Protocol Audit Report

Version 1.0

Cyfrin.io

October 2, 2025

Protocol Audit Report

Cyfrin.io

October 2, 2025

Prepared by: YoYiL Lead Auditors: - YoYiL

Table of Contents

    * [M-2] Missing Critical Event Emissions When State Variables Change (Poor Observability + Reduced Off-chain Integration)
    * [M-3] Division Before Multiplication Causes Precision Loss in Bid Increment Calculation
    * [M-4] Auction Duration Mismatch Creates Extremely Short Auctions
    * [M-5] Incorrect Event Emission During Bid Placement Causes Misleading Auction State
  - Low
    * [L-1] Missing Zero Address Validation in Constructor (Deployment Failure + Contract Unusability)
    * [L-2] Inconsistent Bid Requirements Create Buy-Now Bypass Vulnerability
    * [L-3] Unsafe NFT Transfer to Contract Addresses Can Permanently Lock NFTs
  - Info
    * [I-1] NFT collection name mismatch - constructor uses “Goddie_NFT” instead of “BidBeasts”
    * [I-2] OpenZeppelin version should be specified to ensure Ownable(msg.sender) compatibility
    * [I-3] `mint` function doesn’t follow Checks-Effects-Interactions pattern
    * [I-4] State variable mappings should follow naming convention with `s_` prefix
    * [I-5] Events lack `indexed` parameters for efficient off-chain filtering
    * [I-6] Magic numbers should be replaced with named constants
    * [I-7] Incorrect NatSpec comment for `_payout` function
    * [I-8] Solidity version 0.8.20 may cause deployment issues on certain chains due to PUSH0 opcode
  - Gas
    * [G-1] State variable `CurrenTokenID` should be declared as private to save gas
    * [G-2] State variable `BBERC721` should be declared as `immutable` to save gas
    * [G-3] Public functions not used internally should be marked as external to save gas

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.

Disclaimer

The YoYiL 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

```
*---lib/
*---src/
*   *--- 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

This security audit of the BidBeasts NFT Marketplace protocol identified **21 findings** across multiple severity levels, with critical vulnerabilities in access control, fund management, and core auction mechanics that require immediate attention.

Issues found

Severity	Number of issues found
High	2
Medium	5
Low	3
Info	8
Gas	3
Total	21

Findings

High

[H-1] `src/BidBeasts_NFT_ERC721.sol: BidBeasts::burn(uint256 _tokenId)` lacks access control, allowing any caller to burn any user's NFT

Description

- Normally, only the NFT owner (or an authorized operator/approved address) can burn an NFT.
- In this protocol, `src/BidBeasts_NFT_ERC721.sol: BidBeasts::burn(uint256 _tokenId)` lacks access control, allowing anyone to burn any NFT.

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

Risk

Likelihood:

- This occurs whenever a malicious user calls `burn(uint256 _tokenId)`

Impact:

- Any user's NFTs can be irreversibly destroyed.

Proof of Concept

First we need to make a quick fix in `test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTes` `::setUp()`

```
    function setUp() public {
        // Deploy contracts
-       vm.prank(OWNER);
+       vm.startPrank(OWNER);
        nft = new BidBeasts();
        market = new BidBeastsNFTMarket(address(nft));
        rejector = new RejectEther();

        vm.stopPrank();

        // Fund users
        vm.deal(SELLER, STARTING_BALANCE);
        vm.deal(BIDDER_1, STARTING_BALANCE);
        vm.deal(BIDDER_2, STARTING_BALANCE);
    }
```

Please add the following test to `test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTest` :

```
    function testAnyoneCanBurnAnyNFT() public {
        _mintNFT();
        assertEq(nft.ownerOf(TOKEN_ID), SELLER);
        vm.prank(BIDDER_1);
        nft.burn(TOKEN_ID);
        vm.expectRevert(abi.encodeWithSelector(IERC721Errors.
            ERC721NonexistentToken.selector, TOKEN_ID));
        address NFTowner = nft.ownerOf(TOKEN_ID);
    }
```

Then run: `forge test --mt testAnyoneCanBurnAnyNFT`

Output:

```
Ran 1 test for test/BidBeastsMarketPlaceTest.t.sol:
    BidBeastsNFTMarketTest
[PASS] testAnyoneCanBurnAnyNFT() (gas: 73156)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 758.55us
    (70.77us CPU time)
```

Recommended Mitigation

In `src/BidBeasts_NFT_ERC721.sol: BidBeasts::burn(uint256 _tokenId)`:

```
    function burn(uint256 _tokenId) public {
+        address owner = ownerOf(_tokenId);
+        if (!_isAuthorized(owner, msg.sender, _tokenId)) {
+            revert ERC721InsufficientApproval(msg.sender, _tokenId);
+        }
        _burn(_tokenId);
        emit BidBeastsBurn(msg.sender, _tokenId);
    }
```

[H-2] Anyone Can Withdraw Failed Transfer Credits Belonging to Other Users (Access Control Bypass + Funds Theft)

Description

- The `withdrawAllFailedCredits` function is designed to allow users to withdraw their own failed transfer credits that accumulated when direct ETH transfers failed during the auction process.
- However, the function accepts an arbitrary `_receiver` parameter but always sends the withdrawn funds to `msg.sender`, creating a critical vulnerability where any user can withdraw failed credits belonging to other users.

```
    /**
     * @notice Allows users to withdraw funds that failed to be
        transferred directly.
     */
@>  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:

- Failed transfer credits will accumulate whenever bidders are contracts that cannot receive ETH or when recipients have fallback functions that revert.
- Any user can call this function at any time with any other user's address as the _receiver parameter.

Impact:

- Complete theft of all failed transfer credits belonging to any user.
- Undermines the entire failed transfer credit system designed to protect users.

Proof of Concept

First we need to make a quick fix in `test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTes` `::setUp()`

```
    function setUp() public {
        // Deploy contracts
-       vm.prank(OWNER);
+       vm.startPrank(OWNER);
        nft = new BidBeasts();
        market = new BidBeastsNFTMarket(address(nft));
        rejector = new RejectEther();

        vm.stopPrank();

        // Fund users
        vm.deal(SELLER, STARTING_BALANCE);
        vm.deal(BIDDER_1, STARTING_BALANCE);
        vm.deal(BIDDER_2, STARTING_BALANCE);
    }
```

Please add the following test to `test/BidBeastsMarketPlaceTest.t.sol`:

```
contract RejectEther {
    // Intentionally has no payable receive or fallback
    function bid(BidBeastsNFTMarket market, uint256 tokenId, uint256
        amount) external payable {
```

```
        market.placeBid{value: amount}(tokenId);
    }
}

contract BidBeastsNFTMarketTest is Test {
    event BidPlaced(uint256 indexed tokenId, address indexed bidder,
        uint256 amount);
    // existing code
    function testAnyoneCanWithdrawAnyFailedCredits() public {
        _mintNFT();
        _listNFT();
        /* ------------------------------ rejector bids
            ------------------------------ */
        vm.deal(address(rejector), 2 ether);
        console2.log("Rejector's balance before bidding: ", address(
            rejector).balance);
        uint256 rejectorBeforeBidding = address(rejector).balance;
        assertEq(2 ether, rejectorBeforeBidding);
        rejector.bid(market, TOKEN_ID, MIN_PRICE + 1);
        console2.log("Rejector's balance after bidding: ", address(
            rejector).balance);
        uint256 rejectorAfterBidding = address(rejector).balance;
        assertEq(rejectorAfterBidding + MIN_PRICE + 1,
            rejectorBeforeBidding);
        BidBeastsNFTMarket.Bid memory highestBid1 = market.
            getHighestBid(TOKEN_ID);
        assertEq(highestBid1.bidder, address(rejector));
        assertEq(highestBid1.amount, MIN_PRICE + 1);
        assertEq(market.getListing(TOKEN_ID).auctionEnd, block.
            timestamp + market.S_AUCTION_EXTENSION_DURATION());
        /* ------------------------------ BIDDER_1 bids
            ------------------------------ */
        vm.prank(BIDDER_1);
        market.placeBid{value: MIN_PRICE + 1 ether}(TOKEN_ID);
        BidBeastsNFTMarket.Bid memory highestBid2 = market.
            getHighestBid(TOKEN_ID);
        assertEq(highestBid2.bidder, BIDDER_1);
        assertEq(highestBid2.amount, MIN_PRICE + 1 ether);
        assertEq(market.getListing(TOKEN_ID).auctionEnd, block.
            timestamp + market.S_AUCTION_EXTENSION_DURATION());
        console2.log("Rejector's balance after a higher bidding: ",
            address(rejector).balance);
        uint256 rejectorAfterAnotherHigerBidding = address(rejector).
            balance;
        assertEq(rejectorAfterAnotherHigerBidding, rejectorAfterBidding
            );
        /* ----------------- BIDDER_2  withdraws rejector's credits
            ----------------- */
        uint256 bidder2Beforewithdraw = address(BIDDER_2).balance;
        console2.log("BIDDER_2's balance before withdraw: ", address(
            BIDDER_2).balance);
        vm.prank(BIDDER_2);
        market.withdrawAllFailedCredits(address(rejector));
        console2.log("BIDDER_2's balance after withdraw: ", address(
            BIDDER_2).balance);
        assertEq(address(BIDDER_2).balance, bidder2Beforewithdraw +
            MIN_PRICE + 1);
    }
}
```

Then run: `forge test --mt testAnyoneCanWithdrawAnyFailedCredits -vv`

Output:

```
Ran 1 test for test/BidBeastsMarketPlaceTest.t.sol:
    BidBeastsNFTMarketTest
[PASS] testAnyoneCanWithdrawAnyFailedCredits() (gas: 374804)
Logs:
  Rejector's balance before bidding:  2000000000000000000
  Rejector's balance after bidding:  999999999999999999
  Rejector's balance after a higher bidding:  999999999999999999
  BIDDER_2's balance before withdraw:  100000000000000000000
  BIDDER_2's balance after withdraw:  101000000000000000001

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 811.37us
    (220.32us CPU time)
```

Recommended Mitigation

Remove the `_receiver` parameter and ensure users can only withdraw their own credits.

```
- 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");
- }

+ function withdrawAllFailedCredits() external {
+     uint256 amount = failedTransferCredits[msg.sender];
+     require(amount > 0, "No credits to withdraw");
+     failedTransferCredits[msg.sender] = 0;
+
+     (bool success,) = payable(msg.sender).call{value: amount}("");
+     require(success, "Withdraw failed");
+ }
```

Medium

[M-1] Missing tokenURI Implementation Breaks NFT Display and Trading

Description

- The BidBeasts NFT contract should provide metadata for each token through the tokenURI() function, allowing wallets, marketplaces, and other applications to display NFT information such as name, description, and image. This is a fundamental requirement for NFT functionality in the Web3 ecosystem.
- The contract inherits from OpenZeppelin's ERC721 but fails to implement the tokenURI() function or override _baseURI(), resulting in empty metadata for all minted NFTs. This renders the NFTs functionally useless across all major platforms and applications.

```
contract BidBeasts is ERC721, Ownable(msg.sender) {
    // ... other code ...

@>  // Missing tokenURI implementation
@>  // Missing _baseURI override
@>  // Results in empty string returned for all token metadata
}
```

Risk

Likelihood:

- Every minted NFT will have empty metadata since there is no implementation of tokenURI functionality
- All users attempting to view or trade NFTs on any platform will encounter this issue immediately upon interaction

Impact:

- NFTs cannot be displayed properly on major marketplaces (OpenSea, Rarible, etc.) and will appear as "Unnamed" tokens
- The NFT collection loses all commercial value and utility in the broader ecosystem. This will highly disturb `BidBeastsMarketPlace`'s functionality.

Proof of Concept

First we need to make a quick fix in `test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTes`
`::setUp()`

```
    function setUp() public {
        // Deploy contracts
-       vm.prank(OWNER);
+       vm.startPrank(OWNER);
        nft = new BidBeasts();
        market = new BidBeastsNFTMarket(address(nft));
        rejector = new RejectEther();

        vm.stopPrank();

        // Fund users
        vm.deal(SELLER, STARTING_BALANCE);
        vm.deal(BIDDER_1, STARTING_BALANCE);
        vm.deal(BIDDER_2, STARTING_BALANCE);
    }
```

Please add the following test to `test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTest`
:

```
    function testUserCanNotQueryValidNFTMetadata() public {
        _mintNFT();
        vm.prank(SELLER);
        string memory tokenURI = nft.tokenURI(TOKEN_ID);
        assertEq(tokenURI, "");
    }
```

Then run: `forge test --mt testUserCanNotQueryValidNFTMetadata`

Output:

```
Ran 1 test for test/BidBeastsMarketPlaceTest.t.sol:
    BidBeastsNFTMarketTest
[PASS] testUserCanNotQueryValidNFTMetadata() (gas: 87490)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 528.63us
    (60.13us CPU time)
```

Recommended Mitigation

In `src/BidBeasts_NFT_ERC721.sol: BidBeasts`:

```
contract BidBeasts is ERC721, Ownable(msg.sender) {
    // existing events...
+   event MetadataUpdate(uint256 indexed tokenId, string uri);

+   mapping(uint256 => string) private s_tokenURIs;

    // existing state variables...

    // existing constructor...

```

```diff
-   function mint(address to) public onlyOwner returns (uint256) {
+   function mint(address to, string calldata uri) public onlyOwner
  returns (uint256) {
        uint256 _tokenId = CurrenTokenID;
        _safeMint(to, _tokenId);
+       s_tokenURIs[_tokenId] = uri;
+       emit MetadataUpdate(_tokenId, uri);
        emit BidBeastsMinted(to, _tokenId);
        CurrenTokenID++;
        return _tokenId;
    }

    function burn(uint256 _tokenId) public {
        _burn(_tokenId);
+       delete s_tokenURIs[_tokenId];
+       emit MetadataUpdate(_tokenId, "");
        emit BidBeastsBurn(msg.sender, _tokenId);
    }

+   function tokenURI(uint256 tokenId) public view override returns (
  string memory) {
+       _requireOwned(tokenId);
+       return s_tokenURIs[tokenId];
+   }
}
```

Alternative Solutions:

1. Base URI + Token ID Pattern: Override _baseURI() to return a base URL and concatenate with token ID (e.g., https://api.example.com/metadata/{tokenId}.json)
2. Centralized Metadata Server: Implement a single base URI pointing to a centralized metadata service that serves JSON metadata based on token ID
3. IPFS Integration: Store metadata on IPFS and use IPFS hashes as token URIs for decentralized metadata storage

[M-2] Missing Critical Event Emissions When State Variables Change (Poor Observability + Reduced Off-chain Integration)

Description

- Events should be emitted whenever important state variables are modified to provide transparency and enable off-chain monitoring of contract activities.
- The contract fails to emit events for several critical state changes including contract initialization, bid clearing, listing status updates, and failed credit withdrawals, making it difficult for external systems and users to track these important activities.

In `src/BidBeastsNFTMarketPlace.sol`:

```
contract BidBeastsNFTMarket is Ownable(msg.sender) {
    //existing code
    constructor(address _BidBeastsNFT) {
@>      BBERC721 = BidBeasts(_BidBeastsNFT);//Line#57
    }
    //existing code
    function listNFT(uint256 tokenId, uint256 _minPrice, uint256
        _buyNowPrice) external{
        //existing code
        if (listing.buyNowPrice > 0 && msg.value >= listing.buyNowPrice
            ) {
            //existing code
@>          bids[tokenId] = Bid(msg.sender, salePrice);//Line#124
@>          listing.listed = false;//Line#125
            //existing code
        }
        //existing code
    }
    //existing code
    function _executeSale(uint256 tokenId) internal {
        //existing code
@>      listing.listed = false;//Line#210
@>      delete bids[tokenId];//Line#211
        //existing code
    }
    function withdrawAllFailedCredits(address _receiver) external {
        //existing code
@>      failedTransferCredits[msg.sender] = 0;//Line#242
        //existing code
    }
    //existing code
}
```

Risk

Likelihood:

- These state changes occur during normal contract operations - contract deployment, buy-now purchases, auction completions, and failed credit withdrawals.
- Off-chain systems and dApps commonly rely on events to track contract state changes and update their interfaces accordingly.

Impact:

- Reduced transparency and auditability of contract operations.
- Difficulty for off-chain systems, indexers, and dApps to track important state changes.

Proof of Concept

The following code demonstrates the missing event emissions that prevent proper off-chain monitoring and integration:

```
contract BidBeastsNFTMarket is Ownable(msg.sender) {
    //existing code
    constructor(address _BidBeastsNFT) {
        BBERC721 = BidBeasts(_BidBeastsNFT);//Line#57
        // No event emitted - off-chain systems cannot detect contract
            initialization
    }
    //existing code
    function listNFT(uint256 tokenId, uint256 _minPrice, uint256
        _buyNowPrice) external{
        //existing code
        if (listing.buyNowPrice > 0 && msg.value >= listing.buyNowPrice
            ) {
            //existing code
            bids[tokenId] = Bid(msg.sender, salePrice);//Line#124
            // No event emitted - bid placement cannot be tracked
            listing.listed = false;//Line#125
            // No event emitted - NFT unlisting cannot be detected
            //existing code
        }
        //existing code
    }
    //existing code
    function _executeSale(uint256 tokenId) internal {
        //existing code
        listing.listed = false;//Line#210
        // No event emitted - sale completion status change is
            invisible
        delete bids[tokenId];//Line#211
        // No event emitted - bid clearing cannot be monitored
        //existing code
    }
    function withdrawAllFailedCredits(address _receiver) external {
        //existing code
        failedTransferCredits[msg.sender] = 0;//Line#242
        // No event emitted - credit withdrawal cannot be tracked for
            accounting
        //existing code
    }
    //existing code
}
```

Recommended Mitigation

Add appropriate event emissions for all critical state changes to improve transparency and enable proper off-chain integration.

```
contract BidBeastsNFTMarket is Ownable(msg.sender) {
    //existing code
+   event BidBeastsContractSet(address indexed newContract);
+   event BidCleared(uint256 indexed tokenId);
+   event FailedCreditsWithdrawn(address indexed user, uint256 amount);
    constructor(address _BidBeastsNFT) {
        BBERC721 = BidBeasts(_BidBeastsNFT);//Line#57
+       emit BidBeastsContractSet(_BidBeastsNFT);
    }
    //existing code
    function listNFT(uint256 tokenId, uint256 _minPrice, uint256
        _buyNowPrice) external{
        //existing code
        if (listing.buyNowPrice > 0 && msg.value >= listing.buyNowPrice
            ) {
            //existing code
            bids[tokenId] = Bid(msg.sender, salePrice);//Line#124
+           emit BidPlaced(tokenId, msg.sender, salePrice);
            listing.listed = false;//Line#125
+           emit NftUnlisted(tokenId);
            //existing code
        }
        //existing code
    }
    //existing code
    function _executeSale(uint256 tokenId) internal {
        //existing code
        listing.listed = false;//Line#210
+       emit NftUnlisted(tokenId);
        delete bids[tokenId];//Line#211
+       emit BidCleared(tokenId);
        //existing code
    }
    function withdrawAllFailedCredits(address _receiver) external {
        //existing code
        failedTransferCredits[msg.sender] = 0;//Line#242
+       emit FailedCreditsWithdrawn(msg.sender, amount);
        //existing code
    }
    //existing code
}
```

[M-3] Division Before Multiplication Causes Precision Loss in Bid Increment Calculation

Description

- The auction system should calculate the minimum required bid increment by adding 5% to the previous bid amount with proper precision.

- The bid increment calculation performs division before multiplication, causing precision loss due to Solidity's integer arithmetic truncation, which results in lower minimum bid requirements than intended.

```
function placeBid(uint256 tokenId) external payable isListed(tokenId) {
    // ... existing code ...

    if (previousBidAmount == 0) {
        // First bid logic
    } else {
@>      requiredAmount = (previousBidAmount / 100) * (100 +
    S_MIN_BID_INCREMENT_PERCENTAGE); //Division before multiplication
    loses precision
        require(msg.value >= requiredAmount, "Bid not high enough");
        // ... existing code ...
    }
    // ... existing code ...
}
```

Risk

Likelihood:

- This precision loss occurs on every subsequent bid after the first bid in any auction.
- The issue is more pronounced with smaller bid amounts where the truncation has a larger relative impact.
- Happens automatically as part of the core bidding mechanism without any special conditions required.

Impact:

- Protocol loses potential fee revenue from higher final sale amounts.
- Sellers receive lower final sale prices due to insufficient bid increments driving up the price.

Proof of Concept

```
// Example with previousBidAmount = 0.099 ether (99000000000000000 wei)
uint256 previousBidAmount = 0.099 ether; // 99000000000000000 wei
uint256 S_MIN_BID_INCREMENT_PERCENTAGE = 5;

// Current implementation (division first):
uint256 wrongCalculation = (previousBidAmount / 100) * (100 +
    S_MIN_BID_INCREMENT_PERCENTAGE);
// = (99000000000000000 / 100) * 105
// = 990000000000000 * 105
// = 103950000000000000 wei (0.10395 ether)

// Correct implementation (multiplication first):
uint256 correctCalculation = (previousBidAmount * (100 +
    S_MIN_BID_INCREMENT_PERCENTAGE)) / 100;
// = (99000000000000000 * 105) / 100
// = 10395000000000000000 / 100
// = 103950000000000000 wei (0.10395 ether)

// In this case both are equal, but with amounts like 0.0999 ether:
previousBidAmount = 99900000000000000; // 0.0999 ether

// Wrong: (99900000000000000 / 100) * 105 = 999000000000000 * 105 =
    104895000000000000
// Correct: (99900000000000000 * 105) / 100 = 10489500000000000000 /
    100 = 104895000000000000

// The difference becomes significant with smaller decimals:
previousBidAmount = 99; // Very small amount for demonstration

// Wrong: (99 / 100) * 105 = 0 * 105 = 0 (complete loss!)
// Correct: (99 * 105) / 100 = 10395 / 100 = 103
```

Recommended Mitigation

Perform multiplication before division to preserve precision:

```
function placeBid(uint256 tokenId) external payable isListed(tokenId) {
    // ... existing code ...

    } else {
-        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 ...
    }

    // ... rest of function ...
}
```

[M-4] Auction Duration Mismatch Creates Extremely Short Auctions

Description

- The protocol documentation specifies that auctions should have an "Auction deadline of exactly 3 days" to provide sufficient time for competitive bidding.
- The contract implementation sets the auction duration to only 15 minutes (`S_AUCTION_EXTENSION_DURATI`
  `)` instead of the documented 3 days, creating extremely short auction windows that prevent

proper price discovery.

In `src/BidBeastsNFTMarketPlace.sol`:

```
contract BidBeastsNFTMarket is Ownable(msg.sender) {
    // ... existing code ...
@>  uint256 constant public S_AUCTION_EXTENSION_DURATION = 15 minutes;
    //existing code
    function placeBid(uint256 tokenId) external payable isListed(
        tokenId) {
    // ... existing code ...

    if (previousBidAmount == 0) {
        requiredAmount = listing.minPrice;
        require(msg.value > requiredAmount, "First bid must be > min
            price");
@>      listing.auctionEnd = block.timestamp +
    S_AUCTION_EXTENSION_DURATION; //@> Sets 15 minutes instead of 3 days
        emit AuctionExtended(tokenId, listing.auctionEnd);
    }
    // ... rest of function ...
}
```

Risk

Likelihood:

- This occurs on every first bid placed on any NFT listing, as the auction timer is automatically set to the incorrect duration.
- The 15-minute window is far too short for most users to discover and participate in auctions, especially across different time zones.
- Users expect 3-day auctions based on the protocol specification and will miss bidding opportunities.

Impact:

- Sellers receive significantly lower sale prices due to insufficient time for competitive bidding and price discovery.
- Most potential bidders miss auction opportunities due to the extremely short 15-minute window.

Proof of Concept

First we need to make a quick fix in `test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTes` `::setUp()`

```
    function setUp() public {
        // Deploy contracts
```

```
-       vm.prank(OWNER);
+       vm.startPrank(OWNER);
        nft = new BidBeasts();
        market = new BidBeastsNFTMarket(address(nft));
        rejector = new RejectEther();

        vm.stopPrank();

        // Fund users
        vm.deal(SELLER, STARTING_BALANCE);
        vm.deal(BIDDER_1, STARTING_BALANCE);
        vm.deal(BIDDER_2, STARTING_BALANCE);
    }
```

Please add the following test to `test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTest`
:

```
    function testSecondBidderCanNotBidAfter1Day() public {
        _mintNFT();
        _listNFT();
        /* ------------------------------ BIDDER_1 bids ------------------------------ */
        vm.prank(BIDDER_1);
        market.placeBid{value: MIN_PRICE + 1}(TOKEN_ID);

        BidBeastsNFTMarket.Bid memory highestBid = market.getHighestBid(TOKEN_ID);
        assertEq(highestBid.bidder, BIDDER_1);
        assertEq(highestBid.amount, MIN_PRICE + 1);
        assertEq(market.getListing(TOKEN_ID).auctionEnd, block.timestamp + market.S_AUCTION_EXTENSION_DURATION());

        /* ------------------------ BIDDER_2 bids after 1 day ------------------------ */
        vm.warp(block.timestamp + 1 days);
        vm.roll(block.number + 10);
        vm.prank(BIDDER_2);
        vm.expectRevert("Auction ended");
        market.placeBid{value: MIN_PRICE + 1}(TOKEN_ID);
    }
```

Then run `forge test --mt testSecondBidderCanNotBidAfter1Day`

Output:

```
Ran 1 test for test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTest
[PASS] testSecondBidderCanNotBidAfter1Day() (gas: 305572)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 4.54ms (456.07us CPU time)
```

Recommended Mitigation

Update the auction duration constant to match the 3-day specification:

```
+ uint256 constant public S_AUCTION_DURATION = 3 days;
  uint256 constant public S_AUCTION_EXTENSION_DURATION = 15 minutes;

function placeBid(uint256 tokenId) external payable isListed(tokenId) {
    // ... existing code ...

    if (previousBidAmount == 0) {
        requiredAmount = listing.minPrice;
        require(msg.value > requiredAmount, "First bid must be > min
            price");
-       listing.auctionEnd = block.timestamp +
    S_AUCTION_EXTENSION_DURATION;
+       listing.auctionEnd = block.timestamp + S_AUCTION_DURATION;
        emit AuctionExtended(tokenId, listing.auctionEnd);
    } else {
        // Keep extension logic for subsequent bids
        // ... existing extension logic using
            S_AUCTION_EXTENSION_DURATION ...
    }
}
```

[M-5] Incorrect Event Emission During Bid Placement Causes Misleading Auction State

Description

- The AuctionSettled event should only be emitted when an auction is actually completed and finalized, indicating the final winner and sale price.
- The placeBid function incorrectly emits the AuctionSettled event during regular bid placement, even though the auction is still ongoing and no settlement has occurred.

```
function placeBid(uint256 tokenId) external payable isListed(tokenId) {
    // ... existing code ...

    require(msg.sender != previousBidder, "Already highest bidder");
@>  emit AuctionSettled(tokenId, msg.sender, listing.seller, msg.value)
    ; //Incorrectly emits settlement event during active bidding

    // --- Regular Bidding Logic ---
    // ... auction continues after this point ...
}
```

Risk

Likelihood:

- This occurs on every valid bid placement after the buy-now logic check, affecting all regular auction bidding activity.
- The event is emitted regardless of whether the auction will continue or end, creating false settlement signals.
- External monitoring systems and frontend applications will receive incorrect auction state information on every bid.

Impact:

- Off-chain monitoring systems and indexers will incorrectly interpret ongoing auctions as completed, leading to data inconsistencies.
- Frontend applications may display incorrect auction status, confusing users about whether auctions are still active.
- Analytics and reporting tools will show inflated settlement counts and incorrect auction completion data.

Proof of Concept

First we need to make a quick fix in `test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTes` `::setUp()`

```
    function setUp() public {
        // Deploy contracts
-       vm.prank(OWNER);
+       vm.startPrank(OWNER);
        nft = new BidBeasts();
        market = new BidBeastsNFTMarket(address(nft));
        rejector = new RejectEther();

        vm.stopPrank();

        // Fund users
        vm.deal(SELLER, STARTING_BALANCE);
        vm.deal(BIDDER_1, STARTING_BALANCE);
        vm.deal(BIDDER_2, STARTING_BALANCE);
    }
```

Please add the following test to `test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTest` :

```
    event AuctionSettled(uint256 tokenId, address winner, address
       seller, uint256 price);

    function testIncorrectEmit() public {
        _mintNFT();
        _listNFT();
        /* ------------------------------ BIDDER_1 bids
            ----------------------------- */
        vm.prank(BIDDER_1);
        vm.expectEmit(address(market));
        emit AuctionSettled(TOKEN_ID, BIDDER_1, SELLER, MIN_PRICE + 1);
        market.placeBid{value: MIN_PRICE + 1}(TOKEN_ID);
    }
```

Then run `forge test --mt testIncorrectEmit`

Output:

```
Ran 1 test for test/BidBeastsMarketPlaceTest.t.sol:
    BidBeastsNFTMarketTest
[PASS] testIncorrectEmit() (gas: 288422)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 836.25us
    (92.11us CPU time
```

Recommended Mitigation

Remove the incorrect event emission from the placeBid function:

```
function placeBid(uint256 tokenId) external payable isListed(tokenId) {
    // ... existing code ...

    require(msg.sender != previousBidder, "Already highest bidder");
-   emit AuctionSettled(tokenId, msg.sender, listing.seller, msg.value)
    ;

    // --- Regular Bidding Logic ---
    // ... rest of function ...
}
```

Low

[L-1] Missing Zero Address Validation in Constructor (Deployment Failure + Contract Unusability)

Description

- The constructor should validate that critical address parameters are not zero addresses to prevent deployment with invalid configurations that would render the contract unusable.
- The constructor fails to validate the _BidBeastsNFT parameter, allowing the contract to be deployed with a zero address for the NFT contract, which would cause all NFT-related operations to fail.

In `src/BidBeastsNFTMarketPlace.sol`:

```
    constructor(address _BidBeastsNFT) {
@>      BBERC721 = BidBeasts(_BidBeastsNFT);
    }
```

Risk

Likelihood:

- Contract deployment occurs during initial setup when human error in providing constructor parameters is most likely.
- Zero address is a common default value that could be accidentally passed during deployment scripts or manual deployment.

Impact:

- Contract becomes completely unusable as all NFT operations (listing, transferring, ownership checks) will revert.
- Requires expensive redeployment and migration of any existing state or integrations.

Proof of Concept

First we need to make a quick fix in `test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTes` `::setUp()`

```
    function setUp() public {
        // Deploy contracts
-       vm.prank(OWNER);
+       vm.startPrank(OWNER);
        nft = new BidBeasts();
        market = new BidBeastsNFTMarket(address(nft));
        rejector = new RejectEther();

        vm.stopPrank();

        // Fund users
        vm.deal(SELLER, STARTING_BALANCE);
        vm.deal(BIDDER_1, STARTING_BALANCE);
        vm.deal(BIDDER_2, STARTING_BALANCE);
    }
```

Please add the following test to `test/BidBeastsMarketPlaceTest.t.sol`:

```
    function testZeroAddressDepoly() public {
        BidBeastsNFTMarket failMarket = new BidBeastsNFTMarket(address
            (0));
    }
```

Then run `forge test --mt testZeroAddressDepoly`:

Output:

```
Ran 1 test for test/BidBeastsMarketPlaceTest.t.sol:
    BidBeastsNFTMarketTest
[PASS] testZeroAddressDepoly() (gas: 2344864)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 585.95us
    (69.76us CPU time)
```

Recommended Mitigation

```
constructor(address _BidBeastsNFT) {
+    require(_BidBeastsNFT != address(0), "BidBeasts NFT contract cannot
     be zero address");
     BBERC721 = BidBeasts(_BidBeastsNFT);
}
```

[L-2] Inconsistent Bid Requirements Create Buy-Now Bypass Vulnerability

Description

- The auction system should consistently enforce that regular bids must exceed the minimum price while buy-now purchases can be made at exactly the buy-now price.
- The contract has inconsistent bid validation logic where regular bids require `msg.value > minPrice` but buy-now purchases allow `msg.value >= buyNowPrice`, creating a scenario where users cannot bid at the minimum price but can purchase at that exact price when `minPrice == buyNowPrice`.

In `src/BidBeastsNFTMarketPlace.sol`:

```
function placeBid(uint256 tokenId) external payable isListed(tokenId) {
    // Buy-now logic allows exact price match
@>  if (listing.buyNowPrice > 0 && msg.value >= listing.buyNowPrice) {
    //Uses >= allowing exact match
        // ... buy-now execution
        return;
    }

    // Regular bidding logic requires exceeding minimum
    if (previousBidAmount == 0) {
        requiredAmount = listing.minPrice;
@>      require(msg.value > requiredAmount, "First bid must be > min
    price"); //Uses > requiring excess
        //existing code
    }
    //existing code
}
```

Risk

Likelihood:

- Sellers commonly set minPrice == buyNowPrice to create fixed-price sales rather than auctions.
- Users attempting to bid at the minimum price will encounter unexpected reverts while buy-now purchases succeed.
- This inconsistency occurs every time a seller lists an NFT with equal minimum and buy-now prices

Impact:

- Users cannot participate in regular bidding when minPrice == buyNowPrice, forcing them into immediate buy-now purchases.
- Breaks the intended auction mechanism by preventing competitive bidding at the minimum price threshold.

Proof of Concept

First we need to make a quick fix in `test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTes` `::setUp()`

```
    function setUp() public {
        // Deploy contracts
-       vm.prank(OWNER);
+       vm.startPrank(OWNER);
        nft = new BidBeasts();
        market = new BidBeastsNFTMarket(address(nft));
        rejector = new RejectEther();

        vm.stopPrank();

        // Fund users
        vm.deal(SELLER, STARTING_BALANCE);
        vm.deal(BIDDER_1, STARTING_BALANCE);
        vm.deal(BIDDER_2, STARTING_BALANCE);
    }
```

Please add the following test to `test/BidBeastsMarketPlaceTest.t.sol`:

```
    function testBugNowPriceEqualsMinPrice() public {
        /* --------------------------- Seller lists NFT ---------------------------- */
        _mintNFT();
        vm.startPrank(SELLER);
        nft.approve(address(market), TOKEN_ID);
        market.listNFT(TOKEN_ID, MIN_PRICE, MIN_PRICE);
        vm.stopPrank();
        /* ---------------------------- BIDDER_1 bids
            ---------------------------- */
        vm.prank(BIDDER_1);
        market.placeBid{value: MIN_PRICE}(TOKEN_ID);
        address NFTowner = nft.ownerOf(TOKEN_ID);
        assertEq(NFTowner, BIDDER_1);
    }

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

Then run `forge test --mt testBugNowPriceEqualsMinPrice` and `forge test --mt testBidderBidsExactlyMinPrice`.

Output:

```
Ran 1 test for test/BidBeastsMarketPlaceTest.t.sol:
    BidBeastsNFTMarketTest
[PASS] testBugNowPriceEqualsMinPrice() (gas: 286081)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 6.31ms
    (139.61us CPU time)

Ran 1 test for test/BidBeastsMarketPlaceTest.t.sol:
    BidBeastsNFTMarketTest
[PASS] testBidderBidsExactlyMinPrice() (gas: 221672)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.23ms
    (686.99us CPU time
```

Recommended Mitigation

Change the regular bidding logic to use >= for consistency with buy-now logic:

```
function placeBid(uint256 tokenId) external payable isListed(tokenId) {
    // ... existing code ...

    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);
    }

    // ... rest of function ...
}
```

[L-3] Unsafe NFT Transfer to Contract Addresses Can Permanently Lock NFTs

Description

- The protocol should safely transfer NFTs to winning bidders while ensuring the recipient can properly handle ERC721 tokens to prevent permanent asset loss.
- The `_executeSale` function uses `transferFrom` instead of `safeTransferFrom` when transferring NFTs to auction winners, which can permanently lock NFTs in contracts that don't implement proper ERC721 receiver functions.

```
function _executeSale(uint256 tokenId) internal {
    Listing storage listing = listings[tokenId];
    Bid memory bid = bids[tokenId];

    listing.listed = false;
    delete bids[tokenId];

@>  BBERC721.transferFrom(address(this), bid.bidder, tokenId); //Uses
    unsafe transfer - NFT can be locked forever

    uint256 fee = (bid.amount * S_FEE_PERCENTAGE) / 100;
    s_totalFee += fee;

    uint256 sellerProceeds = bid.amount - fee;
    _payout(listing.seller, sellerProceeds);

    emit AuctionSettled(tokenId, bid.bidder, listing.seller, bid.amount
       );
}
```

Risk

Likelihood:

- This occurs whenever a contract address wins an auction and that contract doesn't implement `onERC721Received` or implement it incorrectly
- Smart contract wallets, multisig wallets, or other contract-based bidders are common in NFT marketplaces.

Impact:

- NFTs become permanently locked in recipient contracts that cannot handle ERC721 tokens, making them unrecoverable.
- Winners lose access to NFTs they legitimately purchased, creating disputes and potential legal issues.

Proof of Concept

First we need to make a quick fix in `test/BidBeastsMarketPlaceTest.t.sol:BidBeastsNFTMarketTes` `::setUp()`

```
    function setUp() public {
        // Deploy contracts
-       vm.prank(OWNER);
+       vm.startPrank(OWNER);
        nft = new BidBeasts();
        market = new BidBeastsNFTMarket(address(nft));
        rejector = new RejectEther();

        vm.stopPrank();

        // Fund users
        vm.deal(SELLER, STARTING_BALANCE);
        vm.deal(BIDDER_1, STARTING_BALANCE);
        vm.deal(BIDDER_2, STARTING_BALANCE);
    }
```

Please add the following test to `test/BidBeastsMarketPlaceTest.t.sol`:

```
contract RejectEther {
    // Intentionally has no payable receive or fallback
    function bid(BidBeastsNFTMarket market, uint256 tokenId, uint256
        amount) external payable {
        market.placeBid{value: amount}(tokenId);
    }
}

contract BidBeastsNFTMarketTest is Test {

    // existing code
    function testBadReceiver() public {
        _mintNFT();
        _listNFT();
        /* ------------------------------ rejector bids
            ----------------------------- */
        vm.deal(address(rejector), 8 ether);
        rejector.bid(market, TOKEN_ID, BUY_NOW_PRICE);
        assertEq(address(rejector), nft.ownerOf(TOKEN_ID));
    }
    // existing code
}
```

Then run `forge test --mt testBadReceiver`

Output:

```
Ran 1 test for test/BidBeastsMarketPlaceTest.t.sol:
    BidBeastsNFTMarketTest
[PASS] testBadReceiver() (gas: 290096)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 3.75ms
    (160.29us CPU time)
```

Recommended Mitigation

Use safeTransferFrom instead of transferFrom to ensure safe NFT transfers:

```
function _executeSale(uint256 tokenId) internal {
    Listing storage listing = listings[tokenId];
    Bid memory bid = bids[tokenId];

    listing.listed = false;
    delete bids[tokenId];

-   BBERC721.transferFrom(address(this), bid.bidder, tokenId);
+   BBERC721.safeTransferFrom(address(this), bid.bidder, tokenId);

    uint256 fee = (bid.amount * S_FEE_PERCENTAGE) / 100;
    s_totalFee += fee;

    uint256 sellerProceeds = bid.amount - fee;
    _payout(listing.seller, sellerProceeds);

    emit AuctionSettled(tokenId, bid.bidder, listing.seller, bid.amount
       );
}
```

This ensures that if the recipient is a contract, it must implement onERC721Received properly, preventing NFTs from being permanently locked.

Info

[I-1] NFT collection name mismatch - constructor uses "Goddie_NFT" instead of "BidBeasts"

Description

- The contract is named BidBeasts and all events reference "BidBeasts", but the ERC721 constructor uses "Goddie_NFT" as the token name and "GDNFT" as the symbol.

- This inconsistency creates confusion about the actual identity of the NFT collection and may indicate leftover code from a previous project or template.

```
contract BidBeasts is ERC721, Ownable(msg.sender) {
    event BidBeastsMinted(address indexed to, uint256 indexed tokenId);
    event BidBeastsBurn(address indexed from, uint256 indexed tokenId);

    //Name mismatch: "Goddie_NFT" doesn't match contract name "
        BidBeasts"
@>  constructor() ERC721("Goddie_NFT", "GDNFT") {}
}
```

Recommended Mitigation

```
- constructor() ERC721("Goddie_NFT", "GDNFT") {}
+ constructor() ERC721("BidBeasts", "BEAST") {}
```

[I-2] OpenZeppelin version should be specified to ensure Ownable(msg.sender) compatibility

Description

- The contract uses Ownable(msg.sender) syntax which is only available in OpenZeppelin Contracts v5.0.0 and above.
- In OpenZeppelin v4.x, the Ownable constructor doesn't accept parameters and ownership is automatically set to msg.sender, requiring different syntax.

```
// @> This syntax requires OpenZeppelin v5.0+
contract BidBeasts is ERC721, Ownable(msg.sender) {
    // ...
}
```

Recommended Mitigation

```
# foundry.toml
[dependencies]
"@openzeppelin/contracts" = "5.0.0"
```

[I-3] `mint` function doesn't follow Checks-Effects-Interactions pattern

Description

- The `mint` function performs external interactions (`_safeMint`) before updating the state variable `CurrenTokenID`.

- While this doesn't pose a reentrancy risk in the current implementation (since only the owner can call this function and `_safeMint` is from OpenZeppelin), it violates the Checks-Effects-Interactions (CEI) pattern which is a security best practice.

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

Recommended Mitigation

```
  function mint(address to) public onlyOwner returns (uint256) {
      uint256 _tokenId = s_currentTokenId;
+     // Update state before external call (CEI pattern)
+     s_currentTokenId++;
+
      _safeMint(to, _tokenId);
      emit BidBeastsMinted(to, _tokenId);
-     s_currentTokenId++;
      return _tokenId;
  }
```

[I-4] State variable mappings should follow naming convention with s_ prefix

Description

- The contract uses a naming convention where storage variables are prefixed with `s_` (as seen in `s_totalFee`), but the mapping variables do not follow this convention.
- Consistent naming conventions improve code readability and make it easier to identify storage variables at a glance.

```
// @> These storage mappings don't follow the s_ convention
mapping(uint256 => Listing) public listings;
mapping(uint256 => Bid) public bids;
mapping(address => uint256) public failedTransferCredits;

// @> But this storage variable does
uint256 public s_totalFee;
```

Recommended Mitigation

```
- mapping(uint256 => Listing) public listings;
+ mapping(uint256 => Listing) public s_listings;
```

```

- mapping(uint256 => Bid) public bids;
+ mapping(uint256 => Bid) public s_bids;

- mapping(address => uint256) public failedTransferCredits;
+ mapping(address => uint256) public s_failedTransferCredits;
```

[I-5] Events lack `indexed` parameters for efficient off-chain filtering

Description

- None of the events in the contract use `indexed` parameters, which are essential for efficient filtering and searching of events by off-chain applications and indexers.
- Indexed parameters allow events to be efficiently queried by specific values (e.g., finding all bids for a specific tokenId or by a specific bidder).

```
// @> No indexed parameters
event NftListed(uint256 tokenId, address seller, uint256 minPrice,
    uint256 buyNowPrice);
event NftUnlisted(uint256 tokenId);
event BidPlaced(uint256 tokenId, address bidder, uint256 amount);
event AuctionExtended(uint256 tokenId, uint256 newDeadline);
event AuctionSettled(uint256 tokenId, address winner, address seller,
    uint256 price);
event FeeWithdrawn(uint256 amount);
```

Recommended Mitigation

```
- 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 amount);
```

[I-6] Magic numbers should be replaced with named constants

Description

- The contract uses magic numbers (hardcoded numeric literals) directly in calculations without defining them as named constants.
- Magic numbers reduce code readability and make it harder to understand the business logic and maintain the code.

```
// @> Magic number 100 used for percentage calculations
uint256 fee = (bid.amount * S_FEE_PERCENTAGE) / 100;

// @> Magic number 100 used for bid increment calculation
requiredAmount = (previousBidAmount / 100) * (100 +
    S_MIN_BID_INCREMENT_PERCENTAGE);
```

Recommended Mitigation

```
contract BidBeastsNFTMarket is Ownable(msg.sender) {
    uint256 public constant AUCTION_EXTENSION_DURATION = 15 minutes;
    uint256 public constant MIN_NFT_PRICE = 0.01 ether;
    uint256 public constant FEE_PERCENTAGE = 5;
    uint256 public constant MIN_BID_INCREMENT_PERCENTAGE = 5;
+   uint256 public constant PERCENTAGE_BASE = 100;

    function _executeSale(uint256 tokenId) internal {
        // ...
-       uint256 fee = (bid.amount * S_FEE_PERCENTAGE) / 100;
+       uint256 fee = (bid.amount * FEE_PERCENTAGE) / PERCENTAGE_BASE;
        // ...
    }

    function placeBid(uint256 tokenId) external payable isListed(
        tokenId) {
        // ...
-       requiredAmount = (previousBidAmount / 100) * (100 +
    S_MIN_BID_INCREMENT_PERCENTAGE);
+       requiredAmount = (previousBidAmount * (PERCENTAGE_BASE +
    MIN_BID_INCREMENT_PERCENTAGE)) / PERCENTAGE_BASE;
        // ...
    }
}
```

[I-7] Incorrect NatSpec comment for `_payout` function

Description

- The NatSpec comment for the `_payout` function states "A payout function that credits users if a direct transfer fails", but this description is incomplete and potentially misleading.
- The function actually attempts a direct ETH transfer first, and only credits the `failedTransferCredits` mapping if the transfer fails, which is an important fallback mechanism.

```
/**
 * @notice A payout function that credits users if a direct transfer fails.
 */
// @> Incomplete description - doesn't mention it tries direct transfer first
function _payout(address recipient, uint256 amount) internal {
    if (amount == 0) return;
    (bool success,) = payable(recipient).call{value: amount}("");
    if (!success) {
        failedTransferCredits[recipient] += amount;
    }
}
```

Recommended Mitigation

```diff
  /**
-  * @notice A payout function that credits users if a direct transfer fails.
+  * @notice Attempts to send ETH to a recipient. If the transfer fails, credits the amount to failedTransferCredits for later withdrawal.
+  * @param recipient The address to receive the ETH
+  * @param amount The amount of ETH to send
   */
  function _payout(address recipient, uint256 amount) internal {
      if (amount == 0) return;
      (bool success,) = payable(recipient).call{value: amount}("");
      if (!success) {
          failedTransferCredits[recipient] += amount;
      }
  }
```

[I-8] Solidity version 0.8.20 may cause deployment issues on certain chains due to PUSH0 opcode

Description

- The contract specifies Solidity version 0.8.20, which introduced the PUSH0 opcode as part of the Shanghai EVM upgrade.

- Some blockchain networks (including certain L2s and sidechains) may not yet support the PUSH0 opcode, which would cause deployment failures on those chains.

```
// @> Version 0.8.20 includes PUSH0 opcode
pragma solidity 0.8.20;
```

Recommended Mitigation

use the `--evm-version paris` flag when compiling with 0.8.20+ to avoid PUSH0:

```
forge build --evm-version paris
```

```
   /**
-   * @notice A payout function that credits users if a direct transfer
    fails.
+   * @notice Attempts to send ETH to a recipient. If the transfer fails
    , credits the amount to failedTransferCredits for later withdrawal.
+   * @param recipient The address to receive the ETH
+   * @param amount The amount of ETH to send
    */
   function _payout(address recipient, uint256 amount) internal {
       if (amount == 0) return;
       (bool success,) = payable(recipient).call{value: amount}("");
       if (!success) {
           failedTransferCredits[recipient] += amount;
       }
   }
```

Gas

[G-1] State variable `CurrenTokenID` should be declared as private to save gas

Description

- The `CurrenTokenID` state variable is declared as **`public`**, which automatically generates a getter function that costs gas to deploy.
- Since this variable is only used internally within the contract and external users can track token IDs through events or by querying the NFT collection, making it **`private`** would reduce deployment costs without affecting functionality.

```
@>  uint256 public CurrenTokenID;
```

Recommended Mitigation

```
- uint256 public CurrenTokenID;
+ uint256 private s_currentTokenId;
```

```

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

[G-2] State variable BBERC721 should be declared as `immutable` to save gas

Description

- The `BBERC721` state variable is only assigned once in the constructor and never modified afterwards.
- Declaring it as `immutable` instead of a regular state variable would save significant gas costs by avoiding SLOAD operations and storing the value directly in the contract bytecode.

```
contract BidBeastsNFTMarket is Ownable(msg.sender) {
@>   BidBeasts public BBERC721;

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

Recommended Mitigation

```
contract BidBeastsNFTMarket is Ownable(msg.sender) {
-   BidBeasts public BBERC721;
+   BidBeasts public immutable BBERC721;

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

[G-3] Public functions not used internally should be marked as external to save gas

Description

- The functions `getListing`, `getHighestBid`, and `getOwner` are marked as **`public`** but are never called internally within the contract.

- Public functions cost more gas than external functions because public functions must copy all parameters to memory, while external functions can read directly from calldata.

```
// @> These functions are never called internally
function getListing(uint256 tokenId) public view returns (Listing
    memory) {
     return listings[tokenId];
}

function getHighestBid(uint256 tokenId) public view returns (Bid memory
    ) {
     return bids[tokenId];
}

function getOwner() public view returns (address) {
     return owner();
}
```

Recommended Mitigation

```
- function getListing(uint256 tokenId) public view returns (Listing
    memory) {
+ function getListing(uint256 tokenId) external view returns (Listing
    memory) {
      return listings[tokenId];
  }

- function getHighestBid(uint256 tokenId) public view returns (Bid
    memory) {
+ function getHighestBid(uint256 tokenId) external view returns (Bid
    memory) {
      return bids[tokenId];
  }

- function getOwner() public view returns (address) {
+ function getOwner() external view returns (address) {
      return owner();
  }
```