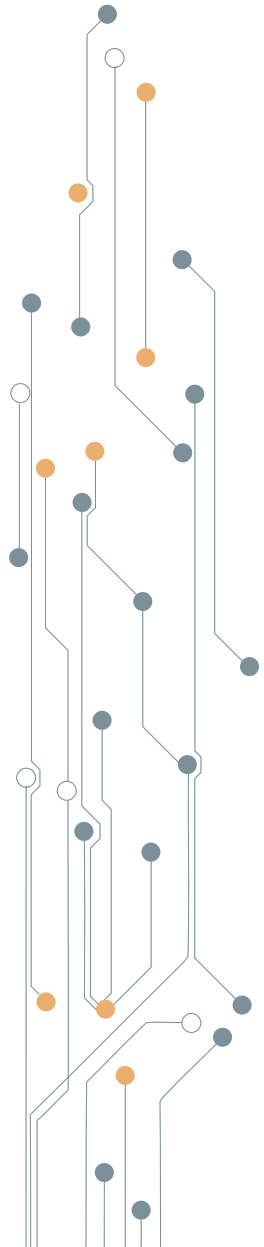
Ataques básicos y anonimato con Phyton

Telefónica EDUCACIÓN DIGITAL

Índice



1. Ataques básicos de red

Una de las finalidades más importantes que se pueden realizar mediante scripts en Python es la automatización de procesos y tareas. En este caso la automatización de ataques de red y de detección de estos ataques es una pieza muy importante en la labor de los especialistas de Seguridad.

1.1 | Detección de ataques de red

Como se ha comentado en puntos anteriores, la librería Scapy 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.

De esta forma, mediante el uso adecuado de esta librería se puede detectar posibles amenazas y ataques como pueden ser:

- Detección de un servidor DHCP falso
- Detección de un ataque ARP Spoofing

Detección de un servidor DHCP falso

En el proceso de administración de una red, es una práctica recomendable el controlar todos los servidores de DHCP que se encuentran en la misma y detectar si existe alguno nuevo, de esta forma se puede evitar ataques de DHCP Spoofing que puedan enviar información falsa al resto de elementos de la red.

Se puede realizar un script en Python que utilice Scapy para realizar un escaneo de la red buscando servidores DHCP. Para ello se construirá una petición que contenga como dirección de destino la dirección de Broadcast para que llegue a todos los dispositivos.

Como se envía a la dirección de Broadcast, es importante configurar Scapy para que no compruebe que el paquete de respuestas corresponde a la dirección que se ha configurado como envío, ya que en este caso esa dirección es la de Broadcast. Por este motivo es necesario poner como False el atributo conf.checkIpaddr.

Después de configurar el paquete en la capa de red y transporte, la siguiente capa del paquete se asignará el tipo BOOTP (Bootstrap Protocol) el cual está diseñado para las asignaciones de IPs en una red. Este paquete tendrá como MAC origen la del interfaz de red local de la máquina (se obtiene mediante el método *get_if_raw_hwaddr()*).

Seguidamente se generará un paquete del tipo DHCP utilizando como tipo de mensaje “discover” para encontrar los posibles servidores DHCP. Es importante añadir a la función “srp” el parámetro “multi=True” ya que en caso contrario no se podrán recoger múltiples respuestas:

```
from scapy.all import *

conf.checkIpaddr = False
f, h = get_if_raw_hwaddr(conf.iface)
dhcp = Ether(dst="ff:ff:ff:ff:ff:ff")/IP(src="0.0.0.0",dst=
"255.255.255.255")/UDP(sport=68,dport=67)/BOOTP(chaddr=h)/
DHCP(options=[("message-type", "discover"), "end"])
print(dhcp)
ans, unans = srp(dhcp, multi=True)
```

Detección de un servidor DHCP falso

El ataque ARP Spoofing es uno de los más utilizados en el entorno de una red local. El protocolo ARP está diseñado para el descubrimiento de las direcciones MAC de los nuevos dispositivos que se conectan a la misma red, pero es posible utilizarlo para engañar a un dispositivo indicándole que una dirección IP de la red con la que se está comunicando se encuentra en otra dirección MAC diferente que controla el atacante. De este modo todo el tráfico de la victima pasará por el equipo del atacante y este podrá inspeccionarlo para obtener información sensible y privada.

Para evitar este tipo de ataques es recomendable utilizar tablas ARP estáticas en cada una de las máquinas de la red, pero esto supone un gran trabajo de administración.

Se puede descubrir este tipo de ataque en un dispositivo concreto mediante un script en Python.

Este se quedará ejecutando e irá esnifando la red capturando todos los paquetes de ARP. Cuando encuentra uno lo analiza y comprueba que la dirección IP y la dirección MAC de este paquete corresponden a las que se han configurado inicialmente. Si es diferente se determinará que se trata de un ataque que está intentando envenenar la tabla ARP de la máquina:

```
from scapy.all import *

IP_Router = "XXXXX"
MAC_Router = "XXXXXX"
def arp_attack_detector(pkt):
        if ARP in pkt and pkt[ARP].op == 2:
                mac = pkt.sprintf("%ARP.hwsrc%")
                ip = pkt.sprintf("%ARP.psrc%")
                if ip == IP_Router and mac != MAC_Router:
                        print("Detectado ataque ARP Spoofing. La
MAC del atacante es: ", mac)
sniff(prn=arp_attack_detector)
```

1.2 | Ataque DHCP Starvation

DHCP Starvation es un ataque del tipo de denegación de servicio, ya que consiste en inundar al servidor DHCP con peticiones DHCP_REQUEST con direcciones MAC de origen falsas. El servidor irá asignando IPs a cada una de estas MAC hasta que se queda sin espacio de direcciones posibles. En este momento es incapaz de responder a otros clientes legítimos de la red ya que no dispone de más IPs que poder asignar.

Este ataque se suele utilizar junto con ataques de DHCP Rogue donde el atacante levanta otro servidor DHCP en la misma red, el cual se encargará de responder las peticiones de direccionamiento de IPs cuando el otro se encuentre inundado.

Para realizar este tipo de ataque desde un script en Python se utilizará el mismo tipo de paquete que el ejemplo anterior (BOOTP), pero en este caso la dirección MAC origen será una cadena aleatoria. Para generar esta cadena aleatoria se utilizará el método RandString.

Se creará un bucle infinito de peticiones DHCP el cual inundará al servidor:

```
conf.checkIpaddr = False
f, h = get_if_raw_hwaddr(conf.iface)
dhcp = Ether(dst="ff:ff:ff:ff:ff:ff") / IP(src="0.0.0.0",dst= "255.255.255.255") / UDP(sport=68,dport=67) / BOOTP
(chaddr=h)/DHCP(options=[("message-type", "discover"), "end"])
print(dhcp)
ans, unans = srp(dhcp, multi=True)
```

1.3 | Ataque MITM

Un ataque MITM (Man In The Middle) se produce cuando el atacante consigue ponerse en medio de la conexión entre 2 equipos, siendo este el intermediario de todos los datos que pasan por la comunicación.

Esto puede realizarse mediante diversas técnicas, pero la más conocida es la técnica de ARP Spoofing. Esta técnica utiliza las debilidades del protocolo ARP para engañar al equipo víctima para hacerle creer que el equipo detrás de una dirección IP es el correcto cuando después del ataque se trata del equipo del atacante (cambiando la dirección física MAC en las tablas ARP).

En este protocolo cuando una máquina quiere conocer la dirección física de otra (teniendo su dirección IP) envía una petición ARP Request a la dirección de Broadcast de la red local. Cuando la máquina aludida (tiene configurada la IP) recibe el paquete en su interfaz de red, esta envía un paquete del tipo ARP Reply indicando su presencia.

Una vez se ha realizado este proceso, la primera máquina almacena temporalmente esta información (IP-MAC) en su tabla ARP para no tener que preguntar de nuevo en futuras conexiones.

El problema de este protocolo es que las máquinas tienen que confiar en los mensajes ARP Request y ARP Reply recibidos ya que no se realiza ningún tipo de comprobación. Por este motivo, un atacante puede generar mensajes falsos indicando que es la MAC de su equipo donde está configurada determinada IP.

Este ataque se puede realizar desde Python utilizando la librería Scapy, con la cual se pueden generar paquetes ARP para engañar al resto de equipos de una red local.

En el siguiente ejemplo se muestra cómo realizar el envenenamiento de ARP tanto para el equipo victima como para el Gateway que tenga configurado para conseguir un MITM completo (controlando los 2 sentidos de la comunicación). El objetivo es generar mensajes ARP especificando en un caso que es enviado por el Gateway a la máquina de la víctima, y al revés. Siendo en los 2 casos la dirección física MAC desde donde se envía la del equipo atacante, con lo que se consigue que las tablas ARP de esas 2 máquinas contengan información falsa:

```
import threading
import sys
from scapy.all import *
class ARPPoison(threading.Thread):
        def __init__(self, srcAddr, dstAddr):
                threading.Thread.__init__(self)
                self.srcAddress = srcAddr
                self.dstAddress = dstAddr
        def run(self):
                try:
                        packet = ARP(pdst = self.dstAddress, psrc =
                        self.srcAddress)
                        send(packet, verbose = False, loop = 1)
                except:
                        print("Error")
```

```
victim = ARPPoison("192.168.1.1", "192.168.1.122")
victim.setDaemon(True)
victim.start()
gateaway = ARPPoison("192.168.1.122", "192.168.1.1")
gateaway.setDaemon(True)
gateaway.start()
```

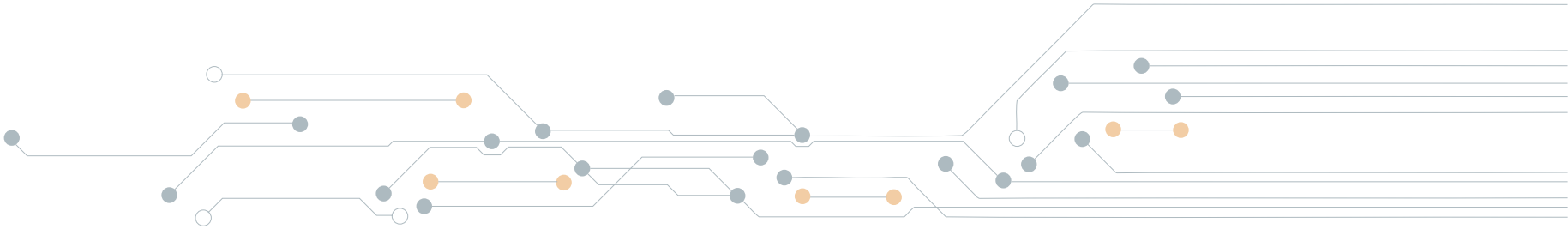
Nota: En sistemas operativos Linux es necesario activar el forwarding de paquetes en el sistema, para que el equipo atacante actúe de pasarela de las comunicaciones correctamente. Para ello se configurará la propiedad /proc/sys/net/ipv4/ip_forward con el valor 1.

1.4 | DNS Spoofing

Los servidores de DNS son los encargados de traducir un nombre de dominio a una IP concreta de internet donde poder realizar una conexión para obtener la información deseada. Los beneficios son evidentes, para acceder a cierta página web, no es necesario recordar la IP concreta del servidor (que puede cambiar a lo largo del tiempo) sino que solo hace falta recordar un nombre de dominio.

El ataque DNS Spoofing tiene como objetivo suplantar el servidor DNS que resuelve los nombres de dominio de una máquina víctima. De este modo, al tener el control sobre la resolución de nombres, se puede desplegar una copia de una página web en otro servidor cualquiera controlado por el atacante, y configurar en el DNS que el nombre del dominio legítimo apunte contra la IP de este servidor.

Este ataque se puede realizar de varias formas diferentes, una de ellas es mediante un ataque MITM donde se controla la comunicación de la máquina víctima. Al pasar todo el tráfico por la máquina del atacante se puede ir inspeccionado todos los paquetes, filtrando los que son de tipo DNS en cuyo caso se decidirá si hay que modificarlo o dejarlo pasar.



2. Anonimato con Python

Anonimato y privacidad en internet son dos términos que están en auge en la actualidad, incrementados por los últimos incidentes de seguridad que han transcendido en la prensa y donde se han hecho públicos una gran cantidad de datos privados de usuarios.

La mayoría de la gente cree que navegar por internet es una actividad anónima, pero poco a poco se van concienciando que en realidad se almacena una gran cantidad de información sobre la navegación del usuario que se puede determinar en cada momento que usuario es y que está haciendo. Por este motivo se está incrementando el uso de herramientas que permiten la navegación lo más anónima posible.

De mismo modo, un atacante necesita realizar sus actividades de forma anónima, eliminando cualquier rastro que pueda identificarle. Los dos proyectos más importantes que permiten una navegación anónima son la red TOR y la red I2P.

2.1 | Conexión con TOR

TOR (The Onion Router) es una red de comunicaciones anónima superpuesta sobre internet, donde una conexión una vez entra por un nodo TOR no se puede identificar al usuario inicial (su IP) y mantiene la privacidad e integridad de la información enviada.

Esto se consigue haciendo pasar el mensaje por varios nodos de la red TOR siguiendo una ruta más o menos aleatoria. Cuando un ordenador A quiere enviar un mensaje, calcula esa ruta y recolecta de cada uno de los nodos la clave pública del mismo. Luego se cifrará en mensaje con cada una de las claves obtenidas (en orden) asemejando una cebolla. De de esta forma ningún nodo va a poder ver el mensaje en claro cuando pase por él.

El primer paso para poder utilizar un script en Python a través de la red TOR es descargarse la herramienta TorBrowser de su página oficial: **https://www.torproject.org/**. En esta página se pueden encontrar los manuales y paquetes de instalación para todos los sistemas operativos.

Esta herramienta es un navegador que tiene integrado todo el sistema de gestión de la red TOR, por lo que al abrir el navegador se inicializa por abajo todos los servicios y endpoints donde se maneja estas comunicaciones.

Una vez instalado, lo ejecutamos desde una consola y lo dejamos corriendo. En este momento de forma transparente se inicializan varios servicios en diferentes puertos locales. Uno de ellos es un proxy SOCKS que se puede utilizar para canalizar todas las comunicaciones del script de Python por él. Este proxy se encuentra escuchando en la dirección IP "127.0.0.1" y en el puerto 9150.

En el siguiente script de Python se va a configurar que todas las conexiones que se realicen utilicen un proxy para enviarse, y este proxy será el listener SOCKS que está arrancado en local en el puerto 9150:

```
import socks
import socket
import requests

socks.setdefaultproxy(socks.PROXY_TYPE_SOCKS5, "127.0.0.1",
9150, True)
socket.socket = socks.socksocket

r = requests.get("https://www.wikipedia.org/")

client.exec_command("nohup /dev/shm/fileName &")
client.close()
```

Con el método setdefaultproxy declaramos el proxy que se va a utilizar que en este caso se encuentra corriendo en "127.0.0.1" en el puerto 9150.

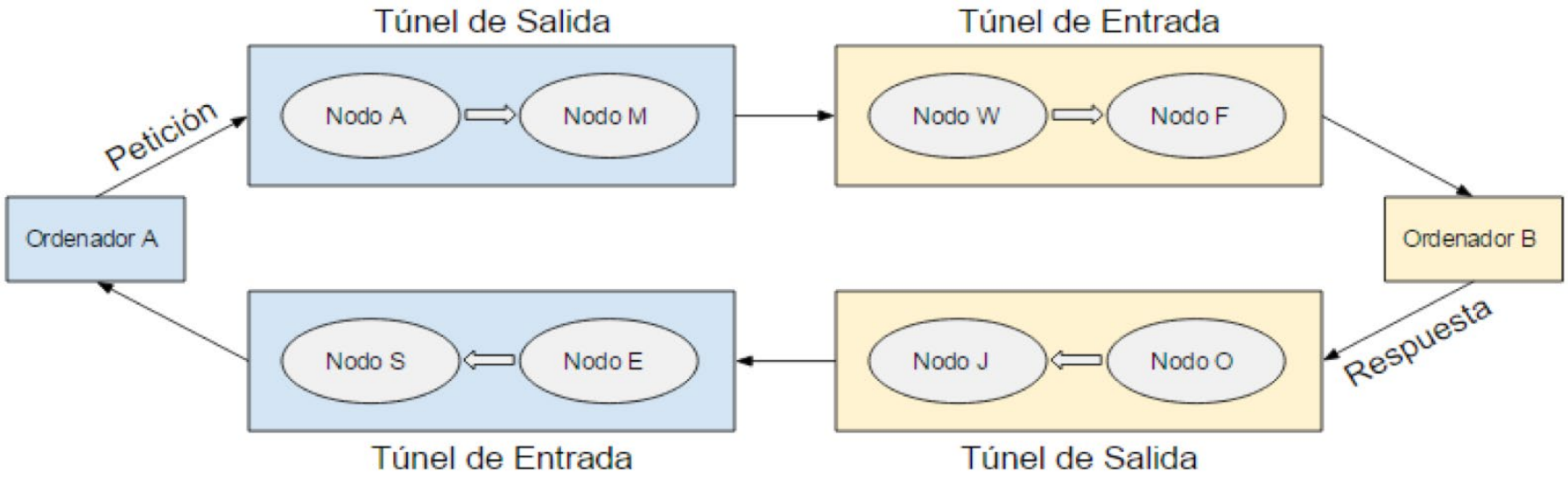
En la siguiente línea definimos que todas las conexiones de red que se utilicen desde este script utilicen la definición cargada en el módulo Socks, por lo que utilizarán el proxy configurado previamente.

Finalmente, la petición realizada a **https://www.wikipedia.org/** será encaminada a través de todos los nodos de la red TOR.

2.2 | Conexión con I2P

A diferencia de la red TOR donde se elegía un camino completo por donde hacer pasar la comunicación, en la red I2P se elijen 2 nodos al azar para crear un túnel entre ellos. Cada máquina que utilice esta red tendrá creado 2 túneles, uno de entrada y otro de salida.

De este modo, si el ordenador A quiere mandar un mensaje al ordenador B, lo enviará a su nodo del túnel de salida. Al salir por el otro nodo (del túnel de salida) llegará el túnel de entrada del ordenador B. Del mismo modo, la respuesta del ordenador B se enviará al nodo del túnel de salida del ordenador B, el cual se comunicará con el túnel de entrada del ordenador A.



I2P permite complicar mucho más el modelo, pudiendo tener varios túneles de salida por donde enviar los mensajes desordenados al ordenador B, y recibiendo las respuestas en el mismo túnel de entrada.

Esta red anónima sigue manteniendo todas las ventajas de la red TOR ya que no se conoce la IP el usuario y los datos de las peticiones van cifrados por lo que no pueden recuperarse, pero además esta solución es mucho más robusta frente a posibles ataques de vulneración de los nodos, ya que tendrían que controlar una mayor cantidad de nodos de la red para poder obtener algo de información.

El primer paso para poder utilizarlo desde un script de Python es instalar I2P. Los paquetes de instalación y manuales se pueden encontrar en su página oficial: **https://geti2p.net/es/**

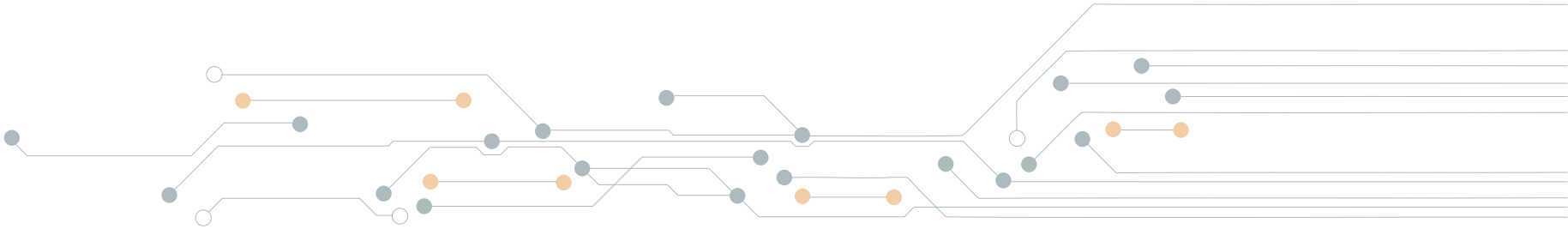
Una vez instalado, hay que ejecutar el fichero i2p y en este momento ya se puede acceder a la consola de administración desde cualquier navegador web del equipo, esta se encuentra en la dirección: **http://localhost:7657/index.jsp**

Pulsando en el menú de la izquierda sobre "Tuneles locales" se abrirá una página con los túneles que tiene levantado en ese momento el programa. Hay que comprobar que el proxy HTTP "I2P HTTP Proxy" está activo y su configuración es "127.0.0.1:4444".

Una vez se dispone del dato del proxy, es fácil realizar una petición Requests desde un script de Python utilizando ese proxy:

```
import requests

proxy = {"127.0.0.1:4444"}
r = requests.get("https://www.wikipedia.org/", proxies = proxy)
```



Telefónica EDUCACIÓN DIGITAL