Babylon7 Security Review

Pashov Audit Group

Conducted by: pashov

June 20th, 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 **Babylon7** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Babylon7

The Babylon7 protocol implements on-chain raffles. It integrates Chainlink's VRF technology using a subscription method and has a main use case of an artist creating a listing, where participants can buy raffle tickets to try and win an ERC721 or some ERC1155 tokens. A raffle has 5 stages: Active, Resolving, Successful, Finalized and Canceled. Raffles also support reserved tickets, for which a whitelisting approach is used with a Merkle tree.

More docs

Observations

The protocol is non-custodial, meaning the token to be raffled is not held in the contract - the contract just expects to have an allowance to move it out of the owner's wallet in the end. If the owner removes his allowance to the contract then this acts as a "cancellation", as anyone can now call `refund` and execute the `listing.state = ListingState.Canceled` code.

The protocol takes a fee from each successfully completed raffle that has a min value of `_minDonationBps` and max value of 100%.

The `Babylon7Core` contract is upgradeable which brings centralization risk for the protocol & its users with it.

Any raffle (listing) has a maximum duration of 7 days (this value can be changed by the protocol owner).

Threat Model

Privileged Roles & Actors

- Random Provider - a contract that can receive randomness requests, it is expected to inherit from `VRFConsumerBaseV2`. Expected to call `resolveClaimer` providing a random value to choose a listing winner
- Treasury - an EOA/contract that will receive the fees from the successfully completed raffles
- Listing Creator - can update listing restrictions, settle listing, cancel listing
- Ticket Minter - can participate in active raffles by minting tickets to try to win an item and can claim a refund if a listing has been canceled
- Allowlist Ticket Minter - like normal Ticket Minter, but has a guaranteed ticket mint slot
- Core Contract Owner - can change `maxListingDuration`, `randomProvider`, `minDonationBps` and `treasury` while a raffle is currently active

If each role was malicious

The random provider might have an incentive to withhold the randomness if it is not beneficial to him (for example he bought tickets with not winning IDs).

The treasury address wants to get as much fees as possible, so increasing `listing.donationBps` is a goal.

The Listing Creator wants to try to get the ticket payouts without sending over his item.

The ticket minter wants to pay as little as possible for as big as possible chance to win the raffle. He could also try to exploit the refunds mechanism to claim more than he put in.

Allowlist ticket minter can try to also mint more tickets than he was allowed to.

The Core Contract Owner can try to exploit the creator's allowances and steal his item.

Security Interview

Q: What in the protocol has value in the market?

A: The items (ERC721/ERC1155) that the protocol has allowance for as well as the raffle participants' ticket payments (in native asset value).

Q: In what case can the protocol/users lose money?

A: If the allowances are exploited or the ticket payments to get stuck in the raffle contract.

Q: What are some ways that an attacker achieves his goals?

A: Forcing the settling of a listing or the refund mechanism to revert or making himself the winner of a raffle.

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* - b5de7e8384df77ac5f04abe012fd6a22388ef436**

***fixes review commit hash* - 524e190518e7b10f4f22eb613d293d66160c1a3c**

Scope

The following smart contracts were in scope of the audit:

- `Babylon7Core`
- `RandomProvider`
- `interfaces/**`

7. Executive Summary

Over the course of the security review, pashov engaged with Babylon7 to review Babylon7. In this period of time a total of **6** issues were uncovered.

Protocol Summary

Protocol Name	Babylon7
Date	June 20th, 2023

Findings Count

Severity	**Amount**
Medium	1
Low	5
Total Findings	**6**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Protocol admin account has a large centralization attack surface	Medium	Resolved
[L-01]	Not following the Checks-Effects-Interactions pattern	Low	Resolved
[L-02]	The cached request should be deleted on randomness received	Low	Resolved
[L-03]	Pull over push pattern is preferred in Solidity	Low	Resolved
[L-04]	The minDonationBps variable needs an upper bound	Low	Resolved
[L-05]	Enum used in a struct should have an invalid default element	Low	Resolved

8. Findings

8.1. Medium Findings

[M-01] Protocol admin account has a large centralization attack surface

Severity

Impact: High, as creators can lose their raffled items and payouts

Likelihood: Low, as it requires a malicious or a compromised admin

Description

A malicious or a compromised admin can execute various rug-like attacks on the protocol:

1. Upgrading the `Babylon7Core` contract to add code to exploit creator's allowances so their raffled items are stolen by the protocol owner
2. Setting the `_treasury` address to one without a `payable` `fallback` or `receive` method (for example `address(0)`), resulting in an inability for the `listing.creator` to execute a `transferETHToCreator` transaction
3. Setting an owner-controlled random provider at any time, so controlling who the winner of a raffle is

There are also smaller problems, like:

1. Setting a very small `_maxListingDuration`
2. Setting a very large `_minDonationBps` (both in `initialize` and in `setMinDonationBps`)
3. The `core` address can be changed in `RandomProvider` which can allow for direct calls to `requestRandom`

Recommendations

Use a TimeLock contract to be the protocol owner, so users can actually monitor protocol upgrades or other actions by the admins. Another option is to make the admin a governance controlled address.

Also you should use a `MINIMUM_MAX_LISTING_DURATION` constant and validate the `maxListingDuration` value in `setMaxListingDuration`, doing the same with a `MAXIMUM_MIN_DONATION_BPS` constant in both `initialize` and `setMinDonationBps` for the `minDonationBps` value. Finally, the `setBabylon7Core` should be made so it is called only once and `core` can't be changed later.

8.2. Low Findings

[L-01] Not following the Checks-Effects-Interactions pattern

In `transferETHToCreator`, even though there is a `nonReentrant` modifier, the CEI pattern is not followed. The `listing.state = ListingState.Finalized;` code is executed after the ether transfers. It is recommended to always first change the state before doing external calls - while the code is not vulnerable right now due to the `nonReentrant` modifier, it is still a best practice to be followed.

[L-02] The cached request should be deleted on randomness received

In `RandomProvider::fulfillRandomWords` it is checked that `requests[_requestId].exists == true`. This is correct, but then after the request is fulfilled (as the name of the event emitted in the method) the request value should be deleted from the mapping, so it doesn't exist anymore.

```
require
          (requests[_requestId].exists, 'RandomProvider: requestId not found');
        _core.resolveClaimer(requests[_requestId].listingId, randomWords[0]);
        emit RequestFulfilled
          (_requestId, requests[_requestId].listingId, randomWords);
+       delete requests[_requestId];
```

[L-03] Pull over push pattern is preferred in Solidity

The `resolveClaimer` method in `Babylon7Core` uses a "push" pattern to force-transfer the ERC721/ERC1155 item to the raffle winner. This opens up some attack vectors like gas griefing, force reverts and callbacks, because of the `safeTransferFrom` methods. It is recommended to use a pull over push approach, where the winner would have to send a transaction to claim the item himself.

[L-04] The `minDonationBps` variable needs an upper bound

Neither in `initialize` nor in `setMinDonationBps` is the `minDonationBps` value checked that it is not `> BASIS_POINTS`, which allows the owner (either maliciously or by mistake) to make it so that

[L-05] Enum used in a struct should have an invalid default element

Currently the default value for `_listingInfos[id].state` is `ListingState.Active` even if the `id` is non-existent. This opens up a vulnerability, as `Active` is both used for a non-existing listing and for an active one. While the code is not vulnerable due to other present checks, it's correct to use an `ListingState.Inactive` as the default value for listings, so it should be added as the first enum element.

can a listing creator buy tickets for a raffle? is this problematic and what if a listing creator becomes the listing winner (same address or different one)