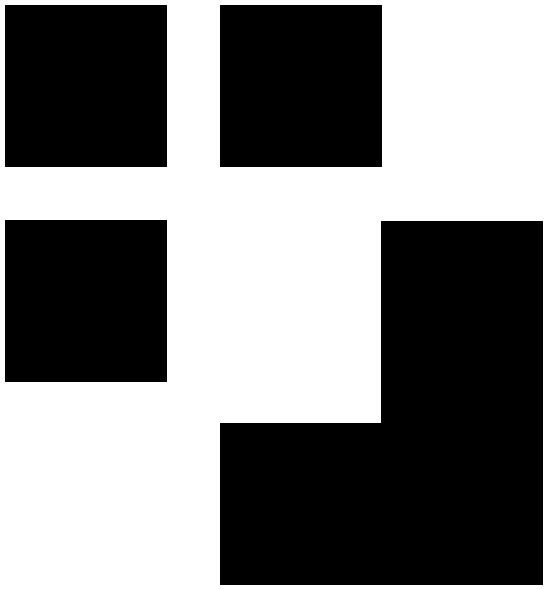

Protocol Audit Report

Version 1.0

Hexific.com

October 1, 2025

Protocol Audit Report

Hexific.com

October 1, 2025

Prepared by: Hexific Lead Security Researcher: - Febri Nirwana

Table of Contents

Protocol Summary

The BidBeast protocol is a decentralized NFT marketplace that enables users to list, and trade unique digital collectibles through an auction-based system. The protocol consists of two main smart contracts: `BidBeasts_NFT_ERC721` for minting ERC-721 compliant NFTs, and `BidBeastsNFTMarketPlace` for managing listings, bids, and auction mechanics.

The protocol is designed to provide a fair and transparent marketplace for NFT trading while protecting both buyers and sellers through automated auction mechanics and secure fund handling.

Disclaimer

The Hexific 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
Likelihood	High	H	H/M	M
	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 the following commit hash:

```
449341c55a57d3f078d1250051a7b34625d3aa04
```

Scope

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

Roles

- Contract Owner: Deploys and manages the marketplace contract, can withdraw accumulated fees from successful NFT sales.
- Bidder (Buyer): Places bids on listed NFTs during auctions, can withdraw failed bid refunds if outbid.
- Seller (NFT Owner): Lists NFTs for auction with minimum and buy-now prices, receives proceeds from successful sales.

Executive Summary

The Hexific team conducted a security audit of the BidBeast protocol smart contracts to evaluate their design, implementation, and resistance to common attack vectors. Our review identified several

critical issues that could lead to fund theft and protocol malfunctions.

The most severe finding includes a critical vulnerability in the fund withdrawal mechanism that allows attackers to steal failed bid refunds intended for legitimate users. Additionally, we found auction timing inconsistencies that deviate from documented behavior, anti-sniping mechanisms that can be exploited to indefinitely extend auctions, and various implementation issues affecting user experience.

The protocol implements an NFT marketplace with auction functionality but contains fundamental flaws in fund handling and auction mechanics that make it unsafe for production use in its current state. Immediate remediation of the High severity findings is strongly recommended before any deployment.

Issues found

Severity	Number of Issues Found
High	1
Medium	2
Low	1
Gas	n/a
Info	7
Total	11

Findings

High

[H-1] Missed transfer destination `BidBeastsNFTMarketPlace::withdrawAllFailedCredits` may result in loss of funds

Description:

This function is intended to manage failed payouts, enabling the recipient—or even third parties—to invoke it later on to withdraw bid funds back to the rightful owner. That owner would be someone who had previously submitted a bid but got overridden by a higher one from another bidder.

That said, there's a coding mistake where `msg.sender` is used instead of what should be `_reciever`. This opens the door for funds to be claimed by unauthorized parties, not just the legitimate recipient. And since the function is marked as `external`, literally anyone can call it.

```
function withdrawAllFailedCredits(address _receiver) external {
    ...
@>  failedTransferCredits[msg.sender] = 0;

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

Risk:

Likelihood: High

- Reason 1: It's incredibly straightforward for anyone to invoke the `withdrawAllFailedCredits` function, given its external visibility.
- Reason 2: An attacker simply has to check the `failedTransferCredits` state or the second-most-recent `BidPlaced` event.

Impact: High

- Impact 1: All funds belonging to the rightful recipients could be drained away by the attacker.
- Impact 2: The protocol may face significant hurdles in recovering or refunding these lost assets.

Proof of Concept:

This Foundry test demonstrates a critical security vulnerability in the function where any attacker can steal funds intended for legitimate bidders.

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

    NonPayable nonPayable = new NonPayable(market);
    vm.deal(address(nonPayable), MIN_PRICE + 1);

    nonPayable.placeBid(TOKEN_ID, MIN_PRICE + 1);
    console.log("Initial contract balance:", address(nonPayable).
        balance);

    uint256 secondBidAmount = MIN_PRICE * 120 / 100; // 20% increase
    vm.prank(BIDDER_1);
    market.placeBid{value: secondBidAmount}(TOKEN_ID);
    console.log("Ending contract balance:", address(nonPayable).balance
        );

    address attacker = makeAddr("attacker");
```

```
17      console.log("Initial attacker balance:", attacker.balance);
18
19      vm.prank(attacker);
20      market.withdrawAllFailedCredits(address(nonPayable));
21      console.log("Ending attacker balance:", attacker.balance);
22  }
```

Output:

```
1  Logs:
2    Initial contract balance: 0
3    Ending contract balance: 0
4    Initial attacker balance: 0
5    Ending attacker balance: 1000000000000000001
```

Recommended Mitigation:

Replace `msg.sender` with `_receiver` in two places in the function.

```
1  function withdrawAllFailedCredits(address _receiver) external {
2      ...
3  -   failedTransferCredits[msg.sender] = 0;
4  +   failedTransferCredits[_receiver] = 0;
5
6  -   (bool success, ) = payable(msg.sender).call{value: amount}("");
7  +   (bool success, ) = payable(_receiver).call{value: amount}("");
8  }
```

Medium

[M-1] `BidBeastsNFTMarketPlace::placeBid` missed auction duration causing the auction to end sooner

Description:

The `placeBid` function also handles the management of the NFT auction timeframe. Based on the documentation, users expect that once an NFT is listed and the first bid arrives, the auction will proceed for precisely 3 days. After those 3 days, it should conclude automatically, with the winner selected based on the highest bid—without any alterations to the duration or extensions, except in cases where a new bid is placed less than 15 minutes before the scheduled end.

The README specifies “Auction deadline of exactly 3 days”. However, the contract implementation sets the auction duration to 15 minutes after the first bid and applies rolling extensions on subsequent bids. This behavior is inconsistent with the documentation and may cause confusion or disputes between users and the platform.

```
if (previousBidAmount == 0) {

    requiredAmount = listing.minPrice;
    require(msg.value > requiredAmount, "First bid must be > min price"
        );
@>  listing.auctionEnd = block.timestamp + S_AUCTION_EXTENSION_DURATION
    ;
    emit AuctionExtended(tokenId, listing.auctionEnd);
}
```

Risk:

Likelihood: High

- Reason 1: Since the auction logic kicks in with every bid, discrepancies in behavior are bound to surface as soon as users read the README and assume a fixed 3-day auction period.
- Reason 2: With the auction duration incorrectly set to 15 minutes instead of 3 days, this issue will affect every listing—it's not just some rare edge case.

Impact: Medium

- Impact 1: The mismatch between user expectations (a guaranteed 3-day wrap-up) and the actual implementation (which could end sooner or keep extending) might lead bidders to make poor decisions.
- Impact 2: Even though it doesn't directly cause financial losses (funds still go to the winner or seller), it could spark disputes, damage the platform's reputation, and introduce potential legal or branding risks.

Proof of Concept:

This Foundry test demonstrates sets the auction duration to 15 minutes after the first bid.

PoC

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

    vm.prank(BIDDER_1);
    market.placeBid{value: MIN_PRICE + 1}(TOKEN_ID);

    BidBeastsNFTMarket.Listing memory listing = market.getListing(0);
    console.log("Current timestamp:", block.timestamp);
    console.log("Auction End:", listing.auctionEnd);
}
```

```
Logs:
  Current timestamp: 1
  Auction End: 901
```

Where 15 min = 900 sec

Recommended Mitigation:

Set the auction duration to 3 days after the first bid.

```
+ uint256 constant public S_AUCTION_DURATION = 3 days;

    if (previousBidAmount == 0) {
        ...
-       listing.auctionEnd = block.timestamp + S_AUCTION_EXTENSION_DURATION;
+       listing.auctionEnd = block.timestamp + S_AUCTION_DURATION;
        emit AuctionExtended(tokenId, listing.auctionEnd);
    }
```

[M-2] `BidBeastsNFTMarketPlace::placeBid` auction time extension stacking bug make sniping attack

Description:

The auction logic should promote fairness by establishing a clear deadline (e.g., 3 days) and only extending the auction if a bid arrives close to the end. An ideal anti-sniping mechanism would reset the timer to “current block.timestamp + extension duration,” ensuring the end time remains consistently aligned with the intended gap from the latest bid.

That said, the current implementation accumulates extra time additively whenever the condition triggers (auctionEnd += 15 minutes). Consequently, the auction could overrun the 3-day threshold by a wide margin if multiple bids keep rolling in near the deadline. This creates an exploit opportunity for attackers to deploy bots and indefinitely postpone the auction’s closure.

```
    if (timeLeft < S_AUCTION_EXTENSION_DURATION) {
@>      listing.auctionEnd = listing.auctionEnd + S_AUCTION_EXTENSION_DURATION;
        emit AuctionExtended(tokenId, listing.auctionEnd);
    }
```

Risk:

Likelihood: High

- Reason 1: This triggers every single time a bid arrives within less than 15 minutes of the deadline, since the extension logic consistently activates.

- Reason 2: It's ripe for exploitation by trading bots that are scripted to fire off repeated bids right as the auction nears its close.

Impact: Medium

- Impact 1: Auctions might stretch well beyond the intended 3-day cap, throwing off the predictable schedules that sellers and buyers rely on.
- Impact 2: The platform could erode user confidence, as the auction mechanics fall short of the documented promise of an "exactly 3 days" timeframe.

Proof of Concept:

This Foundry test demonstrates the auction could overrun the 3-day threshold

```
function test_placeBid_auctionMoreThan3Days() public {
    _mintNFT();
    _listNFT();

    vm.prank(BIDDER_1);
    market.placeBid{value: MIN_PRICE + 1}(TOKEN_ID);
    BidBeastsNFTMarket.Listing memory listing = market.getListing(0);
    console.log("Auction End:", listing.auctionEnd);

    vm.warp(block.timestamp + 15 minutes - 1); // 15 min represent
        actually 30 days

    vm.prank(BIDDER_2);
    market.placeBid{value: MIN_PRICE * 2}(TOKEN_ID);

    BidBeastsNFTMarket.Listing memory listing2 = market.getListing(0);
    console.log("Auction End after second bid:", listing2.auctionEnd);
}
```

```
Logs:
  Auction End: 901
  Auction End after second bid: 1801
```

Recommended Mitigation:

There's 2 option:

1. Keep anti-sniping extension

```
 if (timeLeft < S_AUCTION_EXTENSION_DURATION) {
-    listing.auctionEnd = listing.auctionEnd +
    S_AUCTION_EXTENSION_DURATION;
+    listing.auctionEnd = block.timestamp +
    S_AUCTION_EXTENSION_DURATION;
      emit AuctionExtended(tokenId, listing.auctionEnd);
```

```
5   }
```

1. Fixed hard deadline

```
1  - uint256 constant public S_AUCTION_EXTENSION_DURATION = 15 minutes;
2
3  function placeBid(uint256 tokenId) external payable isListed(tokenId) {
4      ...
5      else {
6          requiredAmount = (previousBidAmount / 100) * (100 +
               S_MIN_BID_INCREMENT_PERCENTAGE);
7          require(msg.value >= requiredAmount, "Bid not high enough");
8
9  -       uint256 timeLeft = 0;
10 -       if (listing.auctionEnd > block.timestamp) {
11 -           timeLeft = listing.auctionEnd - block.timestamp;
12 -       }
13 -       if (timeLeft < S_AUCTION_EXTENSION_DURATION) {
14 -           listing.auctionEnd = listing.auctionEnd +
       S_AUCTION_EXTENSION_DURATION;
15 -           emit AuctionExtended(tokenId, listing.auctionEnd);
16 -       }
17 +       emit AuctionExtended(tokenId, listing.auctionEnd);
18     }
19 }
```

Low

[L-1] `BidBeastsNFTMarketPlace::placeBid` comparison options make unexpected NFT minimum price

Description:

This function is designed to allow bidders or buyers to submit their offers for acquiring the desired NFT, with the minimum bid price set at `listing.minPrice`—as established by the seller during the listing process.

However, a flaw in the comparison operator—using > (greater than) instead—forces bids to exceed `listing.minPrice`. This deviates from the protocol's documentation and the README.md file.

```
1  if (previousBidAmount == 0) {
2
3      requiredAmount = listing.minPrice;
4  @>  require(msg.value > requiredAmount, "First bid must be > min price"
       );
5      listing.auctionEnd = block.timestamp + S_AUCTION_EXTENSION_DURATION
           ;
```

```
6         emit AuctionExtended(tokenId, listing.auctionEnd);
7     }
```

Risk:

Likelihood: High

- Reason 1: It's practically guaranteed that every prospective buyer will call this function to submit their bids.
- Reason 2: The initial bidder or buyer will run into this particular code block.

Impact: Low

- Impact 1: While it won't result in any financial losses, it nonetheless strays from the protocol's intended logic outlined in the documentation.
- Impact 2: It compels the first bidder to submit an offer exceeding `listing.minPrice`.

Proof of Concept:

This Foundry test returned true which mean expected revert has fulfilled.

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

    vm.prank(BIDDER_1);
    vm.expectRevert("First bid must be > min price");
    market.placeBid{value: MIN_PRICE}(TOKEN_ID);
}
```

Recommended Mitigation:

Replace > with >= to allow bids equal to the minimum price.

```
if (previousBidAmount == 0) {

    requiredAmount = listing.minPrice;
-   require(msg.value > requiredAmount, "First bid must be > min price");
+   require(msg.value >= requiredAmount, "First bid must be >= min price");
    listing.auctionEnd = block.timestamp + S_AUCTION_EXTENSION_DURATION;
    emit AuctionExtended(tokenId, listing.auctionEnd);
}
```

Information

[I-1] `BidBeastsNFTMarketPlace` inconsistent naming convention for constants

Description:

Constants are prefixed with `S_` (e.g., `S_AUCTION_EXTENSION_DURATION`), which is typically used for state/storage variables.
This may confuse developers, as constants are not stored in contract storage.

Recommended Mitigation:

Adopt a consistent naming scheme.
For example, use `AUCTION_EXTENSION_DURATION` for constants, and reserve `s_` prefix exclusively for storage state variables.

[I-2] Uncommon state naming on both `BidBeasts_NFT_ERC721` and `BidBeastsNFTMarketPlace` causes confusion

Description:

The common naming of state variable in solidity is using camel case. However the contracts use names `CurrentTokenID` and `BBERC721` as the state variables, which can lead to confusion and hard to maintain.

Recommended Mitigation:

Rename the state variables to common naming convention like use camel case.

```
- uint256 public CurrentTokenID;
+ uint256 public currentTokenId;
...
- BidBeasts public BBERC721;
+ BidBeasts public bidBeasts;
```

[I-3] `BidBeastsNFTMarketPlace` lack of event indexed params make it hard to query

Description:

Indexed params on smart contract events can significantly enhance the efficiency of querying logs. However, the current implementation lacks indexed parameters.

Impact:

when this smart contract is integrated with defi which requires listening to on-chain or smart contract then it will be challenging to filter and retrieve specific events.

Recommended Mitigation:

Add `indexed` params at least one of each events.

```
- event NftListed(uint256 tokenId, address seller, uint256 minPrice, uint256 buyNowPrice);
+ event NftListed(uint256 indexed tokenId, address indexed seller, uint256 minPrice, uint256 buyNowPrice);
- event NftUnlisted(uint256 tokenId);
+ event NftUnlisted(uint256 indexed tokenId);
- event BidPlaced(uint256 tokenId, address bidder, uint256 amount);
+ event BidPlaced(uint256 indexed tokenId, address indexed bidder, uint256 amount);
- event AuctionExtended(uint256 tokenId, uint256 newDeadline);
+ event AuctionExtended(uint256 indexed tokenId, uint256 newDeadline);
- event AuctionSettled(uint256 tokenId, address winner, address seller, uint256 price);
+ event AuctionSettled(uint256 indexed tokenId, address indexed winner, address indexed seller, uint256 price);
- event FeeWithdrawn(uint256 amount);
+ event FeeWithdrawn(uint256 indexed amount);
```

[I-4] `BidBeastsNFTMarketPlace::constructor` lacks zero address check for token parameters

Description:

The `BidBeastsNFTMarketPlace` constructor accept `_BidBeastsNFT` address but doesn't validate that these addresses are not `address(0)`.

Impact:

- Market place becomes completely unusable if deployed with zero addresses
- Requires complete redeployment of the pool contract to fix
- Potential loss of gas costs from failed deployment

Recommended Mitigation:

Add zero address checks in the constructor.

```
constructor(address _BidBeastsNFT) {
-   BBERC721 = BidBeasts(_BidBeastsNFT);
```

```
+    require(_BidBeastsNFT != address(0), "Invalid BidBeastsNFT address");
+    BBERC721 = BidBeasts(_BidBeastsNFT);
 }
```

[I-5] `BidBeastsNFTMarketPlace::listNFT` inconsistent params naming

Description:

The function require 3 params to be called, that's `tokenId`,, `_minPrice`,, and `_buyNowPrice` to list the NFT on the market place. But inconsistent naming conventions are used for these parameters.

Impact:

Make confusing whether this is a parameter or state or even another function, and hard to maintain for developers.

Recommended Mitigation:

1. Rename the params naming.

```
- function listNFT(uint256 tokenId, uint256 _minPrice, uint256 _buyNowPrice) external {
+ function listNFT(uint256 tokenId, uint256 minPrice, uint256 buyNowPrice) external {
```

2. Update the natspec.

```
 /**
  * @notice Lists an NFT for auction.
  * @param tokenId The ID of the token to list.
- * @param _minPrice The starting price for the auction.
+ * @param minPrice The starting price for the auction.
- * @param _buyNowPrice The price for immediate purchase (set to 0 to disable).
+ * @param buyNowPrice The price for immediate purchase (set to 0 to disable).
  */
```

[I-6] `BidBeastsNFTMarketPlace` missing parameters documentation in natspec

Description:

Several functions do not document their parameters in NatSpec comments. This reduces readability and maintainability.

Affected functions:

- unlistNFT(uint256 tokenId)
- placeBid(uint256 tokenId)
- withdrawProceeds()
- … (total: 7 functions)

Recommended Mitigation:

For each function, include proper `@param` tags describing the purpose of every input parameter.
Example

```
1  /**
2   * @notice Allows the seller to unlist an NFT if no bids have been
       made.
3 + * @param tokenId The ID of the token to unlist.
4   */
5  function unlistNFT(uint256 tokenId) external isListed(tokenId)
      isSeller(tokenId, msg.sender) {
```

[I-7] `BidBeastsNFTMarketPlace` magic number should be replaced

Description:

The contract contains multiple hardcoded magic numbers.
This reduces readability and makes maintenance harder if the values need to be updated.

Affected Functions

- placeBid()
- _executeSale()

Recommended Mitigation:

Replace magic numbers with named constants or immutable variables.

```
1 + uint256 constant public S_DENOMINATOR = 100;
2 ...
3 - uint256 fee = (bid.amount * S_FEE_PERCENTAGE) / 100;
4 + uint256 fee = (bid.amount * S_FEE_PERCENTAGE) / S_DENOMINATOR;
5 ...
6 - requiredAmount = (previousBidAmount / 100) * (100 +
     S_MIN_BID_INCREMENT_PERCENTAGE);
7 + requiredAmount = (previousBidAmount / S_DENOMINATOR) * (S_DENOMINATOR
       + S_MIN_BID_INCREMENT_PERCENTAGE);
```