NFT Loots Security Review

Pashov Audit Group

Conducted by: pashov

July 7th, 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 **NFT Loots** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About NFT Loots

NFT Loots is a protocol that implements lootboxes using ERC721 tokens on-chain. The game offers 25 boxes, a player clicks one and sees if he has won. To play, the user should make a "bet" payment that is done with `USDT`. The application is expected to launch on Polygon Mainnet and is making use of Chainlink's VRF technology

Observations

Lootbox games offer 25 boxes that can either hold an NFT or a USD reward. They are time-bounded.

Currently it is possible that two different accounts win the same prize.

The admin can withdraw any ERC20 token any time from the `NFTLootbox` contract

Privileged Roles & Actors

- VRF confirmed owner - can change the `keyHash` and also controls the access control of the `requestRandomWords` method
- VRF admin - can call `requestRandomWords`, this should be only `NFTLootbox` (currently not enforced)
- NFT Lootbox owner - can create actual lootboxes, set the `betCoin`, `vrfV2Consumer` address and withdraw ERC20/ERC721 tokens
- Player - doesn't require authorization, anyone can play and try to win in the NFT lootboxes game

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* - 6b252b1086021c1f57bf7db65506e7d7ead18c42**

***fixes review commit hash* - 9e9c86fed5f44985746l4a4a434d75fd391fe7cf**

Scope

The following smart contracts were in scope of the audit:

- `random/VRFv2Consumer`
- `NFTLootbox`

7. Executive Summary

Over the course of the security review, pashov engaged with NFT Loots to review NFT Loots. In this period of time a total of **7** issues were uncovered.

Protocol Summary

Protocol Name	NFT Loots
Date	July 7th, 2023

Findings Count

Severity	**Amount**
Critical	1
High	1
Medium	4
Low	1
Total Findings	**7**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Polygon chain reorgs will often change game results	Critical	Resolved
[H-01]	Contract might not have enough balance for winner to claim prize	High	Resolved
[M-01]	Two people might win the same prize	Medium	Resolved
[M-02]	Some common non-standard ERC20 tokens are incompatible with the protocol	Medium	Resolved
[M-03]	Lootbox input validation should be present	Medium	Resolved
[M-04]	Centralization attack vectors are present in the code	Medium	Resolved
[L-01]	Contract can't receive NFTs sent with safeTransferFrom method	Low	Resolved

8. Findings

8.1. Critical Findings

[C-01] Polygon chain reorgs will often change game results

Severity

Impact: High, as an already winning user will lose its reward

Likelihood: High, as reorgs with > 3 depth happen often on Polygon

Description

The `REQUEST_CONFIRMATION` constant in `VRFv2Consumer` is set to 3. This value is used to tell the Chainlink VRF service how much blocks do you want to wait at a minimum before receiving randomness. The reason this value was added is because of chain reorganizations - when this event happens, blocks and transactions get reorganized and they change. This is a serious problem in this application as it is expected to be launched on Polygon (mentioned in `README.md`), but as we can see here there are more than 5 block reorganizations a day with depth that is more than 3 blocks. In this article we can even see a recent event where there was a 156 block depth chain reorg on Polygon. This means that it is possible that often the winner of a lootbox game to be changed since when your transaction for requesting randomness from VRF is moved to a different block then the randomness will change as well.

Recommendations

Use a larger `REQUEST_CONFIRMATIONS` value - I would suggest around 60 to be safe. For the past 7 days the deepest chain reorganization had a depth of < 30 blocks. While 60 might not fit your use case for the game, I think anything below 25-30 is potentially dangerous to the project's users and reputation.

8.2. High Findings

[H-01] Contract might not have enough balance for winner to claim prize

Severity

Impact: High, as user will not be able to claim prize value

Likelihood: Medium, as it requires the owner to not input enough balance

Description

The protocol doesn't enforce the actual balance in the `NFTLootbox` contract to be enough to pay out rewards. The first issue comes in the `createLootbox` method, where the code does

```
loot.usdPrizes = prizes;
```

but it doesn't actually transfer `prizes` amount of stablecoin into the contract - it just expects it will have it as a balance, which is not enforcing it and is error-prone. Also, the contract is expected to hold the value of each NFT supplied as reward also in its USD value. So if a BAYC NFT was deposited as a reward, not only the BAYC should be held by the contract but also its USD value in stablecoin. Even though winners will be able to claim either the NFT itself or its USD value, this way the capital efficiency gets cut in half. This is also not enforced.

The situation currently is that the contract is not trustless - there might not be enough rewards to pay winners.

Recommendations

Ensure that all rewards possible to be won are claimable all of the time in a trustless manner. Enforce the contract to hold at least as much balance as the sum of all possible rewards.

8.3. Medium Findings

[M-01] Two people might win the same prize

Severity

Impact: High, as people might not get their prizes

Likelihood: Low, as it requires same `prizeIndex` wins

Description

The current way in `NFTLootbox` to decide if a player wins and what it wins is the `getPrizeIndex` method, which has the following implementation:

```
uint256[] storage _probabilities = lootboxes[_lootboxId].probabilities;
uint256 sum;

// Calculate the cumulative sum of probabilities and find the winning prize
for (uint256 i; i < _probabilities.length; ++i) {
    sum += _probabilities[i];
    if (_randomNumber <= sum) {
        return i;
    }
}

// If no prize is won, return a missing prize index (100001)
return MAX_PROBABILITY + 1;
```

This shows that the smaller `randomNumber` you get, the bigger chance you have of winning. The flaw is that multiple people might draw a small `randomNumber` and get the same `prizeIndex` returned, resulting in them being able to claim the same reward. This is also amplified by the fact that the `probabilities` array is not enforced to be sorted - if the first values in the `probabilities` array are big then it is more possible that winners will get the same `prizeIndex` and prize.

While the game currently looks like it handles this, as multiple users can claim the same prize with the same `prizeIndex`, this shouldn't be the case, as it means there is a race condition for the first person to get an NFT's `prizeIndex`, because front-running can be used to get the ERC721 token from another winner even if you played later than him (given that you got the same `prizeIndex` win). Also, if it is a USD based prize, then it is possible that multiple people win it but it is not enforced that the contract has this balance. This can mean some people lose their expected rewards.

Recommendations

Enforce only 1 winner per `prizeIndex` and also enforce the `probabilities` array to be sorted.

[M-02] Some common non-standard ERC20 tokens are incompatible with the protocol

Severity

Impact: Medium, because it won't leave to a loss of funds, outside of gas for redeployment

Likelihood: Medium, because such token is listed in the docs

Description

The code in `NFTLootbox` is directly using ERC20's `transfer` and `transferFrom` methods. There are two problems with this:

1. Some tokens do not revert on failure in `transfer` or `transferFrom`, but instead return false (example is `ZRX`)
2. Some tokens do not return a bool on `transfer` or `transferFrom` call (examples are `USDT`, `BNB`, `OMG`).

The application is incompatible with either of those. The more problematic one is `USDT` as it is widely known that it has those flaws and it is actually directly listed in the documentation. Still, by looking at the implementation code of the `USDT` token on Polygon, which is different from the Ethereum one, it looks like the issue is not present there. This is why this issue is only marked as Medium severity, but it still requires handling to be extra safe.

Recommendations

Use OpenZeppelin's `SafeERC20` library and its `safeTransfer`/`safeTransferFrom` methods.

[M-03] Lootbox input validation should be present

Severity

Impact: High, as it can leave NFTs stuck in the contract forever

Likelihood: Low, as it requires a fat-finger or misconfiguration by the `NFTLootbox` owner

Description

Both the `_priceForPlay` and `_duration` values should be properly validated in `NFTLootbox::createLootbox`. The `_priceToPlay` has to have an upper bound, because if it's too big then no one will want to participate and until the duration passes the NFTs will be stuck in the contract. For the `_duration` value there should be a lower and an upper bound, as too low of a duration doesn't make sense but too big of a duration can leave NFTs stuck in the contract forever. If the owner fat-fingers the duration and adds one or two digits it can become a big problem.

Recommendations

Add a lower & upper bound checks for `_duration` and a max value check for `_priceForPlay` in the `createLootbox` method.

[M-04] Centralization attack vectors are present in the code

Severity

Impact: High, as some accounts can execute a rug pull or brick the game

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

Description

The owner accounts of both `NFTLootbox` & `VRFv2Consumer` contracts have the power to break the game while it is running.

- `NFTLootbox::withdrawERC20` can be used by the contract owner to execute a rug pull by withdrawing all of the reward tokens from the contract
- `NFTLootbox::withdrawERC721` can be used to steal the ERC721 tokens if the lootbox has closed but the game winner is still about to claim them
- `VRFv2Consumer::requestRandomWords` should only be callable by `NFTLootbox` - currently it can be forbidden that it is called from `NFTLootbox`, resulting in a DoS in a running game
- `NFTLootbox::changeVrfV2Consumer` can be used maliciously to update the randomness provider to an admin controlled one
- `NFTLootbox::changeBetCoin` shouldn't be callable while a game is running as if it's not a stablecoin then the `priceForPlay` will be very different

Recommendations

Limit the usage of those methods by either making them callable only in special conditions or with specific arguments.

8.4. Low Findings

[L-01] Contract can't receive NFTs sent with `safeTransferFrom` method

The `NFTLootbox` contract is expected to receive and hold ERC721 tokens, but some smart wallets or contracts send an nft using `safeTransferFrom` which checks for the `onERC721Received` method if the recipient is a contract. Make sure to extend `ERC721Holder` from OpenZeppelin to handle this.