**REPORT ON PHISHING ATTACK AND ANALYSIS**

Phishing Attack and Analysis

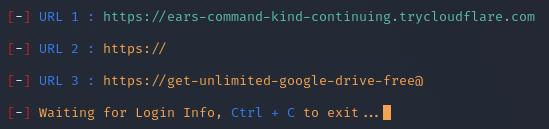
**Introduction**

This report provides a detailed analysis of the identified vulnerabilities related to Broken Authentication and Broken Access Control within the target application. The purpose of this penetration test was to assess the security posture of the application, specifically focusing on the mechanisms that protect user identity, session management, and access restrictions.

First we perform a phishing attack:

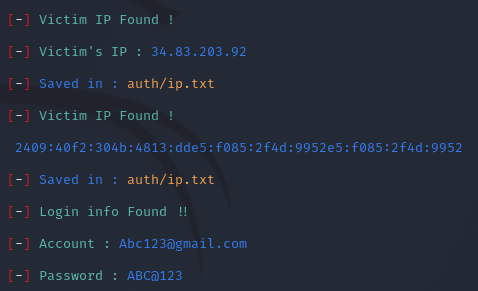
What is Phishing: Phishing is a type of cyberattack where attackers attempt to deceive individuals into revealing sensitive information, such as usernames, passwords, credit card details, or other personal data. This is typically done by impersonating a trusted entity, like a bank, social media platform, or other legitimate organization.

We perform phishing attack using these links:



These Phishing links are generated using a Tool (Zphisher).

Result:



Using these URL we can retrieve information like email address and password.

**Tool built:**

This Python script analyzes a target website's security. It detects Web Application Firewalls (WAF), evaluates HTTP security headers, scans for open ports using Nmap, checks for outdated CMS versions, and assesses safety. Based on findings, it highlights vulnerabilities and recommends improvements to enhance the website's security posture.

**Scope of Testing**

* **Authentication Mechanisms**: To identify weaknesses in the processes that verify the identity of users.
* **Access Control Mechanisms**: To verify the enforcement of permissions and restrictions to prevent unauthorized actions or data access.

**Objectives**

* To uncover flaws that might allow attackers to bypass authentication mechanisms.
* To identify weaknesses that could enable unauthorized users to access restricted functionalities or sensitive data.

**Methodology**

1. **Input Collection**:
   * A text input field accepts the domain (e.g., example.com).
   * Input validation ensures that the field is not empty before proceeding.
2. **Port Scanning**:
   * Used Python's socket library to scan common ports such as 80, 443, 8080, and 8443.
   * Identified and recorded ports that accept connections.
3. **Application Information Gathering**:
   * Sent HTTP and HTTPS requests to each using the requests library.
   * Extracted information including:
     + Server type from the Server HTTP header.
4. **Data Presentation**:
   * Organized output into categories such as:
     + Open Ports.
     + Application URLs.

**Tools Used**

* **Kali Linux**: A Debian-based Linux distribution designed for digital forensics and penetration testing, offering a wide array of security tools.
* **Nmap**: An open-source tool used for network discovery and security auditing, capable of scanning ports, identifying services, and detecting vulnerabilities.

**Code:**

import requests

from bs4 import BeautifulSoup

import os

def detect\_waf(url):

"""Check if the site has a Web Application Firewall (WAF)."""

print("\n[+] Detecting WAF...")

try:

result = os.popen(f"wafw00f {url}").read()

print(result)

if "No WAF detected" in result:

return False

else:

return True

except Exception as e:

print(f"Error detecting WAF: {e}")

return False

def check\_headers(url):

"""Check security-related HTTP headers."""

print("\n[+] Checking HTTP Headers...")

try:

response = requests.get(url, timeout=10)

headers = response.headers

print("Security Headers:")

print("-" \* 30)

for header, value in headers.items():

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

# Check for specific security headers

security\_headers = ['Content-Security-Policy', 'Strict-Transport-Security', 'X-Frame-Options', 'X-XSS-Protection']

missing\_headers = [h for h in security\_headers if h not in headers]

if missing\_headers:

print("\n[!] Missing Security Headers:")

for h in missing\_headers:

print(f" - {h}")

else:

print("\n[+] All important security headers are present.")

return missing\_headers

except Exception as e:

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

return []

def scan\_ports(url):

"""Scan open ports on the target website."""

print("\n[+] Scanning open ports...")

try:

domain = url.split("://")[1].split("/")[0]

result = os.popen(f"nmap -sV {domain}").read()

print(result)

if "443/tcp" in result:

return True

else:

return False

except Exception as e:

print(f"Error scanning ports: {e}")

return False

def check\_cms(url):

"""Check for outdated CMS versions (e.g., WordPress)."""

print("\n[+] Checking for outdated CMS...")

try:

response = requests.get(url, timeout=10)

soup = BeautifulSoup(response.text, 'html.parser')

generator = soup.find('meta', attrs={'name': 'generator'})

if generator:

print(f"CMS Detected: {generator['content']}")

return generator['content']

else:

print("No CMS detected.")

return None

except Exception as e:

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

return None

def is\_website\_safe(waf, headers, https, cms):

"""Determine if the website is safe based on scan results."""

print("\n[+] Is the Website Safe?")

print("-" \* 30)

issues = []

if not waf:

issues.append("No Web Application Firewall (WAF) detected.")

if headers:

issues.append(f"Missing security headers: {', '.join(headers)}.")

if not https:

issues.append("No HTTPS detected; using HTTP only.")

if cms:

issues.append(f"Detected CMS: {cms} (check for vulnerabilities).")

if issues:

print("Current Status: The website has vulnerabilities, making it not entirely safe.")

print("\n[!] Issues Detected:")

for issue in issues:

print(f" - {issue}")

print("\n[+] Next Steps:")

print(" - If this is your website, prioritize fixing the issues listed above.")

print(" - If it's a third-party website, avoid sharing sensitive data with it until these issues are resolved.")

else:

print("Current Status: The website is relatively safe with no critical issues detected.")

print("\n[+] Keep monitoring and maintaining security configurations.")

print("-" \* 30)

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

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

print("\nStarting Website Scan...")

print("=" \* 40)

waf = detect\_waf(target\_url)

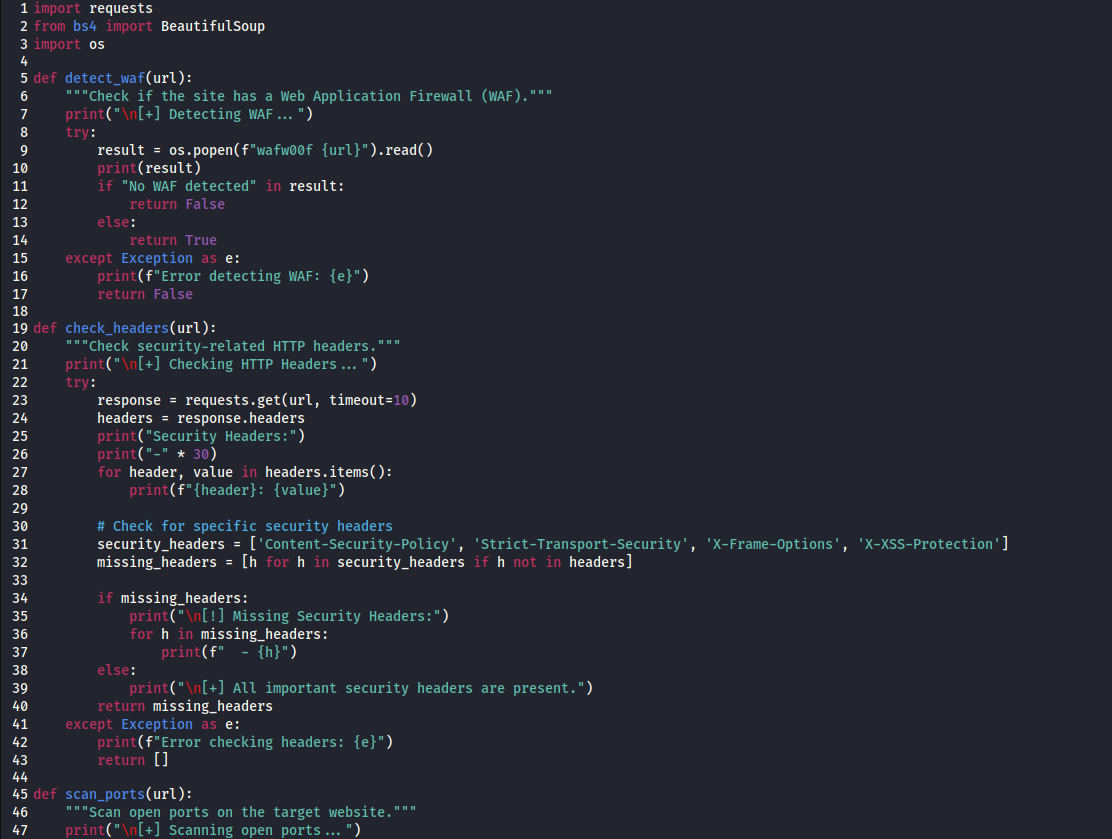
headers = check\_headers(target\_url)

https = scan\_ports(target\_url)

cms = check\_cms(target\_url)

is\_website\_safe(waf, headers, https, cms)

print("\nScan Completed!")

****

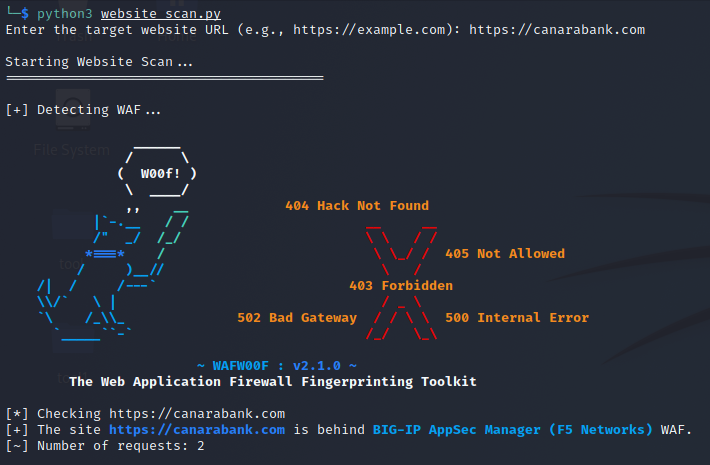
**Step-by-Step Process**

1. Enter the Python file.
2. Enter the domain name into the textbox.
3. Retrieve information about ports and services.
4. Enter the URL copied from the scanner and generate a report using the vulnerability scanner.

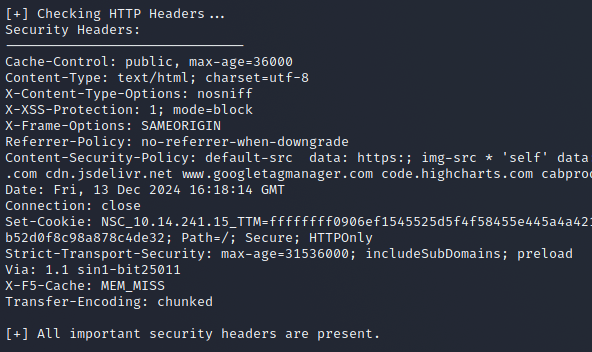
**Results**

**Target: https://canarabank.com**

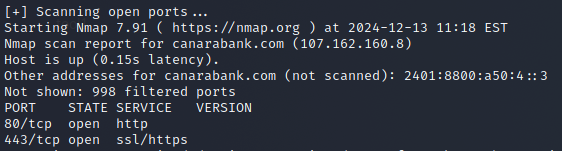
* **Web Application Firewall (WAF)**: Detected BIG-IP AppSec Manager (F5 Networks).
  + A WAF is a security layer designed to protect the website from attacks such as SQL injection or XSS.



* **HTTP Headers**: Security headers found include Cache-Control, Content-Type, X-Content-Type-Options, X-XSS-Protection, X-Frame-Options, Referrer-Policy, CSP, and HSTS.



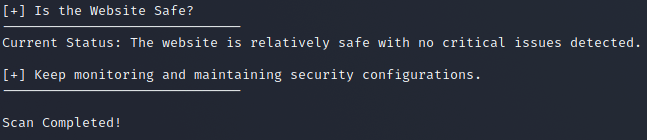
* **Open Ports**: Detected open ports 80/tcp (HTTP) and 443/tcp (HTTPS).



* **Outdated CMS**: No CMS detected.



* **Website Safety**: Relatively safe with no critical issues detected.



**Experiments Performed on:**

**Target: https://lanes-deserve-wholesale-n.trycloudflare.com**

* **Web Application Firewall (WAF)**: Unable to verify WAF due to timeout.
* **HTTP Headers**: Missing critical security headers like CSP, HSTS, X-Frame-Options, and X-XSS-Protection.
* **Open Ports**: Detected open ports 80/tcp, 443/tcp, 8080/tcp, and 8443/tcp.
* **Outdated CMS**: No CMS detected.
* **Website Safety**: Not entirely safe. Address missing headers promptly.

**General Recommendations**

1. Implement and verify WAF configurations to block malicious traffic.
2. Include essential security headers like CSP, HSTS, X-Frame-Options, and X-XSS-Protection.
3. Regularly audit open ports to ensure they are necessary and secure.
4. Update CMS versions to prevent vulnerabilities.
5. Monitor website configurations and traffic continuously to detect and respond to threats.

**Conclusion**

* **https://canarabank.com**: Demonstrates good security practices with no critical issues detected.
* **https://lanes-deserve-wholesale-n.trycloudflare.com**: Needs improvement in implementing security headers and ensuring server responsiveness.

**Report Summary:**

This report analyzes the security of two websites. **Canara Bank** demonstrates robust security with a detected WAF , essential HTTP headers like CSP and HSTS, limited open ports (80 and 443), and no outdated CMS. It is relatively secure with no critical vulnerabilities. In contrast, **lanes-deserve-wholesale-n.trycloudflare.com** lacks key security headers (CSP, HSTS, X-Frame-Options), has multiple open ports (80, 443, 8080, 8443), Recommendations include implementing WAF, adding missing headers, auditing ports, and ensuring continuous monitoring. The first website is secure, while the second needs significant improvements.