Ejercicios Prácticos : Unidad 5

Tomás F. Melli

October 2025

Índice

1 Ejercicio 1

En este ejercicio tenemos que prestar atención al camino cuando Alice le manda un mail a Bob:

Recordemos los protocolos que se mencionan en la consigna :

Protocolos

- **SMTP (Simple Mail Transfer Protocol)** : Es el protocolo utilizado para **enviar correos electrónicos**. Toma el mensaje del remitente y lo entrega al servidor destino. Es utilizado tanto por los clientes (al enviar un correo) como por los servidores (al intercambiar mensajes entre dominios).
  Puertos comunes: 25 (SMTP tradicional), 587 (submission con STARTTLS), 465 (SMTPS — SMTP sobre TLS).
  Seguridad: se recomienda el uso de TLS (STARTTLS o SMTPS) para cifrar la conexión y proteger credenciales y contenido.
- **POP3 (Post Office Protocol v3)**: Permite a un cliente **descargar los correos** desde el servidor al dispositivo local del usuario. O sea, descarga y, opcionalmente, elimina los mensajes del servidor.
  Puertos comunes: 110 (POP3), 995 (POP3S — POP3 sobre TLS).
  Seguridad: se recomienda usar POP3S para cifrar la conexión y proteger las credenciales y el contenido.
- **IMAP (Internet Message Access Protocol)**: Permite a un cliente **sincronizar correos con el servidor**. Los mensajes permanecen almacenados en el servidor, y el cliente refleja carpetas, estados de lectura y borradores. Es ideal para acceder desde varios dispositivos.
  Puertos comunes: 143 (IMAP), 993 (IMAPS — IMAP sobre TLS).
  Seguridad: usar IMAPS cifra la conexión y protege credenciales y contenido en tránsito.

Ahora bien, qué pasa en cada tramo ?

Recorrido `alice@gmail.com` → `bob@outlook.com`

1. **Cliente emisor → Servidor SMTP del remitente**
   Alice escribe el correo y su cliente lo envía al servidor de Gmail. El mensaje y sus credenciales viajan cifrados si se usa TLS. Si la conexión no estuviera cifrada, un atacante podría interceptar la contraseña o el contenido del mensaje.
   Protocolo: SMTP sobre TLS (`SMTPS` o `STARTTLS`).
2. **Servidor SMTP de Gmail → Servidor SMTP de Outlook**
   Gmail busca los registros MX de `outlook.com` y entrega el mensaje al servidor destino. Si ambos servidores soportan TLS, la comunicación se cifra; de lo contrario, el correo puede viajar en texto plano.
   Protocolo: SMTP entre servidores (con posible uso de `STARTTLS`).
3. **Servidor receptor (Outlook) → Almacenamiento del buzón**
   Este paso es interno dentro de la infraestructura de Microsoft. El servidor receptor guarda el correo en el buzón de Bob. Aquí no se aplica TLS, ya que no es un tramo de red pública; la seguridad depende del control interno del proveedor.
4. **Cliente receptor → Servidor IMAP/POP de Outlook**
   Bob se conecta para descargar o sincronizar su correo. Si la conexión usa TLS, tanto las credenciales como los mensajes están protegidos en tránsito.
   Si no se utiliza TLS, un atacante en la misma red (por ejemplo, una Wi-Fi pública) podría capturar contraseñas y contenido.
   Protocolos: IMAPS (puerto 993) o POP3S (puerto 995).

Ahora que ya son claros los tramos, concluimos que no sólo es fundamental que se utilice SMTPS para la comunicación entre servidores, sino que es fundamental que se cifre la comunicación entre Alice y el servidor de su servidor, y luego, en el tramo en que Bob accede al correo de Alice. En esta última etapa podemos utilizar dos protocolos según convenga.

Qué pasa con la integridad ?

Como ya vimos, TLS nos va a garantizar que nadie chusmee lo que mandamos por cierto canal (confidencialidad) y también que esos mensajes en tránsito no sean modificados (integridad). Sin embargo, TLS nunca hace un chequeo de que `alice@gmail.com` sea Alice. Esto es porque SMTP permite que el campo `From:` sea modificado fácilmente. Hay mecanismos como **DKIM (DomainKeys Identified Mail)** (El servidor firmante (por ejemplo, Gmail) agrega una firma criptográfica en el encabezado del correo. El dominio receptor puede verificarla usando la clave pública publicada en el DNS del dominio del remitente), **PGP (Pretty Good Privacy)** o **S/MIME (Secure/Multipurpose Internet Mail Extensions)** que mediante ciertos mecanismos nos permiten garantizar autenticidad del emisor.

No-repudio

El **no repudio** es una propiedad de seguridad que impide que el emisor niegue haber enviado un mensaje, y que el receptor niegue haberlo recibido. Requiere evidencia criptográfica vinculada a una identidad verificable. TLS usa claves efímeras para cada sesión; cuando esta termina, no queda prueba de la comunicación más allá del contenido transmitido. No hay una firma persistente del emisor que pueda verificarse luego. Los certificados TLS identifican a los servidores, no a los usuarios finales (como Alice o Bob). Por lo tanto, aunque la conexión fue segura, no se puede demostrar jurídicamente quién escribió el mensaje. Existen mecanismos como

- **Firmas digitales personales (PGP o S/MIME)**: la firma se adjunta al mensaje y puede ser verificada en cualquier momento.
- **Infraestructura de clave pública (PKI)**: permite vincular legalmente una identidad (por ejemplo, una persona o empresa) con una clave criptográfica.

2 Ejercicio 2

Contexto

Una empresa decide crear su propio protocolo en una aplicación cliente-servidor (el usuario se debe autenticar con el servidor). La propuesta de la empresa es, no vamos a cifrar toda la comunicación, vamos a cifrar sólo la contraseña cuando el usuario se quiere autenticar. En particular, la empresa propone que el algoritmo de flujo simétrico para cifrar únicamente la contraseña antes de enviarla, sin usar vector de inicialización (IV) o nonce. Cliente y servidor comparten una clave simétrica K. Recordemos cómo funcionan estos algortimos :
Un cifrador de flujo típico genera una secuencia de bytes pseudoaleatoria llamada *keystream* KS a partir de la clave simétrica K (y, normalmente, un IV/nonce). El cifrado y descifrado se realizan byte-a-byte mediante la operación XOR:

$$\text{Cifrado:} \qquad C = P \oplus \text{KS},$$

donde P es el *plaintext* (en este contexto, la contraseña en bytes), KS es el keystream (del mismo largo que P), y C es el *ciphertext* enviado.

$$\text{Descifrado:} \qquad P = C \oplus \text{KS},$$

puesto que $(P \oplus \text{KS}) \oplus \text{KS} = P$.
El IV/nonce sirve para que, aun usando la misma clave K, el keystream KS sea distinto en cada cifrado. Si no se utiliza IV/nonce, entonces el keystream será idéntico en cada ejecución (mientras K no cambie). Esto implica que si se cifran la contraseña `pwd1` en una sesión y `pwd2` en otra, ambos se cifran con el mismo KS.
La consecuencia es:

$$C_1 \oplus C_2 = (P_1 \oplus \text{KS}) \oplus (P_2 \oplus \text{KS}) = P_1 \oplus P_2.$$

El keystream KS desaparece del XOR y queda el XOR directo entre los dos plaintexts P_1 y P_2.
Esto genera:

- **Reutilización de keystream (el fallo central).**
  Al cancelar KS entre dos capturas, el atacante obtiene $P_1 \oplus P_2$. Con esa información puede aplicar *crib-dragging*: probar palabras o patrones conocidos (por ejemplo `password`, `admin`, prefijos o formatos habituales) para recuperar fragmentos legibles de P_1 o P_2. Con suficientes capturas o supuestos razonables sobre el formato de la contraseña, es posible reconstruir la contraseña completa.
- **Ataque por diccionario offline / fuerza bruta.**
  Si el atacante dispone de una única captura C, puede iterar candidatos p' del diccionario y calcular

  $$\text{KS}' = C \oplus p'.$$

Si existen otras capturas cifradas con el mismo KS, puede comprobar la consistencia de KS$'$; alternativamente, puede verificar candidatos mediante intentos online contra el servidor.

- **Compromiso del cliente (acceso local).**
  Si el atacante compromete el cliente, puede extraer la clave simétrica K y, con ella, generar o recuperar cualquier keystream necesario para descifrar comunicaciones. Este es el escenario más grave: el esquema queda trivialmente roto.
- **Falta de integridad y autenticidad.**
  Un cifrado de flujo sin un mecanismo de autenticación (MAC/AEAD) permite modificar ciphertexts de forma controlada (bit-flipping), lo que produce cambios previsibles en el plaintext. Si el servidor no valida correctamente, un atacante podría manipular la contraseña o introducir datos indebidos.
- **Replay.**
  Si el protocolo no incorpora nonces/contadores o mecanismos anti-replay, un ciphertext válido capturado puede re-enviarse al servidor. En ausencia de protección contra replay, el atacante puede reutilizar un ciphertext capturado para autenticarse sin necesidad de descifrarlo.

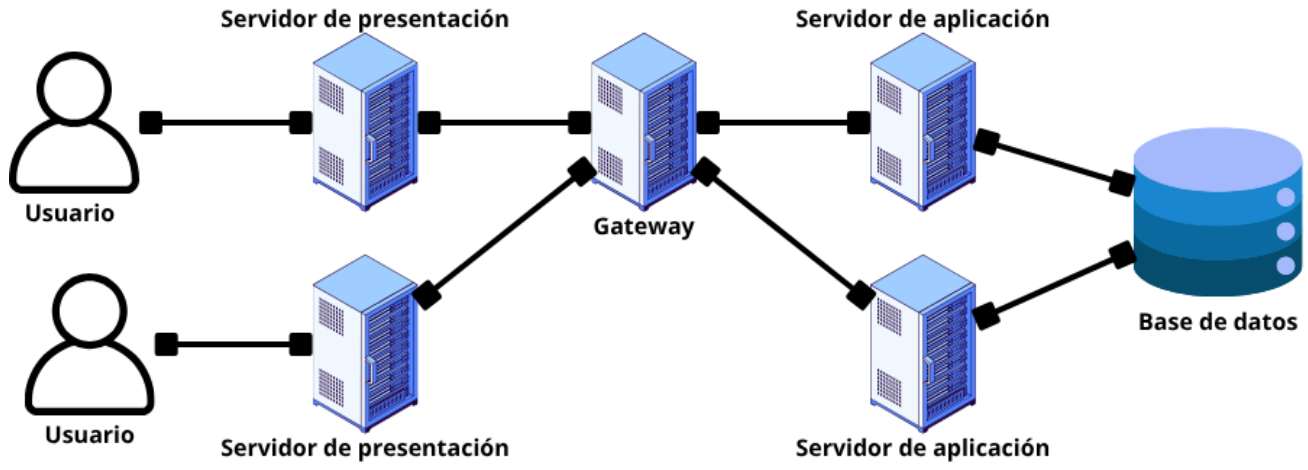
La empresa piensa que si alguien que está en la red local (por ejemplo, en la misma Wi-Fi), que abre Wireshark y Sniffea los paquetes, y ve la contraseña si va en texto plano, con implementar su idea está excenta de ataques. Pero efectivamente no es así, ya que puede hacer un replay muy fácilmente, reenviando el mismo paquete aunque cifrado que mandó el usuario autorizado. Esto funciona ya que el protocolo de la empresa no implementa **challenge** por parte del servidor por cada intento y exige la respuesta cifrada sobre ese nonce (challenge-response). Respondiendo a la última pregunta, el atacante no necesita, en este escenario, acceder al cliente.
Conclusión : usar TLS resuelve todos los problemas ya que evita que el atacante snifee la red.

3 Ejercicio 3

Un sistema ERP (Planificación de Recursos Empresariales) es un software de gestión que integra y centraliza todas las operaciones y procesos de una empresa, como finanzas, RR. HH., ventas, inventario y logística, en una única plataforma. La idea es :

1. El cliente realiza una solicitud al **servidor de presentación** (UI).
2. El servidor de presentación envía llamadas al **Gateway** para obtener/ejecutar tareas en los servidores de aplicación.
3. El Gateway decide a qué **servidor de aplicación** enviar la tarea (balanceo/ruteo).
4. El servidor de aplicación procesa la tarea (puede acceder a la base de datos) y devuelve la respuesta al cliente a través del servidor de presentación.

La topología se ve algo así :



Con esto en mente, qué pasa en cada uno de estos componentes ?

Componentes

Dónde está el ERP ?

El núcleo lógico del ERP —la parte que ejecuta las reglas de negocio, procesa transacciones, accede a la base de datos— reside en los **servidores de aplicación**. Estos servidores contienen los módulos centrales del ERP (por ejemplo: contabilidad, inventario, facturación).

Qué hacen los servidores de presentación?

Los **servidores de presentación** se encargan de la interacción con los usuarios: sirven la interfaz (web o clientes), manejan sesiones, validan entrada y envían peticiones al Gateway para que éste distribuya el trabajo a los servidores de aplicación.

Qué hace el Gateway?

El **Gateway** se sitúa entre los servidores de presentación y los servidores de aplicación y realiza funciones de control: balanceo de carga entre instancias de aplicación, registro de instancias, ruteo de peticiones.

Todos los servidores están en el mismo segmento de red

Significa que los servidores (presentación, Gateway, aplicación y, probablemente, la BD) comparten el mismo dominio de broadcast. Esto facilita ciertas comunicaciones pero también hace factibles ataques de la capa 2 (por ejemplo ARP spoofing o MAC spoofing) para un atacante que tenga acceso a esa red.

Contexto de autenticación

1. **Cliente → Servidor de Presentación**: Identifica al usuario final y permite acceso a la interfaz del ERP. El cliente envía credenciales al servidor de presentación, que las valida y crea una sesión (cookie, token JWT). Si la conexión no está cifrada, un atacante puede capturar credenciales.

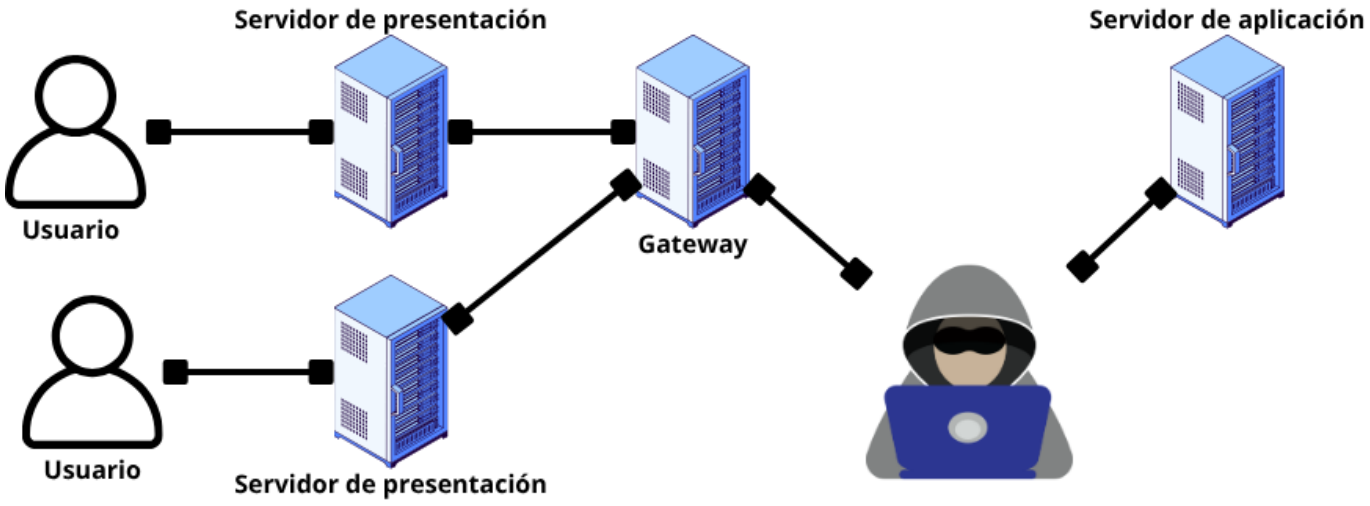
2. **Servidor de Presentación → Gateway**: los servidores de presentación envían solicitudes a los servidores de aplicación. En este escenario **no hay autenticación; el Gateway acepta cualquier registro.**

3. **Gateway → Servidor de Aplicación**: asegura que solo el Gateway legítimo envíe tareas a los servidores de aplicación. En este escenario, **no hay control; cualquier registro es aceptado.**

4. **Servidor de Aplicación → Base de Datos**: acceder a los datos necesarios para procesar tareas del ERP. Para autenticarse puede ser usuario/contraseña del servidor de aplicación, conexiones cifradas (TLS/SSL), roles y permisos dentro de la base de datos.

Cómo puede ocurrir un MITM entre el Gateway y el servidor de aplicación?

Un MITM entre Gateway ↔ App ocurre cuando el atacante consigue que el tráfico que debería ir directo entre estos dos nodos pase por su host (o sea procesado por él) de forma transparente. Esto es :



Se puede dar:

- **Suplantación a nivel de aplicación (registro falso / impersonation).**
- **Ataques de red: ARP spoofing / MAC spoofing / route/DNS poisoning (L2/L3).**
- **Compromiso de un host legítimo (pivot).**

Registro falso / impersonation (nivel aplicación)

El atacante envía al Gateway un mensaje de registro haciéndose pasar por un servidor de aplicación. Si el Gateway no autentica el registro, lo aceptará y empezará a rutear tráfico a la instancia del atacante. Para que esto suceda, el atacante debe :

- Poder alcanzar el endpoint de registro del Gateway (puerta TCP/IP).
- Conocer (o deducir) el formato del protocolo de registro.

Es decir, el atacante deberá:

1. Abrir una conexión TCP al puerto de registro del Gateway.
2. Enviar el payload de registro con la información requerida (por ejemplo: identificador, IP, puerto).
3. Si el Gateway acepta la nueva instancia, empezarará a recibir peticiones delegadas.
4. Actuar como proxy: inspeccionar/modificar y reenviar al servidor legítimo, o responder directamente.

ARP spoofing / ARP cache poisoning (capa 2)

El atacante, en el mismo segmento L2, envía ARP replies falsas que asocian la IP del Gateway (o del servidor de aplicación) con la MAC del atacante. El tráfico que debería ir al destino real pasa por el atacante. El atacante deberá tener:

- Acceso al mismo segmento de red (misma VLAN / broadcast domain).
- Capacidad para inyectar paquetes L2 (herramientas: `arpspoof`, `ettercap`, `bettercap`).

Lo que tendrá que hacer es :

1. Habilitar IP forwarding en la máquina atacante:
2. Lanzar ARP spoofing contra ambos extremos. O sea, lanzar ARP_Reply que digan que la IP del servidor de aplicación está asociada a su MAC (la del atacante), con lo cual el Gateway actualiza su tabla ARP con esta info. Por el otro lado, debe mandar ARP_Reply al servidor de aplicación diciendo que la IP del gateway está asociada a su MAC. Esto funciona, en particular porque ARP admite que haya muchos replies (después si sobre esto se implementan mecanismos de detección, es otra cosa).
3. Recibir paquetes, inspeccionarlos/modificarlos y reenvíarlos al destino real.

Compromiso de un host legítimo (pivot)

El atacante compromete (por ejemplo por phishing) una máquina dentro de la red y desde ahí realiza ARP spoofing, o hace registros falsos hacia el Gateway, actuando como MITM.

Respondiendo las preguntas

- **Protecciones posibles sin cambiar la arquitectura ni los (no)mecanismos de autenticación:**
  - **Controles L2 en switches:** Port security (limitar MACs por puerto), sticky/static MAC binding, prevención de CAM flooding.
  - **Hardening DHCP/DNS/switch:** DHCP snooping, Dynamic ARP Inspection (DAI) y bindings IP-MAC estáticos para servidores críticos.
  - **ACLs y filtrado:** Configurar ACLs en Gateway/routers para aceptar conexiones solo desde IPs/MACs conocidas de servidores.
  - **Segmentación lógica:** Mover servidores a VLANs separadas o usar private VLANs para aislarlos de hosts no confiables.

- **Monitoreo y operación:** Logging de registraciones en el Gateway, detección de nuevos servidores, alertas y procedimientos de respuesta.
- **Seguridad física y de administración:** Control de acceso físico a los switches/puertos y protección de credenciales administrativas.

- **Sirve un IDS/IPS?**
  - **IDS:** Útil para *detectar* ARP spoofing, duplicados IP/MAC y registraciones anómalas. No bloquea por sí solo; sirve para alertar y activar respuestas operativas.
  - **IPS (inline):** Puede bloquear ciertos vectores si está correctamente posicionado y configurado, pero requiere afinamiento para evitar falsos positivos y no sustituye medidas L2 como DAI o port security.

- **Y una VPN?**
  - Una VPN (por ejemplo IPsec entre Gateway y servidores) cifra y autentica el tráfico IP, impidiendo que un interceptador pueda leer o modificar datos sin la clave correcta. Por tanto, una VPN es una defensa efectiva contra MITM.
  - Implementar VPNs requiere gestión de claves/certificados y cambios operativos; si se acepta añadir esa capa, es una de las mejores formas de mitigar MITM aun cuando el registro al Gateway no tenga control.

4 Ejercicio 4

Recordemos algunos conceptos antes de responder las preguntas :

Firewall

Un **firewall** es un sistema que controla el tráfico de red según un conjunto de reglas. Su objetivo es **permitir o bloquear paquetes** entre redes con distintos niveles de confianza (por ejemplo, entre Internet y una red interna).

Hay dos grandes tipos:

- **Stateless (sin estado):**
  Revisa cada paquete individualmente, sin recordar conexiones previas.
  ⇒ *Ejemplo:* bloquea todo paquete TCP con puerto 23, sin importar de dónde viene o si pertenece a una conexión existente.

- **Stateful (con inspección de estado)** o **Stateful Packet Inspection (SPI):**
  El firewall mantiene una **tabla de estado** con información sobre las conexiones activas (por ejemplo, IPs origen/destino, puertos, número de secuencia TCP, etc.).
  ⇒ Esto le permite **distinguir entre tráfico legítimo y respuestas esperadas.**

Ejemplo: si un host interno inicia una conexión TCP hacia Internet, el firewall agrega esa conexión a su tabla de estado. Luego permitirá el tráfico de respuesta, aunque venga en sentido contrario, porque reconoce que pertenece a una conexión ya establecida.

Stateful Packet Inspection (SPI)

Es la técnica que implementan los **firewalls stateful** para:

- Examinar cabeceras y, a veces, el contenido del paquete.
- Mantener contexto de las conexiones activas.
- Permitir únicamente paquetes que formen parte de una sesión válida.

Por ejemplo: Si llega un paquete TCP con el flag `ACK` pero no hay una sesión en la tabla de estado, el firewall lo descarta.

IDS (Intrusion Detection System)

Un **IDS** es un sistema de **detección de intrusiones**.

- Monitorea el tráfico o los registros para detectar patrones sospechosos o anómalos.
- Puede ser **basado en firmas** (reconoce patrones conocidos) o **basado en anomalías** (detecta comportamientos fuera de lo normal).
- No bloquea el tráfico, solo alerta.

Ejemplo: Si detecta paquetes que intentan explotar una vulnerabilidad conocida (según su base de firmas), genera una **alarma**.

IPS (Intrusion Prevention System)

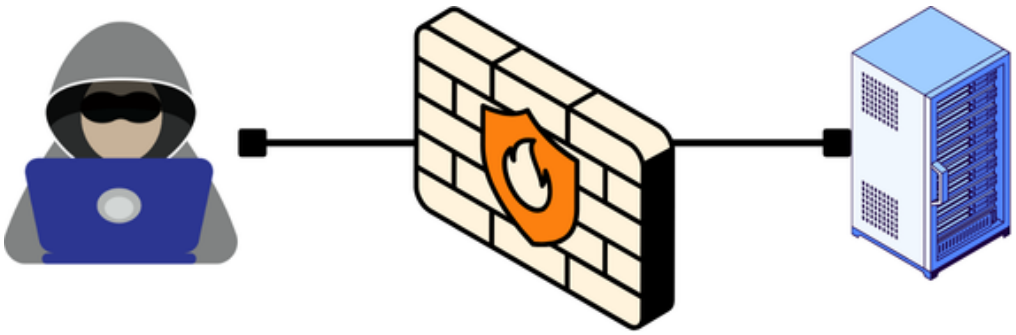
Un **IPS** es como un IDS pero **en línea (inline)**, es decir, intercepta el tráfico.

- Puede detectar y también bloquear o modificar el tráfico malicioso.
- Es un **mecanismo preventivo** (no solo reactivo).

Ejemplo: Si detecta un ataque de denegación de servicio, puede bloquear temporalmente la IP origen.

Respuestas

Esquema



Cómo se podría detectar que un firewall implementa SPI?

La idea es simular un **three-way handshake** pero como estamos auditando el firewall, vamos a fingir demencia y mandar por el puerto 80/TCP, como si estuviesemos haciendo una HTTP Request, pero en vez de hacer todo el biri biri, le mandamos directo un `ACK` al servidor. Esto nos va a servir para deducir el tipo de firewall ya que un stateless, si admite conexiones al puerto 80, lo deja pasar al paquete y como el servidor chequea si hay una conexión activa con ese `oring/destino/puerto/nro de seq` y no existe, manda un `RST`. Como consecuencia, recibimos el `RST`. Caso diferente con el stateful, que mantiene la tabla de conexiones, si le mandamos un `ACK` al servidor, el firewall detecta que no hay conexión activa y descarta el paquete.

Podría un IDS detectar este ataque ?

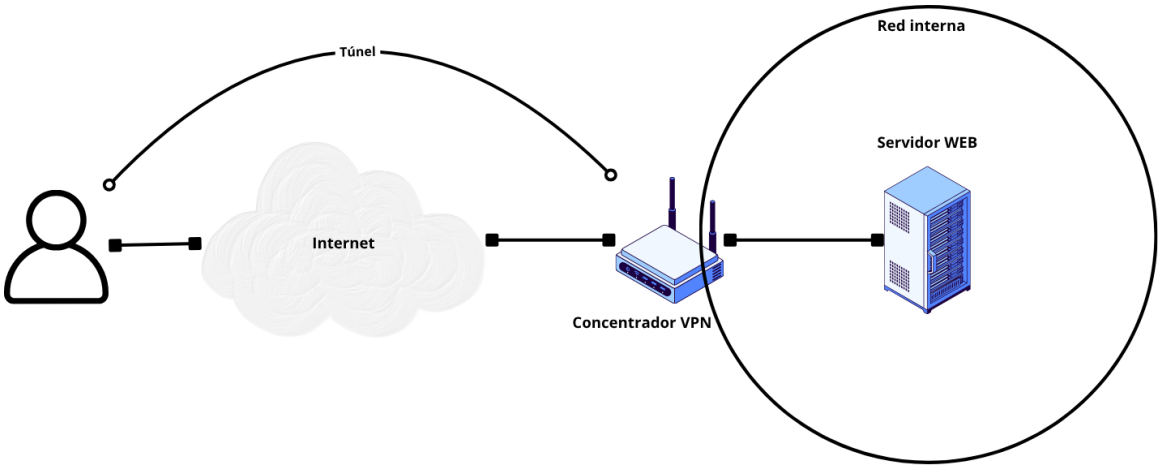
Como hablamos, en la medida en que esté configurado para que el patrón sospechoso sea *Paquete TCP sin conexión previa*, lo va a detectar. La idea es detectar un Firewall probing (es la acción de enviar tráfico deliberado hacia un firewall o hacia hosts protegidos por un firewall con el objetivo de determinar su comportamiento, reglas abiertas, o identificar qué puertos/servicios permiten pasando. Es una forma de reconocimiento activo orientada a descubrir la configuración y restricciones del perímetro).

Podría un prevenirlo un IPS ?

Un IPS puede prevenir el ejemplo que vimos antes, donde se manda un `ACK` sin conexión previa. El tema es si logra el atacante filtrar algún paquete (dado que el IPS no puede interferir con ese tipo) y medir tiempos de respuesta, podría no evitar un Firewall probing. Depende de la naturaleza del ataque, lo que sí ayuda a reducir la superficie de ataque.

5 Ejercicio 5

Primero veamos un esquema de la situación :



El contexto nos dice que nos queremos conectar con cierto servidor WEB (HTTPS) que está en una red remota y que para ello tenemos que establecer una VPN (un túnel cifrado) contra un concentrador VPN (el dispositivo que autentica a los usuarios, establece los túneles y luego enruta el tráfico hacia la red interna). El tema es,
Si al conectarme al servidor Web que utiliza SSL me da un error de certificado porque no tengo instalado el certificado de la autoridad certificante que emitió el certificado utilizado por el servidor Web, ¿puedo confiar en que me estoy conectando al servidor Web real ya que estoy utilizando una VPN para acceder al segmento de red dónde está conectado el servidor?

En otras palabras, si recibo un error de certificado es porque no puedo verificar la autenticidad del certificado del servidor, porque no confío en la CA que lo firmó.
Veamos un poco de contexto: para conectarse con el servidor web con SSL se hace el SSL Handshake que consta de un Hello por parte del usuario y el servidor le responde con su certificado digital, el usuario chequea nombre/firma/duration, y luego se decide si se establece la comunicación.

Volviendo al ejercicio. La verificación falló, o sea, ese servidor no es confiable. Tener acceso a la red remota por la VPN, no implica que todo con lo que interactúe no sea malicioso. Esto se podría dar si uno de los usuarios dentro de esa red es malicioso y se hace pasar por el servidor mediante un spoofing (ARP o DNS).
Y si en lugar de un servidor Web fuese un servidor SSH que no tengo almacenado en el archivo de hosts conocidos?
Veamos contexto : `/.ssh/` es un directorio que contiene archivos de configuración y claves de SSH. Dentro de este directorio, `known_hosts` es un archivo que almacena las claves públicas de los servidores a los que nos hemos conectado previamente. Cada entrada tiene esta pinta :

$$192.168.1.10 \quad ssh-rsa \quad AAAAB3NzaC1yc2EAAAABIwAAAQE...$$

Que nos dice la IP del servidor, el tipo de clave y la clave pública codificada en Base64. Sirve para autenticar automáticamente un servidor en futuras conexiones. Cuando la conexión es nueva, la verificación de la huella digital es manual ya que SSH no puede confiar de una. Dicho esto, si confiamos de una, no sabemos a quién nos conectamos, lo cual es peligroso. Por tal motivo, lo mejor es obtener la huella digital de la clave pública del servidor desde una fuente confiable y comparar con la que salta en la alerta SSH en la primera conexión y sólo si coinciden agregar la entrada en `/.ssh/known_hosts`.

6 Ejercicio 6

Para este ejercicio necesitamos Wireshark. Corremos `sudo apt install wireshark`. Abrimos entonces con `wireshark x-forwarded-for-filtered.pcap` parados en el directorio de la guía. Un poco de detalles de esta interfaz, vamos a tener varias cosas :

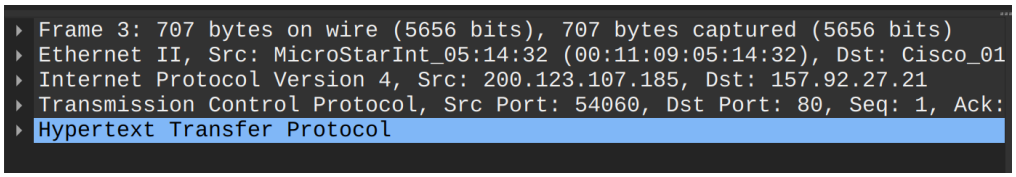
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	200.123.107.185	157.92.27.21	TCP	74	54060 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM TSval=1883255312…
2	0.005453	200.123.107.185	157.92.27.21	TCP	66	54060 → 80 [ACK] Seq=1 Ack=1 Win=5856 Len=0 TSval=1883255314 TSecr=732452…
3	0.005526	200.123.107.185	157.92.27.21	HTTP	707	GET / HTTP/1.0
4	0.453402	200.123.107.185	157.92.27.21	TCP	66	54060 → 80 [ACK] Seq=642 Ack=1449 Win=8736 Len=0 TSval=1883255426 TSecr=7…
5	0.455209	200.123.107.185	157.92.27.21	TCP	66	54060 → 80 [ACK] Seq=642 Ack=2897 Win=11648 Len=0 TSval=1883255426 TSecr=…
6	0.457169	200.123.107.185	157.92.27.21	TCP	66	54060 → 80 [ACK] Seq=642 Ack=4345 Win=14528 Len=0 TSval=1883255427 TSecr=…
7	0.460688	200.123.107.185	157.92.27.21	TCP	66	54060 → 80 [ACK] Seq=642 Ack=5793 Win=17440 Len=0 TSval=1883255427 TSecr=…
8	0.460882	200.123.107.185	157.92.27.21	TCP	66	54060 → 80 [ACK] Seq=642 Ack=6159 Win=20320 Len=0 TSval=1883255428 TSecr=…

- **No. (Número de paquete)** : Es un contador secuencial que indica el orden en que Wireshark capturó los paquetes. Sirve para referenciar paquetes específicos en el análisis.
- **Time (Tiempo)** : Muestra el tiempo transcurrido desde el inicio de la captura hasta el momento en que se capturó el paquete. Por defecto, se expresa como tiempo relativo, pero puede configurarse para mostrar la hora absoluta o la diferencia con el paquete anterior. Ejemplo: `0.005453` indica que el paquete fue capturado 0.005453 segundos después del primero.
- **Source (Origen)** : Dirección IP o MAC del equipo que envió el paquete. En una comunicación cliente-servidor, suele ser la dirección del cliente cuando se trata de una petición, o la del servidor cuando es una respuesta.
- **Destination (Destino)** : Dirección IP o MAC del equipo receptor del paquete. En tráfico bidireccional, permite identificar hacia dónde se dirige cada paquete (cliente o servidor).
- **Protocol (Protocolo)** : Indica el protocolo utilizado por el paquete. Algunos ejemplos comunes son: `ARP`, `ICMP`, `TCP`, `UDP`, `HTTP`, `TLS`, `SSH`, `DNS`, `DHCP`. Wireshark lo determina analizando las cabeceras y los puertos involucrados.
- **Length (Longitud)** : Representa el tamaño total del paquete en bytes, incluyendo las cabeceras y los datos. Permite detectar fragmentaciones, retransmisiones o paquetes de tamaño inusual.
- **Info (Información)** : Contiene un resumen textual del contenido del paquete, adaptado al protocolo correspondiente.

Como en este ejercicio tenemos paquetes de los protocolos TCP y HTTP, vamos a ver algunos ejemplos para poder entender qué nos pide el enunciado :

- **Caso HTTP** : la columna *Info* muestra el tipo de mensaje (petición o respuesta) y el recurso involucrado.
  - Ejemplos:
    - `GET /index.html HTTP/1.1` → el cliente solicita el recurso `/index.html`.
    - `GET /login.php HTTP/1.1 Host: www.example.com` → el cliente envía una petición al dominio `www.example.com`.
    - `HTTP/1.1 200 OK` → el servidor responde correctamente a la solicitud.
    - `HTTP/1.1 302 Found` → el servidor indica una redirección.
    - `HTTP/1.1 404 Not Found` → el recurso solicitado no existe.
    - `Continuation or non-HTTP traffic` → indica tráfico cifrado (HTTPS), que Wireshark no puede interpretar.
  - En resumen, la columna *Info* en HTTP indica si el cliente está realizando una petición (GET, POST, etc.) o si el servidor está respondiendo (códigos 200, 302, 404, etc.). En el caso de HTTPS, el contenido no se muestra porque está cifrado bajo TLS.
  - La estructura es `<Método> <Recurso> <Versión del protocolo>`
- **Caso TCP** : la columna *Info* resume los campos principales del encabezado TCP, mostrando la dirección de comunicación, los flags activos y los números de secuencia y acuse.
  - Ejemplo: `443 → 51234 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0`
  - Quiere decir...:
    - **443 → 51234:** comunicación desde el puerto 443 (servidor HTTPS) hacia el 51234 (cliente).
    - **[SYN, ACK]:** flags TCP activos; en este caso, sincronización y acuse.
    - **Seq=0:** número de secuencia (inicio del flujo de bytes).
    - **Ack=1:** número de acuse, confirma recepción del byte anterior.
    - **Win=65535:** tamaño de ventana, cantidad de bytes que el receptor puede aceptar antes de requerir confirmación.
    - **Len=0:** longitud del segmento TCP (sin datos de aplicación).
  - Ejemplo de intercambio TCP (three-way handshake):
    - `51234 → 443 [SYN] Seq=0 Win=65535 Len=0` → el cliente inicia la conexión.
    - `443 → 51234 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0` → el servidor acepta y responde.
    - `51234 → 443 [ACK] Seq=1 Ack=1 Win=65535 Len=0` → el cliente confirma, estableciendo la conexión.

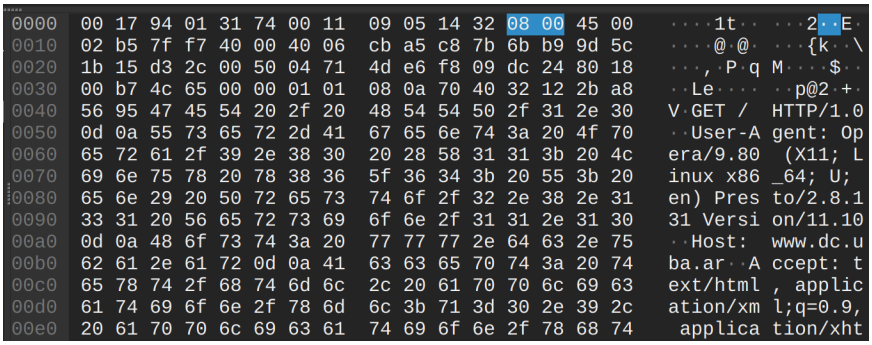
Ahora bien, si seleccionamos una entrada, por ejemplo, la del paquete HTTP, vamos a ver un recuadro con :



Donde cada entrada corresponde a una capa del paquete:

- **Frame:** datos capturados por la tarjeta de red. Contiene información general del paquete, como el tamaño en bytes y el tiempo de captura.
- **Ethernet II:** capa de enlace. Incluye las direcciones físicas *MAC* de origen y destino, así como el tipo de protocolo encapsulado.
- **IP (IPv4 o IPv6):** capa de red. Define las direcciones IP de origen y destino, el *Time To Live (TTL)*, el identificador de protocolo, y el *checksum* de cabecera.
- **TCP o UDP:** capa de transporte. Contiene los números de puerto de origen y destino, los *flags* de control (como SYN, ACK, FIN), los números de secuencia y acuse, y el tamaño de ventana.
- **HTTP / TLS / DNS / SSH:** capa de aplicación. Representa el protocolo de más alto nivel, correspondiente a la comunicación directa entre aplicaciones (por ejemplo, solicitudes HTTP, sesiones TLS cifradas, consultas DNS o conexiones SSH).

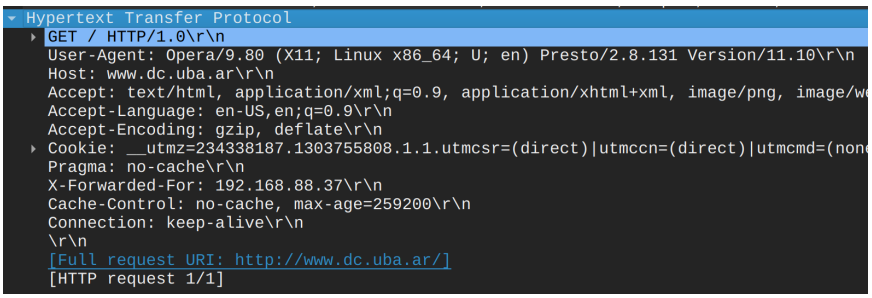
Y finalmente, en el recuadro de la derecha, tenemos el **Packet Bytes Pane** que es el panel que muestra el contenido del paquete en formato hexadecimal y ASCII :



Donde las columnas nos indican :

- **Offset :** Posición del byte dentro del paquete (en hexadecimal).
- **Hexadecimal :** Valor binario de los bytes en formato hex.
- **ASCII**: Representación textual (si hay caracteres imprimibles).

Volviendo al ejercicio, nos pide detectar si el cliente hizo la conexión mediante u proxy. Entonces, lo que vamos a hacer es chusmear el paquete HTTP :



Si miramos atentamente, la entrada `X-Forwarded-For : 192.168.88.37` vemos que no es la misma IP que el del source del panel superior. Por tanto :

- El cliente real es `192.168.88.37`

- El paquete capturado muestra como origen la IP pública `200.123.107.185`. Esto indica que otro equipo (un **proxy** o **gateway**) está reenviando la solicitud HTTP.
- El encabezado `X-Forwarded-For` fue agregado por ese proxy para que el servidor destino (`157.92.27.21`) pueda conocer cuál era la IP original del cliente que realizó el pedido.
- Concluimos que:
  - El proxy o NAT público tiene IP `200.123.107.185`.
  - El cliente interno era `192.168.88.37`.
  - El servidor destino es `157.92.27.21`.

7 Ejercicio 7

Antes de encarar el ejercicio, como las peticiones son de protocolos diferentes a los del ejercicio anterior, vamos a introducir la información que nos proporciona wireshark :

- **DNS (Domain Name System)** En los paquetes DNS, el campo `Info` muestra un resumen de la consulta o respuesta realizada.
  - **Campos en Info:**
    - **Transaction ID**: identificador único de la consulta.
    - **Query / Response**: indica si el paquete es una solicitud o una respuesta.
    - **Nombre de dominio solicitado**.
    - **Tipo de registro**: qué tipo de dato querés obtener del servidor DNS (una IP, un alias, un servidor de correo, ...) (`A`, `AAAA`, `MX`, `CNAME`, `PTR`,...).
    - **Respuesta (si aplica)**: dirección IP u otro valor resuelto.
  - **Ejemplos:**
    - `Standard query 0x1a2b A www.google.com` → El cliente consulta la IP (tipo A) del dominio `www.google.com`.
    - `Standard query response 0x1a2b A 142.250.78.132` → El servidor responde que la IP del dominio es `142.250.78.132`.
    - `Standard query 0x0023 PTR 132.78.250.142.in-addr.arpa` → Consulta inversa: se solicita el nombre asociado a una dirección IP.
- **ICMP (Internet Control Message Protocol)** En los paquetes ICMP, el campo `Info` resume el tipo de mensaje de control o error transportado por IP.
  - **Campos en Info:**
    - **Tipo de mensaje ICMP**: `Echo Request`, `Echo Reply`, `Time Exceeded`, `Destination Unreachable`, ...
    - **Identificador y secuencia**: utilizados para correlacionar solicitudes y respuestas de ping.
    - **Código adicional**: especifica detalles del tipo de error o control.
    - **Datos del paquete original**: en mensajes de error puede incluir parte del datagrama IP original.
  - **Ejemplos:**
    - `Echo (ping) request id=0x1234, seq=1/256, ttl=64` → Un host envía una solicitud de ping.
    - `Echo (ping) reply id=0x1234, seq=1/256, ttl=64` → El host destino responde al ping recibido.
    - `Destination unreachable (Port unreachable)` → Indica que no se puede alcanzar el puerto destino del paquete original.
    - `Time-to-live exceeded (TTL=0)` → El paquete fue descartado porque el TTL llegó a cero (típico en traceroute).

Ahora veamos, un DNS spoofing ocurre cuando un atacante responde falsamente una consulta DNS con información incorrecta, antes de que llegue la respuesta legítima. El objetivo es que el cliente resuelva un dominio legítimo hacia una IP falsa controlada por el atacante. Entonces, lo que podría suceder es que una consulta DNS legítima va del cliente (src) al servidor DNS configurado (dst). La respuesta legítima debe venir de ese mismo servidor DNS. Si aparece una respuesta con el mismo Transaction ID, pero desde otra IP, puede ser falsa. Si miramos en el wireshark, tenemos 4 paquetes DNS :

- La petición del cliente para resolver el dominio www.dc.uba.ar.

```
No.: 1
Time: 0.000000
Src: 172.16.78.128  cliente interno
Dst: 172.16.78.2  servidor DNS
Protocol: DNS
Info: Standard query 0x1bd5 A www.dc.uba.ar
```

- La respuesta del servidor DNS.

```
No.: 2
Time: 0.001628
Src: 172.16.78.2  servidor DNS
Dst: 172.16.78.128  cliente interno
Protocol: DNS
Info: Standard query response 0x1bd5 A www.dc.uba.ar CNAME www-1.dc.uba.ar CNAME dc.uba.ar A
    157.92.27.21
```

- La petición del cliente para resolver el dominio www.dc.uba.ar.

```
No.: 7
Time: 20.785478
Src: 172.16.78.128 cliente interno
Dst: 172.16.78.2  servidor DNS
Protocol: DNS
Info: Standard query 0x9c32 A www.dc.uba.ar
```

- La respuesta del servidor DNS.

```
No.: 8
Time: 20.787001
Src: 172.16.78.2  servidor DNS
Dst: 172.16.78.128  cliente interno
Protocol: DNS
Info: Standard query response 0x9c32 A www.dc.uba.ar A 127.0.0.1
```

Si miramos atentamente, en el paquete 1-2 (consulta/respuesta inicial) devolvía la IP 157.92.27.21 y el paquete 7-8 devuelve 127.0.0.1. Si bien, los números de transacción están bien, lo que llama la atención es que resuelve a dos IPs diferentes. En particular **127.0.0.1** (local host) lo que es evidente que no es la IP legítima de www.dc.uba.ar. Da la sensación que el atacante estaba **on-path** como un MITM, si vemos el tiempo que tardó en responder el server en la primer consulta en comparación con la última, la spoofeada es mucho más veloz. Concluimos que se trata de un DNS spoofing con un atacante on-path que ve el tráfico y manipula la IP de la consulta DNS.

8 Ejercicio 8

Vamos a introducir qué campos son los que vamos a poder observar en un paquete SMTP:

- **Comando/Respuesta SMTP:**
  - Comandos (cliente → servidor): `HELO` / `EHLO` (Primer comando que envía un cliente SMTP al servidor para identificarse. Se envía el nombre del cliente), `MAIL FROM`(Indica el remitente del correo), `RCPT TO`(Indica el destinatario del mensaje), `DATA`(Señala el inicio del contenido del correo (cabeceras + cuerpo)), `RSET`(Resetea la sesión SMTP.), `VRFY`(Pregunta al servidor si un usuario o dirección de correo es válida.), `NOOP`(Comando "sin operación"), `QUIT`(Termina la sesión), `AUTH LOGIN`(Comando de autenticación para enviar credenciales en base64.), `STARTTLS`(Comando que indica al servidor que el cliente quiere elevar la sesión a TLS para cifrar toda la comunicación a partir de ese punto), etc.
  - Respuestas (servidor → cliente): líneas con código numérico y texto, por ejemplo:
    - `220 smtp.example.com ESMTP Postfix`: Código 220: Servicio listo. Se envía al cliente al inicio de la conexión para indicar que el servidor SMTP está disponible. El texto indica el nombre del servidor y el software que corre (Postfix).
    - `250 Ok`: Código 250: Comando completado correctamente.
    - `354 End data with <CRLF>.<CRLF>`: Código 354: El servidor está listo para recibir los datos del correo después del comando DATA. El cliente debe enviar el mensaje y terminar con una línea que contenga solo un punto (.) para indicar el fin del contenido.
    - `550 Requested action not taken` : Código 550: Error permanente. El comando no se puede ejecutar, por ejemplo si el destinatario no existe o el servidor bloquea el mensaje por política de spam.

- **Códigos de respuesta SMTP (status codes):** El primer dígito indica clase:
  - `2xx` → éxito (ej. 250)
  - `3xx` → continuar (ej. 354 para inicio de DATA)
  - `4xx` → error transitorio (reintentar)
  - `5xx` → error permanente (rechazado)
- **Cabeceras y contenido de correo (dentro del comando DATA):**
  - Después de `DATA`, el cliente envía el mensaje completo en formato RFC-5322.
  - Headers típicos: `From:`, `To:`, `Date:`, `Subject:`, `Message-ID:`, `MIME-Version:`, `Content-Type:`, ...
  - Cuerpo del mensaje: texto plano o MIME .
- **MIME / adjuntos:**
  - Si el mensaje usa `Content-Type: multipart/mixed; boundary=...`, se ven las partes MIME y los adjuntos codificados (p. ej. base64).
  - Se pueden seleccionar los bytes en el panel inferior para exportarlos y decodificar manualmente.
- **Autenticación y STARTTLS:**
  - Comandos como `AUTH LOGIN` / `AUTH PLAIN` aparecen en texto (a veces las credenciales están en base64 si no se usa TLS).
  - `STARTTLS` inicia negociación TLS: tras un `220 Ready to start TLS`.

Ahora que sabemos todo esto, podemos chusmear los paquetes:

- El cliente indica que quiere autenticarse usando CRAM-MD5, que es un mecanismo de autenticación basado en challenge-response.

```
No.: 10
Time: 0.000271
Src: 127.0.0.1  cliente
Dst: 127.0.0.1  servidor
Protocol: SMTP
Info: C: AUTH CRAM-MD5
```

- 334 indica que el servidor está enviando un challenge en base64. Este challenge es un valor aleatorio que el cliente va a usar para generar un hash HMAC-MD5 con su contraseña. O sea,

$$\text{response} = \text{username} + \text{HMAC-MD5}(\text{challenge, password})$$

```
No.: 12
Time: 0.000323
Src: 127.0.0.1  servidor
Dst: 127.0.0.1  cliente
Protocol: SMTP
Info: S: 334 PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9uLm1jaS5uZXQ+
```

- Responde el challenge : dXNlcm5hbWUgMDZkMGEzMjVmMDU0NjQ4NjQ2ZTA3MmNkNGZlYjE3YzQ=

```
No.: 14
Time: 0.000462
Src: 127.0.0.1  cliente
Dst: 127.0.0.1  servidor
Protocol: SMTP
Info: C: DATA fragment, 58 bytes
```

- Se autentica correctamente

```
No.: 16
Time: 0.000512
Src: 127.0.0.1  servidor
Dst: 127.0.0.1  cliente
Protocol: SMTP
Info: S: 235 Okay
```

Ahora tenemos que encontrar la contraseña. Para eso, decodificamos de base64
PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9uLm1jaS5uZXQ+
$\Longrightarrow$ <1896.697170952@postoffice.reston.mci.net> y el response del cliente :
dXNlcm5hbWUgMDZkMGEzMjVmMDU0NjQ4NjQ2ZTA3MmNkNGZlYjE3YzQ
$\Longrightarrow$ username 06d0a325f054648646e072cd4feb17c4
Este último nos interesa ya que tiene el **HMAC-MD5(challenge, password)** para poder sacar la contraseña. Vamos a usar John The Ripper. Lo primero es chequear el formato :

```
 ./john --list=formats
descrypt, bsdicrypt, md5crypt, md5crypt-long, bcrypt, scrypt, LM, AFS,
...
OpenVMS, vmx, VNC, vtp, wbb3, whirlpool, whirlpool0, whirlpool1, wpapsk,
wpapsk-pmk, xsha, xsha512, zed, ZIP, ZipMonster, plaintext, has-160,
HMAC-MD5, HMAC-SHA1, HMAC-SHA224, HMAC-SHA256, HMAC-SHA384, HMAC-SHA512,
dummy, crypt
```

Dentro de los formatos que acepta, en particular está **HMAC-MD5**, así que chiche. Vamos a crear el `challenge.hash` con el user + challenge, o sea :

$$username :< 1896.697170952@postoffice.reston.mci.net > \#06d0a325f054648646e072cd4feb17c4$$

A continuación vamos pasarle a John lo siguiente :

```
    ./john --format=HMAC-MD5 --wordlist=.../wordlists/SecLists/Passwords/Cracked-Hashes/milw0rm-dictionary
        .txt --rules .../SegInf/challenge.hash

Using default input encoding: UTF-8
Loaded 1 password hash (HMAC-MD5 [password is key, MD5 256/256 AVX2 8x3])
Warning: poor OpenMP scalability for this hash type, consider --fork=4
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
Enabling duplicate candidate password suppressor using 256 MiB
password         (username)
1g 0:00:00:00 DONE (2025-10-15 19:17) 7.692g/s 567138p/s 567138c/s 567138C/s jack1549..sony5491
Use the "--show --format=HMAC-MD5" options to display all of the cracked passwords reliably
Session completed
```

O sea que la encontró, y la contraseña es **password**. Podemos usar la opción `--show` :

```
 ./john --format=HMAC-MD5 --show .../SegInf/challenge.hash
username:password

1 password hash cracked, 0 left
Session completed
```

9 Ejercicio 9

En este ejercicio nos piden analizar el tráfico de un cliente que bajó un binario y lo ejecutó en su máquina y como consecuencia fue atacado. La página que nos permitirá analizar ese binario es VirusTotal. Lo primero que vemos es que el usuario hace un `GET` de un recurso :

```
No.: 4
Time: 0.178280
Src: 10.200.2.252  cliente
Dst: 185.91.175.64  servidor
Protocol: HTTP
Info: C: GET /jsaxo8u/g39b2cx.exe HTTP/1.1
```

Luego el servidor le da el OK y devuelve el header (tamaño,...):

```
No.: 98
Time: 1.152502
Src: 185.91.175.64  servidor
Dst: 10.200.2.252  cliente
Protocol: HTTP
Info: S: HTTP/1.1 200 OK
```

A partir de este momento, se manda fragmentado en paquetes TCP el binario. Por suerte, podemos hacer el seguimiento. Nos paramos en el `GET`, click derecho, *follow* > *TCP Stream* y ponemos `show as RAW`. La guardamos como `conversation-ojo-malware.ra`
y se la pasamos a VirusTotal :

AliCloud	Worm:Win/Drixed.D	ALYac	Trojan.Dridex.C
Arcabit	Trojan.Dridex.C	Avast	Win32:Dridex-Q [Cryp]
AVG	Win32:Dridex-Q [Cryp]	BitDefender	Trojan.Dridex.C
ClamAV	Win.Trojan.Dridex-22	CTX	Zip.trojan.dridex
DrWeb	Trojan.Dridex.76	Emsisoft	Trojan.Dridex.C (B)
Fortinet	W32/Dridex.HLItr	GData	Trojan.Dridex.C
Google	Detected	Ikarus	Trojan-Downloader.Win32.Drixed
Kaspersky	Worm.Win32.Cridex.pxs	Lionic	Worm.Text.Dridex.o!c
Microsoft	TrojanDownloader:Win32/Drixed.D	NANO-Antivirus	Trojan.Win32.Cridex.dqawyj
Sophos	Mal/EncPk-AOI	Symantec	Trojan.Gen.2
Trellix ENS	Downloader-FARC!107A3BEF0DA9	TrendMicro	TSPY_DRIDEX.UKL
TrendMicro-HouseCall	TSPY_DRIDEX.UKL	Varist	W32/ABTrojan.NWRI-6261
VIPRE	Trojan.Dridex.C	Acronis (Static ML)	Undetected

Es decir que el binario es reconocido como Malware, en particular un Troyano Dridex. Un troyano bancario que roba credenciales y puede descargar otros malwares.
En la captura tenemos al principio unos encabezados:

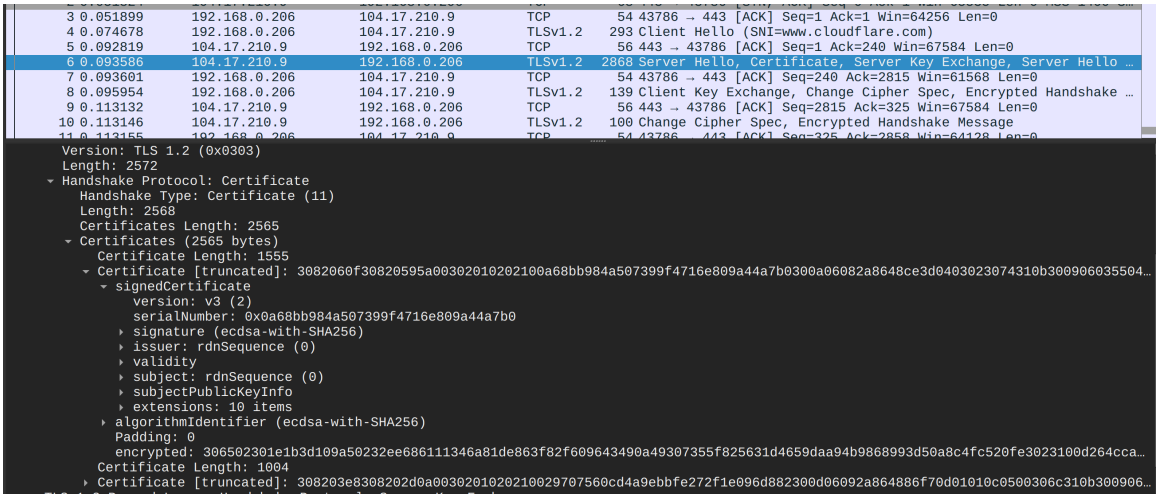
```
GET /jsaxo8u/g39b2cx.exe HTTP/1.1
Host: 185.91.175.64
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: Microsoft-IIS/8.5
Date: Tue, 31 Mar 2015 11:32:49 GMT
Content-Type: application/octet-stream
Content-Length: 84480
Connection: keep-alive
Last-Modified: Tue, 31 Mar 2015 07:41:13 GMT
ETag: "5000238-14a00-51290b915fcee"
Accept-Ranges: bytes
```

10 Ejercicio 10

Para este ejercicio nos vamos a encontrar con paquetes que se corresponden con el protocolo TLS (Transport Layer Security). Si miramos el tráfico en `tlsb.pcap` la versión de TLS es la 1.2 y nos podemos encontrar con que el handshake TLS en esta versión ocurre como sigue :

- **Client Hello** : El cliente inicia la conexión, propone versión de TLS, *cipher suites* y extensiones (como SNI).
- **Server Hello** : El servidor responde con la versión y cifrado seleccionados, y envía su número aleatorio.
- **Certificate** : El servidor envía su certificado para que el cliente pueda verificar su identidad.
- **Server Key Exchange (opcional)** : Envía parámetros de cifrado adicionales si el algoritmo lo requiere.
- **Server Hello Done** : Indica que el servidor terminó su parte del handshake.
- **Client Key Exchange** : El cliente envía su parte para generar la clave simétrica compartida.
- **Change Cipher Spec y Finished** : Ambos lados confirman que usarán la clave simétrica y verifican que el handshake fue correcto. A partir de aquí, el tráfico de aplicación se transmite cifrado.

En esta captura, vemos que el certificado está en claro:



La diferencia con la versión 1.3 es que el certificado se envía dentro del canal cifrado, por lo que no se ve en la captura. Además es más seguro ya que el handshake es más corto y simplificado. Muchos mensajes se combinan : el "Server Key Exchange" y "Server Hello Done" desaparecen como mensajes separados. Quedaría :

- **Client Hello** : El cliente inicia la conexión, propone versión TLS, *cipher suites*, extensiones (como SNI) y, si aplica, su clave efímera para el intercambio de claves.
- **Server Hello** : El servidor responde con la versión y *cipher suite* seleccionadas, su número aleatorio y su parte de la clave efímera.
- **Encrypted Extensions** : Mensaje cifrado del servidor con información adicional opcional (como ALPN).
- **Certificate (cifrado)** : El servidor envía su certificado dentro del canal cifrado, por eso no es visible en la captura.
- **Certificate Verify** : Mensaje que prueba criptográficamente que el servidor posee la clave privada del certificado.
- **Finished** : Confirma que ambas partes calcularon la misma clave de sesión. Desde este punto, todo el tráfico de aplicación se transmite cifrado.

Como consecuencia, TLSv1.3 al ocultar el certificado entre otras cosas, logra:

- **Protección de la identidad del servidor**: Evita que un observador pasivo vea el certificado y, por lo tanto, a qué servidor se está conectando el cliente.
- **Prevención de fingerprinting** : Dificulta que un atacante identifique servidores específicos a partir de la información del certificado. Esto se debe a que los datos del certificado (emisor, dominio, claves) pueden ser usados para reconocer servidores.
- **Mayor privacidad para el usuario** : La conexión a un dominio específico revela hábitos o información sensible del usuario.
- **Reducción de exposición de datos sensibles** : Información contenida en el certificado (subdominios, organización, emails) no queda visible en la red. Esto previene que atacantes recopilen información que pueda ser usada para ataques futuros o ingeniería social.

11 Ejercicio 11

En este ejercicio vamos a ver una captura de una comunicación wiresless y vamos a usar Aircrack-ng para descifrarla. Primero miremos la captura por arriba, nos vamos a encontrar con por ejemplo:
Un **beacon** es un mensaje publicitario que el AP envía constantemente para anunciar la existencia de su red Wi-Fi. Con este mensaje permite a los dispositivos:

- Saber que la red existe (SSID).
- Conocer el AP que la transmite (BSSID / MAC).
- Saber qué tipo de cifrado usa (WEP/WPA/WPA2).
- Conocer capacidades como velocidad soportada, canal, modo de operación.

Si miramos en wireshark, vamos a ver lo siguiente :

- **Número de paquete**.
- **Tiempo relativo**.
- **Remitente (BSSID):** CiscoLinksys_82:b2:55 — MAC del emisor; suele ser el punto de acceso (AP).
- **Destino:** Broadcast — indica que la trama va dirigida a todos los dispositivos.
- **Protocolo:** 802.11 — tipo de trama (Wi-Fi).
- **Longitud:** tamaño total del paquete.
- **Tipo de trama:** Beacon frame — las beacons son anuncios periódicos del punto de acceso que informan de la existencia y capacidades de la red.
- **SN (Sequence Number):** número de secuencia del paquete.
- **FN (Fragment Number):** número de fragmento (0 si no está fragmentado).
- **BI (Beacon Interval):** BI=100, intervalo de emisión de beacons en unidades de 1.024 ms (100 $\approx$ 102.4 ms).
- **SSID:** SSID="Coherer" — nombre (identificador) de la red anunciada.
- **Notas adicionales:**
  - Las beacons incluyen parámetros etiquetados (tagged parameters) como tasas soportadas, capacidades y, si aplica, el elemento RSN/WPA que indica el tipo de cifrado (p. ej. WPA2/CCMP).
  - *Una beacon no contiene la contraseña.* Para romper o verificar una passphrase WPA/WPA2 necesitamos el **4-way handshake** (paquetes EAPOL).

Encontraremos paquetes de tipo **probe request** un mensaje que envía un cliente para descubrir APs. Contiene el SSID que el cliente quiere encontrar.
Busquemos en el panel el 4-way handshake :

- El cliente busca la red “Coherer” enviando un mensaje de sondeo. No contiene datos ni indica cifrado; solo pregunta si la red existe.

```
No.: 58
Time: 5.180060
Src: Apple_82:36:3a  cliente
Dst: Broadcast
Protocol: 802.11
Info: Probe Request, SSID="Coherer"
```

- El cliente intenta autenticarse con el AP. Forma parte del proceso inicial de conexión. Por sí sola, esta trama no indica cifrado.

```
No.: 78
Time: 5.643955
Src: Apple_82:36:3a  cliente
Dst: CiscoLinksys_82:b2:55  AP
Protocol: 802.11
Info: Authentication
```

- El cliente solicita al AP asociarse a la red. Esta es la segunda fase del proceso de conexión, aún sin cifrado de datos.

```
No.: 82
Time: 5.645953
Src: Apple_82:36:3a  cliente
Dst: CiscoLinksys_82:b2:55  AP
Protocol: 802.11
Info: Association Request, SSID="Coherer"
```

- El AP acepta la solicitud de asociación del cliente, permitiendo que continúe el proceso de cifrado.

```
No.: 84
Time: 5.647953
Src: CiscoLinksys_82:b2:55  AP
Dst: Apple_82:36:3a  cliente
Protocol: 802.11
Info: Association Response
```

- Trama de control que indica al cliente que puede transmitir datos. No contiene información de cifrado.

```
No.: 86
Time: 5.648961
Src: CiscoLinksys_82:b2:55  AP
Dst: (RA)
Protocol: 802.11
Info: Clear-to-send
```

- Primer paquete del 4-way handshake WPA/WPA2. Confirma que la red "Coherer" está cifrada; este mensaje inicia el intercambio de claves que protegerá los datos.

```
No.: 87
Time: 5.649953
Src: CiscoLinksys_82:b2:55  AP
Dst: Apple_82:36:3a  cliente
Protocol: EAPOL
Info: Key (Message 1 of 4)
```

- Segundo paquete del 4-way handshake. El cliente responde al AP con parte de la información necesaria para generar la clave de cifrado temporal.

```
No.: 89
Time: 5.650959
Src: Apple_82:36:3a  cliente
Dst: CiscoLinksys_82:b2:55  AP
Protocol: EAPOL (802.1X Authentication)
Info: Key (Message 2 of 4)
```

- Tercer paquete del handshake. El AP envía al cliente la información restante para derivar la clave de cifrado y confirma la autenticación.

```
No.: 92
Time: 5.655957
Src: CiscoLinksys_82:b2:55  AP
Dst: Apple_82:36:3a  cliente
Protocol: EAPOL (802.1X Authentication)
Info: Key (Message 3 of 4)
```

- Cuarto y último paquete del 4-way handshake. El cliente confirma la recepción de la clave derivada. A partir de este momento, el tráfico entre AP y cliente estará cifrado.

```
No.: 94
Time: 5.655973
Src: Apple_82:36:3a  cliente
Dst: CiscoLinksys_82:b2:55  AP
Protocol: EAPOL (802.1X Authentication)
Info: Key (Message 4 of 4)
```

A continuación vemos el siguiente paquete : una trama de datos enviada desde el AP al cliente.

```
No.: 102
Time: 5.846994
Src: CiscoLinksys_82:b2:53  AP
Dst: Apple_82:36:3a  cliente
Protocol: 802.11
Info: Data, Protected
```

- Está cifrada usando la clave derivada del 4-way handshake WPA/WPA2.
- La letra 'p' en las flags indica que es un **Protected Frame**.

Sabemos que la está protegida, y en particular que se trata de WPA/WPA2 por que dentro del beacon, en el campo de **tagged parameters** tenemos el **Tag : RSN(Robust Security Network) information**:

- **Tag Number: 48 (RSN Information)** : Este campo indica que la red utiliza **WPA2/WPA** y define que se trata de un RSN (Robust Security Network).
- **RSN Version: 1** : Versión del estándar RSN utilizado por la red.
- **Group Cipher Suite: TKIP** : Cifrado usado para tráfico broadcast/multicast es decir, paquetes enviados a todos los clientes de la red.
- **Pairwise Cipher Suite List: AES (CCM), TKIP** : Cifrado usado para tráfico unicast entre el AP y cada cliente individual.
- **Auth Key Management (AKM) List: PSK** : Método de autenticación: PSK (Pre-Shared Key indica que la red es WPA/WPA2 Personal y usa una contraseña compartida.
- **RSN Capabilities: 0x0000** : Opciones adicionales de seguridad definidas por el estándar, como soporte para preautenticación o control de replay.

Adelantemos un poco el ejercicio, vemos todo como `EAPOL` o `802.11`, tenemos que descifrar la comunicación y traernos todos los pedidos HTTP. Para ello, necesitamos:

- **Tener el 4-way handshake completo.**
- **Conocer la passphrase WPA/WPA2.**
- **Configurar Wireshark para descifrar:**
  - Ir a: `Edit` → `Preferences` → `Protocols` → `IEEE 802.11`
  - Marcar `Enable decryption`.
  - En `Decryption Keys` agregar la clave en el formato adecuado, por ejemplo:
    - `wpa-pwd:mi_clave:Coherer` (passphrase en texto + SSID)
    - o `wpa-psk:0123456789abcdef...` (PSK en hexadecimal)
- **Verificar el descifrado:** Tras añadir la clave y si la captura contiene el handshake correcto, las tramas que antes aparecían como `Data, Protected` deberían mostrarse con protocolos superiores (IP/TCP/HTTP).
- **Filtrar pedidos HTTP.**

Vamos a hallar la contraseña. En este caso, no es como en ejercicios anteriores que hay un hash que lo atacamos y sale. En este caso, el funcionamiento es como sigue:

- **PMK (Pairwise Master Key)**: Clave maestra derivada de la **passphrase** y el **SSID** mediante `PBKDF2-HMAC-SHA1` (4096 iteraciones). Habitualmente tiene una longitud de 256 bits (32 bytes). La PMK no cifra el tráfico directamente; se usa como entrada para derivar la PTK, que contiene las subclaves utilizadas en la sesión.
- **PTK y KCK** : A partir de la PMK y de los nonces/MACs del handshake se deriva la **PTK (Pairwise Transient Key)**. La PTK se divide en subclaves, entre ellas la **KCK (Key Confirmation Key)**, que se usa para calcular el MIC.
- **MIC (Message Integrity Code)**: Código de integridad calculado sobre el mensaje EAPOL usando la subclave **KCK** y el algoritmo MAC negociado en RSN (por ejemplo HMAC-SHA1 truncado o AES-CMAC, según el descriptor). El MIC garantiza que el mensaje no fue alterado y permite confirmar que la parte que envía conoce la clave correcta.
- **Qué hace Aircrack ?**:
  1. Toma una passphrase candidata y calcula la PMK usando el SSID.
  2. Deriva la PTK (y la KCK) a partir de la PMK y los nonces/MAC del handshake.
  3. Calcula el MIC esperado sobre el EAPOL con la KCK y lo compara con el MIC observado en la captura.
  4. Si coinciden, la passphrase candidata es correcta.

Por ello, tenemos que mirar qué nos ofrece `aircrack-ng` y cómo utilizarlo :

```
1
2 usage: aircrack-ng [options] <input file(s)>
3 ...
4
5 WEP and WPA-PSK cracking options:
6
7       -w <words> : path to wordlist(s) filename(s)
8       -N <file>  : path to new session filename
9       -R <file>  : path to existing session filename
```

Por recomendación de la consigna usamos un diccionario en inglés. Vamos a https://contest-2010.korelogic.com/wordlists.html y los descargamos. En el directorio que querramos almacenar diccionarios o cosas (movemos los .dic.gz) y luego:

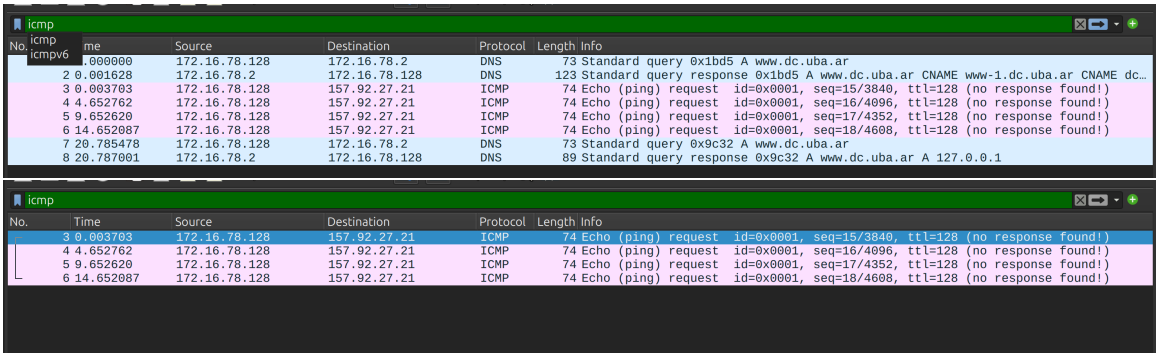
```
gunzip -c dictionary_from_queue_1.dic.gz > dict1.txt
```

Ahora, ya podemos correr el **aircrack** :

```
aircrack-ng -w .../wordlists/dict1.txt wpa-Induction.pcap
Reading packets, please wait...
Opening wpa-Induction.pcap
Read 1093 packets.

   #  BSSID              ESSID                     Encryption

   1  00:0C:41:82:B2:55  Coherer                   WPA (1 handshake, with PMKID)
   2  65:78:F7:B7:30:84                            Unknown
   3  65:78:F7:B7:60:A9                            Unknown
   4  81:F8:47:33:56:BB                            Unknown
   5  92:F3:65:74:D2:DB                            Unknown
   6  98:D3:04:64:FA:55                            WPA (0 handshake)
   7  F4:9F:8F:EA:7B:E6                            Unknown
   8  FF:FF:FF:FF:FF:3F                            WEP (0 IVs)

Index number of target network ? 1

Reading packets, please wait...
Opening wpa-Induction.pcap
Read 1093 packets.

1 potential targets



                               Aircrack-ng 1.7

      [00:01:07] 256279/256279 keys tested (3910.31 k/s)

      Time left: --

                                KEY NOT FOUND


      Master Key     : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

      Transient Key  : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

      EAPOL HMAC     : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

No funcionó con este, hay que seguir intentando.
Hablé con el profe, Ubuntu no tiene preinstalado un diccionario en Inglés y no encontré ninguno. Si tienen Kali, me dijo que sale. Los pasos son como ya mencioné : encontrar la clave (es fácil según me dijo el profe) y con esto seguir los pasos para poder ver en claro cuáles son paquetes HTTP,TCP,..

Después, los filtramos, dejo este ejemplo :



O sea, en la barra, ponemos HTTP. A continuación, queremos guardarlos en otro archivo. Para ello: *File* → *Export Specified Packets...* en la ventana que aparece, *Packet Range* → *Displayed* (sólo se guardan los que cumplen el filtro).