fetch_CVE.py

```
import requests
from config import NVD_API_URL, NVD_API_KEY
from datetime import datetime, timedelta, timezone
import os
import json
import threading
import time

#Where we'll keep scraped CVEs between sessions (local file cache)
CACHE_PATH = "data/cache/cve_cache.json"
CACHE_TIME_MINUTES = 1440   # 24 hours
SCHEDULE_HOUR = 0           # Midnight (server local time)

def _is_cache_fresh():
    #Checks if local CVE cache is recent enough
    if not os.path.exists(CACHE_PATH):
        return False
    mtime = datetime.fromtimestamp(os.path.getmtime(CACHE_PATH))
    now = datetime.now()
    return (now - mtime).total_seconds() < CACHE_TIME_MINUTES * 60

def _load_from_cache():
    #Loads a cached batch of CVEs from disk
    if not os.path.exists(CACHE_PATH):
        return []
    try:
        with open(CACHE_PATH, "r", encoding="utf-8") as f:
            return json.load(f)
    except Exception:
        #Corrupted cache? Just ignore and act empty
        return []

def _save_to_cache(cves):
    #Dump the whole CVE list to disk (creates directory if needed)
    os.makedirs(os.path.dirname(CACHE_PATH), exist_ok=True)
    with open(CACHE_PATH, "w", encoding="utf-8") as f:
        json.dump(cves, f)

def _fetch_from_nvd(days=30):
    #Grabs CVEs from NVD API for the past X days (big batch, paginated)
    now_utc = datetime.now(timezone.utc)
```

```
    all_cves = []
    start_index = 0
    results_per_page = 2000
    pub_end = now_utc
    pub_start = pub_end - timedelta(days=days - 1)

    params = {
        "resultsPerPage": results_per_page,
        "startIndex": start_index,
        "pubStartDate":
pub_start.isoformat(timespec='milliseconds').replace('+00:00', 'Z'),
        "pubEndDate":
pub_end.isoformat(timespec='milliseconds').replace('+00:00', 'Z')
    }
    headers = {"apikey": NVD_API_KEY}
    #Loop the NVD pagination (max 2000 per page)
    while True:
        try:
            response = requests.get(NVD_API_URL, params=params,
headers=headers, timeout=60)
            response.raise_for_status()
            data = response.json()
        except requests.RequestException as e:
            print(f"Error fetching CVEs from NVD: {e}")
            break

        vulnerabilities = data.get("vulnerabilities", [])
        if not vulnerabilities:
            #NVD returned no new CVEs, we're done
            break
        for item in vulnerabilities:
            cve = item.get("cve", {})
            published_str = cve.get("published", "")
            if not published_str:
                continue
            try:
                published_dt = datetime.strptime(published_str[:10], "%Y-%m-
%d").replace(tzinfo=timezone.utc)
            except ValueError:
                continue
            cve_id = cve.get("id", "")
            description = next((desc["value"] for desc in
cve.get("descriptions", []) if desc.get("lang") == "en"), "")
            severity = "UNKNOWN"
            cvss = None
            metrics = cve.get("metrics", {})
```

```
            #Try CVSS v3.1, then v3.0, then v2
            if "cvssMetricV31" in metrics:
                sev_metric = metrics["cvssMetricV31"][0]
                severity = sev_metric["cvssData"].get("baseSeverity",
severity)
                cvss = sev_metric["cvssData"].get("baseScore", cvss)
            elif "cvssMetricV30" in metrics:
                sev_metric = metrics["cvssMetricV30"][0]
                severity = sev_metric["cvssData"].get("baseSeverity",
severity)
                cvss = sev_metric["cvssData"].get("baseScore", cvss)
            elif "cvssMetricV2" in metrics:
                sev_metric = metrics["cvssMetricV2"][0]
                severity = sev_metric.get("baseSeverity", severity)
                cvss = sev_metric["cvssData"].get("baseScore", cvss)
            #Pull out CWE if present
            cwe = None
            for weakness in cve.get("weaknesses", []):
                for desc in weakness.get("description", []):
                    if desc.get("lang") == "en":
                        cwe = desc.get("value")
                        break
                if cwe:
                    break
            #Fill in all fields; "Products" will be empty here
            all_cves.append({
                "ID": cve_id,
                "Description": description,
                "Severity": severity,
                "CVSS_Score": cvss,
                "CWE": cwe,
                "Published": published_str,
                "References": [ref["url"] for ref in cve.get("references",
[])],
                "Products": [],
                "metrics": metrics
            })

        total_results = data.get("totalResults", 0)
        if start_index + results_per_page >= total_results:
            #Got everything, break the pagination loop
            break
        start_index += results_per_page
        params["startIndex"] = start_index

    #Sort from newest to oldest so latest threats are always up top
```

```
    all_cves.sort(key=lambda x: x.get("Published") or "", reverse=True)
    return all_cves

def _refresh_cache():
    #Force a cache update from NVD (used by dashboard and scheduler)
    print("[CVEs] Auto-refreshing CVE cache from NVD...")
    cves = _fetch_from_nvd(days=30)
    _save_to_cache(cves)
    print(f"[CVEs] Cache refreshed. Fetched {len(cves)} CVEs.")

def _auto_refresh_job():
    #Scheduler thread: wakes up at midnight & updates cache (for dashboard
speed)
    while True:
        now = datetime.now()
        next_run = now.replace(hour=SCHEDULE_HOUR, minute=0, second=0,
microsecond=0)
        #If it's after the scheduled hour, aim for next day
        if now >= next_run:
            next_run = next_run + timedelta(days=1)
        delay = (next_run - now).total_seconds()
        print(f"[CVEs] Next auto-refresh scheduled at {next_run}. Sleeping
{int(delay)} seconds...")
        time.sleep(max(delay, 0))
        try:
            _refresh_cache()
        except Exception as e:
            print(f"[CVEs] Cache refresh FAILED: {e}")

def start_auto_cache_scheduler():
    #Start the scheduler thread automatically as soon as module is imported
    t = threading.Thread(target=_auto_refresh_job, daemon=True)
    t.start()

def get_all_cves(max_results=None, year=None, month=None, days=None,
force_refresh=False):
    """
    Returns the latest CVEs. Uses local cache unless filtering by time.
    If cache is old or missing, does a fresh fetch & saves.
    You can filter the results by year/month for more narrow data.
    """
    #Dashboard always uses the cache unless filters or manual refresh
    if not year and not month and not days and not force_refresh and
_is_cache_fresh():
        return _load_from_cache()
    #If dashboard, and cache is missing/outdated, fetch & block until we have
```

```
it
    if not year and not month and not days and not force_refresh and not
_is_cache_fresh():
        _refresh_cache()
        return _load_from_cache()
    #If filtering (or forced refresh), always fetch live from NVD
    cves = _fetch_from_nvd(days=days or 30)

    #Filter by year and month as needed (handy for advanced charts)
    if year and month:
        filtered = []
        for cve in cves:
            pub = cve.get("Published", "")
            if len(pub) >= 7:
                if pub[:4] == str(year) and int(pub[5:7]) == int(month):
                    filtered.append(cve)
        return filtered

    #Otherwise, return all CVEs we just loaded
    elif year:
        filtered = []
        for cve in cves:
            pub = cve.get("Published", "")
            if len(pub) >= 4 and pub[:4] == str(year):
                filtered.append(cve)
        return filtered

    # Otherwise, return all CVEs we just loaded
    return cves

#Fire off background auto-refresh thread at module import (so dashboard cache
is always ready)
start_auto_cache_scheduler()
```