Azuro V2 BetExpress Security Analysis

by Pessimistic

This report is public

March 5, 2024

Abstract

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

The audit showed only two issues of low severity. They do not endanger project security.

After the initial audit, the codebase was updated. The developers increased the number of tests and code coverage. No issues were fixed.

After the recheck #1, the codebase was updated. The developers made a small functionality update and fixed most of the issues. The number of tests of the entire project increased, and the code coverage remained almost the same. All tests of the project successfully passed.

After the recheck #2, the codebase was updated. The developers increased the number of tests and made the small update of the code. No issues were fixed or found.

After the recheck #3, the codebase was updated again. The developers made the small update of the code. No issues were fixed or found.

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

After the recheck #4, the codebase was updated again. The developers updated the functionality of locked liquidity. No issues were fixed. We found the following issues of medium severity: Overpowered owner , "Incorrect order of functions", "Unaccounted profit" and "Incorrect calculation of locked reserves". The total number of tests and the code coverage increased.

After the recheck #5, we contacted the developers and removed the following issues as false positives: "Incorrect order of functions", "Unaccounted profit" and "Incorrect calculation of locked reserves".

The protocol has complex logic of calculating betting odds, which is not properly documented. The logic of liquidity locking is also complex and convoluted. It can lead to unexpected mistakes in reserve accounting.

General recommendations

We recommend fixing the mentioned issues. We also recommend implementing CI to run tests and analyze code with linters and security tools. Also, consider improving public documentation.

Project overview

Project description

For the audit, we were provided with Azuro V2 BetExpress project on a private GitHub repository, commit 314094b933e01878eb532152aa568aed100eca54.

The scope of the audit includes **BetExpress.sol** file + `plugExpress()` function in the **Factory.sol** file.

The documentation for the project includes the detailed description of express bets implementation by Azuro and the logic behind it: https://azuro-protocol.notion.site/f3d4554dca40401886c3c0a095aad964.

239 tests pass successfully, 30 tests are pending. The code coverage is 92.87%.

The total LOC of audited sources is 265.

Codebase update #1

After the initial audit, the codebase was updated. For the recheck, we were provided with commit 0a204cca509a7b6be433d5fcb788e7cb2a88efb3.

The scope of the audit includes updates of **BetExpress.sol** file + `plugExpress()` function in the **Factory.sol** file.

This update did not include any fixes. The number of tests increased. All 248 tests passed and 30 tests had pending status. The code coverage increased to 98.06%.

Codebase update #2

After the recheck #1, the codebase was updated and we were provided with commit 1871a139d354f4b77180d73c8a56afc9a2c75c3.

This update included fixes for low severity issues. All 280 project tests passed, 30 had pending status. The code coverage of this scope was 98.1%.

Codebase update #3

After the recheck #2, the codebase was updated and we were provided with commit a3259c65e58090265ed5c6f7f7842e697e3c91bd.

This update did not include any fixes. The number of tests increased. All 332 tests passed and 30 tests had pending status. The code coverage of this scope was 97.16%.

Codebase update #4

After the recheck #3, the codebase was updated and we were provided with commit [4916d2591507533c4ed4ab466975b0afc6b9c78b](4916d2591507533c4ed4ab466975b0afc6b9c78b).

This update did not include any fixes. All 333 tests passed and 1 test had pending status. The code coverage of this scope was 95.79%.

Codebase update #5

After the recheck #4, the codebase was updated and we were provided with commit [5d36c9cfc98afb1910692df106275cae81750652](5d36c9cfc98afb1910692df106275cae81750652).

This update did not include any fixes but included 4 new issues of medium severity, 3 of them were later removed as false positives. All 411 tests passed and 1 test had pending status. The code coverage of this scope was 99.15%.

Audit process

We started the audit on January 11, 2023 and finished on January 17, 2023.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project. After a discussion, we performed preliminary research and specified those parts of the code and logic that require additional attention during an audit:

- Check if express bets change conditions odds the way it is described in the documentation.
- If users could manipulate odds via express bets for their benefit.
- There is always enough liquidity reserved for express bets with high coefficients.
- The mechanism of margin distribution is consistent with the documentation and common sense.

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 the code is gas efficient.
- The `FixedMath` library is used correctly and calculations cannot lead to over/underflow.
- Access control is implemented correctly.

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

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 tool.

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

We checked the update and whether the issues from the initial audit were fixed.

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 made the recheck for the current scope.

We analyzed 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.

From June 19 to June 28, we performed the recheck. We checked the changes in the code such as odds adjustment and whether the previous issues were fixed. No new issues were discovered.

We scanned the project with the static analyzer Slither and our plugin Slitherin with an extended set of rules and manually verified the output.

Finally, we updated the report.

From October 9 to October 19, we performed the recheck. We checked the changes in the code and whether the previous issues were fixed. No new issues were discovered.

We scanned the project with the static analyzer Slither and our plugin Slitherin with an extended set of rules and manually verified the output.

Finally, we updated the report.

We made the recheck #5 from February 26 to February 28, 2024. The developers updated and optimized the logic of liquidity locking. We checked the new functionality and whether the previous issues were fixed. Also, the developers provided the results of another audit, and we could compare them with ours and make additional checks.

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 also sent the results to the developers in the text file.

We ran tests, calculated the code coverage and updated the report.

On March 5, 2024, we removed M02-M04 issues as false positives.

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.

The audit showed no critical issues.

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. Overpowered owner (new)

The owner can change `reinforcement` and maximum odds that can influence the value of locked liquidity. In case of `reinforcement`, the owner can front-run the `putBet` function and censor bets.

Even thoughs it is the owner of the core and anyone can create core, they still can influence the users.

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. CEI pattern violation (fixed)

The CEI (checks-effects-interactions) pattern is violated in the `putBet()` function of the **BetExpress** contract: the `_shiftOdds()` and `_lockLiquidity()` functions, which contain external calls, are executed before storage writes.

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

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

L02. Redundant code (fixed)

Some blocks of code or expressions can be removed to improve code quality:

- The `using for` expression on line 24 of **BetExpress** contract is redundant as the Math library is not used in the contract.
- The `.toUint64()` expression on line 132 of **BetExpress** contract will not affect the resulting type as `marginAdjustedOdds()` returns a `uint64`.

The first issue has been fixed and is not presented in the current version of the code. The code from the second one has been removed.

L03. Structure naming

The codebase contains multiple contracts that declare a structure named `Bet`: **IBetExpress, FreeBet, ICoreBase**. The structure types are different, so we recommend renaming them to improve code clarity.

L04. No _disableInitializer (new)

The **BetExpress** contract does not disable initializer 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 initializer.

L05. Gas consumption (new)

The `condition` storage variable at line 235 is read multiple times in the `resolvePayout` function. Consider writting it to the local variable to decrease gas consumption.

Notes

N01. Undocumented complex logic

The Azuro V2 protocol allows users to make bets on certain events. Azuro BetExpress presents a feature of express bets, which has different margin and liquidity reserve calculations compared to regular bets. Moreover, express bets also affect the odds of their sub-events via so-called "virtual bets".

The contracts contain a complex logic associated with these calculations, some of which is not documented at all, such as the `marginAdjustedOdds()` function in **CoreTools** or `_applyOdds()` function in **CoreBase** contract. Consider making public documentation on the codebase.

N02. Lack of sub bets amount check (fixed)

The protocol allows making an express bet with just a single sub-bet via `putBet()` function of **BetExpress** contract, which contradicts common sense as an express bet is supposed to be a group of sub-bets on independent conditions. Consider checking that `subBets.length` is greater than 1.

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

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Daria Korepanova, Senior Security Engineer
Ivan Gladkikh, Security Engineer
Oleg Bobrov, Junior Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, Founder

March 5, 2024