CryptoPhunks Market Security Review

Table of Contents

Scope

The scope of the audit is limited to
https://github.com/0xFLIPSY/CryptoPhunksMarket/blob/19951d9c7403e3b07c34cbcb3f8be0f73
768e963/contracts/CryptoPhunksMarket.sol

Summary of Findings

In performing a security audit of CryptoPhunks Market, several issues of concern were found. For each finding, a summary of the issue is documented, along with any other finer details regarding the issue. Security recommendations are also provided where applicable.

The table below shows a breakdown of security findings found categorized by severity or risk and impact. A finding that has been reported is listed as pending, and if that finding is satisfactorily mitigated, it will be categorized as resolved.

Severity	**Resolved**	**Unresolved**	**Total**
Critical	1	0	1
High	0	0	0
Medium	1	0	1
Low	3	1	4
Info	7	1	8

Issues

CPM-001: Reentrancy attack on acceptBidForPhunk

Severity: Critical
Status: Resolved
As the external safeTransferFrom call is made before updating the state of the bid price to 0, reentrancy is possible, allowing any attacker to reenter the acceptBidForPhunk to add more amount to the pendingWithdrawals. The attacker would control both the bidder and the seller to achieve this.

As-is:

```
    phunksContract.safeTransferFrom(msg.sender, bidder, phunkIndex);
    phunksOfferedForSale[phunkIndex] = Offer(false, phunkIndex, bidder, 0, address(0x0));
    uint amount = bid.value;
    phunkBids[phunkIndex] = Bid(false, phunkIndex, address(0x0), 0);
    pendingWithdrawals[seller] += amount;
    emit PhunkBought(phunkIndex, bid.value, seller, bidder);
```

To-be:

```
    phunksOfferedForSale[phunkIndex] = Offer(false, phunkIndex, bidder, 0, address(0x0));
    uint amount = bid.value;
    phunkBids[phunkIndex] = Bid(false, phunkIndex, address(0x0), 0);
    phunksContract.safeTransferFrom(msg.sender, bidder, phunkIndex);
    pendingWithdrawals[seller] += amount;
    emit PhunkBought(phunkIndex, bid.value, seller, bidder);
```

Note: At the point of this review, an actual proof of concept for this issue was not presented to the team. This issue was reported by an external party after the contract was deployed on mainnet.

The attack is explained as such:
1. Bidder contract does enterBidForPhunk for a punk that seller contract owns
2. Seller contract calls acceptBidForPhunk
3. ERC721 token gets transferred to bidder contract.
4. Bidder contract's ERC721 callback function does the transfer of the NFT from bidder to seller contract
5. Seller contract ERC721 callback function then calls acceptBidForPhunk multiple times to add the bid amount to the seller contract's withdrawal balance
6. Seller contract calls withdraw to withdraw the inflated balance

Recommendations
Change the sequence of the code to update the state before making the external call.

Add nonReentrant modifier to all external facing functions.

Resolution
The code sequence has been changed to update the state before making the external call.
A reentrancy on the same function would result in a revert in the following line as the bid's value would have been set to 0.
if (bid.value == 0) revert();

Additionally, the nonReentrant has been added to all external facing functions to prevent other possibilities of reentrancy attacks.

CPM-002: Active bids cannot be withdrawn if phunksContract is changed to contract that causes reverts

Severity: Medium
Status: Resolved
Active bids can be stuck and cannot be withdrawn if setPhunksContract is used to change phunksContract. This is due to this line where the returned address of the ownerOf function call is the zero address, or reverts due to incompatible/unimplemented interface.

```
if (phunksContract.ownerOf(phunkIndex) == address(0x0)) revert();
if (phunksContract.ownerOf(phunkIndex) == msg.sender) revert();
```

Recommendations
Remove the above statements that cause reverts.
The owner of the contract should also be a multisig to prevent unauthorized changes to phunksContract.

Resolutions
The above statements have been removed.

CPM-003: Checks-effects-interaction not adhered to in buyPhunk

Severity: Low
Status: Resolved
Similarly to CPM-001, the checks-effects interaction is not adhered to in buyPunk.

As-is:
```
    phunksContract.safeTransferFrom(seller, msg.sender, phunkIndex);
    phunkNoLongerForSale(phunkIndex);
    pendingWithdrawals[seller] += msg.value;
    emit PhunkBought(phunkIndex, msg.value, seller, msg.sender);
```

To-be:
```
        phunksOfferedForSale[phunkIndex] = Offer(false, phunkIndex, msg.sender, 0,
address(0x0));
        phunksContract.safeTransferFrom(seller, msg.sender, phunkIndex);
        pendingWithdrawals[seller] += msg.value;
        emit PhunkBought(phunkIndex, msg.value, seller, msg.sender);
```

Recommendations
Change the sequence of the code to update the state before making the external call.
Add nonReentrant modifier to all external facing functions.

Resolution
The code sequence has been changed to update the state before making the external call.

Additionally, the nonReentrant has been added to all external facing functions to prevent other possibilities of reentrancy attacks.

CPM-004: withdraw and withdrawBidForPhunk will fail in some situations

Severity: Low
Status: Resolved
The withdraw and withdrawBidForPhunk functions will fail if the seller or bidder, respectively, are the following:
- Contract that does not implement fallback function
- Contract that has fallback function which costs more than 2300 gas

Recommendations
Consider sending WETH instead of ETH if the destination address is a contract.

CPM-005: Wrong logic for checking of buy amount in buyPhunk

Severity: Low
Status: Resolved
The logic for checking if msg.value is valid is wrong:
```
 if (msg.value < offer.minValue) revert();
```

In the dapp UI, the price shown is the offer.minValue, and there is no option to buy at a value greater than the shown price. However, here in buyPhunk, the msg.value can be greater than the minValue, which does not match the functionality at the UI level.

This would allow a buy order higher than the buyout price to go through, causing the user to pay more than the buyout price.

Recommendations
Only allow buyPhunk to be successful if the msg.value is equal to the offer.minValue.

Resolution
The following check is now used:
```
 if (msg.value != offer.minValue) revert();
```

CPM-006: Griefing of bids is possible

Severity: Low
Status: Acknowledged
Griefing of bids can be done by the following:

A is a real bidder who bids 1 ETH on ID1.
B is a griefer who bids 1.1 ETH on ID 1, thus overwriting A's bid.
B then cancels the bid by calling withdrawBidForPhunk. There is no active bid for ID 1 as A's bid has already been canceled.
A will then have to place a new bid.

Recommendations
This is not a simple issue to solve, as removing withdrawBidForPhunk means that the bidder's funds are stuck in the contract unless the seller accepts it.

CPM-007: Funds can be temporarily stuck for bids where the NFT's ownership is transferred to the bidder after a bid is done

Severity: Info
Status: Resolved
Funds can be temporarily stuck for bids where the NFT's ownership is transferred to the bidder after a bid is done.

This is because of this line in withdrawBidForPhunk:
```
if (phunksContract.ownerOf(phunkIndex) == msg.sender) revert();
```

E.g.
User A bids on ID 1 that is owned by user B.
User B transfers ID 1 to User A.
User A is unable to withdraw his bid for ID 1 as it reverts. He will have to send ID 1 to another address before being able to withdraw his bid.

Recommendations
Consider removing that revert as the check is already done in the bid function.

Resolutions

This issue has been resolved together with CPM-002.

CPM-008: Seller can be buyer in buyPhunk

Severity: Info
Status: Resolved
The seller can be the buyer when using the buyPhunk function as there is no check to ensure that both addresses are not the same.

Recommendations
Add a check to ensure that the seller is not the buyer.

Resolution
The following check has been added:
if (seller == msg.sender) revert('seller == msg.sender');

CPM-009: Seller can be bidder in acceptBidForPhunk

Severity: Info
Status: Resolved
It is possible for the bidder to be the same as the owner of the NFT.
Address A bids for ID 1 owned by address B
Address B sends id 1 to address A. Address A is now the owner of id 1
Address A calls acceptBidForPhunk. A is the bidder and the seller.

Recommendations
Add a check to ensure that the seller is not the bidder.

Resolution
The following check has been added:
if (seller == bidder) revert('you already own this token');

CPM-010: More validation to ensure phunksContract is an ERC721

Severity: Info
Status: Resolved
In the constructor, there can be a more effective way of doing a check that the phunksContract set is an ERC721 contract rather than just checking for the zero address.

Recommendations
The following sanity check can be added:

IERC721(initialPhunksAddress).balanceOf(address(this));

Resolution
The following sanity check has been added to the constructor.

CPM-011: Public functions can be made external

Severity: Info
Status: Acknowledged
All public functions that are used can be changed to external if never called within the same function.

Recommendations
Change the visibility of functions that are never called within the same contract to external.

CPM-012: Inconsistent behavior between offerPhunkForSale and offerPhunkForSaleToAddress

Severity: Info
Status: Resolved
The following statement is done in the latter, but not the former.

if (phunksContract.getApproved(phunkIndex) != address(this)) revert();

Recommendations
Consider removing the above statement as it is unnecessary. Sellers can unapprove the market contract after creating an offer.

Resolution
The above statement has been removed.

CPM-013: Inconsistent behavior between buyPhunk and acceptBidForPhunk

Severity: Info
Status: Resolved
Inconsistent behavior between buyPhunk and acceptBidForPhunk for taking off a phunk for sale after it has been sold.

buyPhunk

phunkNoLongerForSale(phunkIndex);

acceptBidForPhunk
phunksOfferedForSale[phunkIndex] = Offer(false, phunkIndex, bidder, 0, address(0x0));

Recommendations
The behavior in the two functions should be standardized. buyPunk does not need to call phunkNoLongerForSale, but can use the same method as acceptBidForPhunk, as long as it is trusted that safeTransferFrom properly transfers the ERC721 from the seller to the buyer.

Resolution
The following change has been made to buyPhunk.
phunksOfferedForSale[phunkIndex] = Offer(false, phunkIndex, msg.sender, 0, address(0x0));

CPM-014: More validation to ensure phunksContract is an ERC721 contract

Severity: Info
Status: Resolved
In the constructor, there can be a more effective way of doing a check that the phunksContract set is an ERC721 contract rather than just checking for the zero address.

Recommendations
The following sanity check can be added:
IERC721(initialPhunksAddress).balanceOf(address(this));

Resolution
The following sanity check has been added to the constructor.