The contract doesn't properly handle historical reward rate changes when calculating rewards.

```
// Add helper function to calculate rewards across rate changes
    function getGeneratedReward(uint256 _fromTime, uint256 _toTime) public view
returns (uint256) {
        if (_fromTime >= _toTime) return 0;

        uint256 totalRewards = 0;
        uint256 totalWeeks = (_toTime - _fromTime) / 1 weeks;
        uint256 startWeek = (_fromTime - poolStartTime) / 1 weeks;

        // Iterate through weeks and add up rewards
        for (uint256 i = 0; i <= totalWeeks; i++) {
            uint256 weekNumber = startWeek + i;
            uint256 weekRate = weeklyShareRate[weekNumber];
            if (weekRate == 0) {
                weekRate = sharePerSecond; // Use current rate if no specific rate
set
            }

            uint256 weekStart = _fromTime + (i * 1 weeks);
            uint256 weekEnd = weekStart + 1 weeks;
            if (weekEnd > _toTime) weekEnd = _toTime;

            totalRewards = totalRewards.add(
                (weekEnd - weekStart).mul(weekRate)
            );
        }

        return totalRewards;
    }
```

Problem Breakdown

The helper function `getGeneratedReward` is supposed to calculate rewards between `_fromTime` and `_toTime`, even if reward rates changed in the middle (e.g., weekly reward halving).

But the current logic has a **flaw**:

- If `weeklyShareRate[weekNumber] == 0`, it **defaults to** `sharePerSecond` **(the *current* rate)**.

- That means if someone claims **after multiple weeks**, the calculation won't respect the *historical reward rates* for past weeks — instead, it applies today's rate retroactively.

Example

- Suppose:
  - Week 1 rate = **2 GHOG/sec**
  - Week 2 rate = **1 GHOG/sec**
- A user stakes at **t=0** and claims at **t=14 days**.

Expected rewards:

- `7 days * 2 GHOG/sec` + `7 days * 1 GHOG/sec` = **(1209600 GHOG + 604800 GHOG) = 1814400 GHOG**

Actual with flawed code (if week 1 not explicitly stored):

- `14 days * current_rate (1 GHOG/sec)` = **1209600 GHOG**

The user loses **604800 GHOG** just because the contract didn't persist the old rate in `weeklyShareRate`.

Impact

- **Users underpaid:** If protocol lowers rate, past high rewards are ignored.
- **Users overpaid:** If protocol raises rate, past low rewards are retroactively boosted.
- **Exploitable edge cases:**
  - An attacker could **delay claiming** until after a rate hike to unfairly earn more than intended.
  - Honest stakers could be **cheated** out of historical high-rate rewards.

This is **not just accounting error — it's reward misdistribution** (high severity).

How to Fix

1. **Store every rate change explicitly in `weeklyShareRate`.**
   - When protocol sets a new `sharePerSecond`, log it for that exact week.

2. **Remove the `if (weekRate == 0)` fallback.**
    - Instead, require that *every week has an explicit rate stored*, so no default “current rate” is applied retroactively.
3. **Alternative approach (gas-efficient):**
    - Store `(timestamp, rate)` checkpoints in an array.
    - Iterate over checkpoints to compute rewards for a given time window.
