Pupniks Security Review

Pashov Audit Group

Conducted by: hickuphh3, carrotsmuggler

February 16th 2024 - February 19th 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our 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 we try to find as many vulnerabilities as possible. We 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 **pupniks-contract** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Pupniks

Pupniks is an ERC721 contract (NFT). Users can mint Pupniks NFTs for ETH and redeem NFTs receiving ETH back.

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 - b2c9fc80fecfbc6b789246aae05b46110464e712

fixes review commit hash - 175ddaa3e1c0194e360e9c12fe0b57989f115708

Scope

The following smart contracts were in scope of the audit:

- `Pupniks`

7. Executive Summary

Over the course of the security review, hickuphh3, carrotsmuggler engaged with Pupniks to review Pupniks. In this period of time a total of **3** issues were uncovered.

Protocol Summary

Protocol Name	Pupniks
Repository	https://github.com/mitche50/pupniks-contract
Date	February 16th 2024 - February 19th 2024
Protocol Type	NFT sale contract

Findings Count

Severity	**Amount**
High	1
Low	2
Total Findings	**3**

Summary of Findings

ID	Title	Severity	Status
[H-01]	Mints can be bricked from NFT redemptions	High	Resolved
[L-01]	Gas rewards can be lost due to the vesting period	Low	Resolved
[L-02]	Incorrect event name breaks ERC7572 compliance	Low	Resolved

8. Findings

8.1. High Findings

[H-01] Mints can be bricked from NFT redemptions

Severity

Impact: High, prevents subsequent Pupnik sales

Likelihood: Medium, can be easily performed by malicious parties

Description

Pupnik NFTs can be redeemed at any point in time after it has been minted. Redeeming a Pupnik whose ID is strictly less than `amountMinted` while the sale is ongoing will cause subsequent sales to revert, because the redemption decrements `amountMinted`, which the sale relies on.

The decrement will attempt to mint the existing ID of `amountMinted` before it is decremented.

```
uint256 currentAmount = amountMinted;

for (uint256 i = 1; i <= quantity;) {
  _mint(msg.sender, currentAmount + i);
  ++i;
}
```

POC

```
function test_dos_sale(uint256 amount) public {
  // setup: mint 2-5 NFTs
  //(exclude 1 because we need to redeem NFT id < lastMinted)
  amount = bound(amount, 2, 5);
  _deploy();

  changePrank(owner);
  pupniks.setSignerAddress(signer);
  pupniks.toggleSaleStatus();
  changePrank(user);

  uint256 nonce = 0;

  (bytes32 hash, uint8 v, bytes32 r, bytes32 s) = getSignature
    (user, nonce, amount, signerPkey);

  pupniks.mintPupnik{value: 0.5 ether * amount}(hash, abi.encodePacked
    (r, s, v), nonce, amount);

  // redeem 1 NFT
  pupniks.redeemPupnik(1);

  // then try to mint more, but it reverts with TokenAlreadyExists()
  nonce = 1;
  (hash, v, r, s) = getSignature(user, nonce, amount, signerPkey);
  vm.expectRevert(ERC721.TokenAlreadyExists.selector);
  pupniks.mintPupnik{value: 0.5 ether * amount}(hash, abi.encodePacked
    (r, s, v), nonce, amount);
}
```

Recommendations

Either have a separate variable that only keeps track of minted Pupniks and never decrements, or delay Pupnik redemptions for a fixed period till the sale is deemed complete.

8.2. Low Findings

[L-01] Gas rewards can be lost due to the vesting period

On Blast, gas rewards generated by the contract can be collected by calling the `claimAllGas()` function on the Blast contract. However, this loses some amount of rewards.

Gas fees consumed by a contract are eligible to be collected by the same contract after a vesting period. Initially, only 50% of the gas fees can be collected, which increases to 100% linearly over a span of 30 days.

When `claimAllGas()` is called, it claims all gas rewards that are currently available and removes all unvested rewards. This can lead to some gas fees being lost, which were still in the process of being vested. This can be different from the result of the `claimableGas` function, which assumes complete vesting.

Consider adding a `claimGasAtMinClaimRate()` function, which will claim gas fees only above a certain vesting threshold.

Refer to Blast docs here

[L-02] Incorrect event name breaks ERC7572 compliance

ERC-7572 states that the `ContractURIUpdated()` event SHOULD be emitted on updates to the contract metadata for off-chain indexers to query the contract.

However, the event name is incorrectly named `ContractURISet()` which also incorrectly includes an argument.

```
- event ContractURISet(string URI);
+ event ContractURIUpdated();
```