Stonewall
Smart Contract Security

MonadGridMarketplace Security Audit Report

Auditor: Stonewall Security **Date:** January 2026 **Version:** 1.0 **Contract:** MonadGridMarketplace.sol **Solidity Version:** ^0.8.28 **Chain:** Monad

Executive Summary

This audit reviews the MonadGridMarketplace smart contract, an NFT marketplace supporting listings, offers, bulk operations, and royalty distribution. The contract demonstrates solid security practices including ReentrancyGuard, proper access control, and safe math (Solidity 0.8+).

Findings Summary

Severity	Count
Critical	0
High	0
Medium	2
Low	4
Informational	2

Fixed Issues (from previous versions)

Severity	Issue	Status
High	Reentrancy in buy() function	**Fixed**
Medium	Missing zero-address check for feeRecipient	**Fixed**

Low	Uncapped royalty from ERC2981	**Fixed**

Scope

File	SLOC
MonadGridMarketplace.sol	~450

Features Reviewed

- Listing creation, update, cancellation (single and bulk)
- Direct purchases and sweep functionality
- Offer placement, cancellation, and acceptance
- Royalty distribution (custom + ERC2981)
- Fee management
- Access control

Findings

[M-01] Stale Listing Vulnerability - NFT Can Be Sold After Transfer and Return

Severity: Medium **Status:** Open

Description: When a seller lists an NFT and later transfers it away, the listing persists in storage. If the NFT is transferred back to the original seller (or if they re-acquire it), the listing becomes valid again without the seller's renewed consent.

Location: `buy()` function (lines 238-267), `sweep()` function (lines 269-338)

```
function buy(address collection, uint256 tokenId) external payable nonReentrant {
    Listing memory listing = _listings[collection][tokenId];
    if (listing.seller == address(0)) revert ListingNotFound();
    if (msg.value != listing.price) revert InvalidPrice();

    delete _listings[collection][tokenId];

    IERC721 token = IERC721(collection);
```

```
    if (token.ownerOf(tokenId) != listing.seller) revert Unauthorized();
    // Listing was created long ago at different price conditions
    // ...
}
```

Impact:

- Seller may have their NFT sold at an outdated price
- Market conditions may have changed significantly
- Particularly problematic for volatile NFT markets

Recommendation:

1. Add listing timestamp and optional expiration:

```
struct Listing {
    address seller;
    uint128 price;
    uint64 createdAt;
    uint64 expiration; // 0 = no expiration
}
```

2. Invalidate listings on NFT transfer by implementing `onERC721Received` hook
3. Or require explicit re-listing after any transfer

[M-02] withdrawFees Can Extract Offer Funds

Severity: Medium **Status:** Open

Description: The `withdrawFees()` function allows the owner to withdraw any amount from the contract balance, including funds that belong to pending offers.

Location: `withdrawFees()` function (lines 498-501)

```
function withdrawFees(address payable recipient, uint256 amount) external onlyOwner {
    if (recipient == address(0)) revert ZeroAddress();
    recipient.sendValue(amount);
}
```

Impact:

- Owner can accidentally or maliciously withdraw user offer funds
- Users with pending offers could lose their ETH/MONAD
- Violates user fund segregation

Recommendation: Track accumulated fees separately:

```
uint256 public accumulatedFees;

function _distributeProceeds(...) private returns (...) {
    // ...
    if (feePaid > 0) {
        remaining -= feePaid;
        accumulatedFees += feePaid;  // Track instead of sending
    }
    // ...
}

function withdrawFees(address payable recipient, uint256 amount) external onlyOwner {
    if (recipient == address(0)) revert ZeroAddress();
    if (amount > accumulatedFees) revert InsufficientFees();
    accumulatedFees -= amount;
    recipient.sendValue(amount);
}
```

[L-01] Listings Have No Expiration Mechanism

Severity: Low **Status:** Open

Description: Listings persist indefinitely until explicitly cancelled or sold. A seller may forget about an old listing at a now-unfavorable price.

Location: `Listing` struct (lines 17-20)

Impact:

- Stale listings at outdated prices
- Poor user experience for both buyers (invalid listings) and sellers (unexpected sales)

Recommendation: Add optional expiration to listings or implement a maximum listing duration.

[L-02] Weak Collection Validation

Severity: Low **Status:** Open

Description: The `_validateCollection()` function only checks for zero address, not whether the address is actually an ERC721 contract.

Location: `_validateCollection()` function (lines 575-577)

```
function _validateCollection(address collection) private pure {
    if (collection == address(0)) revert InvalidCollection();
}
```

Impact:

- Users could create listings for non-NFT addresses
- Could cause confusion and wasted gas

Recommendation: Add ERC165 interface check:

```
function _validateCollection(address collection) private view {
    if (collection == address(0)) revert InvalidCollection();
    try IERC165(collection).supportsInterface(type(IERC721).interfaceId) returns (bool
        if (!supported) revert InvalidCollection();
    } catch {
        revert InvalidCollection();
    }
}
```

[L-03] Front-Running Risk on Offer Acceptance

Severity: Low **Status:** Acknowledged (Mitigated)

Description: When a seller calls `acceptOffer()`, an attacker could front-run with a lower offer that still meets `minPrice`, or the current bidder could cancel their offer.

Location: `acceptOffer()` function (lines 398-434)

Mitigation Present: The function includes `expectedBidder` and `minPrice` parameters which significantly reduce this risk:

```
if (expectedBidder != address(0) && offer.bidder != expectedBidder) revert Unauthorized
```

```
if (offer.price < minPrice) revert InvalidPrice();
```

Residual Risk:

- Bidder can still cancel offer before acceptance TX confirms
- Consider adding time delay on offer cancellation

[L-04] ERC2981 Royalty Could Return Excessive Values

Severity: Low **Status:** Mitigated

Description: External calls to `IERC2981.royaltyInfo()` could return malicious royalty amounts, but the contract properly caps this.

Location: `_distributeProceeds()` function (lines 539-552)

Mitigation Present:

```
uint128 maxRoyalty = uint128((uint256(amount) * MAX_ROYALTY_BPS) / MAX_BPS);
if (royaltyPaid > maxRoyalty) {
    royaltyPaid = maxRoyalty;
}
```

This caps royalties at 10% (MAX_ROYALTY_BPS = 1000).

[I-01] Gas Optimization: Cache Storage Reads

Severity: Informational

Description: Multiple storage reads of the same mapping could be optimized by caching in memory.

Example in `bulkCancelListing()`:

```
// Current - reads storage twice
if (listing.seller != address(0) && listing.seller == msg.sender) {

// Optimized - cache seller
```

```
address seller = listing.seller;
if (seller != address(0) && seller == msg.sender) {
```

[I-02] Consider Adding Emergency Pause

Severity: Informational

Description: The contract lacks a pause mechanism for emergency situations (exploits, market manipulation).

Recommendation: Inherit OpenZeppelin's `Pausable` and add `whenNotPaused` modifier to critical functions.

Fixed Findings

[H-01] Reentrancy in buy() Function - **FIXED**

Severity: High **Status:** Fixed

Previous Vulnerable Code:

```
// OLD - No reentrancy protection
function buy(address collection, uint256 tokenId) external payable {
    // ... transfer logic without nonReentrant
}
```

Current Fixed Code:

```
// FIXED - Added ReentrancyGuard
function buy(address collection, uint256 tokenId) external payable nonReentrant {
    // ... now protected
}
```

Verification: All value-transferring functions now use `nonReentrant` modifier.

[M-03] Missing Zero-Address Check for feeRecipient - **FIXED**

Severity: Medium **Status:** Fixed

Previous Issue: `setFeeRecipient()` could be called with zero address, causing fees to be lost.

Current Fix:

```
function setFeeRecipient(address newRecipient) external onlyOwner {
    if (newRecipient == address(0)) revert InvalidFeeRecipient();
    // ...
}
```

[L-05] Uncapped Royalty from ERC2981 - **FIXED**

Severity: Low **Status:** Fixed

Previous Issue: Malicious ERC2981 implementations could return 100% royalty, draining seller proceeds.

Current Fix: Royalties are now capped at MAX_ROYALTY_BPS (10%):

```
uint128 maxRoyalty = uint128((uint256(amount) * MAX_ROYALTY_BPS) / MAX_BPS);
if (royaltyPaid > maxRoyalty) {
    royaltyPaid = maxRoyalty;
}
```

Security Features (Positive Findings)

1. **ReentrancyGuard**: All value-transferring functions use `nonReentrant`
2. **Solidity 0.8+**: Automatic overflow/underflow protection
3. **Access Control**: Proper `onlyOwner` on admin functions
4. **Safe Value Transfers**: Uses `Address.sendValue()` for ETH transfers
5. **Royalty Cap**: ERC2981 royalties capped at 10%
6. **Approval Checks**: Validates marketplace approval before transfers
7. **Bulk Operation Limits**: MAX_BULK_OPERATIONS = 50 prevents gas griefing
8. **Receive/Fallback Protection**: Rejects direct ETH transfers

---

Architecture Overview

```
MonadGridMarketplace
├── Listings
|   ├── list() - Single listing
|   ├── bulkList() - Multiple listings
|   ├── updateListing() - Price update
|   ├── cancelListing() - Single cancel
|   ├── bulkCancelListing() - Multiple cancel
|   ├── buy() - Direct purchase
|   └── sweep() - Bulk purchase
├── Offers
|   ├── placeOffer() - Create/update offer
|   ├── cancelOffer() - Withdraw offer
|   └── acceptOffer() - Accept as seller
├── Royalties
|   ├── updateRoyalty() - Set custom royalty
|   └── ERC2981 fallback
└── Admin
    ├── setMarketplaceFee()
    ├── setFeeRecipient()
    ├── setCollectionOwner()
    └── withdrawFees()
```

Conclusion

The MonadGridMarketplace contract demonstrates good security practices with ReentrancyGuard, access control, and safe math. The main concerns are:

1. **Stale listing vulnerability** - NFTs can be sold at old prices after re-acquisition
2. **Fee withdrawal overlap** - Owner can withdraw user offer funds

Neither issue is critical, but both should be addressed before mainnet deployment. The contract is suitable for testnet use and beta launch with proper user warnings.

Overall Assessment: Low-Medium Risk

Disclaimer

This audit does not guarantee the absence of vulnerabilities. It represents a time-limited review based on the provided source code. Smart contracts should undergo multiple independent audits

before deployment with significant value.

Stonewall Security *Building Stronger Smart Contracts*