Azuro Security Review

Pashov Audit Group

Conducted by: pashov

May 26th, 2023

Contents

1. About pashov

Krum Pashov, or **pashov**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Check his previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **Azuro** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Azuro

Azuro is a decentralized betting protocol. Anyone can launch a frontend service that connects to the smart contracts and to 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 once by a Data Feed provider (Oracle) for initialization and then odds change based on the betting on the platform. A user bet gets automatically converted to an NFT in the user's wallet.

5. Risk Classification

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

5.1. 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 core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. 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.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

***review commit hash* - 1c475b43e47798ae0a49716fe949b523a2663d0a**

Scope

The following smart contracts were in scope of the audit:

- `BetExpress`

7. Executive Summary

Over the course of the security review, pashov engaged with Azuro to review Azuro. In this period of time a total of **12** issues were uncovered.

Protocol Summary

Protocol Name	Azuro
Date	May 26th, 2023

Findings Count

Severity	**Amount**
Critical	1
Medium	2
Low	1
QA	8
Total Findings	**12**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Value of leaf argument when calling addReserve is hardcoded incorrectly	Critical	Resolved
[M-01]	The protection check for maxBetShare can be gamed	Medium	Resolved
[M-02]	Tokens with a no-op fallback function can be used to steal the ETH balance of LP	Medium	Resolved
[L-01]	Using 0 as an argument value is error-prone	Low	Resolved
[QA-01]	Off-by-one error on timestamp check	QA	Resolved
[QA-02]	The word "core" has multiple meanings in the protocol which raises complexity	QA	Resolved
[QA-03]	Redundant getter	QA	Resolved
[QA-04]	Missing event emission	QA	Resolved
[QA-05]	Missing override keyword	QA	Resolved
[QA-06]	Unused imports	QA	Resolved
[QA-07]	Incorrect comment	QA	Resolved
[QA-08]	Use a safe pragma statement	QA	Resolved

8. Findings

8.1. Critical Findings

[C-01] Value of `leaf` argument when calling `addReserve` is hardcoded incorrectly

Severity

Impact: High, because liquidity won't be returned to the LiquidityTree

Likelihood: High, because the incorrect value is hardcoded and can't be changed

Description

In `BetExpress::resolvePayout` we can see the following code:

```
uint128 reward = lp.addReserve(
    0,
    fullPayout - amount,
    fullPayout - payout,
    0
    );
```

where the last argument is 0 sent as a value for the `leaf` parameter. Since the leafs counting begins at 1, this will always be wrong and the liquidity won't be returned to the LiquidityTree.

Recommendation

The value of `leaf` should be the `leaf` value of each `condition` in the bet. The current design of `resolvePayout` does not allow to work on each `condition` in isolation, so this would need a redesign where you handle each `condition` separately.

8.2. Medium Findings

[M-01] The protection check for `maxBetShare` can be gamed

Severity

Impact: Medium, because a protocol invariant can be broken and the code gives a false sense of security

Likelihood: Medium, as it can easily be gamed but there is no incentive for an attacker

Description

The `lockLiquidity` method tries to block a single bet from taking up too much of the LP's allowed liquidity limit, but this can be gamed by splitting a very large bet into a big number of smaller ones, so this `LargeBet` custom error check would give a false sense of security as it doesn't guarantee what it intended to.

Recommendations

Change the validation to be based on all bets made through `BetExpress` instead of on each bet in isolation.

[M-02] Tokens with a no-op fallback function can be used to steal the ETH balance of `LP`

Severity

Impact: High, because it can lead to stolen funds from the protocol

Likelihood: Low, as it requires a token with a fallback function but without a `withdraw` function

Description

In `LP::withdrawPayout` we have the following code:

```
if (isNative) {
    IWNative(token).withdraw(amount);
    TransferHelper.safeTransferETH(account, amount);
} else {
    TransferHelper.safeTransfer(token, account, amount);
}
```

Now imagine the following scenario:

1. The `token` used in the contract is one that does not have a `withdraw` function but has a fallback function
2. An attacker has a winning bet of 100 * 1e18 tokens
3. Now he calls `withdrawPayout` but sets the `isNative` flag to `true`
4. The `IWNative(token).withdraw(amount);` will not revert but will be a no-op because of the fallback function of `token`
5. The attacker will receive 100 ETH instead of 100 * 1e18 tokens

The attack is similar to this one and even though it requires a special token and the `LP` to hold liquidity it is still a potential attack vector.

Recommendations

You can implement team processes about adding specific `token` contracts to be used in `LP`, where you have a checklist that contains not including tokens with a fallback function that are missing a `withdraw` function. You can also check the balance of `LP` before and after the `withdraw` call so you see it changed accordingly.

8.3. Low Findings

[L-01] Using 0 as an argument value is error-prone

It is a best practice to overload methods so they have signatures that omit the arguments where 0 is a valid value. Intentionally using 0 as a valid value is error-prone and has lead to high severity issues in multiple protocols in the past.

8.4. QA Findings

[QA-01] Off-by-one error on timestamp check

The code in `_conditionIsRunning` reverts when `block.timestamp >= startsAt` but if `block.timestamp == startsAt` this should mean condition is running, so shouldn't result in a revert.

```
- block.timestamp >= startsAt
+ block.timestamp > startsAt
```

[QA-02] The word "core" has multiple meanings in the protocol which raises complexity

The word "core" is used both as a contract name (`Core`, `CoreBase`) as well as a word that means something that is a part of the protocol, for example `BetExpress`. This is non-intuitive and raises the complexity of the protocol which is non-ideal - consider using different wording for both meanings of "core" in the codebase.

[QA-03] Redundant getter

`_baseURI` getter is redundant since there is already `baseURI` getter automatically generated.

[QA-04] Missing event emission

The `setBaseURI` method in `BetExpress.sol` does not emit an event which might not be good for off-chain monitoring. Emit an event on state change.

[QA-05] Missing `override` keyword

Methods `initialize` & `viewPayout` in `BetExpress.sol` are missing override keyword despite inheriting their function's signature from `ICoreBased.sol` & `IBet.sol` respectively.

[QA-06] Unused imports

`import "./libraries/Math.sol"` and `"@uniswap/lib/contracts/libraries/TransferHelper.sol"` are not used in `BetExpress.sol` and can be removed.

[QA-07] Incorrect comment

We have the following comment in `putBet` method in `BetExpress.sol`:

```
@notice Liquidity Pool: See {IBetEngine-putBet}.
```

The right interface in this case is `IBet.sol` instead of `IBetEngine.sol` Also in `putBet` method in `IBet.sol`, the `@return` field is missing. Consider adding one.

[QA-08] Use a safe pragma statement

Always use stable pragma statement to lock the compiler version. Also there are different versions of the compiler used throughout the codebase, use only one. Finally consider upgrading the version to a newer one to use bugfixes and optimizations in the compiler.