Security Recommendations: Land NFT Contracts

High Priority Fixes

1. Re-entrancy Protection

Issue: State updates after external calls in mint() function **Implementation**:

```
function mint(uint256 quantity) public {
    // Validate inputs first
    if(_currentBatch.quantity <= 0) revert NoMoreTokensLeft();
    if(!_currentBatch.active) revert CurrentBatchNotActive();
    if(quantity <= 0) revert QuantityMustBeAboveZero();
    if(quantity >= _maxBuyAmount && !hasRole(SIGNER_ROLE, _msgSender())) {
        revert MaxBuyAmountLimitReached();
    }

    // Update state before external calls
    uint256 newQuantity = _currentBatch.quantity - quantity;
    _currentBatch.quantity = newQuantity;

    // External calls last
    if (!freeParticipant[msg.sender]) {
        if(!_pay(msg.sender, quantity)){
            revert MustPayBeforeMinting();
        }
    }

    _safeMint(msg.sender, quantity);
}
```

2. Payment Validation Enhancement

Issue: Inadequate payment validation in _pay() function **Implementation**:

```
function _pay(address payee, uint256 quantity) internal returns (bool) {
    IERC20Upgradeable token = IERC20Upgradeable(_paymentToken);

    uint256 paymentAmount = _currentBatch.price * quantity;

    // Check allowance
    require(
        token.allowance(payee, address(this)) >= paymentAmount,
        "Insufficient allowance"
    );

    bool success = token.transferFrom(
        payee,
```

```
        _feeCollector,
        paymentAmount
    );

    require(success, "Payment transfer failed");

    emit PaymentProcessed(payee, paymentAmount, success);
    return success;
}
```

Medium Priority Fixes

1. Authorization Control Fix

Issue: Incorrect permission management for free participant controllers **Implementation**:

```
error UnauthorizedAccess();
error InvalidParticipant();

function setFreeParticipant(address participant, bool free) external {
    if (!freeParticipantControllers[msg.sender] && msg.sender != owner()) {
        revert UnauthorizedAccess();
    }
    if (participant == address(0)) {
        revert InvalidParticipant();
    }

    freeParticipant[participant] = free;

    emit FreeParticipantUpdated(participant, free, msg.sender);
}

event FreeParticipantUpdated(
    address indexed participant,
    bool status,
    address indexed updatedBy
);
```

2. Batch State Management Optimization

Issue: Inefficient storage access patterns **Implementation**:

```
struct BatchOperation {
    uint256 quantity;
    uint256 price;
    bool active;
}

function processBatch(uint256 quantity) internal returns (bool) {
```

```
    // Load struct into memory
    BatchOperation memory batch = BatchOperation({
        quantity: _currentBatch.quantity,
        price: _currentBatch.price,
        active: _currentBatch.active
    });

    // Perform operations in memory
    if (batch.quantity < quantity) revert InsufficientQuantity();
    batch.quantity -= quantity;

    // Write back to storage
    _currentBatch.quantity = batch.quantity;

    emit BatchProcessed(quantity, batch.quantity);
    return true;
}
```

Low Priority Improvements

1. Input Validation

Implementation:

```
function setPaymentToken(address newToken) external onlyOwner {
    if (newToken == address(0)) revert InvalidAddress();
    address oldToken = _paymentToken;
    _paymentToken = newToken;
    emit PaymentTokenUpdated(oldToken, newToken);
}

function setFeeCollector(address newCollector) external onlyOwner {
    if (newCollector == address(0)) revert InvalidAddress();
    address oldCollector = _feeCollector;
    _feeCollector = newCollector;
    emit FeeCollectorUpdated(oldCollector, newCollector);
}
```

2. Standardized Error Handling

Implementation:

```
// Custom errors
error NoMoreTokensLeft();
error CurrentBatchNotActive();
error QuantityMustBeAboveZero();
error MaxBuyAmountLimitReached();
error InsufficientAllowance();
error PaymentFailed();
```

```
error InvalidAddress();
error BatchProcessingFailed();

// Remove string error messages and use custom errors
function validateMint(uint256 quantity) internal view {
    if (_currentBatch.quantity <= 0) revert NoMoreTokensLeft();
    if (!_currentBatch.active) revert CurrentBatchNotActive();
    if (quantity <= 0) revert QuantityMustBeAboveZero();
    if (quantity >= _maxBuyAmount && !hasRole(SIGNER_ROLE, _msgSender())) {
        revert MaxBuyAmountLimitReached();
    }
}
```

3. Enhanced Event Logging

Implementation:

```
event BatchProcessed(
    uint256 quantityProcessed,
    uint256 remainingQuantity
);

event PaymentProcessed(
    address indexed payer,
    uint256 amount,
    bool success
);

event PaymentTokenUpdated(
    address indexed oldToken,
    address indexed newToken
);

event FeeCollectorUpdated(
    address indexed oldCollector,
    address indexed newCollector
);
```

4. Code Cleanup

```
// Remove unused tax functionality
- function _tax(address payee) internal virtual returns (bool) {
-     IERC20 token = IERC20(_paymentToken);
-     token.transferFrom(payee, _feeCollector, _txFeeAmount);
-     return true;
- }
```

Documentation Improvements

1. NatSpec Documentation

```
/// @title Land NFT Contract
/// @notice Manages the minting and distribution of land NFTs
/// @dev Implements ERC721 standard with custom minting logic
contract LandNFT is ERC721Upgradeable {
    /// @notice Mints new tokens
    /// @param quantity Number of tokens to mint
    /// @dev Validates batch availability and processes payment if required
    /// @return Boolean indicating success
    function mint(uint256 quantity) external returns (bool) {
        // Implementation
    }

    /// @notice Processes payment for minting
    /// @param payee Address making the payment
    /// @param quantity Number of tokens being minted
    /// @return Boolean indicating payment success
    function _pay(address payee, uint256 quantity) internal returns (bool) {
        // Implementation
    }
}
```