Azuro V2 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 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.

After the recheck #5, the codebase was updated again.

The developers added new functionality in this update and fixed almost all low-severity issues. However, we discovered two new medium severity issues: Unclear formula and Insufficient parameter validation. The number of tests increased, as well as the code coverage percentage.

After the recheck #6, the codebase was updated again.

The developers successfully resolved the majority of issues encountered in the previous update, including Unclear formula and Insufficient parameter validation, and also introduced new functionality. During our assessment, we identified a new issue of medium severity known as the M08, along with a few minor issues of low severity. There was a notable increase in the number of tests. However, it is worth noting that the documentation for the project still needs to be improved.

After the recheck #7, the codebase was updated again.

The medium severity M08 issue was found to be a false positive and removed. Low severity issues were addressed, but new ones of the same severity were discovered. We also updated the Overpowered role issue to accommodate new functionality and found Signature replay attack of medium severity.

The total number of tests increased, but specifically for this scope decreased. Therefore, the code coverage has decreased.

After the recheck #8, the codebase was updated again.

The developers fixed the medium severity issue Signature replay attack and most of the low severity issues. The number of tests and the code coverage increased.

After the recheck #9, the codebase was updated again.

The developers introduced new changes that isolated the core code of the liquidity management part of the project. During the review we have found four new medium severity issues, several low severity issues and one note. Moreover, a new point was added to Overpowered role issue. The number of tests increased, and the code coverage decreased.

After the recheck #10, the codebase was updated again.

The developers fixed the following issues of medium severity:
Losses and profits can be applied to the whole tree,
No revert when there is not enough liquidity in the tree,
No push of changes before the tree traversal and
Incorrect profits/losses distribution may occur. New issues were not found. L35 issue of low severity was removed as false positive.

The number of tests increased, the code coverage was unmeasured.

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 issue. 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%.

Codebase update #6

After the recheck #5, the codebase was updated, and we were provided with commit 1871a139d354f4b77180d73c8a56afc9a2c75c3.

The developers added new functionality and a new contract **StakingPool**. The scope excluded the following contracts: **FreeBet**, **LiveCore**, **ProxyFront**, **ProxyOracle** , **BetExpress** - there is a separate report for the **BetExpress** contract. They also fixed almost all low-severity issues. We discovered two medium-severity issues. All 280 tests passed, and 30 tests had pending status. The code coverage is 97.8%.

Codebase update #7

For this update, we received a commit a3259c65e58090265ed5c6f7f7842e697e3c91bd of the contracts GitHub repository for review.

Following the last recheck, the developers implemented new features in the existing contracts and introduced a new **FreeBet** contract. The scope of the audit included the following contracts:

- **CoreBase.sol (libraries/CoreTools.sol, libraries/Math.sol)**;
- **PrematchCore.sol**;
- **LP.sol**;
- **extensions/ProxyFront.sol**;
- **StakingPool.sol**;
- **FreeBet.sol**.

All out of 332 tests successfully passed, and 30 tests were pending. The code coverage was 97.16%.

Codebase update #8

After the recheck #7, the codebase was updated, and we were provided with commit 4916d2591507533c4ed4ab466975b0afc6b9c78b.

The recheck included previous scope except the **SafeOracle.sol** and **ISafeOracle.sol** contracts.

The developers resolved the majority of low severity issues. However, during the recheck process, several additional low severity issues were discovered. Furthermore, one medium severity issue from the previous rechecks was updated, another new one was added to the report, and another medium severity issue was determined to be a false positive and subsequently removed.

All out of 333 tests successfully passed, and 1 test was pending. The code coverage was 95.79%.

Codebase update #9

After the recheck #8, the codebase was updated, and we were provided with commit 3c473bdd8f9bd31af46869f539f5c4e2e376135c.

The developers fixed one medium severity issue and the majority of low severity issues.

All out of 337 tests successfully passed, and 1 test had pending status. The code coverage was 96.07%.

Codebase update #10

After the recheck #9, the codebase was updated, and we were provided with commit 200ebfec987201cfdfe3fc3e8672ba78d3e58c7e.

The developers extracted some parts of the **LP** to a separate contract called **Vault**, which contains the code that will not have many changes in the future. This update led to several changes in the other parts of the project. Four new medium severity issues, several new low severity issues and one new note were introduced with this update. The scope of the audit included the following files:

- **contracts/Vault.sol;**
- **contracts/LP.sol;**
- **contracts/Factory;**
- **contracts/utils/LiquidityTree.sol;**
- **contracts/interface/ILiquidityTree.sol;**
- **contracts/interface/IVault.sol.**

Other changes introduced in this commit were reviewed during the recheck of the **LiveCore** part of the project that took place from December 5 to December 6, 2023.

All out of 348 tests successfully passed, and 1 test had pending status. The code coverage was 91.91%.

Codebase update #11

After the recheck #10, the codebase was updated, and we were provided with commit f541648c1eb786c8a451fe32c230f24be32930ba. The scope included the contracts from the recheck #10 and all the contracts from previous rechecks.

The developers fixed four medium severity issues. All out of 386 tests passed and one test had pending status. The code coverage was unmeasured.

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 betting 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 were 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 canceled. 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, and 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 the 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.

From April 6 to April 14, we rechecked the current scope.

We checked the update and whether the rest of the issues were fixed.

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

Finally, we updated the report.

Following the last update, we reviewed changes to the previous version of the protocol between June 19 to June 29 of 2023.

The developers provided fixes for the medium-severity issues identified in the previous audit and also added new functionality, including the **FreeBet** contract.

We examined the new functionality to ensure its secure integration into the protocol. We identified a new issue of medium severity, as well as a few low-severity issues. Among other things, we reviewed the following:

- We validated the implementation of formulas within the `_calcCashout()` function to ensure that they do not allow for reward manipulation.
- The `_applyOdds()` function correctly integrates the multiple winning outcomes functionality.
- The new odds calculation mechanism cannot be used for express bet odds or regular bet odds malicious manipulation.
- Since affiliate rewards are removed from the codebase, they should no longer be accounted for in reward distribution.

Also, we scanned the project with the static analyzer Slither and our plugin Slitherin with an extended set of rules and then manually verified all the occurrences found by the tool.

The recheck #8 we made from October 9 to October 19, 2023. We reviewed the changes to the previous verison of the code.

During the recheck, in addition to checking for fixes, we updated the Overpowered role issue due to new functionality involving signatures for off-chain data. Additionally, a previously flagged issue M08 concerning the calculation of reserves was confirmed to be a false positive and subsequently removed.

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.

Finally, we updated the report.

From November 14 to November 16, 2023, we performed recheck #9. This involved a validation of fixes within the prior scope.

We also ran the tests, calculated the code coverage and updated the description of the L27 issue after re-scanning the project with the following tools:

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

As a final step, the report was updated.

From December 25 to December 27, 2023, we performed recheck #10. We validated the liquidity management part of the project. Other changes were reviewed earlier during the **LiveCore** recheck process. The security review introduced several medium severity issues (Losses and profits can be applied to the whole tree,
No revert when there is not enough liquidity in the tree,
No push of changes before the tree traversal, Incorrect profits/losses distribution may occur), several low severity issues and one note. New point was added to Overpowered role issue.

We also ran the tests and calculated the code coverage.

Finally, we updated the report.

During January 22-24, we made the recheck #11. We checked whether the developers fixed the remain issues. Also, we checked the difference between commits from the recheck #9 and recheck #11 since the scope was limited in the recheck #10. 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. 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:

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

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

- `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`);
  - Update the admin of the **Vault**.

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

The system also depends on the off-chain oracle:

- In the `beforeAddLiquidity` function of the **StakingConnector** contract the `stakedAmount` variable is passed as the parameter. And only off-chain oracle can validate it. Also, this value should reflect how many tokens have been staked in the **Staking** contract. And if it is incorrect, it will not regulate how much liquidity each user can add to the **LP** contract;
- The off-chain oracle validates FreeBetData which is passed through the `bet` function of the **FreeBet** contract.

> *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.*

> *The update at the commit* `4916d2591507533c4ed4ab466975b0afc6b9c78b:`
> *The impact of off-chain logic was added.*

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.

M06. Unclear formula (fixed)

The `_applyOdds` function contains an unclear formula since `fund` is divided by `(length - 1)` at line 412 of the **CoreBase** contract. Obscuring undocumented places in the code can lead to incorrect operation of the project.

> *The developers implemented a new formula to support the feature of multiple winning outcomes.*

M07. Insufficient parameter validation (fixed)

The `stake()` function of the **StakingPool** contract contains the parameter `uint48 period`, which is used to increase a `depositLimit` of the caller: `uint256 depositLimit = (amount.mul(depositRate) * period) / ONE_YEAR;`. Anyone can make their `depositLimit` arbitrarily large by staking a small number of tokens on a large enough period, as it is not restricted from above.

There is also a potential griefing attack because the withdrawal amount also depends on the `period`. `lockedReserve` restricts a withdraw amount (lines 140-141), so a malicious user can stake a minimal amount of tokens on a large enough period to increase `lockedReserve` and discourage future staking by restricting `withdrawAmount`.

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

M09. Signature replay attack (fixed)

Validation of the signature during the `FreeBet.bet` call is limited to the `FreeBetData` structure, which does not include information about the betting `token`. Consequently, this setup permits the replay of the same `signature` for different free bets on the same blockchain with the same `manager` (signer) of the `FreeBetData`.

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

M10. Losses and profits can be applied to the whole tree (fixed)

`_addLimit` and `_removeLimit` may apply profits and losses to all the leaves in the tree of the **LiquidityTree** contract. However, it seems that the changes should be applied only up to the last existing leaf and not on all leaves in the tree when the `_isNeedUpdateWholeLeaves` condition is met. Consider changing the `leaf` value at lines 91 and 358 of the **LiquidityTree** contract.

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

M11. No revert when there is not enough liquidity in the tree (fixed)

The validation at line 346 within the **LiquidityTree** contract ensures that there is enough liquidity in the tree to cover all losses. However, if the liquidity is insufficient, the function currently does not revert and simply returns without an error. Additionally, the verification at line 124 in the `LiquidityTree._changeAmount` seems redundant since the root amount is already checked within the `_removeLimit` function. By skipping the subtraction, unexpected behavior may occur. It might be beneficial to consider implementing a revert condition when liquidity is insufficient in the tree or when an overflow occurs during liquidity subtraction.

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

M12. No push of changes before the tree traversal (fixed)

The `isNeedWholeTreeUpdate` function within the **LiquidityTree** contract relies on the values stored inside the `treeNode` mapping at lines 481, 482, 504, and 506. However, before the check, there is no `_push` call, potentially leading to outdated values during the traversal. It is advisable to execute the `_push` operation just before examining the `isNeedWholeTreeUpdate` condition.

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

M13. Incorrect profits/losses distribution may occur (fixed)

The `_isNeedUpdateWholeLeaves` function might return a `true` status, even when updating the entire tree should not occur. Specifically, the update on lines 516-534 is divided between two children, splitting the forwarded amount to each child respectively. However, if only one child contains a subtree with a zero-sum, the expression will still return `true` due to the check at line 482 passing. The correct behavior in this scenario should involve checking only the non-zero child.

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

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 its part back.

The issue has been fixed at the commit e76a3940d0c883f1ef432178bde34709554a5ffa.

L22. View function (fixed)

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

Consider declaring this function as `view`.

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

L23. Initialization of the implementation contract

The common pattern for upgradeable contracts is to move all the contract initialization into `initialize` function. This allows the initialization of 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 (fixed)

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 are enough reserves for already locked liquidity. Consider checking that `reinforcementAbility*reserves >= coreData.lockedLiquidity`.

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

L25. Redundant code

The following code has no impact on the system's state and should be removed:

- (fixed) `CoreTools.calcProbability()` call on lines 527-531 of the **CoreBase** contract.
- `CoreTools.calcProbability()` call on lines 80-84 of the **PrematchCore** contract.

The issue has been partially fixed.

L26. Improper variable usage (fixed)

To ensure relevancy, we recommend using the variable `newTotalNetBets` instead of `condition.totalNetBets` in the condition stated at line 438 of the **CoreBase** contract.

This code has been removed and is not present in the latest version of the contract.

L27. Dubious cast (fixed)

Converting from a `uint256` to a `uint128` is considered an unsafe downcast. Consider using SafeCast library for conversions in the following contracts:

- (commented) at line 204 in `resolvePayout` function of **FreeBet** contract;
- (fixed) at lines 79-81 in the `putBet` function of the **PrematchCore** contract;
- (fixed) at line 457 in the `_applyOdds` function of the **CoreBase** contract.

In addition, this issue occurs throughout the code.

> *Comment from the developers:* ***SafeCast*** *is not used in cases where, according to the logic of the smart contract, the value of a variable cannot exceed its type boundaries.*

The issues have been partially fixed. We also updated the description as we have found it all over the scope.

L28. Non-indexed address in the event (fixed)

Consider declaring `newOracle` as `indexed` in the `OracleChanged` event of the **IStaking** contract.

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

L29. Unnecessary code (fixed)

The `receive` funiction in the **LP** and **FreeBet** contracts is unused.

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

L30. Uninitialized variables

The `timestamp` field of the `Bet` structure is not set in the `putBet` function of the **PrematchCore** contract and is always equal to 0. This field is only used in the `if` at line 313 in the **CoreBase** contract. The second compared `condition.endsAt` variable is also 0 and is not initialized anywhere. Thus, it turns out that this `if` always returns `false`.

This will not lead to anything serious because there are other checks that prevent from bidding after the deadline. But it consumes extra gas when reading from the storage.

L31. Complex modifier (commented)

In the `resolveConditionBase` modifier of the **CoreBase** contract, there is complex logic that calculates the reserves and 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.*

L32. Unsafe cast (part 2)

Unsafe downcasts could be found throughout the code:

- At line 568 of the **LP** contract;
- At lines 169, 200 and 326 of the **LiquidityTree** contract.

Consider using SafeCast library for safe conversions to the smaller types.

L33. Use FixedMath division

There are some parts of the code where the division is performed without the functions from **FixedMath** library. However, this library is used in the other parts of the code with similar functionality. Consider using FixedMath functions at lines 200 and 326 of the **LP** contract.

L34. No _disableInitializer

There are contracts that do not disable initializers inside the constructor during the deployment. It seems that it does not cause any serious problems, but we still advise adding the disablement of the initializers in **LP**, **Factory** and **Vault**.

L36. Indexed arguments (part 2)

`AdminChanged` event of the **IVault** interface contains `address` argument that is not marked as `indexed`.

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`.

N03. Storage breaking changes (commented)

Several changes were made that disrupt the storage layout. It is crucial to either preserve the existing storage layout or create a new contract when modifying the storage layout. The issues can be identified in the following contracts:

- In the **ICoreBase.sol** contract at line 17, the `timestamp` was introduced before the existing `isPaid` variable.
- In the **ILP.sol** at line 35, the `unusedVariable` was added before other fields to the `Game` structure which was used in the `games` mapping of the **LP**.
- In the **LP.sol** at line 50, the `unusedVariable` was inserted before the existing `liquidityManager` storage variable.

> *Comment from the developers: Core - contracts were not upgraded, but deployed again, so changing storage does not affected. LP - the variable was added, then renamed to unused, there were release branches not affected to deployed code, there are actual only history of upgrades.*

N04. Storage breaking changes (part 2)

With the update that separated some **LP** logic to the **Vault**, some storage breaking chages were introduced:

- `vaultBeacon` variable was unsafely inserted at line 23 of the **Factory** contract;
- `vault` variable was unsafely inserted at line 24, and two variables were unsafely removed at lines 36 and 60 of the **LP** contract.

It is important to either preserve the existing storage layout or create a new contract when the modification of the storage layout is unsafe.

This analysis was performed by Pessimistic:

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

January 25, 2024