SuperCowNFT: **Dividend Code Audit**

July 14, 2025

Audit based on code:

https://bscscan.com/address/0x5798499C516Cc4b02892622aa7B0F4151d666666

Unchecked ERC-20 Transfers

You do

```
IERC20(token).transfer(_msgSender(), _amount);
```

but if the token returns `false` instead of reverting, you'll silently ignore the failure.

Suggestion:

Use OpenZeppelin's `SafeERC20` so that failures revert:

```
IERC20(token).safeTransfer(_msgSender(), _amount);
```

Gas-Costly On-Chain Loops & OOG Risk

```
address[] memory holders = IERC721(nft).getAllOwner();

for (uint i = 0; i < holders.length; i++) { … }

for (uint j = 0; j < holdersD.length; j++) { … }
```

If your NFT has hundreds or thousands of holders, this will blow past block gas limits.

Suggestion:

- **Off-chain indexing**: emit events for deposits and let an off-chain subgraph or script calculate per-wallet dividends.
- Or use a cumulative-points approach (similar to ERC-20 dividend trackers) so that you don't need to iterate every distribution.

Incorrect Blacklist Filtering and Counting

// you filter into holdersD[total]++ and then do j < holdersD.length…

You should iterate `j < total`, not `holdersD.length`, otherwise you do extra zero-address checks.

You never filter out the zero-address owner from the original holders list, which could skew your count.

Hard-Coded “1000” Threshold Lacks Decimals Context

```
if (amount > 1000) { … }
```

Is that 1000 wei of your ERC-20? 1000 tokens? 1000 × 10^{-18}? It’s ambiguous.

Either express thresholds in “token smallest units” (`1_000 * 10**decimals`) or make it a constructor-settable parameter, e.g. `uint256 public minDistribution;`.

Odd ETH-Triggered Control Flow

You use the `receive()` hook to both

1. Kick off bookkeeping (if called by the bookkeeper)
2. Trigger `claim()` (if called by a user)
3. Immediately refund all ETH back to the caller

That is a very non-standard UX and makes auditing/usage confusing.

- Why does anyone ever send ETH to the contract?
- Why do you refund it unconditionally?

No Events on Claim, Totals Might Drift

- You emit `BookEvent` during bookkeeping, but you never emit an event when someone actually claims.
- You track `sendTotal` and `receiveTotal` separately, but `receive()` can call `claim()` multiple times, and it's not obvious that `sendTotal−receiveTotal` always equals the outstanding unclaimed balance.

Suggestion:

- Add a `ClaimEvent(address indexed user, uint256 amount)` when a claim succeeds.
- Double-check that

  ```
  IERC20(token).balanceOf(address(this)) + receiveTotal − sendTotal == 0
  ```

  always holds, or simplify by avoiding these running totals and just reading balances.

No Reentrancy Guard on ETH Transfers

`_safeTransferETH` does:

```
(bool success, ) = to.call{value: value}(new bytes(0));
```

If `to` is a malicious contract, its fallback could call back into to contract. Currently the state changes happen before the call, so it will be safe, but it's a fragile pattern.

Suggestion:

- If you must refund ETH, do it via the Checks-Effects-Interactions pattern (which mostly happens now) and consider OpenZeppelin's `ReentrancyGuard` on theexternal entry points.

Scalability & Data Structures

- Storing `usersBook` per-address is fine, but you will eventually hit map size limits if you have thousands of holders.
- You never clean up old entries, so ghost entries could accumulate.

Suggestion:
Consider a “cumulative per-share” approach:

```
uint256 public magnifiedDividendPerShare;

mapping(address => int256) public magnifiedDividendCorrections;

// on distribute:

magnifiedDividendPerShare += (amount * MAGNITUDE) / totalSupply;

// on claim:

uint256 withdrawable = ((magnifiedDividendPerShare * balanceOf(msg.sender)) +
magnifiedDividendCorrections[msg.sender]) / MAGNITUDE;
```

Centralization

Owner can:

Asset withdrawals

- `withdrawToken(address _token, address _to, uint256 _amount)`
  ‣ Transfer any ERC-20 token held by the contract to `_to`.

- `withdraw(address _to, uint256 _amount)`
  ‣ Send `_amount` wei of ETH from the contract to `_to`.

Blacklist management

- `setBL(address _addr, bool _b)`
  ‣ Add or remove `_addr` from the internal blacklist (BL), so it's excluded from future distributions.

Distribution controls

- `setBookkeeper(address _bookkeeper)`
  ‣ Change the address allowed to trigger the bookkeeping logic.

- `setToken(address _token)`
  ‣ Point the contract at a different ERC-20 token for dividends.

- `setNFT(address _nft)`
  ‣ Point the contract at a different ERC-721 collection whose holders receive dividends.

Ownership management (inherited from Ownable)

- `transferOwnership(address newOwner)`
  ‣ Assign the `owner` role to `newOwner`.

- `renounceOwnership()`
  ‣ Relinquish ownership entirely (leaving the contract without an owner).

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.