Azuro V2 Security Analysis

by Pessimistic

This report is public

February 17, 2023

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

Summary

In this report, we considered the security of Azuro V2 smart contracts. We performed our audit according to the audit process described below.

The audit showed several issues of medium severity, including Lost rewards for paused conditions, Bug, and Overpowered role. Moreover, several low-severity issues were found.

After the initial audit, the codebase was updated. The developers fixed two medium severity issues such as Lost rewards for paused conditions, Bug and provided the comment for Overpowered role issue. Also, the developers updated the profit and reward accounting system and added new tests.

After the first recheck, the codebase was updated. We did not find any fixes and discovered two new issues of low severity. Also, the developers increased the number of tests.

After the second recheck, the codebase was updated again. The developers added two new contracts: **SafeOracle** and **Access**, and changed the previous codebase.

The audit showed the critical Locked funds issue, the Uncounted scenario of the condition cancelation issue of medium severity, and several low severity issues. The number of tests increased.

After the third recheck, the codebase was updated. The developers fixed the critical Locked funds issue, Uncounted scenario of the condition cancelation of medium severity issues, and several issues of low severity. We discovered one note and one issue of low severity that was immediately fixed. The number of tests decreased.

The implementation of the logic in the **SafeOracle** contract became simpler. Different scenarios were divided into separate functions.

After the recheck #4, the codebase was updated again.

During this recheck, we updated the Overpowered role, CEI pattern violation issues and found several low severity issues. The number of tests and the code coverage increased.

The documentation for the project is insufficient.

General recommendations

We recommend fixing the mentioned issues and improving documentation.

Project overview

Project description

For the audit, we were provided with Azuro V2 project on a private GitHub repository, commit 2a27165e45168472e07837a432f33fecb5e9e59d.

The scope of the audit included the whole repository except the **LiveCore** contract.

The documentation for the project included the following links:

- https://docs.azuro.org/azuroprotocol/
- https://azuro-protocol.notion.site/Live-betting-b032972dedce4a568aa76039ba6ad4c6

All 94 tests pass successfully. The code coverage is 97.39%.

The total LOC of audited sources is 2540.

Codebase update #1

After the initial audit, the codebase was updated. For the recheck, we were provided with commit 8b28b0419b04ec30392933a8477403cef918125a.

The scope of the audit included the whole repository except the **LiveCore** contract.

This update included fixes for two issues of medium severity and for most of the low severity issues. It also contained the update of accounting functionality. The number of tests increased. All 125 tests passed.

Codebase update #2

After the recheck #1, the codebase was updated and we were provided with commit 2c56905e04a1945b8104afcc33466677562b1774.

The scope of the audit included the whole repository except the **LiveCore** contract.

This update did not include any fixes. However, it contained code improvements. The number of tests increased. 130 tests out of 142 passed, 12 remaining skipped. During the audit, we found two issues of low severity.

Codebase update #3

After the recheck #2, the codebase was updated, and we were provided with commit 2aa6d44d32b815a0eb904fddffca58ed913e96c4.

The audit scope included the **SafeOracle**, **Access** contracts, and the code updates from the previous recheck except for the **FreeBet** and **LiveCore** contracts. During the audit, we found one critical issue, one issue of medium severity, and a few issues of low severity.

The number of tests increased. All 244 tests passed. The code coverage was 97.4%.

Codebase update #4

After the recheck #3, the codebase was updated, and we were provided with commit 4b3b331a825937ab1040a4cd895030350f34f6f5.

The recheck scope included the code updates of all contracts except for the **FreeBet**, **LiveCore**, **BetExpress** contracts and the `plugExpress` function of the **Factory** contract. For the recheck, the developers fixed one critical and one medium severity issue, as well as several low severity issues. However, we found one low severity issue that was immediately fixed and one note. The developers also provided the **SafeOracle** contract scheme.

228 tests out of 258 passed, and 30 tests had pending status. The code coverage was 93.4%.

Codebase update #5

After the recheck #4, the codebase was updated, and we were provided with commit 0a204cca509a7b6be433d5fcb788e7cb2a88efb3.

The recheck scope included the code updates of all contracts except for the **FreeBet**, **LiveCore**, **BetExpress** contracts and the `plugExpress` function of the **Factory** contract.

We updated one medium and low severity issues. Also, we found several low severity issues.

248 tests out of 278 passed, and 30 tests had pending status. The code coverage was 97.7%.

Audit process

The audit began on August 23 and ended on September 09, 2022.

We previously audited the first version of this project. So we contacted the developers to learn about the current changes and identify contracts and features that needed additional attention during the audit:

- Gradual lock of liquidity reserves. Liquidity did not have to be lost completely due to manipulations of odds or bet sizes.
- New logic for awarding affiliates. They needed to have a common incentive with the protocol to win. Rewards had to be correctly counted for regular bets.
- Correct calculation of payments to bet participants in case of winning, game or event cancellations, etc.

During the work, we kept in touch with the developer and figured out all the obscure points.

In our audit, we considered the following crucial features of the code:

- Whether the code was secure.
- Whether the code corresponded to the documentation.
- Whether the code met best practices.

All contracts from a given scope was fully analyzed manually, and their logic was checked. Some of the contracts properties we checked during the audit:

- **AzuroBet** and **LP** tokens complied with `ERC1155` and `ERC721` standards.
- Gas usage was optimized, especially with storage operations.
- Standard solidity checks.
- It was not possible to place a bet after condition resolution or to front-run a condition resolution transaction.
- The bet had to be returned when the condition or game was cancelled. It was safe to transfer a bet to someone else, and the reward would be taken by the current owner of the token.
- Affiliate rewards were calculated correctly for regular bets.
- The calculation of the odds was correctly implemented, taking into account the margin.
- It was impossible to unbalance odds so as to take away all the locked Liquidity.
- Profit calculation was correct for condition resolution.

We scanned the project with slither-analyzer and manually verified all the occurrences found by it.

To check logical conditions, we created and ran Echidna properties. Logical properties that we checked using fuzzing:

- Assert in the `updateContribution` function of **AffiliateHelper** library never happened regarding smart contract functions.
- The identity of a sum of coefficients and the margin identity never failed.

After the initial audit, we received a new commit with the updated codebase.

We checked if the issues from the initial audit were fixed. We also inspected whether the logic of the updated functionality, which was responsible for accounting payouts, bets, blocked liquidity, calculation of odds, was implemented correctly.

After the first update, we had a new commit again with the following updated codebase.

First, we checked the fixes for the remaining issues from the previous update. After that, we reviewed the new improvements to the source code and checked whether the previous logic was preserved.

After the audit, the developers fixed mentioned new issues.

The next update of the codebase that we received included the changes to the previous scope and two new contracts: **SafeOracle** and **Access**. The audit began on December 13 and ended on December 21, 2022.

During the audit, we contacted the developers to clarify unclear parts of the code and ambiguous logic. We also did the following checks:

- Whether all changes were correct;
- If there were any problems at the junction of the new functionality and the previous one;
- Whether all possible scenarios in SafeOracle were implemented;
- Whether insurance by oracle or disputer could be locked into the SafeOracle contract;
- Whether there were collisions when adding roles to Access.

Also, we scanned the project with slither-analyzer and manually verified all the occurrences found by it.

The next update of the codebase that we received included the changes to the scope from the previous update. The recheck of the **Access** contract was on December 28, 2022. And the recheck of the residual scope began on January 9 and ended on January 11, 2023.

We checked the fixes and ran through possible scenarios in the updated version of the **SafeOracle** contract.

Also, we scanned the project with slither-analyzer and manually verified all the occurrences found by it.

Since we did not find anything dangerous, but only one low severity issue and a note, we suggested that the developers fix them or comment on them.

In the recheck #5, we checked changes to the scope from the previous update. We also added several low severity issues to the report and updated the Overpowered role and CEI pattern violation issues.

Also, we scanned the project with slither-analyzer and manually verified all the occurrences found by it.

Finally, we 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. **Locked funds** (fixed)

The insurance may be locked in the **SafeOracle** contract.

If the condition is already created and then, at some point, the game is canceled via `LP.cancelGame`, the following may happen in `SafeOracle.applyProposal` function:

1. The `if (success)` case is impossible since the external call to the core at line 216 is not successful due to the "whether the game is canceled" check inside the `CoreBase._resolveCondition` function.
2. In the `else if (disputed)` case the `CoreBase.cancelCondition` function also reverts with the `ConditionAlreadyResolved` error since there is a check inside that the game is canceled.
3. In the `else` case, it seems that everything is ok, and the owner is able to provide a solution via the `_resolve` function of the **SafeOracle** contract. However, the function reverts with the `IncorrectSolution` error since the external call with the solution to the core is not successful.

As a result, it may happen that insurance of oracle will be locked in the **SafeOracle** contract since balances will not be updated. The same may happen to insurance of a disputer.

The issue has been fixed and is not present in the latest version of the code.The developers added the `handleCanceledCondition` function for this scenario.

Medium severity issues

Medium 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. Lost rewards for paused conditions (fixed)

In **Core** contract, `resolveAffiliateReward` function iterates through `conditions` mapping and deletes even those which have `PAUSED` state. Thus, even if these conditions change their state, affiliates will not be able to collect rewards from them.

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

M02. Bug (fixed)

In **AzuroBet** contract, `_addTokenToOwnerEnumeration` function adds a new element to `_ownedTokens[to]` array and increases `_ownedTokensCount[to]` value by `1`. `_removeTokenFromOwnerEnumeration` function is considered to perform the opposite operation. It deletes the last element from `_ownedTokens[from]` array, but it only subtracts `1` from `_ownedTokensCount[from]` value without decreasing the variable at line 193.

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

M03. **Overpowered role** (commented)

The system relies heavily on special roles:

1. `Maintainer` role can:
    - Stop and cancel conditions;
    - Change max banks ratio;

2. `Oracle` role can:
    - Cancel conditions;
    - Reject batches;
    - Set batch blocks;
    - Cancel games;
    - Shift the moment when a game starts;

3. `Owner` role can:
    - Update the beacon of **Core**;
    - Update core in **LP** contract;
    - Update the address of `Maintainer` role;
    - Change fees for factory's owner, affiliates, and oracle;
    - Change the value of the minimum deposit;
    - Change the reinforcement ability and the timeout of withdraws;
    - Change dispute period and insurance of the **SafeOracle** contract. And zero check of dispute period is not enough since at least one block has to pass (it was added at the commit `0a204cca509a7b6be433d5fcb788e7cb2a88efb3`).

We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Comment from the developers:* ***PoolFactory*** *owner is a DAO. Anyone can create his own pool of liquidity. Therefore, the risks associated with large* `Oracle` *and* `Maintainer` *powers are the responsibility of the pool creator.*

> *The update at the commit* `2aa6d44d32b815a0eb904fddffca58ed913e96c4`*:*
> *All roles are now granted and managed through the* ***Access*** *contract.*

> *The update at the commit* `0a204cca509a7b6be433d5fcb788e7cb2a88efb3`*:*
> *The new functionality of the* ***SafeOracle*** *owner was added to this issue since it can be used in front-running. However, according to NatSpecs the owner is a DAO.*

M04. **Insufficient documentation**

The project has documentation. However, it does not explain crucial behavior of the system, including:

1. when a bookmaker should stop accepting bets;
2. when the condition resolution begins;
3. how liquidity is gradually locked in reserves;
4. how the rewards distribution between affiliates occurs;
5. whether an affiliate can go into deficit;
6. how odds are recalculated;
7. etc.

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.

M05. **Uncounted scenario of the condition cancelation** (fixed)

The `applyProposal` function of the **SafeOracle** contract has a mechanism to automatically cancel the condition if the owner does not send their decision or the oracle has provided the wrong solution. Condition is canceled only in case of a dispute and after `DECISION_PERIOD` has passed. No automatic cancelation occurs if there is no dispute, oracle`s solution does not pass, and the owner does not provide its decision.

This may lead to the fact that some liquidity providers are not able to withdraw their tokens since the condition status is not `RESOLVED` or `CANCELED`. It means that the total value of locked liquidity does not decrease, and therefore the number of available tokens for withdrawal is the same as before the creation of the condition.

The issue has been fixed and is not present in the latest version of the code.The developers added the `applyCancelCondition` function for this scenario.

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. Precision loss (fixed)

Performing division operations before multiplication can result in a loss of precision. Consider dividing in the last step in `marginAdjustedOdds` function of **CoreTools** contract at line 49.

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

L02. Redundant check (fixed)

In **LP** contract, the condition `lockedReserve >= 0` at line 604 always evaluates to `true` since the type of `lockedReserve` variable is `uint128`.

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

L03. Name shadowing (fixed)

In **AzuroBet** contract, `owner` parameter of `tokensOfOwner` and `tokenOfOwnerByIndex` functions shadows the name of `owner` function of **OwnableUpgradeable** contract.

The name of the local variable should not duplicate the name of the function or storage variable.

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

L04. Functions visibility (fixed)

Consider declaring functions as `external` instead of `public` when possible to improve code readability and optimize gas consumption.

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

L05. CEI pattern violation (commented)

The [CEI (checks-effects-interactions) pattern] is violated in:

1. In **Factory** contract, `createPool` function updates `registeredLPs` storage variable at line 71 after an external call.
2. In the `addLiquidity` and `addLiquidityNative` functions of the **LP** contract there are external calls at lines 240, 254 before storage update (it was added at the commit `0a204cca509a7b6be433d5fcb788e7cb2a88efb3`).

We highly recommend following CEI pattern to increase the predictability of the code execution and protect from some types of re-entrancy attacks.

> *Comment from the developers: The external calls specified in the report are addressed to contracts that are part of the protocol, whose behavior is predictable.*

> *The update at the commit `0a204cca509a7b6be433d5fcb788e7cb2a88efb3`:*
> *The new case from the **LP** contract was added.*

L06. Code duplication (fixed)

Core and **CoreTools** contracts declare the same `getTokenId` function. The function of **Core** contract is not used.

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

L07. Redundant code (fixed)

In **LiquidityTree** contract, internal removeLimit function is never used.

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

L08. Extra reads from storage (fixed)

Reading the same value from storage multiple times is expensive. Consider writing it to a local variable to optimize gas consumption:

- In **Core** contract, `_maxPayout` function reads `payouts[0]` and `payouts[1]` fields of `Condition` struct multiple times.

  > *The `_maxPayout` function has been removed from the latest version of the code.*

- In **AffiliateHelper** library, `updateContribution` function reads `totalNetBets` and `payouts[i]` fields of **Contribution** struct from storage multiple times at lines 59, 60, 63, and 64.
- In **Core** contract, `_cancel` function reads `state` field of **Condition** struct from storage multiple times at lines 357, 358.

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

L09. Dependency management (commented)

hardhat.config.js uses inadequate runs value. Consider increasing this value since your contracts will be called multiple times.

> *Comment from the developers: Closer to the launch on the main network, we will increase the runs parameter to the limit of the bytecode size*

L10. Unused function (fixed)

The `_asSingletonArray` function of **AzuroBet** contract is never used. Consider removing redundant code.

The issue has been fixed with pull request 4ff3a37ead1d55b382568692fac49e03dff3f33e.

L11. Excessive increase of the index (fixed)

The condition at line 240 in `tokenOfOwnerByIndex` function of **AzuroBet** contract does not work properly in some cases since the index is increased by 1.

The issue has been fixed with pull request e75d4c197f7ed6231ee6e420486ba234a3e4b96f.

L12. Overcomplicated modifier

In the modifier `resolver` in **SafeOracle** contract, there is complex logic that depends on a `msg.sender`. Consider simplifying the logic, this will avoid unnecessary errors and improve the readability of the code.

L13. Indexed arguments (fixed)

Using indexed properties for events is useful when they are considered for further search. The following events have no indexed properties:

- `Created`, `Disputed`, `Proposed`, and `Resolved` events in `ISafeOracle` interface.
- `RoleAdded`, `RoleRenamed`, `RoleBound`, `RoleUnbound`, `RoleGranted` and `RoleRevoked` events in `IAccess` interface.

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

L14. Memory vs calldata (fixed)

Consider using `calldata` instead of `memory` for arguments in external functions to save gas at lines 32, and 74 in **Access** contract.

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

L15. Contradictory check (fixed)

In **AzuroBet** contract, at line 302, according to the message inside the `require`, `amount > 0` should be checked instead of `fromBalance > 0`.

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

L16. Incorrect role (fixed)

In `resolveCondition` function of **Core** contract, `this.resolveCondition.selector` should be passed to `restricted` modifier, instead of `this.createCondition.selector`.

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

L17. Missing check (fixed)

In **Access** contract lines 34 and 78 convert `string` variable `name` to `bytes32` type. Since `bytes32` can only contain 32 bytes, in case when name is longer than that, part of the name will not be stored. We recommend checking name length.

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

L18. Code logic (fixed)

In case when owner wants to revoke certain role from the user, they need to call the `burn` function from the **Access** contract with the corresponding `tokenId`. However, in the current implementation there is no easy way to determine which token is held by the user. We recommend adding **ERC721EnumerableUpgradeable** contract of OpenZeppelin to the inheritance list.

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

L19. Missing check of condition status (fixed)

In **SafeOracle** contract, `_resolve` function theoretically can be invoked when condition already had `RESOLVED` status. The correctness of that scenario is guaranteed by the external modules: **CoreBase** contract should revert the transaction that tries to resolve condition for the second time. Nevertheless, we recommend adding check for the condition status since other modules can be updated in future versions of the code.

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

L20. Redundant code (fixed)

In **SafeOracle** contract, line 230 is redundant since it is reachable only in case when condition status is `CREATED`.

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

L21. Misleading comment (fixed)

The `handleCanceledCondition` function of **SafeOracle** contract has the NatSpec comment, "If proposed solution is disputed, the disputer receives the oracle's stake" at line 285. However, according to line 304, the disputer only receives their part back.

The issue has been fixed at the commit e76a3940d0c883f1ef432178bde34709554a5ffa.

L22. View function (new)

The `addCondition` function of the **LP** contract does not change the storage.

Consider declaring this function as `view`.

L23. Initialization of the implementation contract (new)

The common pattern for upgradeable contracts is to move all the contract initialization into `initialize` function. This allows to initialize proxy storage. However, there is an attack vector that exploits `initialize` function by making a call directly to the logic contract. The current implementation is safe from this type of attack. Nevertheless, we recommend preventing logic contract initialization in order to make future versions of the code more secure.

L24. No liquidity accessibility check (new)

The `updateCoreSettings()` function of the **LP** contract resets a `reinforcementAbility` of the specified core. In case the `reinforcementAbility` decreases, there is no guarantee that there is enough reserves for already locked liquidity. Consider checking that `reinforcementAbility*reserves >= coreData.lockedLiquidity`.

Notes

N01. Different roles with the same names

In `addRole` and `renameRole` functions of **Access** contract, it is not checked if the role with the new name already exists. Thus, it is possible to create different roles with the same role names.

N02. Assert

The `applyCancelCondition` function of **SafeOracle** contract has `assert` at line 277. According to the Solidity documentation: "assert should only be used to test for internal errors, and to check invariants". In this case, we can consider it as an invariant check in the complex project. However, the best practice is to replace it with `require`.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Daria Korepanova, Security Engineer
Ivan Gladkikh, Security Engineer
Pavel Kondratenkov, Security Engineer
Nikita Kirillov, Security Engineer
Yhtyyar Sahatov, Junior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

February 17, 2023