CD SECURITY

# AUDIT REPORT

Wizard Gallery
March 2024

# Introduction

A time-boxed security review of the **Wizard Gallery** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

# Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

# About **Wizard Gallery**

Wizard Gallery is a NFT marketplace built for the Core ecosystem.

There are two main contracts for this audit:

- `WizardLaunchpad` which is designed to facilitate the launching and initial distribution of NFT collections.
- `WizardNFTMarketplace` which serves for listing NFTs for sale and auction.

Users can buy and sell NFTs using the Core token.

# Severity classification

| Severity | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| **Likelihood: High** | Critical | High | Medium |
| **Likelihood: Medium** | High | Medium | Low |
| **Likelihood: Low** | Medium | Low | Low |

**Impact** - the technical, economic, and reputation damage of a successful attack

**Likelihood** - the chance that a particular vulnerability gets discovered and exploited

**Severity** - the overall criticality of the risk

# Security Assessment Summary

***review commit hash* - 2621f4f742033276acee9297cdb433d95db22f9c**

Scope

The following smart contracts were in scope of the audit:

- `./ERC721.sol`
- `./ERC721ForCollection.sol`
- `./WizardLaunchpad.sol`
- `./Ilaunchpad.sol`
- `./WizardNFTMarketplace.sol`

The following number of issues were found, categorized by their severity:

- Critical & High: 2 issues
- Medium: 1 issues
- Low: 4 issues
- Informational: 6

---

# Findings Summary

| ID | Title | Severity |
|---|---|---|
| [C-01] | Users can get NFTs for as little as 1 wei | Critical |
| [C-02] | Anyone can change the base URI | Critical |
| [M-01] | Missing input validation | Medium |
| [L-01] | Use two-step ownership transfer approach | Low |
| [L-02] | Use `call()` instead of `transfer()` when sending ETH | Low |
| [L-03] | Use `safeTransferFrom` instead of `transferFrom` for ERC721 | Low |
| [L-04] | Discrepancy between comments and code | Low |
| [I-01] | Missing event emissions in state changing methods | Informational |
| [I-02] | Solidity safe pragma best practices are not used | Informational |
| [I-03] | Prefer Solidity Custom Errors over `require` statements with strings | Informational |
| [I-04] | No need to check if `uint` is less than 0 | Informational |
| [I-05] | Redundant and unused code | Informational |
| [I-06] | Typos | Informational |

# Detailed Findings

# [C-01] Users can get NFTs for as little as 1 wei

## Severity

**Impact:** High

**Likelihood:** High

## Description

Inside the `WizardNFTMArkeplace` token owners can create auctions and users can bid for a specific NFT through the `placeBid` function.

```
    function placeBid(
        address collectionAddr,
        uint256 tokenId
    ) external payable nonReentrant {
        MarketItem storage item = idToMarketItem[collectionAddr][tokenId];
        ...
        if (msg.value < item.highestBid) {
            revert BidShouldGreater();
        }

        if (item.highestBidder != address(0)) {
            payable(item.highestBidder).transfer(item.highestBid);
        }

        item.highestBidder = payable(msg.sender);
        item.highestBid = msg.value;

        emit BidPlaced(tokenId, msg.sender, msg.value);
    }
```

It can be seen that if a user sends more native tokens, the previous `highestBid` amount will be refunded to the `highestBidder` before updating the state variables with the new `highestBid` and new `highestBidder`. If the owner of the NFT decides to end an ongoing auction without accepting any bids through `endAuction`, the `highestBid` will again be refunded to the `highestBidder`.

```
    function endAuction(
        address collectionAddr,
        uint256 tokenId
    ) external onlyTokenOwner(collectionAddr, tokenId) {
        MarketItem storage item = idToMarketItem[collectionAddr][tokenId];
        if (!item.isOnAuction) {
            revert NotOnAuction();
        }

        payable(item.highestBidder).transfer(item.highestBid);
        ...
    }
```

However, this push approach can lead to a scenario where a malicious user can buy an NFT for as little as 1 wei or brick the auction. If the bidder is a contract that doesn't have any `receive` or `fallback` methods, the `transfer` call will always fail. Thus, the only possible way to unbrick the auction will be to call `acceptBid` and send the NFT to the malicious contract.

### Recommendations

Use pull instead of push approach by implementing separate `withdraw` function that allows outbid users to get their funds back.

### Client

Fixed

## [C-02] Anyone can change the base URI

### Severity

**Impact:** High

**Likelihood:** High

### Description

When a collection is deployed through `WizardLaunchpad::deployCollection`, an `ipfsHash` parameter is passed by the user to set the `_baseuri`. However, the `setbaseUri` function in `ERC721ForCollection` is a public function and can be called by anyone. This means that anyone can change the base URI.

```
    function setbaseUri(string memory uri)public {
        _baseuri = uri;
    }
```

### Recommendations

Add an appropriate access control modifier.

### Client

Fixed

## [M-01] Missing input validation

### Severity

**Impact:** High

**Likelihood:** Low

## Description

Throughout the codebase there are couple of places where important input validation is missing.

- `setPlatformFeePercent()`
- `setMaxWalletLimit()`

Not constraining the parameters of these functions can create problems due to an error or malicious activities from a compromised owner when calling them.

Also in `listNFTForAuction()` we can observe that a check for `endAt` param is missing. This can cause the auction to end even 1 second after the current block.timestamp or in a time far in the future.

## Recommendations

You can create a check where `exampleParam` should be less than x and greater than y, otherwise revert the transaction.

## Client

Partially fixed

# [L-01] Use two-step ownership transfer approach

The `owner` role is crucial for the protocol as there are a lot of functions with the `onlyOwner` modifier. Make sure to use a two-step ownership transfer approach by using `Ownable2Step` from OpenZeppelin as opposed to `Ownable` as it gives you the security of not unintentionally sending the `owner` role to an address you do not control.

Also the owner in `ERC721.sol` cannot be changed if wanted. It will be best if you import `Ownable2step` there as well, put the `onlyOwner` modifier in `mintToken()`, `setBaseUri()` and remove the require check from these functions.

## Client

Fixed

# [L-02] Use `call()` instead of `transfer()` when sending ETH

Couple of functions in the contract are using the `transfer` method to send ETH. These addresses are possible to be a smart contract that have a `receive` or `fallback` function that takes up more than the 2300 gas which is the limit of `transfer`. Examples are some smart contract wallets or multi-sig wallets, so usage of `transfer` is discouraged.

Use .call with value like in `withdraw()` but put the `nonReentrant` modifier in these functions as well.

### Client

Fixed

## [L-03] Use `safeTransferFrom` instead of `transferFrom` for ERC721

Use of transferFrom method for ERC721 transfer is discouraged and recommended to use safeTransferFrom whenever possible by OpenZeppelin. This is because transferFrom() cannot check whether the receiving address know how to handle ERC721 tokens.

If this msg.sender is a contract and is not aware of incoming ERC721 tokens, the sent token could be locked up in the contract forever.

Reference: https://docs.openzeppelin.com/contracts/3.x/api/token/erc721

### Client

Fixed

## [L-04] Discrepancy between comments and code

The `createProject` function in `WizardLaunchpad` is an external function and is not protected by any modifiers.

```
     * @notice Only the contract owner can call this function.  //@audit
     */

    function createProject(
        string memory name,
        string memory symbol,
        string memory ipfsHash,
        Stage[] calldata stagesData,
        uint256 maxCap,
        uint256 _maxWallet,
        bool isLimitedEdition
    ) external {
```

However, the NatSpec says that it should be callable only by the owner. Either change the comment or add an access control to the function.

### Client

Fixed

# [I-01] Missing event emissions in state changing methods

It's a best practice to emit events on every state changing method for off-chain monitoring.The following methods are missing event emissions, which should be added:

- ERC721:setBaseUri()
- WizzardLaunchpad:setTreasuryWallet()
- WizzardNFTMarketplace:setTreasuryWallet()
- WizzardNFTMarketplace:setCollectionRoyalty()

## Client

Fixed

# [I-02] Solidity safe pragma best practices are not used

Always use a stable pragma to be certain that you deterministically compile the Solidity code to the same bytecode every time. The `WizardLaunchpad.sol` contract is currently using a floatable version.

## Client

Fixed

# [I-03] Prefer Solidity Custom Errors over `require` statements with strings

Using Solidity Custom Errors has the benefits of less gas spent in reverted transactions, better interoperability of the protocol as clients of it can catch the errors easily on-chain, as well as you can give descriptive names of the errors without having a bigger bytecode or transaction gas spending, which will result in a better UX as well. Consider replacing the `require` statements with custom errors.

You are using custom errors in most places but in couple of places you are using require checks. Change these to custom errors as well.

## Client

Fixed

# [I-04] No need to check if `uint` is less than 0

There are couple of places that are checking if certain variables are less than zero but these variables are `uint`s so this is not possible and no need to check. You can remove the `<` sign from these places:

- `if (msg.value <= 0)`
- `if (offerAmount <= 0)`
- `if (offeredAmount <= 0)`

### Client

Fixed

## [I-05] Redundant and unused code

These getter functions in are redundant since the return variables that they are returning are public storage variables that have automatically generated getters:

- `getProjectDetails()`
- `getAllCollections()`

Also these two are not used anywhere:

- `mapping collectionToRoyaltySetter;`
- `error NoOfferFoundToCancel();`

### Client

Fixed

## [I-06] Typos

- `error AunctionEnds();` - Auction
- `error ShouldLessThanMaxRyalty();` - Royalty

### Client

Fixed