Smart Contract Audit

Date : 2024-07-16

NFT Marketplace

Executive Summary

Type	Smart Contract Audit
Audit Timeline	4 days
Runtime Environment	EVM
Languages	Solidity

Scope

./Marketplace.sol

./NFT_Contract.sol

Summary of Findings

ID	Name	Description	Severity
H-01	Incorrect Max Supply of KenduChad in nft_contract::numTotalkenduChads() function	Correct the Max Supply of KenduChad	High
M-01	Potential Denial of Service (DoS) using marketplace::withdrawBidForChad()	The withdrawBidForChad function allows users to place a bid and then immediately withdraw it, effectively preventing other bids from being processed	Medium
M-02	Centralization Risk: Funds can be frozen when critical key holders lose access to their keys	Lost or stolen keys can pose a serious risk to the protocol	Medium
M-03	Incorrect Cost Calculation for Minting in nft_contract::getCostForMintingkenduChads()	The getCostForMintingkenduChads function calculates the cost of minting as 10,000 Kendu tokens per NFT	Medium
M-04	Inconsistent Platform Fee Percentage	The platformFeePercentage variable is set to 5%, whereas the documentation specifies that the platform fee should be 2.5%	Medium

L–01	Misleading Function Name nft::freeRollMint()	The name freeRollMint could be misleading if users assume it is free. It is better to name it something like mintWithFreeRoll to avoid confusion.	Low
L-02	Redundant Allowance and Balance Checks in ‘mint’	The mint function includes redundant allowance and balance checks before transferring tokens. These checks are unnecessary because the transferFrom function will fail if the allowance is insufficient or the balance is too low.	Low
L-03	Incorrect Free Rolls Check in ‘seedFreeRolls’	The check for the maximum number of free rolls is performed only for the first element of the numOfFreeRolls array. This is incorrect as each element should be validated within the loop to ensure that no address is assigned more than 3 free rolls.	Low
L-04	Inconsistent ERC20 Token Behavior Handling	The contract directly uses token.transfer and token.transferFrom methods, which could lead to issues with tokens that do not follow the ERC20	Low
L-05	marketplace::buyChad() function	The buyChad function requires msg.value to be exactly equal	Low

	requires offer.minValue to be equal to msg.value	to offer.minValue. This restricts users from sending more than the minimum value	
G-01	Using `calldata` instead of `memory` for read-only arguments saves gas	Changing Function visibility can help save some gas on function calls	Gas
G-02	Unnecessary Variable in Bid Struct	The Bid struct contains a boolean variable hasBid that is not being used in the contract logic.	Gas
G-03	Use of constant Keyword for token_contract_addr	The token_contract_addr variable is declared as a public state variable but is assigned a fixed address upon contract deployment. This variable should be declared as constant to optimize gas usage	Gas

Findings

[H-01] Incorrect Max Supply of KenduChad in nft_contract::numTotalkenduChads() function

Severity: High

Location: nft_contract.sol

Description:

The **numTotalkenduChads** function returns a total supply of 10, while the documentation states that 10,000 NFTs will be minted. This discrepancy can lead to incorrect assumptions and behaviors in the contract's operations.

Impact:

- **Minting Limits:** Users might be unable to mint beyond 10 tokens, conflicting with the intended supply.
- **Operational Errors:** Functions relying on the total supply may behave unexpectedly, causing failures in transactions and operations.

Recommendation:
Update the **`numTotalkenduChads`** function to reflect the correct total supply of 10,000 NFTs.

Proof of Concept

```
function numTotalkenduChads() public view virtual returns (uint256) {
    return 10;
}
```

[M-01] Potential Denial of Service (DoS) using marketplace::`withdrawBidForChad()`

Severity: Medium

Location: marketplace.sol

Description:

The `withdrawBidForChad` function allows users to place a bid and then immediately withdraw it, effectively preventing other bids from being processed. This can lead to a denial of service (DoS) attack where the marketplace remains in an eternal state of bid withdrawals, disrupting the bidding process for other users. Consider this scenario :

1) A malicious actor can see a bid and bid a higher amount on the chad.
2) The they can immediately withdraw the bid then clearing out the previous bid
3) This can leave the system in an eternal state of DOS as the valid bids would be cleared out by the bad actor and then that bid would be immediately withdrawn

Impact:

1) **Denial of Service:** Malicious users can repeatedly place and withdraw bids, preventing legitimate bids from being processed and leading to a disruption in the marketplace's functionality.
2) **User Frustration:** Legitimate users may be unable to place bids, leading to frustration and a loss of trust in the marketplace.

Recommendation

Implement a time-based bid withdrawal system to ensure that bids can only be withdrawn after a certain time period. This prevents immediate withdrawals and mitigates the risk of a DoS attack.

```
function withdrawBidForChad(uint chadIndex) public nonReentrant {
    if (chadIndex >= 10000) revert("token index not valid");
    Bid memory bid = chadBids[chadIndex];
    if (bid.bidder != msg.sender)
        revert("the bidder is not message sender");

    // Add a time delay of 1 hour (3600 seconds) before allowing withdrawal
    require(block.timestamp >= bid.bidTimestamp + 3600, "Cannot withdraw
bid within 1 hour of placing it");
```

```
    emit ChadBidWithdrawn(chadIndex, bid.value, msg.sender);
    uint amount = bid.value;
    chadBids[chadIndex] = Bid(false, chadIndex, address(0x0), 0, 0);
    // Refund the bid money
    payable(msg.sender).transfer(amount);
}
```

Proof of Concept

```
    function withdrawBidForChad(uint chadIndex) public nonReentrant {
        if (chadIndex >= 10000) revert("token index not valid");
        Bid memory bid = chadBids[chadIndex];
        if (bid.bidder != msg.sender)
            revert("the bidder is not message sender");
        emit ChadBidWithdrawn(chadIndex, bid.value, msg.sender);
        uint amount = bid.value;
        chadBids[chadIndex] = Bid(false, chadIndex, address(0x0), 0);
        // Refund the bid money
        payable(msg.sender).transfer(amount);
    }
```

[M-02] Centralization Risk: Funds can be frozen when critical key holders lose access to their keys

Severity: Medium

Location: nft_contract.sol , marketplace.sol

Description: The Owner can update key parameters and withdraw funds from the protocol. This introduces a high centralization risk, which can cause funds to be frozen in the contract if the key holders lose access to their keys or are stolen.

Impact

The withdrawal functionality of the contract will be bricked meaning that no withdrawal can be made in case of a protocol failure or a hack.
Recommendation

It is a best practice to use a two-step ownership transfer pattern, meaning ownership transfer gets to a pending state and the new owner should claim his new rights, otherwise the old owner still has control of the contract or use MultiSig wallets for owners.

Proof of Concept

```
contract KenduChad is Ownable, ERC721Enumerable, ReentrancyGuard {
```

[M-03] Incorrect Cost Calculation for Minting in nft_contract::getCostForMintingkenduChads()

Severity: Medium

Location : nft_contract.sol

Description:

The **getCostForMintingkenduChads** function calculates the cost of minting as **10,000** Kendu tokens per NFT. According to the documentation, **5 million** Kendu tokens are required to mint one NFT. This discrepancy results in users being charged incorrect fees for minting.

Impact:

- **Incorrect Minting Fees:** Users are charged significantly less than the intended fee, leading to financial discrepancies and potential loss of funds for the contract's intended operation.
- **Cost Calculation Error:** The function does not accurately reflect the required cost, leading to potential financial mismanagement and user mistrust.

Recommendation

Update the calculation to reflect the correct cost of 5 million Kendu tokens per NFT.

Proof of Concept :

```
  function getCostForMintingkenduChads(
      uint256 _numToMint
  ) public view returns (uint256) {
      require(
          totalSupply() + _numToMint <= numTotalkenduChads(),
          "There aren't this many kenduChads left."
      );
      if (_numToMint >= 1 && _numToMint <= 10) {

          return 10_000 * _numToMint * 10 ** 18; // 10K Kendu Tokens
equivalent in tokens per nft upto 10 nfts, adjust based on token decimals
      } else {
          revert("Unsupported mint amount");
      }
  }
```

[M-04] Inconsistent Platform Fee Percentage

Severity: Medium

Location : marketplace.sol

Description:

The `platformFeePercentage` variable is set to 5%, whereas the documentation specifies that the platform fee should be 2.5%. This discrepancy can lead to users being charged higher fees than expected, resulting in potential trust issues and financial discrepancies.

Impact:

- Inconsistent Fee Charges: Users will be charged a higher fee than documented, leading to potential trust issues and discrepancies between expected and actual fees.
- User Trust Issues: Users may lose trust in the platform if they realize they are being charged more than what is stated in the documentation.
- Financial Discrepancies: The contract will collect more fees than intended, which can lead to financial mismanagement and user dissatisfaction.

Recommendation:

Update the `platformFeePercentage` to match the documented fee of 2.5%. This ensures consistency between the contract's behavior and the provided documentation, maintaining user trust and financial accuracy.

Proof of Concept:

```
    uint8 public platformFeePercentage = 5;
```

[L-01] Misleading Function Name nft::freeRollMint()

Severity: Low

Location: nft_marketplace.sol

Description: The name `freeRollMint` could be misleading if users assume it is free. It is better to name it something like `mintWithFreeRoll` to avoid confusion.

Impact:

- **User Confusion:** Users might misunderstand the function's purpose, assuming it allows free minting without any conditions.

Recommendation:

Rename the function to something more descriptive like `mintWithFreeRoll`.

```
function mintWithFreeRoll() public nonReentrant {
   uint256 toMint = freeRollkenduChads[msg.sender];
   freeRollkenduChads[msg.sender] = 0;
   uint256 remaining = numTotalkenduChads() - totalSupply();
   if (toMint > remaining) {
       toMint = remaining;
   }
   _mint(toMint);
}
```

[L-02] Redundant Allowance and Balance Checks in 'mint'

Severity: Low

Location: nft_marketplace.sol

Description
The `mint` function includes redundant allowance and balance checks before transferring tokens. These checks are unnecessary because the `transferFrom` function will fail if the allowance is insufficient or the balance is too low.

Recommendation
Remove the redundant allowance and balance checks to optimize gas usage. The `transferFrom` function already ensures that the transfer will not succeed if the allowance or balance is insufficient.

POC
```
 function mint(uint256 _numToMint) public nonReentrant {
       require(isSaleOn, "Sale hasn't started.");
       uint256 totalSupply = totalSupply();
       require(
           totalSupply + _numToMint <= numTotalkenduChads(),
           "There aren't this many kenduChads left."
       );
```

```
        uint256 costForMintingkenduChads = getCostForMintingkenduChads(
            _numToMint
        );
        uint256 feeRecipientAmount =
_calculateFee(costForMintingkenduChads);

        // Check allowance
        // @audit LR these checks are not needed as the transfer will fail
if these checks are not met
        uint256 allowance = token_contract_addr.allowance(
            msg.sender,
            address(this)
        );
        require(allowance >= costForMintingkenduChads, "Allowance too
low");

        // Check balance
        uint256 balance = token_contract_addr.balanceOf(msg.sender);
        require(
            balance >= costForMintingkenduChads,
            "Insufficient token balance"
        );

        // Transfer the cost to this contract
        require(
            token_contract_addr.transferFrom(
                msg.sender,
                address(this),
                costForMintingkenduChads
            ),
            "Token transfer failed"
        ); // @audit LR consider using safeTransferFrom

        // Transfer 10% to the fee recipient from contract
        require(
            token_contract_addr.transfer(feeRecipient, feeRecipientAmount),
            "Fee transfer failed"
        );

        // Proceed with minting the kenduChads
```

```
    _mint(_numToMint);
}
```

[L-03] Incorrect Free Rolls Check in ‘seedFreeRolls’

Severity: Low

Location: nft_marketplace.sol

Description
The check for the maximum number of free rolls is performed only for the first element of the `numOfFreeRolls` array. This is incorrect as each element should be validated within the loop to ensure that no address is assigned more than 3 free rolls.

Impact:

- **Incorrect Validation:** Only the first element is validated, potentially allowing other elements to have more than 3 free rolls, which can lead to inconsistencies and unintended behavior.

Recommendation
Move the check inside the loop to validate each element of the `numOfFreeRolls` array

Proof of Concept:

```
function seedFreeRolls(
    address[] memory tokenOwners,
    uint256[] memory numOfFreeRolls
) public onlyOwner {
    require(
        !saleHasBeenStarted,
        "cannot seed free rolls after sale has started"
    );
    require(
        tokenOwners.length == numOfFreeRolls.length,
            "tokenOwners does not match numOfFreeRolls length"
        );

        // light check to make sure the proper values are being passed
        require(numOfFreeRolls[0] <= 3, "cannot give more than 3 free
rolls");

        for (uint256 i = 0; i < tokenOwners.length; i++) {
            freeRollkenduChads[tokenOwners[i]] = numOfFreeRolls[i];
        }
    }
```
.

[L-04] Inconsistent ERC20 Token Behavior Handling

Severity: Low

Location: nft_marketplace.sol

Description
The contract directly uses **token.transfer** and **token.transferFrom** methods, which could lead to issues with tokens that do not follow the ERC20

Recommendation
To ensure compatibility across all types of **ERC20 tokens**, including those that revert on failure, integrate OpenZeppelin's **SafeERC20** library. This library provides methods like **safeTransfer** and **safeTransferFrom**, which handle both returning false and reverting, making token interactions safer and more predictable.

Proof of Concept:

```
  function mint(uint256 _numToMint) public nonReentrant {

        require(isSaleOn, "Sale hasn't started.");

        uint256 totalSupply = totalSupply();

        require(
            totalSupply + _numToMint <= numTotalkenduChads(),

            "There aren't this many kenduChads left."

        );

        uint256 costForMintingkenduChads = getCostForMintingkenduChads(

            _numToMint

        );

        uint256 feeRecipientAmount =
_calculateFee(costForMintingkenduChads);



        // Check allowance

        // @audit LR these checks are not needed as the transfer will fail
if these checks are not met

        uint256 allowance = token_contract_addr.allowance(

            msg.sender,

            address(this)

        );

        require(allowance >= costForMintingkenduChads, "Allowance too
low");



        // Check balance

        uint256 balance = token_contract_addr.balanceOf(msg.sender);

        require(

            balance >= costForMintingkenduChads,

            "Insufficient token balance"
```

```
    );


    // Transfer the cost to this contract

    require(

        token_contract_addr.transferFrom(

            msg.sender,

            address(this),

            costForMintingkenduChads

        ),

        "Token transfer failed"

    ); // @audit LR consider using safeTransferFrom


    // Transfer 10% to the fee recipient from contract

    require(

        token_contract_addr.transfer(feeRecipient, feeRecipientAmount),

        "Fee transfer failed"

    );


    // Proceed with minting the kenduChads

    _mint(_numToMint);

}
```

[L-05] marketplace::buyChad() function requires offer.minValue to be equal to msg.value

Severity: Low

Location: marketplace.sol

Description
The buyChad function requires msg.value to be exactly equal to offer.minValue. This restricts users from sending more than the minimum value, potentially causing transactions to fail unnecessarily if users accidentally send a higher amount.

Impact:

- **Transaction Failure:** Users may accidentally send more ETH, causing the transaction to fail due to the strict equality check.
- **User Frustration:** This strict check can lead to user frustration, especially if they are required to retry transactions with the exact amount.

Recommendation
Modify the check to allow `msg.value` to be greater than or equal to `offer.minValue`. This ensures that users can send more than the minimum required amount, preventing unnecessary transaction failures. Add a refund mechanism if you don't want the users to spend more than the min value .

Proof of Concept:

```
if (msg.value != offer.minValue) revert("not enough ether");
```

[G-01] Using `calldata` instead of `memory` for read-only arguments saves gas

Severity: Gas

**Location : **nft_contract.sol

Description
Use calldata for arguments that are not changed while the function calls.

Recommendation
Change memory to calldata in the function parameter

POC
```
   function seedFreeRolls(
       address[] memory tokenOwners,
       uint256[] memory numOfFreeRolls
   ) public onlyOwner {
```

```

 function seedInitialContractState(
       address[] memory tokenOwners,
       uint256[] memory tokens
   ) public onlyOwner {
```

[G-02] Unnecessary Variable in Bid Struct

Severity: Gas

**Location : **marketplace.sol

Description
The `Bid` struct contains a boolean variable `hasBid` that is not being used in the contract logic. Storing unused variables increases the gas cost for transactions involving this struct, leading to inefficiencies.

Recommendation

Remove the hasBid variable from the Bid struct to optimize gas usage. This change will reduce the gas cost for creating, updating, and deleting instances of the Bid struct.

POC

```
 struct Bid {
        bool hasBid;
        uint chadIndex;
        address bidder;
        uint value;
  }
```

[G-03] Use of constant Keyword for token_contract_addr

Severity: Gas

**Location : **nft_contract.sol

Description

The token_contract_addr variable is declared as a public state variable but is assigned a fixed address upon contract deployment. This variable should be declared as constant to optimize gas usage, as constant variables are stored directly in the bytecode, making access cheaper than regular state variables.

Recommendation

Declare the token_contract_addr variable as constant to optimize gas usage and improve code clarity.

```
IERC20 public constant token_contract_addr =
IERC20(0xc99bb5E1d9C3C4B0D4D2Da86296E73C2097c6Df2);
```

POC

```
IERC20 public token_contract_addr =
IERC20(0xc99bb5E1d9C3C4B0D4D2Da86296E73C2097c6Df2);
```