Azuro Security Review

Pashov Audit Group

Conducted by: pashov

March 27th, 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* - 7c6f477ca345ef8ca7a1c1f697daf479174b7060**

Scope

The following smart contracts were in scope of the audit:

- `Access`
- `AzuroBet`
- `Core`
- `CoreBase`
- `Factory`
- `LP`
- `interface/**`
- `libraries/**`
- `utils/**`

Contracts `SafeOracle`, `FreeBet`, `BetExpress` and `LiveCore` were out of scope for this audit.

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 **24** issues were uncovered.

Protocol Summary

Protocol Name	Azuro
Date	March 27th, 2023

Findings Count

Severity	**Amount**
Medium	5
Low	7
QA	12
Total Findings	**24**

Summary of Findings

ID	Title	Severity	Status
[M-01]	claimTimeout is not checked for first claim by an account	Medium	Resolved
[M-02]	Protocol can't use smaller decimals tokens as bet tokens	Medium	Resolved
[M-03]	Missing admin input sanitization	Medium	Resolved
[M-04]	OwnableUpgradeable uses single-step ownership transfer	Medium	Resolved
[M-05]	Admin privileges are dangerous	Medium	Resolved
[L-01]	Prefer using _safeMint over _mint	Low	Resolved
[L-02]	Missing event emission	Low	Resolved
[L-03]	Protocol won't work with tokens with a fee-on-transfer or a rebasing mechanism	Low	Resolved
[L-04]	The coreAffRewards mapping is not checked in claimAffiliateReward	Low	Resolved
[L-05]	Call to azuroBet.mint() can reenter	Low	Resolved
[L-06]	Code is lacking technical documentation	Low	Resolved
[L-07]	Unused method is not working as intended	Low	Resolved
[QA-01]	Use braces around operators with uncertain precedence	QA	Resolved
[QA-02]	The stopCondition method can start a condition as well as stopping	QA	Resolved
[QA-03]	Move not essential logic to off-chain computations	QA	Resolved
[QA-04]	Redundant code	QA	Resolved

[QA-05]	Open TODO in code	QA	Resolved
[QA-06]	Unused imports	QA	Resolved
[QA-07]	Method inherited from interface is missing the override keyword	QA	Resolved
[QA-08]	Use a safe pragma statement	QA	Resolved
[QA-09]	Small issues in initializer methods	QA	Resolved
[QA-10]	Typos in NatSpec	QA	Resolved
[QA-11]	Wrong import	QA	Resolved
[QA-12]	Consider using custom errors instead of require statements with string error	QA	Resolved

8. Findings

8.1. Medium Findings

[M-01] `claimTimeout` is not checked for first claim by an account

Severity

Likelihood: High, because it will happen for each account's first claim

Impact: Low, because there is no loss of funds, but code is not working as intended

Description

In `claimRewards` in `LP.sol` there is the following check

```
if ((block.timestamp - reward.claimedAt) < claimTimeout)
    revert ClaimTimeout(reward.claimedAt + claimTimeout);
```

which basically forces an account that claims his rewards to wait for at least `claimTimeout` amount of time. The problem is, in `addReserve` the reward amount is set, but `reward.claimedAt` is not set to `block.timestamp`. This means that `reward.claimedAt` will be 0 the first time `claimRewards` is called for an address, so the `claimTimeout` check will pass even though the time might have not passed yet.

Recommendations

When setting the reward amount in `addReserve` also set `reward.claimedAt = block.timestamp`

[M-02] Protocol can't use smaller decimals tokens as bet tokens

Severity

Likelihood: Medium, because such tokens are widely used and accepted

Impact: Medium, because it limits the functionality of the protocol

Description

The current implementation of the protocol allows it to only use higher (for example 18) decimals tokens like `DAI` for betting and liquidity provision. This is enforced by the `minDepo` property in `LP.sol` which can't be less than 1e12 for adding liquidity, as well as the check for `amount` in `putBet` in `Core.sol` where `amount` should be >1e12. If a smaller decimals tokens is to be used (for example `USDT`, `USDC`, `wBTC`) then the users and LPs will need to have a very high amount of capital to interact with the platform.

Recommendations

Revisit the validations for `minDepo` and `amount`, one possible approach is to calculate those based on the token's decimals

[M-03] Missing admin input sanitization

Severity

Likelihood: Low, because it requires a malicious/compromised admin or an error on admin side

Impact: High, because important protocol functionality can be bricked

Description

It is not checked that the `claimTimeout` property in `LP.sol` both in its setter function and in `initialize` does not have a very big value. Same thing for the setter function of `withdrawTimeout`. Also, the `checkFee` method in `LP.sol` has a loose validation - the max sum of all fees should be much lower than

100%. Finally the `startsAt` argument of `shiftGame` in `LP.sol` is not validated that it is not after the current timestamp.

Recommendations

Add an upper cap for `claimTimeout` & `withdrawTimeout`. Make the max sum of all fees to be lower - for example 20%. In `shiftGame` check that `startsAt >= blockTimestamp`.

[M-04] `OwnableUpgradeable` uses single-step ownership transfer

Severity

Likelihood: Low, because it requires an error on the admin side

Impact: High, because important protocol functionality will be bricked

Description

Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever. The ownership pattern implementation for the protocol is in `OwnableUpgradeable.sol` where a single-step transfer is implemented.This can be a problem for all methods marked in `onlyOwner` throughout the protocol, some of which are core protocol functionality.

Recommendations

It is a best practice to use two-step ownership transfer pattern, meaning ownership transfer gets to a "pending" state and the new owner should claim his new rights, otherwise the old owner still has control of the contract. Consider using OpenZeppelin's `Ownable2Step` contract

[M-05] Admin privileges are dangerous

Severity

Likelihood: Low, because it requires a malicious/compromised admin

Impact: High, because a rug pull can be executed

Description

A malicious or a compromised admin can execute a 100% rug pull in the following way:

1. The `LP` admin calls the `Factory` contract to add a malicious `core` to the `LP`
2. The malicious `core` returns the `LP` contract balance when its `resolveAffiliateReward` method is called
3. Now calling `claimAffiliateReward` with the fake `core` as an argument will result in a 100% of the `LP` balance stolen

Same thing applies to `withdrawPayout`.

Recommendations

Make the process of adding a new `coreType` or calling `plugCore` to be safer. One possible approach is by adding a time delay before a core is added to the `LP`, up until which the request will be pending.

8.2. Low Findings

[L-01] Prefer using `_safeMint` over `_mint`

Both `Access::grantRole` and `LP::_addLiquidity` use ERC721's `_mint` method, which is missing the check if the account to mint the NFT to is a smart contract that can handle ERC721 tokens. The `_safeMint` method does exactly this, so prefer using it over `_mint` but always add a `nonReentrant` modifier, since calls to `_safeMint` can reenter.

[L-02] Missing event emission

The `changeLockedLiquidity` method in `LP.sol` does not emit an event which might not be good for off-chain monitoring. Emit a proper event in both paths, adding liquidity and reducing it. Same problem exists in `AzuroBet::setURI` - emit an event on state change.

[L-03] Protocol won't work with tokens with a fee-on-transfer or a rebasing mechanism

Some tokens on the blockchain make arbitrary changes to account balances. Examples are fee-on-transfer tokens and tokens with rebasing mechanisms. There is no specific handling for such tokens, as the amount held by the `LP` contract might actually be less than it has accounted for. Think about handling such tokens or document the list of ERC20 tokens you intend to support in the protocol.

[L-04] The `coreAffRewards` mapping is not checked in `claimAffiliateReward`

When calling `claimAffiliateReward` for a core in `LP.sol` the `coreAffRewards` mapping is not checked to see if the core has actually earned enough rewards for the claim. Add a validation for the mapping.

[L-05] Call to `azuroBet.mint()` can reenter

The `mint` function in `azuroBet` does an external call to check if the recipient is a smart contract that can handle such tokens. This call is unsafe, as the recipient can be malicious and do a reentrancy call. Consider adding a `nonReentrant` modifier to methods in `Core.sol`

[L-06] Code is lacking technical documentation

In multiple places throughout the code there is a need for technical documentation as dev assumptions are not clear and some math formulas are used but it is not clear why. One example for this is `CoreBase::_applyOdds` - consider adding technical documentation to complex code for easier understandability by users & auditors. Also revisit existing NatSpec docs and add information for all parameters, as that is missing in multiple places.

[L-07] Unused method is not working as intended

The method `getLeavesAmount` in `LiquidityTree` will return just the `node` amount and won't consider the amounts in its leaves. Method is not used anywhere, consider removing it.

8.3. QA Findings

[QA-01] Use braces around operators with uncertain precedence

In `Access::roleGranted` we see the following code

```
return userRoles[account] & roleBit == roleBit;
```

In Solidity the `&` operator will be executed before `==` but this might not always be clear and might be different in other languages. I suggest adding braces around `userRoles[account] & roleBit` to clarify operator precedence.

[QA-02] The `stopCondition` method can start a condition as well as stopping

The `stopCondition` method and its event show intention that they only have functionality for stopping a condition, but they can start it again. Use different wording, for example `updateConditionStatus`.

[QA-03] Move not essential logic to off-chain computations

The `game.conditions` array is written to in `LP::addCondition` but is never read in the system. Consider moving this logic to the front end services.

[QA-04] Redundant code

The `assert(affiliateProfits[i] >= oldProfit - newProfit);` check is redundant, since the next line of code `affiliateProfits[i] -= (oldProfit -`

`newProfit).toUint128();` will fail if condition checked is false. Consider removing redundant code.

[QA-05] Open `TODO` in code

The `_resolveCondition` method in `CoreBase.sol` has an open `TODO`, consider fixing or deleting it.

[QA-06] Unused imports

All imports in `IBet.sol` are unused, consider removing those. Same for `OwnableUpgradeable` import in `Core`

[QA-07] Method inherited from interface is missing the `override` keyword

The `changeOdds` method in `CoreBase` is missing the `override` keyword, consider adding it.

[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.

[QA-09] Small issues in initializer methods

In `AzuroBet::initialize` the call to `__ERC165_init` is missing and should be added. Also `__Ownable_init_unchained` is called in `Access::initialize` which does not initialize call the `Context` initializer, call `__Ownable_init` instead.

[QA-10] Typos in NatSpec

In `IWNative`, the NatSpec has two typos - `interrface` -> `interface` and `vbased` -> `based`. Also move the NatSpec to be just above the interface declaration, not before the `pragma` statement.

[QA-11] Wrong import

Change the `ICore` import in `Factory` to `ICoreBase` since that is the one that is used.

[QA-12] Consider using custom errors instead of require statements with string error

Custom errors reduce the contract size and can provide easier integration with a protocol. Consider using those instead of require statements with string error