**Assignment 1**

Web security auditing is a critical process for ensuring the safety and integrity of web applications and the data they handle. In today's interconnected world, websites are prime targets for cyberattacks, ranging from data breaches and defacement to denial of service and malware distribution. A web security audit is a systematic evaluation designed to identify vulnerabilities and weaknesses in a website's security posture, allowing developers and administrators to proactively address them before they can be exploited by malicious actors.

The Python code we've discussed performs a basic form of automated web security auditing. Its focus is on identifying common, easily detectable security misconfigurations and weaknesses. The underlying theory behind each check stems from established web security principles and common attack vectors.

Security Headers: HTTP security headers are directives sent by the web server to the browser, instructing it on how to behave to enhance security. They are a crucial "defense-in-depth" mechanism. For example, X-Frame-Options aims to prevent clickjacking attacks by controlling whether a webpage can be embedded within an <iframe> on another site. Content-Security-Policy (CSP) is a powerful header that mitigates Cross-Site Scripting (XSS) by defining a whitelist of sources from which the browser is allowed to load resources like scripts, stylesheets, and images. X-Content-Type-Options prevents MIME-sniffing attacks, where browsers might misinterpret file types, potentially executing malicious code disguised as a different file. Strict-Transport-Security (HSTS) enforces HTTPS, ensuring that browsers only connect to the website over secure connections after the first visit, preventing downgrade attacks. Referrer-Policy and Permissions-Policy (or older Feature-Policy) control the amount of information shared in HTTP referer headers and restrict access to browser features, respectively, enhancing privacy and mitigating certain information leakage and attack vectors. The absence of these headers signifies a potential vulnerability or a missed opportunity to bolster security.

HTTPS (Hypertext Transfer Protocol Secure): HTTPS is fundamental for web security. It encrypts communication between the user's browser and the web server using SSL/TLS. This encryption protects sensitive data like passwords, personal information, and financial details from being intercepted and read by eavesdroppers during transmission (Man-in-the-Middle attacks). Ensuring a website uses HTTPS is a basic but vital security measure, especially for websites handling sensitive information or requiring user authentication.

Robots.txt and Sitemap.xml: While not direct vulnerabilities, these files can be valuable for attackers during the information gathering phase. robots.txt instructs search engine crawlers which parts of a website should not be indexed. Disallowed paths in robots.txt can sometimes unintentionally reveal sensitive or administrative areas of a website that an attacker might then target. sitemap.xml provides a structured list of a website's pages, which can help attackers understand the site's structure and potentially identify entry points or interesting content to probe for vulnerabilities. Checking for these files is a part of a broader reconnaissance effort.

CMS (Content Management System) Detection: Identifying the CMS (like WordPress, Joomla, Drupal) or framework used by a website is crucial because specific CMSs and their versions often have known vulnerabilities. Attackers frequently target websites running outdated or vulnerable versions of popular CMSs. Detecting the CMS allows for more targeted vulnerability research and exploitation attempts. Our basic code uses simple heuristics like looking for common file paths or headers associated with specific CMSs.

Cookie Flags (HttpOnly and Secure): Cookies are small pieces of data stored in the user's browser by websites. HttpOnly and Secure flags enhance cookie security. HttpOnly prevents client-side scripts (like JavaScript) from accessing the cookie, mitigating XSS attacks that might attempt to steal cookies. The Secure flag ensures that the cookie is only transmitted over HTTPS connections, preventing interception over insecure HTTP. Checking for these flags helps assess the security of cookie handling.

Basic Port Scanning: Port scanning is a technique to determine which network ports are open on a server. Open ports indicate services running on the server. Common ports like 80 (HTTP), 443 (HTTPS), 21 (FTP), and 22 (SSH) are often targets. While not a vulnerability in itself, open ports represent potential attack vectors. For instance, an outdated FTP or SSH service might have known vulnerabilities. Basic port scanning is a preliminary step in understanding the attack surface of a web server.

In conclusion, the basic web security audit performed by the Python code is grounded in fundamental web security principles. It aims to quickly identify easily rectifiable security weaknesses related to headers, HTTPS, information disclosure, technology identification, and basic network service exposure. However, it's crucial to remember that this is only a starting point. Comprehensive web security auditing requires deeper, more sophisticated techniques, including vulnerability scanning, penetration testing, and manual code review to uncover more complex vulnerabilities and ensure robust security.

**Code:**

import requests

import urllib.parse

import socket

import ssl

from bs4 import BeautifulSoup  # Optional, for more advanced content analysis later

def check\_security\_headers(url):

    """Checks for common security headers."""

    try:

        response = requests.get(url, verify=True, timeout=10)  # verify=True for SSL cert validation

        headers = response.headers

        print("\n--- Security Header Checks ---")

        security\_headers = {

            "X-Frame-Options": "Clickjacking Protection",

            "Content-Security-Policy": "Cross-Site Scripting (XSS) Prevention",

            "X-Content-Type-Options": "MIME-Sniffing Protection",

            "Strict-Transport-Security": "HTTPS Enforcement (HSTS)",

            "Referrer-Policy": "Referrer Information Control",

            "Permissions-Policy": "Browser Feature Control (Modern)", # Or Feature-Policy (older)

        }

        for header, description in security\_headers.items():

            if header in headers:

                print(f"[+] {header}: Present - {description}")

                header\_value = headers.get(header)

                if header == "Content-Security-Policy":

                    print(f"    Value: {header\_value}") # Consider further analysis of CSP value

                elif header == "Strict-Transport-Security":

                    print(f"    Value: {header\_value}") # Consider checking max-age, includeSubDomains

            else:

                print(f"[-] {header}: Missing - {description} (Potential vulnerability)")

    except requests.exceptions.RequestException as e:

        print(f"Error checking headers: {e}")

    except ssl.SSLError as e:

        print(f"SSL Certificate Error: {e}")

def check\_https(url):

    """Checks if the website uses HTTPS."""

    parsed\_url = urllib.parse.urlparse(url)

    if parsed\_url.scheme != "https":

        print("\n--- HTTPS Check ---")

        print(f"[-] Website does not use HTTPS by default. Consider enforcing HTTPS redirection.")

    else:

        print("\n--- HTTPS Check ---")

        print(f"[+] Website uses HTTPS.")

def check\_robots\_txt\_sitemap(url):

    """Checks for robots.txt and sitemap.xml."""

    print("\n--- Robots.txt and Sitemap Checks ---")

    files\_to\_check = ["robots.txt", "sitemap.xml"]

    for file in files\_to\_check:

        file\_url = urllib.parse.urljoin(url, file)

        try:

            response = requests.get(file\_url, verify=True, timeout=5)

            if response.status\_code == 200:

                print(f"[+] {file}: Found at {file\_url}")

                if file == "robots.txt":

                    # You could potentially parse robots.txt to look for disallowed paths (info gathering)

                    pass

                elif file == "sitemap.xml":

                    # You could parse sitemap.xml to understand site structure (info gathering)

                    pass

            elif response.status\_code == 404:

                print(f"[-] {file}: Not found (This is not necessarily a vulnerability)")

            else:

                print(f"[!] {file}: Status code {response.status\_code} - Investigate.")

        except requests.exceptions.RequestException as e:

            print(f"Error checking {file}: {e}")

def basic\_cms\_detection(url):

    """Basic CMS/Framework detection (very rudimentary)."""

    print("\n--- Basic CMS/Framework Detection ---")

    common\_indicators = {

        "WordPress": ["wp-content", "wp-admin", "wordpress"],

        "Joomla": ["/administrator/", "joomla"],

        "Drupal": ["/sites/default/", "drupal"],

        "Magento": ["/skin/frontend/", "/js/mage/", "magento"],

        "PHP": [".php"], # Very broad, PHP is widely used

        "ASP.NET": [".aspx", ".cshtml"], # Also broad

        # Add more CMS/framework indicators as needed

    }

    try:

        response = requests.get(url, verify=True, timeout=5)

        if response.status\_code == 200:

            content = response.text.lower()

            headers = response.headers

            detected\_cms = []

            for cms, indicators in common\_indicators.items():

                for indicator in indicators:

                    if indicator in content.lower() or indicator in str(headers).lower() : # Check headers too

                        detected\_cms.append(cms)

                        break # Avoid adding same CMS multiple times if multiple indicators found

            if detected\_cms:

                print(f"[+] Potential CMS/Framework detected: {', '.join(detected\_cms)}")

            else:

                print("[-] CMS/Framework detection inconclusive (basic checks).")

        else:

            print(f"[!] Could not fetch website for CMS detection (Status code: {response.status\_code})")

    except requests.exceptions.RequestException as e:

        print(f"Error during CMS detection: {e}")

def check\_cookies\_flags(url):

    """Checks for HttpOnly and Secure flags on cookies."""

    try:

        response = requests.get(url, verify=True, timeout=5)

        cookies = response.cookies

        print("\n--- Cookie Flag Checks ---")

        if cookies:

            for cookie in cookies:

                print(f"Cookie: {cookie.name}")

                if cookie.has\_nonstandard\_attr('httponly'): # Use has\_nonstandard\_attr for httponly

                    print(f"  [+] HttpOnly Flag: Present (Good)")

                else:

                    print(f"  [-] HttpOnly Flag: Missing (Consider adding)")

                if cookie.secure:

                    print(f"  [+] Secure Flag: Present (Good - for HTTPS)")

                else:

                    print(f"  [-] Secure Flag: Missing (Consider adding, especially over HTTPS)")

        else:

            print("[-] No cookies set by the website.")

    except requests.exceptions.RequestException as e:

        print(f"Error checking cookies: {e}")

def basic\_port\_scan(url):

    """Basic TCP port scan for common ports (limited and rudimentary)."""

    print("\n--- Basic Port Scan (Limited) ---")

    parsed\_url = urllib.parse.urlparse(url)

    hostname = parsed\_url.netloc or parsed\_url.path # Handle cases with just domain or full URL

    if ":" in hostname: # Remove port if present in URL

        hostname = hostname.split(":")[0]

    common\_ports = [80, 443, 21, 22, 25] # HTTP, HTTPS, FTP, SSH, SMTP

    try:

        ip\_address = socket.gethostbyname(hostname)

        print(f"Scanning host: {hostname} ({ip\_address}) for common ports...")

        for port in common\_ports:

            sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

            sock.settimeout(2) # Short timeout for each port

            try:

                result = sock.connect\_ex((ip\_address, port))

                if result == 0:

                    print(f"  [+] Port {port}: Open")

                else:

                    print(f"  [-] Port {port}: Closed/Filtered") # Could be filtered, not necessarily closed

            except Exception as e:

                print(f"  [!] Error checking port {port}: {e}")

            finally:

                sock.close()

    except socket.gaierror:

        print(f"[-] Could not resolve hostname: {hostname}")

    except Exception as e:

        print(f"Error during port scan: {e}")

def web\_security\_audit(url\_to\_audit):

    """Performs basic web security audit checks."""

    print(f"Starting basic security audit for: {url\_to\_audit}")

    if not url\_to\_audit.startswith("http://") and not url\_to\_audit.startswith("https://"):

        url\_to\_audit = "https://" + url\_to\_audit  # Default to HTTPS if no scheme provided

    check\_https(url\_to\_audit)

    check\_security\_headers(url\_to\_audit)

    check\_robots\_txt\_sitemap(url\_to\_audit)

    basic\_cms\_detection(url\_to\_audit)

    check\_cookies\_flags(url\_to\_audit)

    basic\_port\_scan(url\_to\_audit)

    print("\n--- Audit Finished ---")

    print("Note: This is a basic automated audit. Further manual testing and analysis are recommended.")

if \_\_name\_\_ == "\_\_main\_\_":

    target\_url = input("Enter the URL to audit (e.g., https://example.com): ")

    if target\_url:

        web\_security\_audit(target\_url)

    else:

        print("Please provide a URL to audit.")

**Output**

**PS D:\Data\Sem 8\Cyber\_Security> python -u "d:\Data\Sem 8\Cyber\_Security\prac1.py"**

**Enter the URL to audit (e.g., https://example.com): https://www.google.com/url?sa=E&q=https%3A%2F%2Fowasp.org%2Fwww-project-webgoat%2F**

**Starting basic security audit for: https://www.google.com/url?sa=E&q=https%3A%2F%2Fowasp.org%2Fwww-project-webgoat%2F**

**--- HTTPS Check ---**

**[+] Website uses HTTPS.**

**--- Security Header Checks ---**

**[-] X-Frame-Options: Missing - Clickjacking Protection (Potential vulnerability)**

**[+] Content-Security-Policy: Present - Cross-Site Scripting (XSS) Prevention**

**Value: object-src 'none';base-uri 'self';script-src 'nonce-RK7SSRIREXgnvBaBrmNpjw' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https://csp.withgoogle.com/csp/gws/other**

**[-] X-Content-Type-Options: Missing - MIME-Sniffing Protection (Potential vulnerability)**

**[-] Strict-Transport-Security: Missing - HTTPS Enforcement (HSTS) (Potential vulnerability)**

**[-] Referrer-Policy: Missing - Referrer Information Control (Potential vulnerability)**

**[-] Permissions-Policy: Missing - Browser Feature Control (Modern) (Potential vulnerability)**

**--- Robots.txt and Sitemap Checks ---**

**[+] robots.txt: Found at https://www.google.com/robots.txt**

**[+] sitemap.xml: Found at https://www.google.com/sitemap.xml**

**--- Basic CMS/Framework Detection ---**

**[-] CMS/Framework detection inconclusive (basic checks).**

**--- Cookie Flag Checks ---**

**Cookie: NID**

**[-] HttpOnly Flag: Missing (Consider adding)**

**[-] Secure Flag: Missing (Consider adding, especially over HTTPS)**

**--- Basic Port Scan (Limited) ---**

**Scanning host: www.google.com (142.250.183.196) for common ports...**

**[+] Port 80: Open**

**[+] Port 443: Open**

**[-] Port 21: Closed/Filtered**

**[-] Port 22: Closed/Filtered**

**[-] Port 25: Closed/Filtered**

**--- Audit Finished ---**

**Note: This is a basic automated audit. Further manual testing and analysis are recommended.**

**PS D:\Data\Sem 8\Cyber\_Security> python -u "d:\Data\Sem 8\Cyber\_Security\prac1.py"**

**Enter the URL to audit (e.g., https://example.com): https://www.google.com/url?sa=E&q=http%3A%2F%2Fwww.itsecgames.com%2F**

**Starting basic security audit for: https://www.google.com/url?sa=E&q=http%3A%2F%2Fwww.itsecgames.com%2F**

**--- HTTPS Check ---**

**[+] Website uses HTTPS.**

**--- Security Header Checks ---**

**[-] X-Frame-Options: Missing - Clickjacking Protection (Potential vulnerability)**

**[+] Content-Security-Policy: Present - Cross-Site Scripting (XSS) Prevention**

**Value: object-src 'none';base-uri 'self';script-src 'nonce-3q7G63\_wRPbwowUMSy31ew' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https://csp.withgoogle.com/csp/gws/other**

**[-] X-Content-Type-Options: Missing - MIME-Sniffing Protection (Potential vulnerability)**

**[-] Strict-Transport-Security: Missing - HTTPS Enforcement (HSTS) (Potential vulnerability)**

**[-] Referrer-Policy: Missing - Referrer Information Control (Potential vulnerability)**

**[-] Permissions-Policy: Missing - Browser Feature Control (Modern) (Potential vulnerability)**

**--- Robots.txt and Sitemap Checks ---**

**[+] robots.txt: Found at https://www.google.com/robots.txt**

**[+] sitemap.xml: Found at https://www.google.com/sitemap.xml**

**--- Basic CMS/Framework Detection ---**

**[-] CMS/Framework detection inconclusive (basic checks).**

**--- Cookie Flag Checks ---**

**Cookie: NID**

**[-] HttpOnly Flag: Missing (Consider adding)**

**[-] Secure Flag: Missing (Consider adding, especially over HTTPS)**

**--- Basic Port Scan (Limited) ---**

**Scanning host: www.google.com (142.250.183.196) for common ports...**

**[+] Port 80: Open**

**[+] Port 443: Open**

**[-] Port 21: Closed/Filtered**

**[-] Port 22: Closed/Filtered**

**[-] Port 25: Closed/Filtered**

**--- Audit Finished ---**

**Note: This is a basic automated audit. Further manual testing and analysis are recommended.**

**PS D:\Data\Sem 8\Cyber\_Security> python -u "d:\Data\Sem 8\Cyber\_Security\prac1.py"**

**Enter the URL to audit (e.g., https://example.com): https://www.google.com/url?sa=E&q=http%3A%2F%2Ftestphp.vulnweb.com%2F**

**Starting basic security audit for: https://www.google.com/url?sa=E&q=http%3A%2F%2Ftestphp.vulnweb.com%2F**

**--- HTTPS Check ---**

**[+] Website uses HTTPS.**

**--- Security Header Checks ---**

**[-] X-Frame-Options: Missing - Clickjacking Protection (Potential vulnerability)**

**[+] Content-Security-Policy: Present - Cross-Site Scripting (XSS) Prevention**

**Value: object-src 'none';base-uri 'self';script-src 'nonce-3sndMQnM\_2dQ-SBGsKmP\_g' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https://csp.withgoogle.com/csp/gws/other**

**[-] X-Content-Type-Options: Missing - MIME-Sniffing Protection (Potential vulnerability)**

**[-] Strict-Transport-Security: Missing - HTTPS Enforcement (HSTS) (Potential vulnerability)**

**[-] Referrer-Policy: Missing - Referrer Information Control (Potential vulnerability)**

**[-] Permissions-Policy: Missing - Browser Feature Control (Modern) (Potential vulnerability)**

**--- Robots.txt and Sitemap Checks ---**

**[+] robots.txt: Found at https://www.google.com/robots.txt**

**[+] sitemap.xml: Found at https://www.google.com/sitemap.xml**

**--- Basic CMS/Framework Detection ---**

**[-] CMS/Framework detection inconclusive (basic checks).**

**--- Cookie Flag Checks ---**

**Cookie: NID**

**[-] HttpOnly Flag: Missing (Consider adding)**

**[-] Secure Flag: Missing (Consider adding, especially over HTTPS)**

**--- Basic Port Scan (Limited) ---**

**Scanning host: www.google.com (142.250.67.132) for common ports...**

**[+] Port 80: Open**

**[+] Port 443: Open**

**[-] Port 21: Closed/Filtered**

**[-] Port 22: Closed/Filtered**

**[-] Port 25: Closed/Filtered**

**--- Audit Finished ---**

**Note: This is a basic automated audit. Further manual testing and analysis are recommended.**

Output