Withdrawal Limit Based on Mutable Balance State

The bug arises when **withdrawal limits are calculated based on `address(this).balance` at the *time of withdrawal*** — rather than fixing the baseline at the start of the limit window (e.g., per day). Because this balance is reduced with each withdrawal, the **limit keeps shrinking**, potentially rejecting valid withdrawals even if the total withdrawn amount is within the allowed quota (e.g., 10%).

This breaks user expectations and **makes the limit effectively tighter than intended**.

Cause

The contract uses the **current contract balance** (`address(this).balance`) *after* withdrawals have already happened within the window to validate future withdrawals. This causes the maximum allowed withdrawal amount to **dynamically decrease** during the day.

For example:

- Initial balance: 1000 ETH → 10% limit = 100 ETH/day
- User 1 withdraws 50 ETH → balance = 950
- User 2 tries to withdraw 50 ETH → 50 + 50 > 10% of 950 (which is 95 ETH) → **fails**, even though **total withdrawal is within 100 ETH**

Where to Look

1. **Rate-limited withdrawal systems**
   - Daily/weekly/monthly limits on withdrawals
   - Bridges, DAOs, treasuries, multisigs
2. **Code that uses dynamic values in invariant conditions**
   - Especially expressions like:

     ```
     require(amount + withdrawnSoFar <= address(this).balance * factor / 100)
     ```

3. **Systems with "sliding window" logic** but use **state-mutated variables** like `balance` instead of snapshotting or fixed references.
4. **Bridges** or **layer-2 escape hatches** — where withdrawal caps are a common safety control.

Why This Happens

Developers mistakenly assume that using `address(this).balance` reflects the system's capacity to withdraw funds proportionally. However:

- The **balance changes during the limit period**,
- Making the **reference point unstable** and prone to failure with multiple withdrawals,
- Especially in **multi-user environments**, where prior withdrawals reduce the allowable ceiling for later ones.

This is a form of **non-idempotent state logic**.

Recommended Solutions

1. **Snapshot the Reference Balance at Window Start**
   Record `startingBalance` when the withdrawal window (e.g., daily) resets, and use it consistently:

   ```
   uint256 limit = withdrawalFactor * s.snapshotBalance / 100;
   require(_amount + withdrawnSoFar <= limit, "limit exceeded");
   ```

2. **Avoid Using `address(this).balance` in Repeated Limit Checks**
   It's mutable and can be manipulated or unintentionally altered.
3. **Separate Balance Accounting**
   Maintain internal accounting for limits independent of actual ETH balance.
4. **Implement Window Resets Properly**
   Update snapshot variables like `snapshotBalance` at the start of each new limit window:

   ```
   if (block.timestamp >= s.lastReset + 1 days) {
       s.snapshotBalance = address(this).balance;
       s.lastReset = block.timestamp;
       s.withdrawnAmountInWindow = 0;
   }
   ```

5. **Unit Test for Multi-Withdrawal Behavior**
   Always test scenarios where multiple withdrawals happen sequentially within a time window.