**Informe Técnico**

**Analizador de Cabeceras HTTP con Python – Lección 1**

**1. Escenario**

Durante las fases de **reconocimiento en pruebas de penetración (recon)** es esencial obtener información del servidor objetivo. Una técnica común es analizar las **cabeceras HTTP** devueltas por el servidor al realizar una petición, ya que revelan tecnologías, configuraciones de seguridad y posibles vectores de ataque. El objetivo del laboratorio fue desarrollar un **script en Python** que automatice este proceso y muestre de forma destacada los encabezados más relevantes.

**2. Metodología**

La actividad se llevó a cabo aplicando las siguientes etapas:

1. **Lenguaje de programación:** Python 3.x.
2. **Dependencias:** librería requests para realizar peticiones HTTP.
3. **Lógica implementada:**
   * Validación de la URL ingresada por el usuario.
   * Ejecución de petición GET.
   * Impresión en consola de todas las cabeceras recibidas.
   * Identificación destacada de headers críticos: Server, X-Powered-By, Content-Type, Set-Cookie, Strict-Transport-Security.
   * Advertencias si la respuesta no es 200 (OK).

**3. Código Implementado (header\_analyzer.py)**

import requests

def analyze\_headers(url):

# Verificar esquema

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

url = "http://" + url

try:

response = requests.get(url)

print(f"\n[+] Analizando cabeceras de: {url}")

print(f"[+] Código de respuesta: {response.status\_code}\n")

if response.status\_code != 200:

print("[!] Advertencia: el servidor no devolvió estado 200 (OK)\n")

# Listar todas las cabeceras

for header, value in response.headers.items():

print(f"{header}: {value}")

# Destacar cabeceras importantes

print("\n=== Cabeceras destacadas ===")

for key in ["Server", "X-Powered-By", "Content-Type",

"Set-Cookie", "Strict-Transport-Security"]:

if key in response.headers:

print(f"[+] {key}: {response.headers[key]}")

else:

print(f"[-] {key}: No presente")

except Exception as e:

print(f"[!] Error al conectar: {e}")

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

target = input("Ingrese la URL objetivo: ")

analyze\_headers(target)

**4. Ejemplo de Ejecución**

Ingrese la URL objetivo: www.ejemplo.com

[+] Analizando cabeceras de: http://www.ejemplo.com

[+] Código de respuesta: 200

Date: Mon, 26 Aug 2024 22:00:00 GMT

Server: Apache

Content-Type: text/html; charset=UTF-8

Set-Cookie: sessionid=xyz123; HttpOnly

=== Cabeceras destacadas ===

[+] Server: Apache

[-] X-Powered-By: No presente

[+] Content-Type: text/html; charset=UTF-8

[+] Set-Cookie: sessionid=xyz123; HttpOnly

[-] Strict-Transport-Security: No presente

**5. Análisis de Seguridad**

* **Cabecera Server:** identifica el software del servidor (ej. Apache, Nginx). Información útil para fingerprinting.
* **X-Powered-By:** revela tecnologías usadas (PHP, ASP.NET), lo cual puede exponer vectores de ataque.
* **Content-Type:** importante para prevenir ataques de *MIME sniffing*.
* **Set-Cookie:** permite validar si se aplican flags de seguridad (HttpOnly, Secure, SameSite).
* **Strict-Transport-Security (HSTS):** fundamental para forzar uso de HTTPS.

**6. Recomendaciones de Ciberseguridad**

1. Configurar el servidor para **ocultar o limitar cabeceras sensibles** (Server, X-Powered-By).
2. Asegurar que las cookies incluyan **HttpOnly, Secure y SameSite**.
3. Implementar **Strict-Transport-Security (HSTS)** en entornos productivos.
4. Monitorear periódicamente cabeceras con herramientas propias y automatizadas.
5. Integrar este script como parte del flujo de **auditorías internas de seguridad**.

**7. Reflexión**

Este laboratorio demostró que cabeceras HTTP, aunque invisibles para el usuario, pueden filtrar información sensible de un servidor. La automatización con Python agiliza el proceso de reconocimiento y refuerza la importancia de aplicar el principio de **seguridad por defecto** en la configuración del servidor.

**8. Conclusión**

El script implementado permitió obtener, analizar y destacar cabeceras HTTP críticas de forma automatizada. Con ello, se comprobó la utilidad de esta técnica en fases de **reconocimiento de pentesting** y se evidenció la necesidad de aplicar configuraciones seguras en servidores web. Este ejercicio refuerza que la **seguridad no depende solo del código de aplicación**, sino también de la correcta configuración de los servicios que la soportan.