Assuming All ERC-20 Tokens Transfer Full Amounts (Ignoring Fee-on-Transfer Behavior)

Explainer

Fee-on-transfer (FoT) tokens deduct a **fee or tax** when a transfer happens. For example:

```
transfer(to, 1000); // recipient only receives 950
```

But most DeFi protocols assume:

```
- transfer() = exact amount received
- transferFrom() = exact amount moved
```

This mismatch causes:

- **Underfunded swaps**, **vaults**, or **deposits**
- **Failed logic or reverts** when checking balances post-transfer
- **Lost funds** if FoT tokens are forwarded without accounting
- **Broken accounting** in lending, staking, pools, or bridge systems

---

Cause

- The ERC-20 spec does **not guarantee** that `transfer()` or `transferFrom()` moves the full amount.
- Many DeFi protocols **do not check the actual balance delta** — they assume the token is vanilla.
- FoT tokens **subtract fees internally**, meaning:
  - Sender sends 1000
  - Recipient receives 950
  - 50 goes to burn/fund/dev pool

Where to Look (General)

Look for any logic that assumes **transfer is exact**, especially:

1. **ERC20 → Protocol → Vault** patterns:

```
token.transferFrom(msg.sender, address(this), amount); // Assumes `amount` received
```

2. **Swaps**

```
token.transfer(pair, amount);
pair.swap(...); // Assumes `amount` was fully received by pair
```

3. **Lending protocols**

```
require(token.balanceOf(address(this)) >= expectedCollateral, "Underfunded");
```

4. **Bridges and escrow**

- User deposits FoT token but the bridge doesn't receive full amount

5. **Reward calculations / fees**

- `userDepositAmount` logged as 1000, but only 900 was received → unfair interest or slashing

---

Why This Happens

- ERC-20 standard doesn’t enforce return values or strict behavior
- FoT tokens are often used for speculative purposes or tax mechanisms
- Protocol devs optimize for common tokens (DAI, USDC) and don’t consider fee tokens

---

General Recommended Solutions

Always measure the **actual received amount**:

```
uint256 before = token.balanceOf(address(this));
token.transferFrom(msg.sender, address(this), amount);
uint256 after = token.balanceOf(address(this));
uint256 received = after - before;
```

Avoid relying on `transfer()`'s return value for amount validation

For swaps/pools:

- Use wrapper functions or safe routers that measure exact in/out

For bridges/staking:

- Log and act on received amount, not sent amount

Add token type whitelists (if you only want to support vanilla ERC-20s)

Warn in docs: “Only standard ERC20s supported” — or actively check for FoT behavior

Consider using `safeTransferFrom()` wrappers that enforce `balanceOf()` checks

Example Vulnerabilities

Protocol	**Bug**
Rari Capital	Underfunded vault deposits due to FoT tokens
Cream / Forks	Incorrect accounting for user collateral when token taxed
Some DEXes	Liquidity pools failed due to unexpected token inflows