Gas Bad NFT Marketplace Audit Report

SnavOhBurmaa

June 6, 2026

Gas Bad NFT Marketplace Audit Report

Prepared by: SnavOhBurmaa Lead Auditors:

- SnavOhBurmaa

Assisting Auditors:

- None

Table of contents

See table

About Me

Hi, I'm Khant Wai Yan Aung (SnavOhBurmaa). I'm currently focusing on smart contract security audit. This is one report in my ongoing audit series. Thanks for reading.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

Audit Details

Source repository: https://github.com/Cyfrin/12-gas-bad-nft-marketplace-audit

The findings described in this document correspond to the following commit hash:

```
425b2b32199366ab8b30d2192ef4d2a80c40aced
```

Scope

```
src/
--- GasBadNftMarketplace.sol   (optimized, deployed to mainnet)
--- NftMarketplace.sol         (reference implementation)
--- INftMarketplace.sol        (shared interface)
```

- **Solc version:** 0.8.20
- **Chain:** Ethereum mainnet

- **Tokens:** None (the marketplace escrows arbitrary ERC-721s; no ERC-20 is handled)

Methodology

The audit combined four techniques:

1. **Manual review** of both contracts, line-by-line, with special attention to the inline-assembly event emitters in `GasBadNftMarketplace.sol`.
2. **Differential analysis** between the reference and optimized contracts (the whole point of the project is that they must behave identically).
3. **Formal verification** with Certora (equivalence rule + an "every storage write emits an event" invariant).
4. **Static analysis** with Slither and Aderyn, plus an independent recomputation of every hard coded `keccak256` event topic with `cast keccak`.

Protocol Summary

The Gas Bad NFT Marketplace is an escrow-based ERC-721 marketplace. A seller lists an NFT (which is transferred into the marketplace), a buyer pays ETH to receive it, and the seller later pulls their proceeds. The project ships **two** implementations:

- `NftMarketplace.sol` — a plain-Solidity *reference* implementation.
- `GasBadNftMarketplace.sol` — a gas-optimized version that emits all four events using raw `log4` opcodes in inline assembly instead of Solidity `emit`. It is the contract intended for mainnet deployment.

The two are meant to be *behaviourally identical*; Certora's equivalence rule proves this at the storage level.

Function	Purpose
`listItem`	Escrow an NFT and create a listing (`price > 0`).
`cancelListing`	Seller withdraws their unsold NFT.
`updateListing`	Seller changes the price.
`buyItem`	Buyer pays ETH, receives the NFT, seller is credited.
`withdrawProceeds`	Seller pulls accumulated ETH.

Roles

- **Seller:** Lists, updates, and cancels NFTs, and withdraws proceeds.
- **Buyer:** Pays ETH to receive a listed NFT.
- There is **no owner/admin role** and **no upgradeability** — the contract is immutable once deployed

Findings

Summary of Findings

ID	Title	Severity
M-1	NFTs sent directly to the marketplace are permanently locked	Medium
M-2	`buyItem` credits `msg.value` instead of `price` (overpayment captured)	Medium
L-1	`withdrawProceeds` emits no event	Low
L-2	No validation that `nftAddress` is a real ERC-721	Low
G-1	`<= 0` comparisons on `uint256`	Gas/QA
G-2	`buyItem` re-reads the listing struct	Gas/QA
I-1	Acknowledged front-running / MEV issues	Info
I-2	PUSH0 opcode restricts deployment chains	Info
I-3	Aderyn "state change without event" false positive	Info

Medium

[M-1] NFTs sent directly to the marketplace via `safeTransferFrom` are permanently locked

Location:

- `src/GasBadNftMarketplace.sol:191-199` — `onERC721Received` (unconditional accept, return at line 198)
- `src/GasBadNftMarketplace.sol:73-77` — `cancelListing` seller check (the only non-sale exit), reverts at line 75
- `src/NftMarketplace.sol:141-147` — same `onERC721Received` in the reference contract

Description: The marketplace implements `onERC721Received` and returns the ERC-721 "magic value" *unconditionally*:

```
function onERC721Received(address, address, uint256, bytes
calldata) external pure returns (bytes4) {
    return bytes4(0x150b7a02);
}
```

This means the contract will accept any ERC-721 pushed to it with `safeTransferFrom`, not just NFTs that arrive through `listItem`. When an NFT arrives outside of `listItem`:

No `s_listings` entry is created (the listing's `seller` stays `address(0)`). There is **no admin, no rescue, and no sweep function** anywhere in the contract.

`cancelListing` is the only way an NFT can leave the contract without a sale, and it requires `msg.sender == s_listings[nftAddress][tokenId].seller`. For a directly-deposited token that seller is `address(0)`, so the call reverts with `NftMarketplace__NotOwner`. The NFT is stuck forever.

The contract is immutable and has no owner, so recovery is **impossible** after the fact.

Risk: MEDIUM

Impact: HIGH — permanent, irreversible loss of an NFT. The asset is not stolen, but it can never be retrieved by anyone. **Likelihood: LOW** — requires a user to call `safeTransferFrom` directly (a common mistake when people see the marketplace already holds NFTs), or an NFT contract / other protocol that auto-pushes tokens. It does not happen on the normal `listItem` path.

Proof of Concept:

```
function test_directNftTransferIsPermanentlyLocked() public {
    // Stranger mints an NFT, sends it directly to the
marketplace with safeTransferFrom.
    vm.startPrank(stranger);
    nft.mint();
    uint256 tokenId = nft.totalSupply() - 1;
    console2.log("stranger mints tokenId          :", tokenId);
    console2.log("owner before direct transfer    :",
nft.ownerOf(tokenId));
    nft.safeTransferFrom(stranger, address(market), tokenId);
    vm.stopPrank();

    // The marketplace accepted it (onERC721Received returns the
magic value)
    console2.log("owner after direct transfer     :",
nft.ownerOf(tokenId));
    console2.log("marketplace address             :",
address(market));
    assertEq(nft.ownerOf(tokenId), address(market), "marketplace
should hold the NFT");

    // but NO listing was ever created.
    INftMarketplace.Listing memory listing =
market.getListing(address(nft), tokenId);
    console2.log("listing.seller (0 = no listing) :",
listing.seller);
    console2.log("listing.price                   :",
listing.price);
    assertEq(listing.seller, address(0), "no listing exists for
the deposited NFT");
    assertEq(listing.price, 0);

    // The original owner cannot cancel: the listing's seller is
address(0), not them
    vm.prank(stranger);
```

```
vm.expectRevert(GasBadNftMarketplace.NftMarketplace__NotOwner.selector);
    market.cancelListing(address(nft), tokenId);
    console2.log("cancelListing by stranger       : REVERTED
(NotOwner)");

    // There is no admin / rescue function anywhere in the
contract, so the NFT
    // is now permanently locked. The marketplace still owns it
    assertEq(nft.ownerOf(tokenId), address(market), "NFT is
permanently stuck");
    console2.log("RESULT                           : NFT is
permanently locked in the marketplace");
}
```

Run `forge test --match-test test_directNftTransferIsPermanentlyLocked -vv`:

```
[PASS] test_directNftTransferIsPermanentlyLocked()
Logs:
  stranger mints tokenId          : 0
  owner before direct transfer    :
0x49052147F5D97A723DEBdf07680fFFaDAd29A5dC
  owner after direct transfer     :
0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f
  marketplace address             :
0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f
  listing.seller (0 = no listing) :
0x0000000000000000000000000000000000000000
  listing.price                   : 0
  cancelListing by stranger       : REVERTED (NotOwner)
  RESULT                          : NFT is permanently locked in
the marketplace
```

The owner flips from the stranger to the marketplace, the listing is empty (`seller = 0x0`), and the only non-sale exit (`cancelListing`) reverts, confirming the NFT can never leave.

Recommended Mitigation: The cleanest fix is to only accept NFTs that the contract is actively pulling in during `listItem`. A common pattern is a transient flag:

```
+ bool private s_inListItem;

  function listItem(address nftAddress, uint256 tokenId, uint256
        price) external {
      if (price <= 0) revert
        NftMarketplace__PriceMustBeAboveZero();
      s_listings[nftAddress][tokenId] = Listing(price,
        msg.sender);
      // ...emit...
+     s_inListItem = true;
```

```
        IERC721(nftAddress).safeTransferFrom(msg.sender,
          address(this), tokenId);
+       s_inListItem = false;
    }

    function onERC721Received(address, address, uint256, bytes
          calldata) external view returns (bytes4) {
+       if (!s_inListItem) revert
          NftMarketplace__DirectTransferNotAllowed();
        return bytes4(0x150b7a02);
    }
```

If gas is the priority, an alternative is to keep accepting deposits but add an owner-gated `rescueNft` function. Either way, "accept and forget" must be removed.

[M-2] `buyItem` credits `msg.value` instead of `price`, so buyer overpayment is silently captured by the seller

Location:

- `src/GasBadNftMarketplace.sol:118` — `s_proceeds[seller] += msg.value;` (inside `buyItem`, lines 104-136)
- `src/NftMarketplace.sol:90` — `s_proceeds[listedItem.seller] += msg.value;` (inside `buyItem`, lines 79-96)

Description: `buyItem` checks `msg.value >= price` but then credits the **entire** `msg.value` to the seller and never refunds the difference:

```
if (msg.value < price) {
    revert NftMarketplace__PriceNotMet(nftAddress, tokenId,
price);
}
s_proceeds[seller] += msg.value;   // <-- should be += price,
with a refund of the rest
```

Any buyer who sends more than the listing price loses the excess: it is added to the seller's withdrawable proceeds rather than returned. Both the reference and the optimized contract behave this way.

This is more than a theoretical foot-gun. Combined with the *acknowledged* front-running issue, a seller can raise the price with `updateListing` right after a buyer submits a `buyItem` with a generous `msg.value`; as long as the new price is still `<= msg.value`, the buyer overpays and the seller pockets the difference.

Risk: MEDIUM

- **Impact: MEDIUM** — direct, unbounded loss of the overpaid amount for the buyer.

- **Likelihood: MEDIUM** — overpayment is easy to trigger: stale UI prices, a price updated between quote and execution, slippage buffers, or simply rounding the value up.

Proof of Concept:

```
function test_buyItemOverpaymentIsCreditedToSellerNotRefunded()
public {
    uint256 price = 1 ether;
    uint256 sent = 1.5 ether; // buyer overpays by 0.5 ether
    uint256 tokenId = _list(seller, price);

    console2.log("listing price (wei)             :", price);
    console2.log("buyer sends    (wei)             :", sent);
    console2.log("buyer balance before             :",
buyer.balance);

    vm.deal(buyer, sent);
    vm.prank(buyer);
    market.buyItem{value: sent}(address(nft), tokenId);

    // The seller is credited the FULL msg.value, not the listing
price.
    console2.log("seller proceeds after buy       :",
market.getProceeds(seller));
    console2.log("expected (if price only)        :", price);
    console2.log("overpayment captured by seller  :",
market.getProceeds(seller) - price);
    console2.log("buyer balance after (no refund) :",
buyer.balance);
    assertEq(market.getProceeds(seller), sent, "seller captured
the overpayment");
    assertGt(market.getProceeds(seller), price, "proceeds exceed
the listed price");

    // The buyer received the NFT but got NO refund of the 0.5
ether excess.
    assertEq(nft.ownerOf(tokenId), buyer);
    assertEq(buyer.balance, 0, "buyer was not refunded the
overpayment");
    console2.log("RESULT                           : buyer lost
0.5 ETH overpayment to seller");
}
```

The companion test `test_refMarketAndOptimizedShareOverpaymentBug` confirms the reference contract behaves identically, proving this is a shared logic bug (not an assembly defect):

```
function test_refMarketAndOptimizedShareOverpaymentBug() public {
    // List the same scenario on the reference contract and
confirm the
    // identical (buggy) accounting, proving the assembly version
did not
```

```
    // introduce/fix the issue - it is a logic bug in both.
    uint256 price = 1 ether;
    uint256 sent = 2 ether;

    vm.startPrank(seller);
    nft.mint();
    uint256 tokenId = nft.totalSupply() - 1;
    nft.approve(address(refMarket), tokenId);
    refMarket.listItem(address(nft), tokenId, price);
    vm.stopPrank();

    vm.deal(buyer, sent);
    vm.prank(buyer);
    refMarket.buyItem{value: sent}(address(nft), tokenId);

    console2.log("reference listing price (wei)   :", price);
    console2.log("buyer sends    (wei)            :", sent);
    console2.log("reference seller proceeds       :",
refMarket.getProceeds(seller));
    assertEq(refMarket.getProceeds(seller), sent, "refMarket also
captures overpayment");
    console2.log("RESULT                          : reference ==
optimized (shared logic bug)");
}
```

Run `forge test --match-test test_buyItemOverpaymentIsCreditedToSellerNotRefunded -vv`:

```
[PASS] test_buyItemOverpaymentIsCreditedToSellerNotRefunded()
Logs:
  listing price (wei)             : 1000000000000000000
  buyer sends    (wei)            : 1500000000000000000
  buyer balance before            : 0
  seller proceeds after buy       : 1500000000000000000
  expected (if price only)        : 1000000000000000000
  overpayment captured by seller  : 500000000000000000
  buyer balance after (no refund) : 0
  RESULT                          : buyer lost 0.5 ETH
overpayment to seller
```

The seller is credited the full `1.5 ETH` instead of the `1.0 ETH` list price, the buyer is refunded `0`, and the `0.5 ETH` overpayment is silently captured.

The cross-check test prints the identical result against the reference contract:

```
[PASS] test_refMarketAndOptimizedShareOverpaymentBug()
Logs:
  reference listing price (wei)   : 1000000000000000000
  buyer sends    (wei)            : 2000000000000000000
  reference seller proceeds       : 2000000000000000000
  RESULT                          : reference == optimized
(shared logic bug)
```

Recommended Mitigation: Credit the seller only the listed `price` and refund the remainder to the buyer (pull-over-push is safest, but a direct refund after state changes is acceptable here because CEI is preserved):

```
- s_proceeds[seller] += msg.value;
+ s_proceeds[seller] += price;
  delete s_listings[nftAddress][tokenId];
  // ...emit...
  IERC721(nftAddress).safeTransferFrom(address(this), msg.sender,
        tokenId);
+ if (msg.value > price) {
+     (bool ok,) = payable(msg.sender).call{value: msg.value -
        price}("");
+     if (!ok) revert NftMarketplace__TransferFailed();
+ }
```

Low

[L-1] `withdrawProceeds` emits no event

Location:

- `src/GasBadNftMarketplace.sol:175-189` — `withdrawProceeds`
- `src/NftMarketplace.sol:125-139` — `withdrawProceeds`

Description: `withdrawProceeds` zeroes a seller's balance and sends ETH, but emits no event. Every other state-changing function emits one. This is listed by the team under "Known Issues", and is independently flagged by Aderyn (L-2). Off-chain indexers cannot track withdrawals.

Risk: LOW

- **Impact: LOW** — no funds at risk; observability/accounting only.
- **Likelihood: HIGH** — every withdrawal is unobservable on-chain via events.

Proof of Concept:

```
function test_withdrawProceedsEmitsNoEvent() public {
    uint256 price = 1 ether;
    uint256 tokenId = _list(seller, price);

    vm.deal(buyer, price);
    vm.prank(buyer);
    market.buyItem{value: price}(address(nft), tokenId);
    console2.log("seller proceeds before withdraw :",
market.getProceeds(seller));

    vm.recordLogs();
    vm.prank(seller);
    market.withdrawProceeds();
```

```
    Vm.Log[] memory logs = vm.getRecordedLogs();

    console2.log("seller proceeds after withdraw  :",
market.getProceeds(seller));
    console2.log("events emitted by withdraw      :",
logs.length);
    assertEq(logs.length, 0, "withdrawProceeds emits no event at
all");
    console2.log("RESULT                           : withdrawal is
invisible to off-chain indexers");
}
```

Run `forge test --match-test test_withdrawProceedsEmitsNoEvent -vv`:

```
[PASS] test_withdrawProceedsEmitsNoEvent()
Logs:
  seller proceeds before withdraw : 10000000000000000000
  seller proceeds after withdraw  : 0
  events emitted by withdraw      : 0
  RESULT                          : withdrawal is invisible to
off-chain indexers
```

Proceeds drop from `1.0 ETH` to `0`, yet `events emitted by withdraw = 0` the state change is completely invisible to event based indexers.

Recommended Mitigation:

```
+ event ProceedsWithdrawn(address indexed seller, uint256
        amount);

  function withdrawProceeds() external {
      uint256 proceeds = s_proceeds[msg.sender];
      if (proceeds <= 0) revert NftMarketplace__NoProceeds();
      s_proceeds[msg.sender] = 0;
+     emit ProceedsWithdrawn(msg.sender, proceeds);
      (bool success,) = payable(msg.sender).call{value: proceeds}
        ("");
      if (!success) revert NftMarketplace__TransferFailed();
  }
```

[L-2] No validation that `nftAddress` is a contract / a real ERC-721

Location:

- `src/GasBadNftMarketplace.sol:38-64` — `listItem` (storage write at line 45, before the external call at line 63)
- `src/NftMarketplace.sol:38-50` — `listItem` (storage write at line 45, before the external call at line 49)

Description: `listItem` accepts any `nftAddress`. The `safeTransferFrom` call to an EOA or a non-ERC-721 contract reverts, which mostly protects `listItem`. However, the

listing is written to storage **before** the external call (CEI ordering), so a malicious or non-standard token that returns success from `safeTransferFrom` without moving a token could create a listing that does not correspond to a real escrowed asset. The marketplace trusts the NFT contract entirely.

Risk: LOW

- **Impact: LOW** — limited to interactions with intentionally malicious token contracts; honest ERC-721s are unaffected.
- **Likelihood: LOW.**

Recommended Mitigation: Document that only standards-compliant ERC-721 contracts are supported, and optionally check `nftAddress.code.length > 0` and `IERC165(nftAddress).supportsInterface(type(IERC721).interfaceId)` in `listItem`.

Gas

[G-1] `<= 0` comparisons on `uint256` are misleading and slightly wasteful

Location:

- `src/GasBadNftMarketplace.sol:40` / `src/NftMarketplace.sol:40` — `price <= 0` in `listItem`
- `src/GasBadNftMarketplace.sol:148` / `src/NftMarketplace.sol:108` — `newPrice <= 0` in `updateListing`
- `src/GasBadNftMarketplace.sol:178` / `src/NftMarketplace.sol:128` — `proceeds <= 0` in `withdrawProceeds`

`price <= 0`, `newPrice <= 0`, and `proceeds <= 0` are applied to unsigned integers, where `x <= 0` is exactly `x == 0`. The `<=` form reads as if negatives were possible and costs a tiny amount more.

```
- if (price <= 0) {
+ if (price == 0) {
      revert NftMarketplace__PriceMustBeAboveZero();
  }
```

Applies to `listItem`, `updateListing`, and `withdrawProceeds` in **both** contracts.

[G-2] `buyItem` re-reads the listing struct into multiple locals

Location: `src/GasBadNftMarketplace.sol:104-115` — `buyItem` (struct copy + locals at lines 105-107, zero-check at line 109)

`GasBadNftMarketplace.buyItem` copies the whole `Listing` struct into memory and then re-reads `listedItem.seller` for the zero-check. Reading `seller/price` once and reusing the cached locals is marginally cheaper

Informational

[I-1] Acknowledged known issues (front-running / MEV)

Location: `README.md` (Known Issues section); affected code: `cancelListing` (`src/GasBadNftMarketplace.sol:73-96`) and `updateListing` (`src/GasBadNftMarketplace.sol:146-170`)

The README explicitly acknowledges:

- A seller can front-run a purchase and `cancelListing`.
- A seller can front-run a purchase and `updateListing` (raising the price).
- General MEV / front-running exposure.

[I-2] PUSH0 opcode limits deployment to PUSH0-enabled chains

Location: `src/GasBadNftMarketplace.sol:2`, `src/NftMarketplace.sol:2`, `src/INftMarketplace.sol:2` (`pragma solidity 0.8.20;`)

Compiling with solc 0.8.20 defaults to the Shanghai EVM, emitting `PUSH0`. The README targets Ethereum mainnet (which supports it), so this is informational. If the contracts are ever ported to a chain without PUSH0, set `evm_version` explicitly.

[I-3] Static-analysis false positive: "state change without event" on assembly emitters

Location: `src/GasBadNftMarketplace.sol` — `listItem:38`, `cancelListing:73`, `buyItem:104`, `updateListing:146` (events emitted via `log4` at lines 48-60, 82-92, 121-132, 158-169 respectively)

Aderyn reports L-2 *"State Change Without Event"* against `listItem`, `cancelListing`, `buyItem`, and `updateListing` in `GasBadNftMarketplace.sol`. **This is a false positive.** Those functions do emit events via raw `log4` in inline assembly, which AST-based tools cannot see. Only `withdrawProceeds` genuinely lacks an event. This is a useful reminder that automated tools under report on hand rolled assembly, and why the topics had to be verified manually.