Azuro V3 (PayMaster)

Protocol Audit Report

Prepared by: Maxim Timofeev (@max_timo)

16.04.25-17.04.25

Table of Contents

Protocol Summary

Azuro is a decentralized betting protocol. Anyone can launch a frontend service that connects to the smart contracts and receive an affiliate bonus for each bet made through the given frontend. Different betting events can be hosted, for example a football game. Odds are provided by a Data Feed provider (Oracle) for every bet. A user bet gets automatically converted to an NFT in the user's wallet. Different types of bets can be made, e.g. ordinary, combo bets.

The new PayMaster contract introduces sponsored bets (free bets), and sponsored fees. Free bets can be with returnable body or not. Returnable means that a bettor gets only winning payout (the difference between payout and amount) in case of a win. Frontends can deposit token amount, which will then be used by oracle-defined bettors to pay for betting and relayer fees.

Disclaimer

Auditor makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the auditor is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a protocol functioning can be harmed.
- Low - can lead to any kind of unexpected behaviour or slightly affect the protocol functioning.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

Audit Details

Review commit hash: 9a0ac86466b33b59ab1a7c30bfeba3b5f8463bce

Scope

```
contracts/extensions/PayMaster.sol

And related changes:
contracts/extensions/Relayer.sol
contracts/LP.sol
contracts/LiveCore.sol
contracts/utils/OrderTools.sol
```

Executive Summary

Issues found

Severity	Number

Severity	Number
Critical	2
High	1
Medium	1
Informational	2
Gas	1

Findings

Critical

[C-1] `IsLocked` mechanism can be exploited to freeze LP liquidity forever

Severity

Impact: High, it lets malicious bettor to lock high amount of LP liquidity forever using combo bets.

Likelihood: High, anyone can perform an attack.

Description

The `isLocked` mechanism is designed to prevent manually calling `withdrawPayout` via the `LP` for free bets with a returnable body. This ensures that free bets can only be resolved through the `PayMaster`, where the calculations and state updates related to the free bet take place. The following lines in the `LP` are responsible for this mechanism:

```
        if (account.code.length == 0) return false;

        (bool success, bytes memory result) = account.staticcall(
            abi.encodeCall(IPayMaster.IsLocked, (tokenId))
        );

        if (!success) return false;

        return (result.length >= 32 && abi.decode(result, (bool)) == true);
```

Here, `account` refers to the `betOwner`. The check implies that the bet owner can be the `PayMaster` contract, in which case it is queried whether the bet is locked for payout. However, the `betOwner` can be any contract that implements the `IsLocked()` function. If this function returns `false`, it will cause the transaction to revert, preventing the `withdrawPayout` call for that bet.

Since this function contains a call to `resolvePayout` — and for combo bets, liquidity is unlocked internally via `addReserve` — if the function always reverts, the liquidity tied to the combo bet will never be unlocked. A malicious bettor could exploit this by placing a small bet amount with high odds in a free combo bet and deploying a contract that causes reverts, permanently locking a large amount of liquidity.

Recommendation

In `_isLocked`, we should add a check to verify that the `account` is the `PayMaster` contract (e.g., `account == relayer.payMaster()`). Only if this check passes should we call `account.isLocked()`; otherwise, the function should return `false`.

Status

Fixed

[C-2] Non-returnable free bets can be cashed out, bringing guaranteed payout to a bettor

Severity

Impact: High, sponsored bets with non-returnable amount may be exploited to get positive payout with 100% chance.

Likelihood: High, everyone having a non-returnable free bet can perform an attack.

Description

Non-returnable free bets are bets whose entire amount is sponsored by the `PayMaster` and awarded to the bettor upon winning. Such a bet is transferred entirely to the bettor's ownership, meaning they can use cashout for it. Since the cashout payout is always greater than 0, the bettor is guaranteed to receive winnings from such a free bet and can exploit this attack multiple times, draining the `PayMaster` contract of a significant amount of money.

Recommendation

Cashout functionality must be disabled for non-returnable free bets.

Status

Fixed

High

[H-1] Non-returnable free bets body amount is returned to a bettor after rejection or condition cancellation

Severity

Impact: High, sponsored bets with non-returnable amount may be intentionally tried to be rejected, or a cancellation may happen, bringing a bettor unfair payout.

Likelihood: Medium, rejection or cancellation is not guaranteed to happen, though is very likely.

Description

Non-returnable free bets are bets where the entire amount is sponsored by the `PayMaster` and awarded to the bettor upon winning, with full ownership transferred to the bettor. This means that if the bet is rejected or its condition is canceled, the `amount` will still be returned to the bettor. This is not entirely fair, as a cancellation or rejection essentially means the bet "did not play out" and has no valid outcome — yet the bettor still receives a payout. In the case of rejections, a bettor could intentionally place a free bet right after a goal is scored or after a condition is resolved to trigger a rejection and secure a positive payout, though this attack isn't always guaranteed to succeed.

Recommendation

The stake should be returned to the `PayMaster` contract in case of a rejection or canceled condition, allowing the free bet to be reused for other bets.

Status

Fixed

Medium

[M-1] Possible inequality of LPs used for betting and for withdrawing payouts in `PayMaster`

Severity

Impact: High, it may lead to free bets stuck unresolved in `PayMaster` contract, meaning frozen funds.

Likelihood: Low, it may happen in case of adding or changing a relayer with connected LP which is different from the LP saved in `PayMaster`.

Description

The `lp` in `PayMaster` is set during initialization and is used in the `withdrawPayouts` function instead of the `Relayer`'s `lp`. Additionally, the `PayMaster` can support multiple relayers due to its `relayerAllowed` mapping. This leads to several implications:

1. The `PayMaster` cannot functionally work with multiple relayers. Since an `lp` is tied to a single `relayer`, and the `PayMaster` only has one `lp`, the `PayMaster` can effectively only have one `relayer`.

2. `lp` and `relayer.lp()` may be completely unrelated. If they somehow differ (which is not checked anywhere), free bets cannot be resolved through the `PayMaster` because it would interact with the wrong `lp` — the one where the bet was not originally placed.

3. The `lp` is immutable and cannot be updated. If the relayer is changed via `activateRelayer` and the new relayer uses a different `lp`, the `PayMaster` will again fail to correctly resolve free bets, with no way to adjust the `lp` to maintain contract compatibility.

Recommendation

Option 1: Store the `relayer` or `lp` in the free bet itself, and modify line 222 to use the `lp` (or `relayer.lp()`) stored in the free bet instead of the fixed `lp`. Option 2: Remove support for multiple relayers, restricting the `PayMaster` to a single `relayer`, and enforce that the `lp` used for free bets matches the one used in `withdrawPayouts`.

Status

Fixed

Low

Informational

[I-1] Wrong function call interface

Description

There is a `getOrderBetsAmounts` call in `LP.sol:324`:

```
        uint128 totalAmount = IBetting(core).getOrderBetsAmounts(order);
```

The function is from `OrderTools` contract and has a different number of returned parameters: `returns (uint128 totalAmount, uint128[] memory amounts)`. There is no error caused by this discrepancy, but it slightly harms a readability of code.

Recommendation

Update the assignment to display correct interface usage: `(uint128 totalAmount,) = ...`

Status

Fixed

[I-2] Flawed namings

Description

1. `PayMaster.IsLocked => PayMaster.isLocked` (mapping and function names should start with lower case letter)
2. `returns (uint128 amountPayBettor, uint256 feePayBettor)` - more grammatically correct naming would be: `amountPaidByBettor, feePaidByBettor`

Status

Fixed

Gas

[G-1] Logical condition can be reduced

Description

In `Relayer.sol` lines 122-123, the "if" condition can be safely reduced like this: `if (!data.isBetSponsored || !data.isSponsoredBetReturnable)`

Status

Fixed