Task 2

Web Application Vulnerability Scanner

CODE:-

```
    import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin

def fetch_url(url):
    """Fetches the content of a given URL."""
    try:
        response = requests.get(url)
        response.raise_for_status() # Raise an HTTPError for bad responses (4xx or 5xx)
        return response
    except requests.exceptions.RequestException as e:
        print(f"Error fetching {url}: {e}")
        return None

def parse_html(html_content):
    """Parses HTML content using BeautifulSoup."""
    return BeautifulSoup(html_content, 'html.parser')

def find_forms(soup, base_url):
    """Finds forms in the HTML and extracts relevant information."""
    forms = []
    for form in soup.find_all('form'):
        form_details = {}
        # Get the form action URL (handle relative URLs)
```

```
        action = form.get('action')
        form_details['action'] = urljoin(base_url, action) if action else base_url

        # Get the form method (GET or POST)
        form_details['method'] = form.get('method', 'get').lower()

        # Find input fields
        inputs = []
        for input_tag in form.find_all(['input', 'textarea', 'select']):
            input_details = {
                'type': input_tag.get('type', 'text'),
                'name': input_tag.get('name'),
                'value': input_tag.get('value', '')
            }
            inputs.append(input_details)
        form_details['inputs'] = inputs

        forms.append(form_details)
    return forms

# --- Basic Vulnerability Checking (Conceptual) ---
# This is where you would add checks for specific vulnerabilities.
# This is highly simplified and for demonstration only.

def check_sql_injection(url, forms):
    """
    Conceptual function to check for basic SQL injection points.
    This is a placeholder and requires much more sophisticated logic.
    """
```

```
    print(f"\n[*] Checking {url} for potential SQL Injection...")
    # A real check would involve injecting various payloads into form inputs
    # and URL parameters and analyzing the responses (e.g., error messages,
    # changes in content). This is a complex process.

    # Example: Check URL parameters (very basic)
    if '?' in url:
        print("    Potential GET parameters found. Needs further testing.")

    # Example: Indicate forms need testing
    if forms:
        print(f"    Found {len(forms)} forms. Each form input needs testing.")

def check_xss(url, response, forms):
    """
    Conceptual function to check for basic Cross-Site Scripting (XSS).
    This is a placeholder and requires much more sophisticated logic.
    """
    print(f"\n[*] Checking {url} for potential XSS...")
    # A real check would involve injecting XSS payloads into various parts
    # of the request (URL, headers, form inputs) and seeing if they are
    # reflected in the response and executed by the browser.

    # Example: Check if simple script tags are reflected (very basic and unreliable)
    if response and "<script>" in response.text.lower():
        print("    Simple script tag found in response. Could indicate reflection, but needs verification.")

    # Example: Indicate forms need testing for reflected/stored XSS
    if forms:
```

```
        print(f"    Found {len(forms)} forms. Inputs need testing for XSS.")


# --- Main Scanner Logic ---

def scan_website(target_url):
    """Basic scanner function."""
    print(f"[*] Starting scan of: {target_url}")

    # 1. Fetch the initial page
    response = fetch_url(target_url)
    if not response:
        return

    # 2. Parse the HTML
    soup = parse_html(response.text)

    # 3. Find forms (common injection points)
    forms = find_forms(soup, target_url)
    if forms:
        print(f"[*] Found {len()} forms on {target_url}")
        # You would typically iterate through forms and their inputs here
        # to perform vulnerability checks.

    # 4. Perform basic conceptual vulnerability checks
    check_sql_injection(target_url, forms)
    check_xss(target_url, response, forms)

    # You would expand this to crawl links, test different pages,
```

```
    # and implement specific checks for various vulnerability types (CSRF, file upload, etc.)


if __name__ == "__main__":
    # Replace with the target URL you have permission to scan
    target_website = "http://testphp.vulnweb.com/" # Example vulnerable website for testing ONLY
    scan_website(target_website)
```