## Donation-Based Share Price Manipulation

### Explainer

In vaults that distribute shares based on the ratio of *total assets* to *total shares*, an attacker can send assets directly to the strategy or vault (a “donation”) to **artificially inflate the share price**. This causes **subsequent depositors to mint 0 or near-0 shares**, effectively transferring their funds to early depositors (usually the attacker).

---

### Cause

The root cause is the **share-minting logic relying purely on totalAssets() / totalSupply()**, without accounting for untracked or unintentional inflows (e.g., donations). This allows attackers to game the asset/share ratio.

In particular:

- Shares = (deposit amount * total shares) / total assets
- If `total assets` is inflated, `shares minted` approaches 0

---

### Where to Look

1. **Vaults** that issue shares in return for deposits (`mintShares()` or `deposit()` functions).
2. **Strategies** that manage funds on behalf of vaults and may receive untracked tokens.
3. **Functions** using:

   ```
   shares = (amount * totalSupply()) / totalAssets();
   ```

4. **Protocols with composable integrations**, e.g., where any address can transfer funds directly to a vault or strategy without restrictions.

---

### Why This Happens

Most vault designs assume a linear, predictable flow of assets and ignore the **possibility of external asset inflows** that don’t mint shares. These inflows distort the accounting because:

- The asset count increases,
- But the share count remains the same,
- Making shares more expensive (i.e., fewer minted per deposit),
- Causing new users to get fewer shares,
- Letting early depositors benefit disproportionately.

This is especially dangerous when:

- The protocol does **not** restrict where assets can come from,
- Strategies auto-deposit all assets they hold,
- There's no buffer mechanism to mitigate initial price manipulation.

---

### Recommended Solutions

1. **Seed Shares / Burn Initial Liquidity**
   Send the first 1000 (or more) shares to `address(0)` or lock them permanently to prevent disproportionate ownership when asset values are low.
   *(Used by Uniswap V2, Balancer, etc.)*
2. **Minimum Share Mint Threshold**
   Ensure that deposits must mint **non-zero shares**:

   ```
   require(shares > 0, "Zero shares");
   ```

3. **Donations Ignored in Share Math**
   Track only *intentional deposits* when calculating `totalAssets()`:
   - Use internal accounting rather than raw ERC20 balances.
   - E.g., maintain a `trackedAssets` variable updated only during controlled flows.
4. **Limit or Sanitize External Transfers**
   Prevent users from directly sending tokens to the vault or strategy (e.g., via a whitelist or enforced deposit flow).
5. **Use TWAP or Oracle Valuation (Advanced)**
   For more sophisticated systems, pricing should not rely directly on token balances alone but on external price feeds.