Onchain Heroes Security Review

Pashov Audit Group

Conducted by: Koolex, mahdiRostami, Ch_301

January 13th 2025 - January 16th 2025

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 **onchain-heroes/och-contracts** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Onchain Heroes

Onchain Heroes is a game where players send NFT heroes into dungeons to earn tokens. Success brings rewards, but losing means the hero is lost until the season ends.

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 - 42d0262d32aba8e06cad0f6e2def2ec9e0e711f0

fixes review commit hash - beb094fca6d90dccb810ecc5ad0eafc42a3cec5d

Scope

The following smart contracts were in scope of the audit:

- `GachaToken`
- `Endgame`
- `HeroERC721A`

7. Executive Summary

Over the course of the security review, Koolex, mahdiRostami, Ch_301 engaged with Onchain Heroes to review Onchain Heroes. In this period of time a total of **2** issues were uncovered.

Protocol Summary

Protocol Name	Onchain Heroes
Repository	https://github.com/onchain-heroes/och-contracts
Date	January 13th 2025 - January 16th 2025
Protocol Type	Game

Findings Count

Severity	**Amount**
Critical	2
Total Findings	**2**

Summary of Findings

ID	**Title**	**Severity**	**Status**
[C-01]	Unauthorized access to burn Gotcha token	Critical	Resolved
[C-02]	Malicious users can mint the double Guaranteed Mints per RING	Critical	Resolved

8. Findings

8.1. Critical Findings

[C-01] Unauthorized access to burn Gotcha token

Severity

Impact: High

Likelihood: High

Description

Two functions are affected here: `burnFrom` and `brun`

- `burnFrom`: `GachaToken::burnFrom` has four inputs, `address by, address from, uint256 id, uint256 amount`, here is a description for `address by`:

```
// - If `by` is not the zero address, it must be either `from`,
        //   or approved to manage the tokens of `from`.
```

An attacker could set `by` as zero or any approved address and burn tokens.

- `burn`: this function calls `_burn` function, in `_burn`:

```
function _burn(address from, uint256 id, uint256 amount) internal virtual {
        _burn(address(0), from, id, amount);
    }
```

It sets `by` as a zero address (just like the above attack path).

Here is the POC for setting `by` as a zero address.

POC:

```
function testBurn_gotcha_token() public {
        vm.prank(address(endgame));
        gachaToken.mint(owner, 1, 100);
        assertEq(gachaToken.balanceOf(owner, 1), 100);

        vm.prank(user1);
        // gachaToken.burnFrom(by, from, id, amount);
        //@audit by setting `by` as 0, it should be able to burn from any
        // address
        gachaToken.burnFrom(address(0), owner, 1, 100);
        assertEq(gachaToken.balanceOf(owner, 1), 0);
    }
```

Recommendations

1. Delete `burnFrom`.
2. Modify `burn` as follows:

```
function burn(address from, uint256 id, uint256 amount) external {
        _burn(msg.sender, from, id, amount);
    }
```

[C-02] Malicious users can mint the double Guaranteed Mints per RING

Severity

Impact: High

Likelihood: High

Description

`HeroERC721A.sol` is an NFT contract, It guarantees the first phase **OG Phase** the whitelisted addresses will get:

- 1 Free Mint
- 2 Guaranteed Mints per **RING** (RING is NFT, so they take a Snapshot for holders)

`HeroERC721A.sol#ogMint()` function is how users mint tokens for OG whitelisted addresses. Users who are eligible for the two types of OG mint have the flexibility to call `ogMint()` multiple times, So they can take the free mint and the guaranteed mint in two different transactions.

Malicious users can take this privilege to mint double the number of Guaranteed Mints tokens by:

- 1. Call `ogMint()` and set `freeMint` to false and set `paidMintQty = 10` so `numOfRing = 5`. It mints 10 NFT and this is how the `aux` will be:

```
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
 1010 0000 0000 0000 0000 0000 0000
```

- 2. Call `ogMint()` and set `freeMint` to true and set `paidMintQty = 0`. It mints 1 NFT (the free one), this is how the `aux` will be:

```
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
 0000 0000 0000 0000 0000 0000 1000
```

We can see that [24-63] is now empty. This is because `_ogMint()` deleted the old values in this step:

```
uint64 aux = (_getAux(to) >> 3) & 0x0fffff;
```

- 3. Call `ogMint()` and set `freeMint` to false and set `paidMintQty = 10` so `numOfRing = 5` It mints 10 NFT and the `aux` will be:

```
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
 1010 0000 0000 0000 0000 0000 1000
```

So, this Malicious has mint 20 from the OG paid mints when he only has 5 RING.

Note: Aux Bit Layout The aux is a 64-bit number with the following bit layout:

[0-1]: Tracking `publicMint` status [2-2]: Hero list mint tracking [3-23]: Tracking the number of OG free mints (capacity : 2****20 -1) [24-63]: Tracking the number of OG paid mints (capacity : 2****40 -1)

POC:

```
function testMinting() public {
        // og mint
        vm.warp(uint256(hero721.OG_MINT_TIMESTAMP()));
        bytes32[] memory b = new bytes32[](3);
        b[0] = bytes32
          (0xf6cad13a027fce5304a21819a989779362a06a4fbd07521b18209cd6ddc4e41c);
        b[1] = bytes32
          (0xc41bc74587e0ba059b462f9e0d8a6abd430876040e77e050ca8ec3f11d8bab38);
        b[2] = bytes32
          (0x31d93cf3542a493f93d6d5304dc473819fb4d0bf018b05c910f1802ea619ab5d);

        hero721.ogMint{value: 4e17}(ALICE, false, 4, 2, b);

        hero721.ogMint{value: 0}(ALICE, true, 0, 2, b);

        hero721.ogMint{value: 4e17}(ALICE, false, 4, 2, b);
        assertEq(hero721.balanceOf(ALICE), 10);
    }
```

Recommendations

```
function _ogMint(address to, uint64 numOfRings) internal {
+        uint64 _aux = _getAux(to);
+        uint64 aux = (_aux >> 3) & 0x0fffff;
-        uint64 aux = (_getAux(to) >> 3) & 0x0fffff;
        if (aux == numOfRings) revert AlreadyMinted();
-        _setAux(to, uint64(aux | (numOfRings << 3))); // Set OG free mint flag
+        _setAux(to, uint64(_aux | (numOfRings << 3))); // Set OG free mint flag
        _safeMint(to, numOfRings);
    }
```