
dtf
.is
the Decentralized Technology Firm

CryptoPunks Quick Audit

auditing@dtf.is

September 2, 2021

Contents

1 Project Overview **4**

2 Legal **4**

3 Overview **4**

4 Methodology **4**

5 Contract **4**

6 Findings **4**

6.1 Malicious CryptoPunk owner 5

6.1.1 Preconditions 5

6.1.2 Exploit steps 5

6.1.3 Mitigation 6

6.2 Spurious events 6

6.3 Self bidding 7

6.4 Mitigation 7

6.5 Unused variables and spurious comments 7

1 Project Overview

Date Conducted: September 1, 2020

Focus of Assessment: Fast audit of Cryptopunks

2 Legal

All work product by the auditor DTF.is is provided “AS IS”. Auditor makes no other warranties, express or implied, and hereby disclaims all implied warranties, including any warranty of merchantability and warranty of fitness for a particular purpose. The auditor takes no responsibility for loss of funds, or other assets that may occur due to any disclosure.

3 Overview

DTF has done an unsolicited quick audit of the CryptoPunks contract on the Ethereum network. This was not a work for hire, but was undertaken due to recent enthusiasm. The purpose was to expose any contract breaking bugs, or behavior that breaks either implicit assumptions or expected behavior.

4 Methodology

DTF quick audits are limited to exactly one hour. Given this limitation they are not complete or cohesive audits. This is only a starting point, and is meant to expose obvious flaws. Only manual code review was done based on the published source code for the contract.

5 Contract

Ethereum contract address:
0xb47e3cd837ddf8e4c57f05d70ab865de6e193bbb
Official site: Larvalabs
Source code: github

Source code was verified to match contract deployment via Etherscan. Larvalabs explicitly takes no responsibility for this deployed contract.

6 Findings

During the course of the one hour audit 2 issues were found. Neither results in direct loss of a CryptoPunk nor compromise of the funds in the contracts.

However, there is a major functionality breaking bug in the auction process for the CryptoPunks which absolutely breaks the implicit guarantee that funds put into the contract are safe in all cases. It is this auditors assessment that funds locked for pending bids are not safe.

6.1 Malicious CryptoPunk owner

Severity: severe

Given a Malicious CryptoPunk owner, someone with a low enough cost basis, or simply willing to cause economic damage can lock a bidders funds indefinitely in the contract, and deny them access to the CryptoPunk in question.

6.1.1 Preconditions

The malicious CryptoPunk owner must own the Punk in question, and be able to offer it for sale.

6.1.2 Exploit steps

1. List a Punk for sale via

```
function offerPunkForSale(uint punkIndex, uint
    minSalePriceInWei)
function offerPunkForSaleToAddress(uint punkIndex,
                                   uint minSalePriceInWei,
                                   address toAddress)
```

   Set the minSalePriceInWei to a large value to maximize damage

2. User enters a bid for cryptopunk, but does not buy it outright

```
        function enterBidForPunk(uint punkIndex) payable
```

   Now there is a pending high bid in punkBids array

3. Malicious CryptoPunk owner transfers the punk to the 0x0 address.

```
        function transferPunk(address to, uint punkIndex) {
        if (!allPunksAssigned) throw;
        if (punkIndexToAddress[punkIndex] != msg.sender) throw;
        if (punkIndex >= 10000) throw;
        if (punksOfferedForSale[punkIndex].isForSale) {
            punkNoLongerForSale(punkIndex);
        }
        punkIndexToAddress[punkIndex] = to;
        balanceOf[msg.sender]--;
        balanceOf[to]++;
```

```
11          Transfer(msg.sender, to, 1);
12          PunkTransfer(msg.sender, to, punkIndex);
13          // Check for the case where there is a bid from the new
                owner and refund it.
14          // Any other bid can stay in place.
15          Bid bid = punkBids[punkIndex];
16            if (bid.bidder == to)
                // Kill bid and refund value
                pendingWithdrawals[to] += bid.value;
                punkBids[punkIndex] = Bid(false, punkIndex, 0x0, 0);

17      }
```

Notice that burning a Punk is allowed.
Notice that bid.bidder != to, so the bid is not refunded.

4. User is now denied the punk, and the submitted funds for the bid.

```
1           function withdrawBidForPunk(uint punkIndex) {
2           if (punkIndex >= 10000) throw;
3           if (!allPunksAssigned) throw;
4            if (punkIndexToAddress[punkIndex] == 0x0) throw;
5           if (punkIndexToAddress[punkIndex] == msg.sender) throw;
6           Bid bid = punkBids[punkIndex];
7           if (bid.bidder != msg.sender) throw;
8           PunkBidWithdrawn(punkIndex, bid.value, msg.sender);
9           uint amount = bid.value;
10          punkBids[punkIndex] = Bid(false, punkIndex, 0x0, 0);
11          // Refund the bid money
12          msg.sender.transfer(amount);
13      }
```

Through this series of steps the malicious punk owner can punk the bidder. It is worth noting that there is a strong economic incentive not to do this, but since some punks are probably owned by users with very low acquisition cost the possibility definitely exists.

6.1.3 Mitigation

Potential buyers should not bid on punks but should outright buy the punk with the buyPunk function over the minValue. This forces an immediate transfer, and prevents a malicious punk owner from locking a bidders funds. Ideally punks would be voluntarily migrated to a successor contract that fixes this issue.

6.2 Spurious events

Severity: low

Some functions spawn spurious events for actions that should be explicitly disallowed.

```
function transferPunk(address to, uint punkIndex)
```

Allows self transfer of a punk to the same address that owns the punk. This results in a meaningless PunkTransfer event, and makes the punk no longer for sale, generating a PunkNoLongerForSale event.

6.3 Self bidding

Severity: observation

Since there is no way to enforce identity in the bidding process, it is very easy to create a trail of self traded auction volume. This can easily create the illusion of interest and future profitability without any real economic activity.

6.3.1 Mitigation

Future versions of the contract should burn a percentage of each successful bid in order to discourage and marginally penalize self bidding behaviors.

6.4 Unused variables and spurious comments

Severity: observation

There are various unused variables and comments, this is just a code-smell issue. There should be no dangling declarations in a well designed contract.

```
uint public nextPunkIndexToAssign = 0;
//mapping (address => uint) public addressToPunkIndex;
//        balanceOf[msg.sender] = initialSupply;              // Give
     the creator all initial tokens
```