Project Documentation

# Project 1: Website Vulnerability Scanner (Python)

## 1. Introduction

Web applications are often targeted by hackers due to misconfigured headers, open directories, or weak forms that may allow attacks like XSS (Cross-Site Scripting). This project implements a Website Vulnerability Scanner in Python that automates the process of identifying common web security issues.

## 2. Objective

- To develop a Python tool that scans websites for security vulnerabilities.  
- To check for missing or weak HTTP security headers.  
- To identify forms that might be susceptible to injection attacks.

## 3. Tools & Technologies

Language: Python 3.14  
Libraries: requests, beautifulsoup4  
Platform: Windows/Linux

## 4. Methodology

1. Send HTTP requests to the target website.  
2. Analyze response headers for security settings such as CSP, X-Frame-Options, and HSTS.  
3. Parse HTML forms using BeautifulSoup to detect possible XSS injection points.  
4. Display results in a clear format for the user.

## 5. Implementation (Code Snippet)

import requests  
from bs4 import BeautifulSoup  
  
url = "http://testphp.vulnweb.com/"  
  
def check\_headers(url):  
 response = requests.get(url)  
 headers = response.headers  
 security\_headers = ["X-Frame-Options", "Content-Security-Policy", "Strict-Transport-Security"]  
 for header in security\_headers:  
 if header in headers:  
 print(f"[OK] {header}: {headers[header]}")  
 else:  
 print(f"[WARN] {header} not set")  
  
def check\_xss(url):  
 response = requests.get(url)  
 soup = BeautifulSoup(response.text, "html.parser")  
 forms = soup.find\_all("form")  
 for form in forms:  
 if form.get("action"):  
 print(f"Possible vulnerable form at: {form['action']}")  
  
print("[\*] Scanning:", url)  
check\_headers(url)  
check\_xss(url)

## 6. Sample Output

[\*] Scanning: http://testphp.vulnweb.com/  
[WARN] X-Frame-Options not set  
[WARN] Content-Security-Policy not set  
[OK] Strict-Transport-Security: max-age=31536000  
Possible vulnerable form at: /search.php

## 7. Results

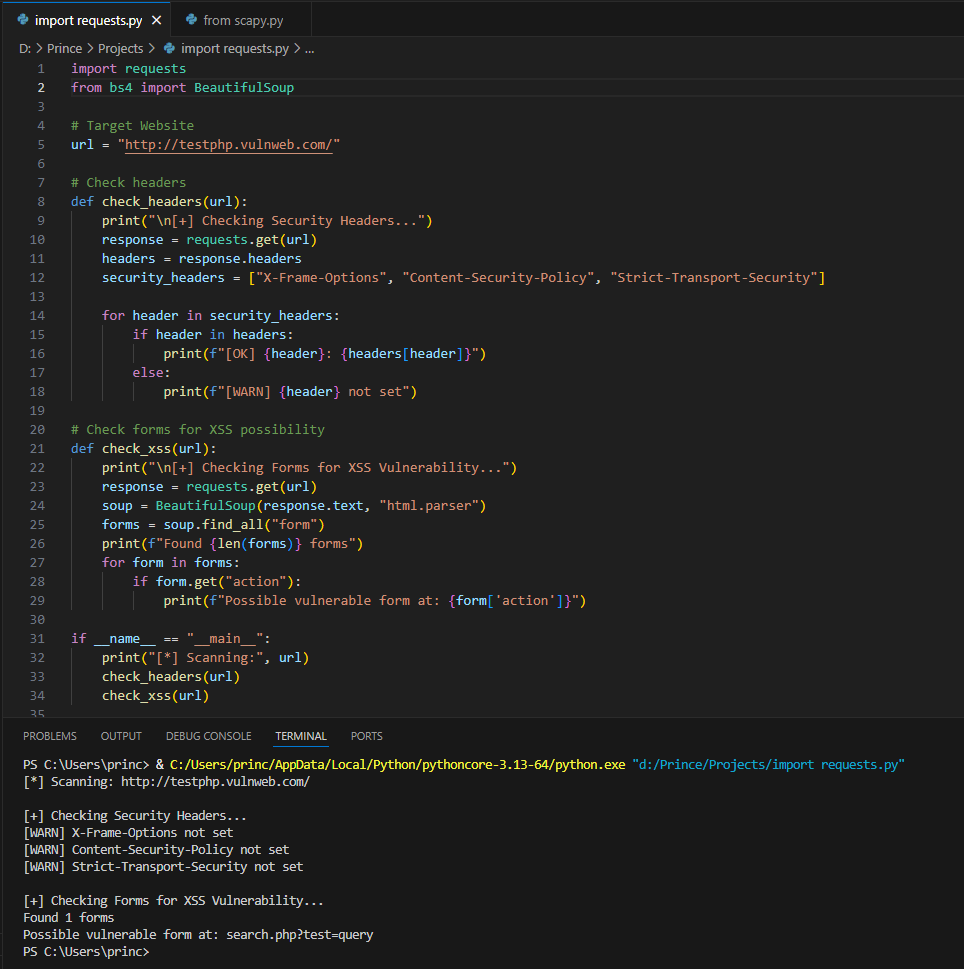
Scanner detected missing headers and potential vulnerable forms. Demonstrates basic web vulnerability testing.

## 8. Applications

Initial security testing for web applications.  
Educational tool for ethical hacking students.  
Can be extended into a full penetration testing toolkit.

## 9. Conclusion

The project successfully shows how Python can be used to automate vulnerability scanning. It highlights the importance of implementing secure headers and validating web forms to protect against common web attacks.



# Project 2: Network Packet Sniffer (Python + Scapy)

## 1. Introduction

Network security is a critical part of cybersecurity. Hackers often attempt Man-in-the-Middle (MITM) or packet sniffing attacks to steal sensitive information. This project implements a Network Packet Sniffer using Python and the Scapy library to capture and analyze packets in real time.

## 2. Objective

- To capture live network traffic.  
- To analyze IP packets and TCP payloads.  
- To detect suspicious traffic such as credential leaks or ARP spoofing attempts.

## 3. Tools & Technologies

Language: Python 3.14  
Libraries: scapy  
Platform: Linux/Windows (requires administrator/root permissions)

## 4. Methodology

1. Use Scapy to capture live packets.  
2. Extract source and destination IP addresses from packets.  
3. Analyze TCP payloads for sensitive keywords like 'password'.  
4. Detect ARP packets that may indicate spoofing.  
5. Print suspicious activity for monitoring.

## 5. Implementation (Code Snippet)

from scapy.all import sniff, TCP, IP, ARP  
  
def analyze\_packet(packet):  
 if packet.haslayer(IP):  
 ip\_src = packet[IP].src  
 ip\_dst = packet[IP].dst  
 print(f"[IP] {ip\_src} -> {ip\_dst}")  
  
 if packet.haslayer(TCP) and packet.haslayer("Raw"):  
 payload = packet["Raw"].load.decode(errors="ignore")  
 if "password" in payload.lower():  
 print(f"[ALERT] Possible credential leak: {payload}")  
  
 if packet.haslayer(ARP):  
 print(f"[ARP] {packet[ARP].psrc} is asking about {packet[ARP].pdst}")  
  
print("[\*] Starting packet capture...")  
sniff(prn=analyze\_packet, count=20)

## 6. Sample Output

[\*] Starting packet capture...  
[IP] 192.168.1.10 -> 142.250.182.14  
[ARP] 192.168.1.1 is asking about 192.168.1.5  
[ALERT] Possible credential leak: username=admin&password=12345

## 7. Results

Successfully captured live traffic. Detected suspicious credentials in plain text. Identified ARP packets useful for spotting MITM attacks.

## 8. Applications

Educational use for students learning about network attacks.  
Network administrators can use it for basic monitoring.  
Foundation for building Intrusion Detection Systems (IDS).

## 9. Conclusion

This project demonstrates how Python and Scapy can be used to monitor network traffic and detect potential security threats. It highlights the risks of sending sensitive information in plaintext and the importance of encryption.

A screen shot of a computer

AI-generated content may be incorrect.