

Bullseye Prediction Pools

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	GameFi	Documentation quality	Medium
Timeline	2023-10-09 through 2023-11-10	Test quality	Medium
Language	Solidity	Total Findings	15 Fixed: 9 Acknowledged: 5 Mitigated: 1
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review	High severity findings ⓘ	2 Fixed: 2
Specification	Bullseye.cash	Medium severity findings ⓘ	3 Fixed: 2 Mitigated: 1
Source Code	Bullseye-DevOrg/bullseye-contracts ⓘ #c331723 ⓘ	Low severity findings ⓘ	6 Fixed: 2 Acknowledged: 4
Auditors	- Gelei Deng Auditing Engineer - Nikita Belenkov Auditing Engineer - Valerian Callens Senior Auditing Engineer	Undetermined severity findings ⓘ	0
		Informational findings ⓘ	4 Fixed: 3 Acknowledged: 1

Summary of Findings

Bullseye Prediction Pools offers an on-chain GameFi-style lottery system. Users can join the lottery by purchasing tickets through the contract and later claim their winnings. The Bullseye team determines lottery outcomes using external mechanisms and oversees the distribution of tokens and winnings within the system.

During the audit, the Quantstamp team found several issues disrupting the system's proper functionality. Concerns were also raised about unaudited external components posing potential centralized risks to the game. While the project is well-documented and coded, these identified issues need to be addressed to minimize risks. The documentation can be further improved, and the test quality can be improved by adding more checks covering edge cases handling and false inputs.

During the fix review process, Bullseye team addressed the identified issues by updating the implementation and providing more detailed user-facing documentation. In the end, all the issues have been properly addressed, or acknowledged by the team.

Update: The original repository has been rebranded to Bullseye. The fix commits referenced in the audit report refer to the original repository from the initial audit. The original repository was forked starting from the fix review commit into a [private repository](#), where `Pricefeed`, `PredictionPool`, and `TicketManager` were renamed to `BullseyePriceFeed`, `Bullseye`, and `BullseyeTicketManager`, respectively.

The [private repository](#) fix review commit is `d50724b`, corresponding to commit `5bcc21b` in the [public repository](#).

ID	DESCRIPTION	SEVERITY	STATUS
REL-1	Collisions Are Possible Between Ticket Ids of Different Lotteries	• High ⓘ	Fixed
REL-2	<code>transferProceeds()</code> Does Not Check the Two Lotteries Have the Same <code>collateralToken</code>	• High ⓘ	Fixed
REL-3	Risks of the Current Mechanism to Update Prices	• Medium ⓘ	Mitigated
REL-4	Absence of Constraint for the Main Roles of the System	• Medium ⓘ	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
REL-5	Impact of Roles Unavailability on the System	• Medium ⓘ	Fixed
REL-6	Integration Requirements for Smart Contracts Are Not Documented	• Low ⓘ	Fixed
REL-7	Impact of Using Non-Standard ERC-20 Tokens as Collateral Tokens	• Low ⓘ	Fixed
REL-8	Mismatch Between Documentation and Code Regarding Role Privilege	• Low ⓘ	Acknowledged
REL-9	Durations of Lottery Phases Are Not Enforced in the Code	• Low ⓘ	Acknowledged
REL-10	The <code>claim()</code> Process Reverts if One of the Submitted Ticket Does Not Win	• Low ⓘ	Acknowledged
REL-11	The Price Update Could Be Front-Run, Leading to More Favourable Pricing	• Low ⓘ	Acknowledged
REL-12	The Option to Buy Tickets for a Lottery Can Be Disabled by the Role <code>PRICE_SETTER_ROLE</code>	• Informational ⓘ	Fixed
REL-13	Unlocked Pragma	• Informational ⓘ	Fixed
REL-14	Token Description Used when the Lottery Is Created May Not Match the Underlying Token of the Lottery.	• Informational ⓘ	Fixed
REL-15	Ticket Uris and Description of Lottery's Token Could Contain Malicious Executable Scripts	• Informational ⓘ	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
 1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
 2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.

3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
 1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
 2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

All the files in the codebase is within the audit scope.

Files Included

Repo: [https://github.com/Bullseye-DevOrg/bullseye-contracts\(c3317230219f51d50245539b2a35f9844d69fad5\)](https://github.com/Bullseye-DevOrg/bullseye-contracts(c3317230219f51d50245539b2a35f9844d69fad5))

Files:

- src/PredictionPool.sol
- src/TicketManager.sol
- src/interfaces/IPredictionPool.sol
- src/interfaces/ITicketManager.sol
- src/interfaces/IPriceFeed.sol

Update: The original repository has been rebranded to Bullseye. The fix commits referenced in the audit report refer to the original repository from the initial audit. The original repository was forked starting from the fix review commit into a [private repository](#), where `Pricefeed`, `PredictionPool`, and `TicketManager` were renamed to `BullseyePriceFeed`, `Bullseye`, and `BullseyeTicketManager`, respectively.

Findings

REL-1

Collisions Are Possible Between Ticket Ids of Different Lotteries

• High ⓘ Fixed

✓ Update

Marked as "Fixed" by the client. The issue has been fixed by adding `bucketLowerBound`, and setting related variables to `uint128` type.

Addressed in: `ead868ec765a2e8e95299c309d6197d789849c30`.

File(s) affected: `PredictionPool.sol`

Description: The ID of a ticket is a `uint256` variable representing a bet for a particular bucket on a particular lottery. It is calculated with the formula `(lotteryId << 128) + bucketLowerBound`. Collisions are possible when `bucketLowerBound > 2^128`. This can happen in special cases, for instance, if a price has more than 128 decimals. Also, users can use any arbitrary value for the parameter `bucketLowerBound` of the function `buyTickets()`. As a result, as described in the exploit section below, a user could buy a cheap ticket for lottery A (using the minimum price) and mint a ticket for lottery B with a `bucketLowerBound` corresponding to an expensive ticket.

Exploit Scenario:

Assumptions for this scenario:

- Lottery A with `lotteryID == 1` has a `bucketSize` of `2^2` and follows the price of token `tA`. The most likely outcome of `tA` at maturity date is `2^4`.
- Lottery B with `lotteryID == 2` has a `bucketSize` of `2^2` and follows the price of token `tB`. The most likely outcome of `tB` at maturity date is `2^10`.
- Lotteries A and B are open at the same time.

Problematic sequence of actions:

- Attacker executes the function `buyTickets()` with the parameters:
 - `lotteryId == 1` → lottery A
 - `bucketLowerBound == (2^128) + (2^10)` → betting for a price of `tA` of `(2^10)` at maturity date.
 - `buyTicketCount == 10`
- `bucketLowerBound` is a multiple of `bucketSize == 2^2`;
- Attacker receives 10 tokens of the `ticketId == (1 * (2^128)) + (2^128) + (2^2)` which represents a ticket that can be claimed for lottery B at `bucketLowerBound == 2^10`, by betting for a price of `tA` of `2^10` at the maturity date.

Recommendation: Consider replacing `uint256` with `uint128` for the type of the parameter `bucketLowerBound` of the function `buyTickets()`.

REL-2

`transferProceeds()`
`collateralToken`

Does Not Check the Two Lotteries Have the Same

• High ⓘ

Fixed

✓ Update

Marked as "Fixed" by the client. This issue has been fixed by adding checks in `transferProceeds()` as recommended, which ensures that the two lotteries have the same `collateralToken`.

Addressed in: `b2835585229d8d47155fcc3e3f2146a60320857d`.

File(s) affected: `PredictionPool.sol`

Description: In `transferProceeds()`, the `collateralToken` of the two lotteries are not checked. In case that the admin accidentally sets the two lotteries with different `collateralToken`, the underlying calculation will be wrong, and the users may not be able to receive the correct proceeds.

Recommendation: We recommend the team to add a check on the `collateralToken` of the two lotteries, and revert the transaction if the two lotteries are not the same.

REL-3 Risks of the Current Mechanism to Update Prices

• Medium ⓘ

Mitigated

ⓘ Update

Marked as "Mitigated" by the client. The client has updated the user documentation to declare the trust model and associated risks. The threat model is discussed, but the detailed price mechanism is not disclosed.

The client provided the following explanation:

The oracle contract is not in scope of the audit. Updated user documentation to discuss the oracle trust model and associated risks.

File(s) affected: `PredictionPool.sol`

Description: An on-chain price is fetched from a `PRICE_FEED` contract by the lottery to determine the winning bucket. As this process is dependent to external processes outside of the audit scope, we consider the following potential risks:

1. The address in charge of updating the price could get compromised and update the price with a malicious value. It could also choose not to update the price.
2. The updated price can be any value between `0` and `type(uint256).max`. As a result, no assumption can be made about the number of decimals used for the new value.
3. The data source of the updated price is not known and may be vulnerable to price manipulation attacks.
4. There is no mechanism in place to react in case an abnormal value is submitted to the oracle contract.

Recommendation: Consider adding more user-facing documentation so that users could properly validate the system implementation with proper transparency. Consider implementing on-chain and off-chain measures to improve the process of updating the price feed to mitigate the risks described above.

Even if the prices are set by the Bullseye team, some validations can be performed:

1. Use a variable to record when the last update of the price occurred, and pause the lottery if the price has not been updated.
2. (not really in scope) The `PRICE_FEED` contract could implement validations to avoid setting invalid prices.

REL-4 Absence of Constraint for the Main Roles of the System

• Medium ⓘ

Fixed

✓ Update

Marked as "Fixed" by the client. The latest implementation ensures that there is at least one `DEFAULT_ADMIN_ROLE` within the system.

Addressed in: `1ed67e60518f6c4896cc77651a1e09bc248812a3`.

File(s) affected: `PredictionPool.sol`

Description: Four roles are defined in the system:

1. `DEFAULT_ADMIN_ROLE`
2. `PRICE_SETTER_ROLE`
3. `FEE_RECIPIENT_ROLE`
4. `LOTTERY_MANAGER_ROLE`

Any of these roles can be granted and revoked by an address owning the role `DEFAULT_ADMIN_ROLE`. It means that each of these roles can be owned at a given time by `0` to `n` addresses. However, if only one address owns the role `DEFAULT_ADMIN_ROLE` and it revokes it:

- it will not be possible anymore to update the roles of other addresses;
- if the contract is currently paused with funds used as collateral, these funds would remain locked forever as it will not be possible anymore to call the functions `claim()` or `transferProceeds()`.

Recommendation: Consider assessing if constraints should be enforced in the code for these roles. For instance:

1. Should it be allowed to remain without address owning a given role?
2. Should it be allowed to remain with more than one address owning a given role?
3. Should an update of the role `DEFAULT_ADMIN_ROLE` be done in two steps to avoid giving the role to an address owned by no one?

REL-5 Impact of Roles Unavailability on the System

• Medium ⓘ

Fixed

✓ Update

Marked as "Fixed" by the client. The client has updated the documentation to explain the access control roles and the off-chain mechanisms.

The client provided the following explanation:

This issue relates to off-chain monitoring and operations. Confirmed with the auditor team that no protocol changes are required. Fixed by updating user documentation to explain the access control roles, associated risks, and off-chain mechanisms used to mitigate risks.

File(s) affected: `PredictionPool.sol`

Description: Four roles are defined in the system:

1. `DEFAULT_ADMIN_ROLE`
2. `PRICE_SETTER_ROLE`
3. `FEE_RECIPIENT_ROLE`
4. `LOTTERY_MANAGER_ROLE`

The `DEFAULT_ADMIN_ROLE` role can be used to pause/unpause the system, update the fee rate, and update the ticket URI. However, we can imagine a scenario where addresses with this role become unavailable and unable to unpause the system, where funds are currently locked as collateral.

The `PRICE_SETTER_ROLE` role is used to update the lottery tickets' price at a given frequency. However, we can imagine a scenario where addresses with this role become unavailable and unable to update prices, giving a possible advantage to users for buying potential winning tickets at a discount price.

The `FEE_RECIPIENT_ROLE` role is used to receive the fees. Its unavailability would have a limited impact on users.

The `LOTTERY_MANAGER_ROLE` role is used to create lotteries, withdraw fees, add proceeds to a lottery, and transfer proceeds from a matured lottery to an open one. Here, we can imagine a scenario where addresses with this role become unavailable and unable to withdraw fees or transfer proceeds from a matured lottery to an open one.

Also, it is important to have protections in place for such roles, for example, having them as a multi-sig wallet. Currently, this is not documented in the project.

Recommendation: Consider assessing the impact of the unavailability of any of these roles and the maximum acceptable duration where these roles can remain unavailable without affecting the system. If required, implement additional measures to reduce the likelihood of such an event and also to quickly react if an unavailability of these roles is detected.

Also, consider having privileged roles as a multi-sig.

REL-6

Integration Requirements for Smart Contracts Are Not Documented

• Low ⓘ

Fixed

✓ Update

Marked as "Fixed" by the client. The client has updated the user documentation. Also after discussion with the development team, the severity of this issue has been updated from "Medium" to "Low".

The client provided the following explanation:

Fixed by updating user documentation.

File(s) affected: `PredictionPool.sol`

Description: When an address buys a ticket via the function `buyTickets()`, a ticket that is an `ERC1155` is minted to the address. Then, when the address wants to claim a winning ticket, it has to call the function `claim()` and the ticket will be burned. The address could also have to call the function `resolveLottery()`. If the address is an EOA, it can trigger all these functions. However, if it is a smart contract, it should be able to interact with at least the functions `buyTickets()` and `claim()`. Right now, that aspect is not documented.

Recommendation: Consider documenting integration requirements for smart contracts.

REL-7

Impact of Using Non-Standard ERC-20 Tokens as Collateral Tokens

• Low ⓘ

Fixed

✓ Update

Marked as "Fixed" by the client. A new method is added as recommended to ensure that balance of the contract is increased by the amount transferred. However, note that the current fix introduces a read-only reentrancy attack vector with the variable `lottery.proceeds` and the view function `getLottery()`.

Addressed in: `d535263cdf231008399e8fd2e53b58788adf7cd2`.

The client provided the following explanation:

NOTE: Discussed with the auditing team. The severity of this item will be updated to Low because the potential issues are contained within the administrative account only.

File(s) affected: `PredictionPool.sol`

Description: The current system lets any `ERC-20` token be used as the collateral token of a lottery. However, some tokens can have specific features. They can:

- be paused;
- upgraded;
- have specific addresses blacklisted;
- a fee mechanism;
- a rebasing mechanism;
- a hook mechanism when performing a transfer, which can lead to reentrancies (ex: `ERC-777`);

As a result, the main flows of the protocol could be severely impacted if, for instance:

- funds cannot be transferred anymore;
- the number of collateral tokens or proceeds accounted for is not what has been received (in functions `addProceeds()`, `buyTickets()`).

Since only an address with the role `LOTTERY_MANAGER_ROLE` can create a lottery, we assessed the severity of this issue as low.

Recommendation: Consider accounting for the amount received by the system by checking the balances before and after the operation, instead of accounting for the expected amount to be received. For other tokens, consider analyzing thoroughly from a security perspective any token you would like to be supported and the impact these unusual features could have on the system.

REL-8

Mismatch Between Documentation and Code Regarding Role Privilege

• Low ⓘ

Acknowledged

✓ Update

Marked as "Fixed" by the client. The team has updated the user documentation.

The client provided the following explanation:

Fixed by updating user documentation.

File(s) affected: `PredictionPool.sol`

Description: According to the documentation, the `withdrawFees()` should only be executable by an address having the role `FEE_RECIPIENT_ROLE`. However, the code only allows an address with the role `LOTTERY_MANAGER_ROLE` to execute that function.

Recommendation: Consider aligning the documentation and the code accordingly.

REL-9

Durations of Lottery Phases Are Not Enforced in the Code

• Low ⓘ

Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

This is the desired behavior and is not a security concern. Users can view all timestamps before deciding to interact with a lottery.

File(s) affected: PredictionPool.sol

Description: According to the documentation, the betting window opens 14 days before the expiry date and closes 3 days before the expiry. However, these durations are not verified when the function `createLottery()` is executed, where an address with the role `LOTTERY_MANAGER_ROLE` provides as parameters the three timestamps `openTimestamp`, `closeTimestamp`, and `maturityTimestamp`. This lack of enforcement may lead to participants not having enough time to make their predictions and would permit last-minute volatility arbitrages.

Recommendation: Consider enforcing in the code the fact that the betting window opens 14 days before the expiry date and closes 3 days before expiry. Another suggestion would be to only provide as a parameter the `openTimestamp` and deduce on-chain the values of `closeTimestamp` and `maturityTimestamp` using constant durations.

REL-10

The `claim()` Process Reverts if One of the Submitted Ticket Does Not Win

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

This is the desired behavior and is not a security concern. Clients have full knowledge of valid claims and are expected to call `claim()` only with lottery IDs corresponding to valid claims. This expectation also holds for smart contracts calling the `claim()` function.

File(s) affected: TicketManager.sol

Description: In `claim()`, the for loop enumerates through the `lotteryIds` passed by the claimer and calculates the wins. In this for loop, there is a `require` statement that reverts the transaction if the ticket does not win. As a result, if one of the tickets does not win, the whole transaction will be reverted. This is not a desired behavior, as the users shall claim the wins of the winning tickets, and ignore the losing tickets. The revert also does not provide the exact information of which ticket is not winning.

Recommendation: We recommend the team to update the logic code logic: (1) use `if` clause for proper claiming if `userWinningTicketBalance > 0`, and emit the winning information; (2) emit the ticket not winning in case of `userWinningTicketBalance == 0`.

REL-11

The Price Update Could Be Front-Run, Leading to More Favourable Pricing

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

Per the recommendation, we will use a private mempool if deemed necessary for fairness. The operation of the protocol and risks will be made clear in user-facing documentation.

File(s) affected: PredictionPool.sol

Description: The `setLotteryTicketsPrice()` allows the `PRICE_SETTER_ROLE` account to set the pricing rules for the current round of the lottery. This price change should be happening every hour, and it is possible for the user to observe the mem pool until a price-changing transaction occurs and either front-run it or back-run it depending on what is more profitable for their position.

Recommendation: Consider using a private mem pool like Flashbots for setting lottery prices. Otherwise, clearly document this potential risk to the participants.

REL-12

The Option to Buy Tickets for a Lottery Can Be Disabled by the Role `PRICE_SETTER_ROLE`

• Informational ⓘ

Fixed

✓ Update

Marked as "Fixed" by the client. A `require` statement has been added to explicitly check that the length of `bucketTicketPrices` is not zero.

Addressed in: `5890ceb3524bb40279da8ff5d695ccc4a09031a9`.

The client provided the following explanation:

NOTE: Discussed with the audit team that the severity will be updated to Informational as this is the intended system design.

File(s) affected: `PredictionPool.sol`

Description: Once a lottery has been created, users can only buy tickets if the list `bucketTicketPrices` of the lottery has a length greater than `0`. This update comes from the execution of the function `setLotteryTicketsPrice()` by an address with the role `PRICE_SETTER_ROLE` and a non-empty list `bucketTicketPrices`. Such a call to `setLotteryTicketsPrice()` with an empty list `bucketTicketPrices` will revert due to a math underflow in `bucketTicketPrices.length - 1`. However the user experience will be improved if a correct error is displayed, rather than the transaction reverting with an underflow.

Recommendation: Consider adding to the function `setLotteryTicketsPrice()` an explicit check to make sure that the list `bucketTicketPrices` is non-empty.

REL-13 Unlocked Pragma

• Informational ⓘ

Fixed

✓ Update

Marked as "Fixed" by the client. The solidity version has been fixed to `0.8.19`.

Addressed in: `053f6b8c21ddd39b095b66bb4f9f2bd9e5339e8d`.

Related Issue(s): `SWC-103`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

REL-14

Token Description Used when the Lottery Is Created May Not Match the Underlying Token of the Lottery.

• Informational ⓘ

Fixed

i Update

Marked as "Mitigated" by the client. The development team have updated the user documentation and documented that users shall be aware of such design. Since this is the intended design, we label the issue status as fixed.

The client provided the following explanation:

Updated user documentation to discuss the oracle trust model and associated risks.

File(s) affected: `PredictionPool.sol`

Description: Addresses with the role `LOTTERY_MANAGER_ROLE` can create a lottery via the function `createLottery()`. The struct `LotteryParams` contains several fields, including a string `token` describing the underlying token of the lottery. Any typo or ambiguity in this string could induce users in error by thinking that the underlying token of the lottery is another one.

Exploit Scenario:

Assumptions for this scenario:

- Lottery A represents the price variation of `BTC`.
- The string `token` describing the underlying token of Lottery A is `BTT`.

Problematic sequence of actions: A user buys tickets to bet on the price of `BTT` at the maturity date when the underlying price used will be the price of `BTC`.

Recommendation: Consider documenting the fact that users should double-check what object is indexed by the lottery and the correctness of the current prices before buying tickets.

REL-15

Ticket Uris and Description of Lottery's Token Could Contain Malicious Executable Scripts

• Informational ⓘ

Acknowledged

i Update

Marked as "Acknowledged" by the client.

The client provided the following explanation:

Frontend web applications should beware of XSS vulnerabilities when rendering untrusted inputs. This risk is mitigated in most modern frontend frameworks.

File(s) affected: PredictionPool.sol

Description: Two string parameters can be set via the functions `setTicketURI()` and `createLottery()`. As the values are not sanitized on-chain, they could contain malicious content such as malicious executable scripts. However, since only addresses with the roles `DEFAULT_ADMIN_ROLE` or `LOTTERY_MANAGER_ROLE` can execute these functions, such a scenario is unlikely.

Exploit Scenario:

We recommend the team to consider the following exploitation scenario mentioned in this [article](#).

Recommendation: Consider adding sanitization mechanisms to limit the length of the strings or prevent the usage of unauthorized characters.

Code Documentation

1. **Acknowledged** Consider improving the readability of the code documentation by systematically using the NatSpec format (<https://docs.soliditylang.org/en/develop/natspec-format.html>). The purpose of the function as well as the expected format of parameters and what it returns should clearly be described.
2. **Fixed** In `PredictionPool.sol`: In the function `buyTickets()`, there is a mismatch between the check and the error message of the 4th require statement. It should be replaced with "count should be greater than 0".

Adherence to Best Practices

1. **Acknowledged** The team can consider to add the minimum lottery period between `openTimestamp`, `closeTimestamp`, and `maturityTimestamp`.
2. **Fixed** Consider adding more view functions to retrieve the status of the lotteries, for example, to allow users to view the `maturityTimestamp` and `winningBucket`.
3. **Fixed** The best practice is to emit event for all status changes. It is recommended to add event emits in the `constructor`.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

- 044...81f ./src/PredictionPool.sol
- 0bd...5f7 ./src/TicketManager.sol
- 01d...c42 ./src/interfaces/IPredictionPool.sol
- 4c3...dae ./src/interfaces/IPriceFeed.sol
- 2d8...a8b ./src/interfaces/ITicketManager.sol

Tests

- f18...6ba ./test/PredictionPool.t.sol

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- [Slither](#) 

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Slither reports 55 issues within the scope. Most of them are false-positives, and the rest of the issues are included in this report.

Test Suite Results

All 12 test cases are passed. Considering the complexity of the project, more test cases shall be added. More details in Code Coverage section.

```
[PASS] testAdminRoleProtection() (gas: 126643)
[PASS] testBuyTickets() (gas: 662606)
[PASS] testBuyTicketsAbovePricedRange() (gas: 455605)
[PASS] testBuyTicketsBelowPricedRange() (gas: 455349)
[PASS] testBuyTicketsMisalignedPrice() (gas: 336898)
[PASS] testFractions() (gas: 969731)
[PASS] testPause() (gas: 89251)
[PASS] testPredictionPool() (gas: 973816)
[PASS] testSetFeeRate() (gas: 76888)
[PASS] testSetPrices_revertIfEmptyPriceList() (gas: 193951)
[PASS] testTranferProceeds() (gas: 1888222)
[PASS] testTransferProceeds_revertIfCollateralMismatch() (gas: 881150)
Test result: ok. 12 passed; 0 failed; 0 skipped; finished in 4.49ms
Ran 1 test suites: 12 tests passed, 0 failed, 0 skipped (12 total tests)
```

Code Coverage

The code coverage rate for the core contract `PredictionPool.sol` is 82.2%, which can be further improved. In particular, more tests should be added to check the branches that lead to unfavourable outcomes, such as false inputs, false control logics, etc.

File	% Lines	% Statements	% Branches	% Funcs
script/PredictionPool.s.sol	0.00% (0/9)	0.00% (0/14)	100.00% (0/0)	0.00% (0/1)
src/PredictionPool.sol	81.37% (83/102)	81.82% (108/132)	57.41% (31/54)	63.64% (14/22)
src/TicketManager.sol	100.00% (7/7)	100.00% (12/12)	100.00% (0/0)	100.00% (4/4)
src/util/AccessControlAdminProtection.sol	83.33% (5/6)	83.33% (5/6)	75.00% (6/8)	100.00% (2/2)
Total	76.61% (95/124)	76.22% (125/164)	59.68% (37/62)	68.97% (20/29)

Changelog

- 2023-10-13 - Initial report
- 2023-11-10 - Final report
- 2025-03-21 - Final report update

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over \$200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no

responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information to, to, called by, referenced by or accessible through the report, its content, or any related related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

