consumption.

The issue has been fixed and is not present in the latest version of the code.

L10. Public -> external (fixed)

Consider declaring functions as `external` instead of `public` in the `getOutcomeIndex` function of the **HostCore** contract to improve code readability and optimize gas consumption.

The issue has been fixed and is not present in the latest version of the code.

L11. Unused imports (fixed)

The following imports in the **ClientCoreBase** contract are not used:

- `@openzeppelin/contracts/utils/cryptography/MerkleProof.sol`;
- `./libraries/CoreTools.sol`.

The issue has been fixed and is not present in the latest version of the code.

Notes

N01. OnlyOwner modifier is never used (commented)

The **HostCore** contract inherits from the **OwnableUpgradeable** contract, even though it does not utilize any ownership-related functionality. It may be beneficial to consider removing the inheritance of the **OwnableUpgradeable** contract, which would reduce the gas consumption and code size.

> *Comment from the developers: Used for settings.*

This analysis was performed by Pessimistic:

Daria Korepanova, Senior Security Engineer
Oleg Bobrov, Junior Security Engineer
Irina Vikhareva, Project Manager

January 25, 2024