cwe_scraper.py

```python
import requests
from bs4 import BeautifulSoup
import json
import os
import time
from functools import lru_cache

#Where I'm saving scraped CWE info (local, fast lookup when available)
CACHE_DIR = "data/cache"
CWE_CACHE_FILE = os.path.join(CACHE_DIR, "cwe_scraped_data.json")
CACHE_EXPIRY_HOURS = 24 #How long until I want to re-scrape?

def ensure_cache_dir():
    #Make the cache folder if it's not there already
    os.makedirs(CACHE_DIR, exist_ok=True)

def is_cache_valid():
    #Check if I've got cached data and it's "fresh" (not too old)
    if not os.path.exists(CWE_CACHE_FILE):
        return False

    file_age = time.time() - os.path.getmtime(CWE_CACHE_FILE)
    return file_age < (CACHE_EXPIRY_HOURS * 3600)

def load_cached_cwe_data():
    #Pull CWE data out of my local cache (if present)
    try:
        with open(CWE_CACHE_FILE, 'r', encoding='utf-8') as f:
            return json.load(f)
    except Exception:
        #Something broke or file is corrupted? Just act like nothing cached.
        return {}

def save_cwe_data_to_cache(data):
    #Save the full CWE dict to disk (so next run is faster)
    ensure_cache_dir()
    try:
        with open(CWE_CACHE_FILE, 'w', encoding='utf-8') as f:
            json.dump(data, f, indent=2, ensure_ascii=False)
    except Exception as e:
        print(f"Failed to save CWE cache: {e}")
```

```
def scrape_cwe_detail(cwe_id):
    #Pulls data for one CWE from MITRE (scrapes their HTML)
    if not cwe_id.startswith('CWE-'):
        return None

    cwe_number = cwe_id.replace('CWE-', '')
    url = f"https://cwe.mitre.org/data/definitions/{cwe_number}.html"

    try:
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'
        }
        response = requests.get(url, headers=headers, timeout=10)
        response.raise_for_status()

        soup = BeautifulSoup(response.content, 'html.parser')

        #Extract CWE name
        name = "Unknown"
        title_elem = soup.find('h2')
        if title_elem:
            title_text = title_elem.get_text(strip=True)
            if ':' in title_text:
                name = title_text.split(':', 1)[1].strip()

        #Extract description
        description = ""
        desc_section = soup.find('div', {'id': 'Description'}) or
soup.find('div', class_='indent')
        if desc_section:
            #Get the first paragraph of description
            paragraphs = desc_section.find_all('p')
            if paragraphs:
                description = paragraphs[0].get_text(strip=True)

        if not description:
            #Fallback: look for any div with description-like content
            desc_divs = soup.find_all('div', class_='indent')
            for div in desc_divs:
                text = div.get_text(strip=True)
                if len(text) > 50:  #Reasonable description length
                    description = text[:500] + "..." if len(text) > 500 else
text
                    break
        #Extract mitigation strategies
        mitigations = []
        mitigation_section = soup.find('div', {'id': 'Potential_Mitigations'})
        if mitigation_section:
            mitigation_items = mitigation_section.find_all(['li', 'p'])
            for item in mitigation_items[:3]:  #Limit to first 3 mitigations
                mitigation_text = item.get_text(strip=True)
                if len(mitigation_text) > 20:  #Filter out short/empty items
                    mitigations.append(mitigation_text[:200])

        #Extract examples
        examples = []
        example_section = soup.find('div', {'id': 'Demonstrative_Examples'})
        if example_section:
            code_blocks = example_section.find_all(['code', 'pre'])
            for code in code_blocks[:2]:  #Limit to first 2 examples
                example_text = code.get_text(strip=True)
                if example_text:
                    examples.append(example_text[:150])

        #Extract relationships
        relationships = []
        related_section = soup.find('div', {'id': 'Related_Attack_Patterns'})
or soup.find('div', {'id': 'Relationships'})
        if related_section:
            links = related_section.find_all('a', href=True)
            for link in links[:5]:  #Limit relationships
                href = link.get('href', '')
                if '/data/definitions/' in href:
                    related_cwe = link.get_text(strip=True)
                    if related_cwe.startswith('CWE-'):
                        relationships.append((related_cwe, 'Related'))

        #return all the details in a nice consistent dict
        return {
            'id': cwe_id,
            'name': name,
            'description': description or f"CWE-{cwe_number} is a software
weakness category tracked by MITRE.",
            'mitigations': mitigations,
            'examples': examples,
            'relationships': relationships,
            'source': 'scraped',
            'scraped_at': time.time()
        }
```

```
    except requests.RequestException as e:
        print(f"Failed to scrape CWE-{cwe_number}: {e}")
        return None
    except Exception as e:
        print(f"Error parsing CWE-{cwe_number}: {e}")
        return None

def get_enhanced_fallback_data():
    # Only key/popular CWEs; real details from MITRE are better
    return {
        "CWE-79": {
            "id": "CWE-79",
            "name": "Improper Neutralization of Input During Web Page
Generation ('Cross-site Scripting')",
            "description": "The software does not neutralize or incorrectly
neutralizes user-controllable input before it is placed in output that is used
as a web page that is served to other users. This allows attackers to inject
malicious scripts that execute in victims' browsers.",
            "mitigations": [
                "Validate all input on the server side using whitelist
validation",
                "Encode all output data before rendering in HTML contexts",
                "Use Content Security Policy (CSP) headers to restrict script
execution",
                "Implement proper session management and use HTTP-only
cookies"
            ],
            "examples": [
                "<script>alert('XSS')</script>",
                "javascript:alert(document.cookie)",
                "<img src=x onerror=alert('XSS')>"
            ],
            "relationships": [("CWE-20", "Parent"), ("CWE-116", "Related")],
        },
        "CWE-89": {
            "id": "CWE-89",
            "name": "Improper Neutralization of Special Elements used in an
SQL Command ('SQL Injection')",
            "description": "The software constructs all or part of an SQL
command using externally-influenced input from an upstream component, but it
does not neutralize or incorrectly neutralizes special elements that could
modify the intended SQL command when it is sent to a downstream component.",
            "mitigations": [
                "Use parameterized queries or prepared statements",
                "Validate input using whitelist validation techniques",
                "Apply principle of least privilege to database accounts",
                "Use stored procedures with proper parameter handling"
            ],
            "examples": [
                "SELECT * FROM users WHERE name = '' OR '1'='1' --'",
                "'; DROP TABLE users; --",
                "UNION SELECT username, password FROM admin_users"
            ],
            "relationships": [("CWE-20", "Parent"), ("CWE-78", "Related")],
        },
        "CWE-20": {
            "id": "CWE-20",
            "name": "Improper Input Validation",
            "description": "The product receives input or data, but it does
not validate or incorrectly validates that the input has the properties that
are required to process the data safely and correctly.",
            "mitigations": [
                "Implement comprehensive input validation using whitelist
approach",
                "Use regular expressions for format validation",
                "Validate input length, type, and range",
                "Reject invalid input rather than attempting to sanitize"
            ],
            "examples": [
                "Unchecked file upload allowing executable files",
                "Missing validation on numeric inputs causing buffer
overflows",
                "Accepting unvalidated URLs leading to SSRF attacks"
            ],
            "relationships": [("CWE-707", "Parent"), ("CWE-79", "Child")],
        },
        "CWE-22": {
            "id": "CWE-22",
            "name": "Improper Limitation of a Pathname to a Restricted
Directory ('Path Traversal')",
            "description": "The software uses external input to construct a
pathname that is intended to identify a file or directory that is located
underneath a restricted parent directory, but the software does not properly
neutralize special elements within the pathname that can cause the pathname to
resolve to a location that is outside of the restricted directory.",
            "mitigations": [
                "Use a whitelist of allowed file names and paths",
                "Validate file paths against a restricted directory
structure",
                "Use canonical path validation to resolve symbolic links",
                "Implement proper access controls and sandboxing"
            ],
            "examples": [
                "../../../etc/passwd",
                "..\\..\\..\\windows\\system32\\config\\sam",
                "....//....//etc/passwd"
            ],
            "relationships": [("CWE-20", "Parent"), ("CWE-59", "Related")],
        },
        "CWE-119": {
            "id": "CWE-119",
            "name": "Improper Restriction of Operations within the Bounds of a Memory Buffer",
            "description": "The software performs operations on a memory buffer, but it can read from or write to a memory location that is outside of the intended boundary of the buffer.",
            "mitigations": [
                "Use memory-safe programming languages",
                "Implement bounds checking for all buffer operations",
                "Use static analysis tools to detect buffer overflows",
                "Enable compiler protections like stack canaries and ASLR"
            ],
            "examples": [
                "strcpy(dest, source) without bounds checking",
                "Array access with unchecked index values",
                "Buffer allocation without proper size calculation"
            ],
            "relationships": [("CWE-664", "Parent"), ("CWE-120", "Child")],
        },
        "CWE-200": {
            "id": "CWE-200",
            "name": "Exposure of Sensitive Information to an Unauthorized Actor",
            "description": "The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information.",
            "mitigations": [
                "Implement proper access controls and authentication",
                "Use encryption for sensitive data at rest and in transit",
                "Remove or mask sensitive data from error messages",
                "Implement proper logging without exposing sensitive information"
            ],
            "examples": [
                "Database errors revealing table structure",
                "Debug information exposed in production",
                "Sensitive data in HTTP response headers"
            ],
```

```python
            "relationships": [("CWE-668", "Parent"), ("CWE-209", "Child")],
        },
        "CWE-287": {
            "id": "CWE-287",
            "name": "Improper Authentication",
            "description": "When an actor claims to have a given identity, the
software does not prove or insufficiently proves that the claim is correct.",
            "mitigations": [
                "Implement multi-factor authentication",
                "Use strong password policies and secure storage",
                "Implement proper session management",
                "Use secure authentication protocols and frameworks"
            ],
            "examples": [
                "Missing password verification in login functions",
                "Weak password reset mechanisms",
                "Authentication bypass through parameter manipulation"
            ],
            "relationships": [("CWE-284", "Parent"), ("CWE-798", "Related")],
        },
        "CWE-120": {
            "id": "CWE-120",
            "name": "Buffer Copy without Checking Size of Input ('Classic
Buffer Overflow')",
            "description": "The program copies an input buffer to an output
buffer without verifying that the size of the input buffer is less than the
size of the output buffer, leading to a buffer overflow.",
            "mitigations": [
                "Use safe string functions like strncpy, strlcpy",
                "Implement input length validation before copying",
                "Use bounds-checking compilers and tools",
                "Consider using memory-safe languages"
            ],
            "examples": [
                "strcpy(buffer, user_input) without size check",
                "gets() function reading unlimited input",
                "sprintf() without format string validation"
            ],
            "relationships": [("CWE-119", "Parent"), ("CWE-131", "Related")],
        }
    }

@lru_cache(maxsize=128)
def get_cwe_data(cwe_id):
    #Main way to get data for a given CWE id.
    #Uses cache → then scraping → then fallback
    cached_data = load_cached_cwe_data() if is_cache_valid() else {}

    #Check if we have this CWE in cache
    if cwe_id in cached_data:
        return cached_data[cwe_id]

    #Try to scrape from MITRE
    print(f"Fetching CWE data for {cwe_id}...")
    scraped_data = scrape_cwe_detail(cwe_id)

    if scraped_data:
        #Save to cache
        cached_data[cwe_id] = scraped_data
        save_cwe_data_to_cache(cached_data)
        return scraped_data

    #Fallback to enhanced static data
    fallback_data = get_enhanced_fallback_data()
    if cwe_id in fallback_data:
        return fallback_data[cwe_id]

    #Last resort: minimal data
    cwe_number = cwe_id.replace('CWE-', '') if cwe_id.startswith('CWE-') else
cwe_id
    return {
        'id': cwe_id,
        'name': f"Common Weakness Enumeration {cwe_number}",
        'description': f"{cwe_id} is a software weakness category. Visit
https://cwe.mitre.org/data/definitions/{cwe_number}.html for official
details.",
        'mitigations': ["Follow secure coding practices", "Implement proper
input validation", "Use security testing tools"],
        'examples': ["Refer to official CWE documentation for specific
examples"],
        'relationships': [],
    }

def get_cwe_dict():
    #Returns a dict mapping CWE-ID to data; combines fallback, cache, and
static mappings
    enhanced_data = get_enhanced_fallback_data()

    #Load cached scraped data if available
    if is_cache_valid():
        cached_data = load_cached_cwe_data()
        enhanced_data.update(cached_data)
    #Add more CWEs from the CWE_TITLES mapping
    from .cwe_map import CWE_TITLES

    for cwe_id in CWE_TITLES.keys():
        if cwe_id not in enhanced_data:
            enhanced_data[cwe_id] = {
                'id': cwe_id,
                'name': CWE_TITLES[cwe_id],
                'description': f"{cwe_id} ({CWE_TITLES[cwe_id]}) is a software
weakness category. Click to fetch detailed information from MITRE.",
                'mitigations': ["Loading detailed information..."],
                'examples': ["Detailed examples will be loaded from MITRE"],
                'relationships': [],
            }

    return enhanced_data

def refresh_cwe_cache():
    #Update the cache for "important/common" CWE IDs
    from .cwe_map import CWE_TITLES

    print("Refreshing CWE cache...")
    cached_data = load_cached_cwe_data() if is_cache_valid() else {}

    #Start by refreshing the most popular/common CWEs
    priority_cwes = ['CWE-79', 'CWE-89', 'CWE-20', 'CWE-22', 'CWE-119', 'CWE-
200', 'CWE-287', 'CWE-120']

    for cwe_id in priority_cwes:
        #If not cached fresh, scrape and cache it anew
        if cwe_id not in cached_data or time.time() -
cached_data[cwe_id].get('scraped_at', 0) > (CACHE_EXPIRY_HOURS * 3600):
            scraped = scrape_cwe_detail(cwe_id)
            if scraped:
                cached_data[cwe_id] = scraped
                time.sleep(1)

    save_cwe_data_to_cache(cached_data)
    print(f"CWE cache refreshed with {len(cached_data)} entries")
```