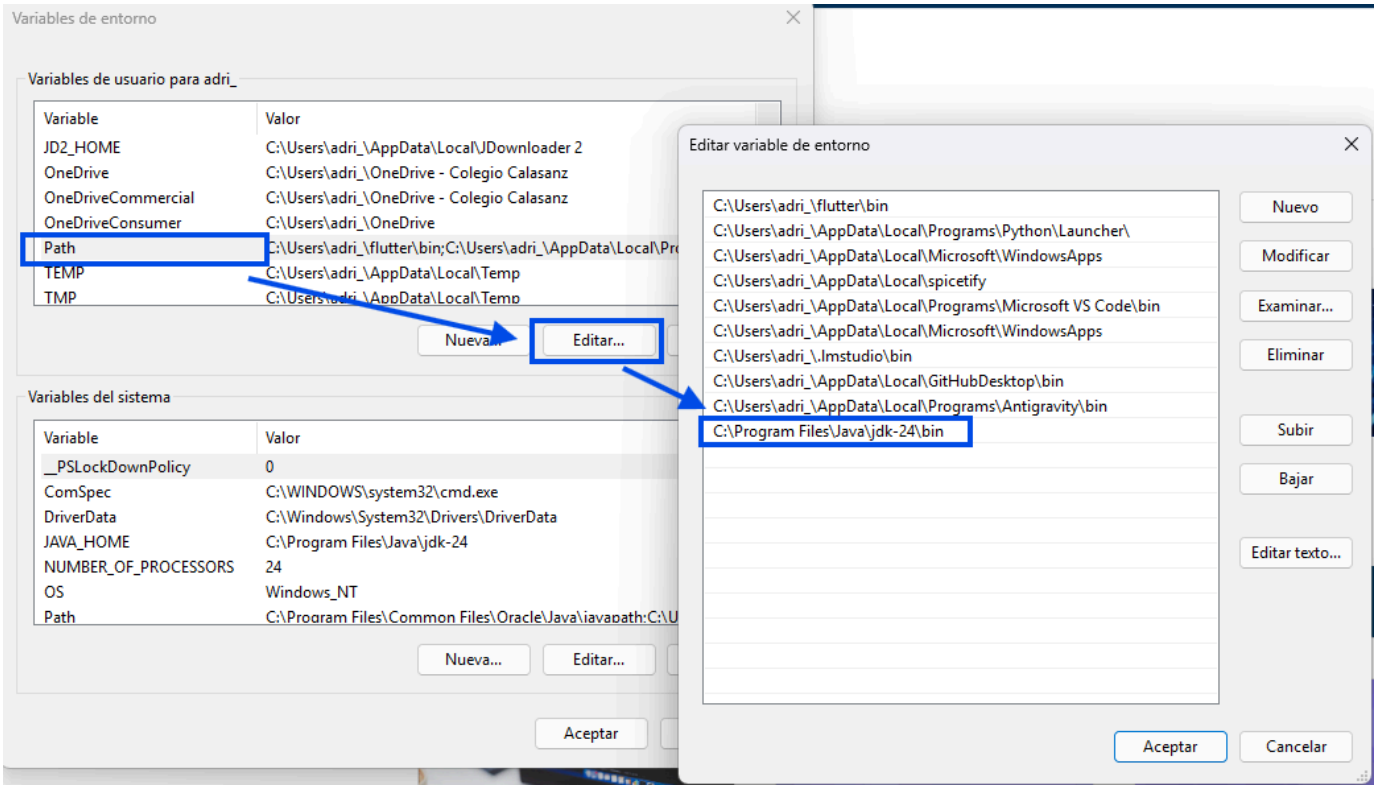
PRÁCTICA: CIFRADO HTTPS CON CERTIFICADOS SSL/TLS EN SPRING BOOT

- En esta práctica implementaremos un certificado SSL/TLS en un aplicación Spring Boot.
- Crearemos un login simple con Spring Boot, para comprobar como sin certificado los datos no están seguros y protegidos. Y luego aprenderemos a protegerlos.

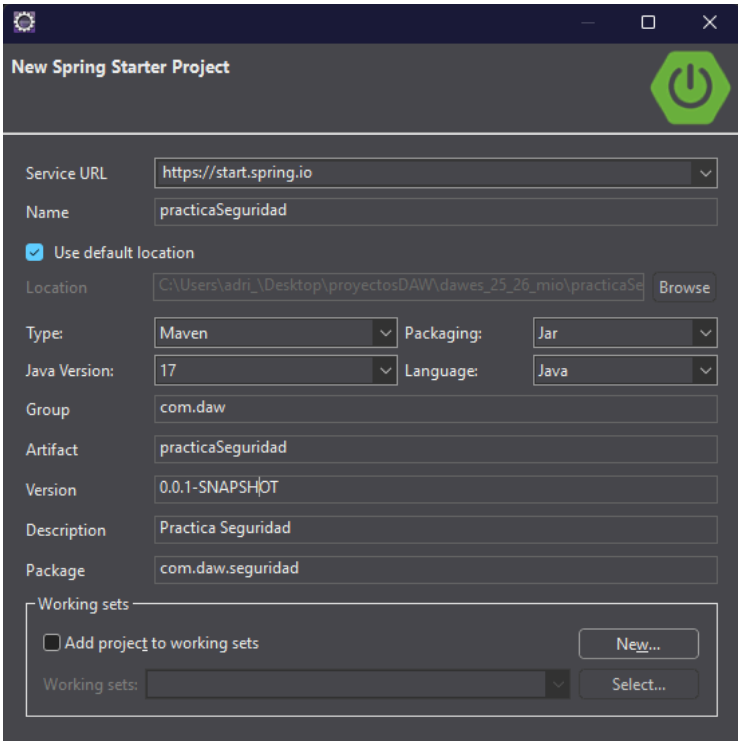
Parte 1: Instalación y configuración

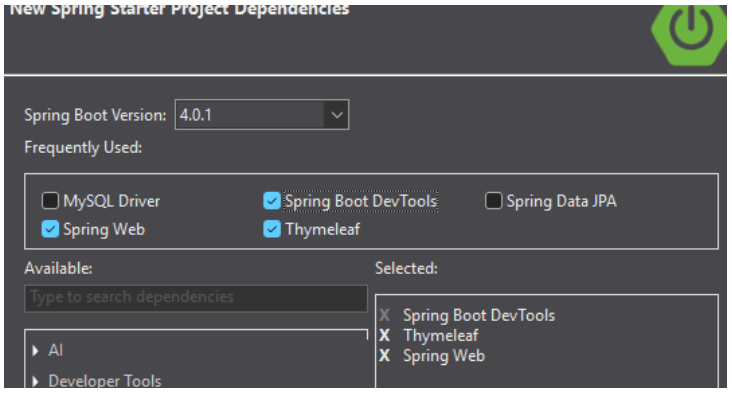
- Ya contamos con lo que necesitamos: El IDE de Eclipse y Whireshark. Si lo ultimo no lo tenéis, descargarlo en https://www.wireshark.org/download.html
- Asegúrate también que tienes configurado el JDK en Java en las variables del PATH de Windows, usaremos un comando que lo necesita



Creación Proyecto Spring Boot

- Crearemos un nuevo proyecto de Spring Boot, con las dependencias Spring Web, Thymeleaf y Spring Boot DevTools. Como hemos hecho más veces en Servidor.





- En este proyecto probaremos un login simple, donde recibiremos un usuario y contraseña. Creamos un nueva clase, que será nuestro controlador controlador. Tendrá dos métodos:
  - `@GetMapping("/")`: Mapeamos la ruta por defecto, para que nos redirija a una vista `login` que crearemos
  - `@PostMapping("/login")`: Cuando el usuario introduzca sus datos, se producirá el post, donde he añadido una validación simple de usuario y contraseña, que devuelve si es correcto o no el usuario. Por simplificar, lo hago todo en el controlador, pero no seria una buena practica.

```java
@Controller
public class ControladorLogin {

    @GetMapping("/")
    public String login() {
        return "login";
    }

    @PostMapping("/login")
```

```
    public String login(@RequestParam("usuario") String usuario,
                        @RequestParam("password") String password,
                        Model model) {

        // Validación simple (NO USAR EN PRODUCCIÓN)
        if ("admin".equals(usuario) && "contraseña1234".equals(password)) {
            model.addAttribute("correcto", "¡Login exitoso! Bienvenido " + usuario);
            return "login";
        } else {
            model.addAttribute("error", "Credenciales inválidas");
            return "login";
        }
    }
}
```

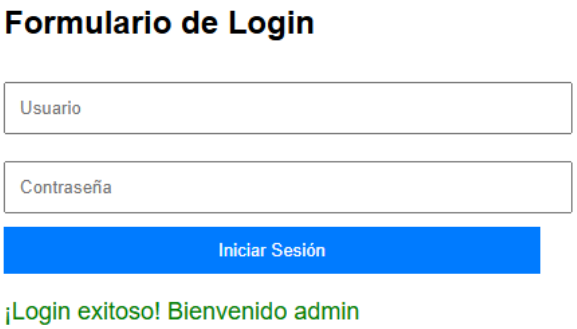
- Crearemos a continuación una simple vista de thymeleaf en la carpeta `resources/templates` que llamaremos `login.html`

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta charset="UTF-8">
        <title>Login - SSL Demo</title>
        <style>
            body { font-family: Arial; max-width: 400px; margin: 50px auto; padding: 20px;}
            input { width: 100%; padding: 10px; margin: 10px 0; }
            button { width: 100%; padding: 10px; background: #007bff; color: white; border: none; cursor: pointer; }
            .error { color: red; }
            .correcto { color: green; font-size: 18px; }
        </style>
    </head>
    <body>
        <h2>Formulario de Login</h2>
        <form method="post" action="/login">
            <input type="text" name="usuario"  placeholder="Usuario" required />
            <input type="password" name="password" placeholder="Contraseña" required />
            <button type="submit">Iniciar Sesión</button>
        </form>
        <p class="error" th:if="${error}" th:text="${error}"></p>
        <p class="correcto"  th:if="${correcto}"  th:text="${correcto}"></p>
    </body>
</html>
```

- Normalmente el puerto por defecto de Spring Boot es 8080, pero vamos a indicarselo en `application.properties`, añadiendo `server.port=8080`:

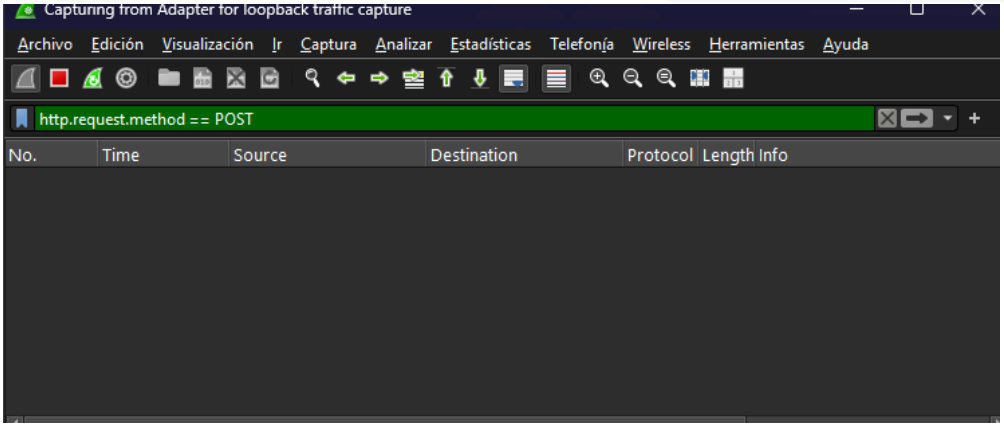
```
spring.application.name=practicaSeguridad
server.port=8080
```

- Si arrancamos el servidor y accedemos a localhost:8080 deberíamos tener algo como esto:

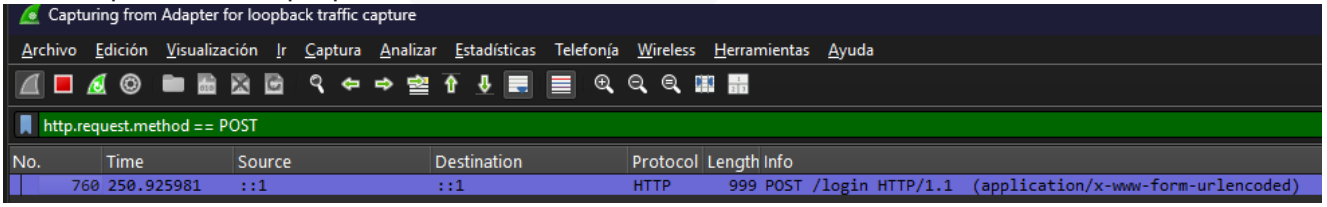


Parte 2: Captura de Tráfico HTTP con Wireshark

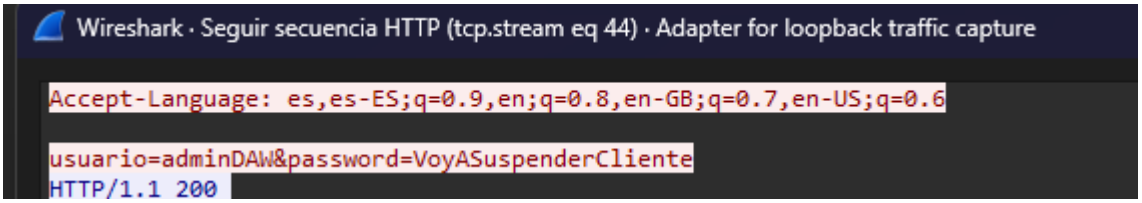
- Vamos a probar ahora a capturar el trafico de nuestra aplicación con Wireshark
- Abrimos Wireshark, seleccionamos la interfaz loopback
  - Filtramos por `http` o si queremos ser más específicos por `http.request.method == POST`



- Vamos a generar el trafico. Realizamos ahora un login en la aplicación, con cualquier usuario o contraseña.
- Debería apareceros el paquete `POST /login HTTP/1.1`



- Si le damos a click derecho → Seguir → HTTP Stream. Se os abrirá una ventana con toda la información del paquete. Entre todo ese contenido deberias ver algo como `usuario=USUARIO&password=CONTRASEÑA`



- Como veis la contraseña y el usuario que habéis puesto. Esto es peligroso. Las credenciales viajan en **texto plano** y cualquier atacante con acceso a la red puede interceptarlas (man-in-the-middle attack)

Parte 3: Generar Certificado SSL Autofirmado

- Vamos a crear ahora un certificado para cifrar las credenciales y usarlo en Spring Boot.
- Para ello, en un cmd, y nos dirigimos al escritorio (u otra que queráis, o podéis dejar la defecto). Yo pongo escritorio para tener localizado el certificado que vamos a generar.

```
cd Desktop
```

- Y ahora que estamos en la carpeta seleccionada, creamos el Certificado con `keytool` un comando que incorpora Java es su JDK (de hay que reviséis si esta en el PATH, si no lo esta no lo reconoce)

```
keytool -genkeypair -alias certificadoSpringBoot -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore keystore.p12 -validity 365 -storepass daw2026
```

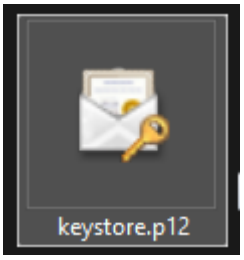
  - Parámetros explicados:
    - `-genkeypair`: Genera un par de claves (pública/privada)
    - `-alias certificadoSpringBoot`: Nombre identificativo del certificado
    - `-keyalg RSA`: Algoritmo de cifrado asimétrico
    - `-keysize 2048`: Longitud de la clave (bits)
    - `-storetype PKCS12`: Formato estándar del keystore
    - `-keystore keystore.p12`: Archivo de salida
    - `-validity 365`: Días de validez del certificado
    - `-storepass daw2026`: Contraseña del keystore
- Os pedirá una serie de datos. **Lo importante es lo primero. Tenéis que poner como nombre, como CN, localhost.**

```
C:\Users\adri_>cd Desktop

C:\Users\adri_\Desktop>keytool -genkeypair -alias certificadoSpringBoot -keyalg
store keystore.p12 -validity 365 -storepass daw2026
Enter the distinguished name. Provide a single dot (.) to leave a sub-component
value in braces.
What is your first and last name?
  [Unknown]:  localhost
What is the name of your organizational unit?
  [Unknown]:  DAW
What is the name of your organization?
  [Unknown]:  Calasanz
What is the name of your City or Locality?
  [Unknown]:  Salamanca
What is the name of your State or Province?
  [Unknown]:  Castilla y Leon
What is the two-letter country code for this unit?
  [Unknown]:  ES
Is CN=localhost, OU=DAW, O=Calasanz, L=Salamanca, ST=Castilla y Leon, C=ES corre
  [no]:  YES

Generating 2048-bit RSA key pair and self-signed certificate (SHA384withRSA) wit
        for: CN=localhost, OU=DAW, O=Calasanz, L=Salamanca, ST=Castilla y Leon,
```

- Os habrá creado un certificado en la carpeta seleccionada:



- Y podemos verificarlo con `keytool -list -v -keystore keystore.p12 -storepass daw2026`

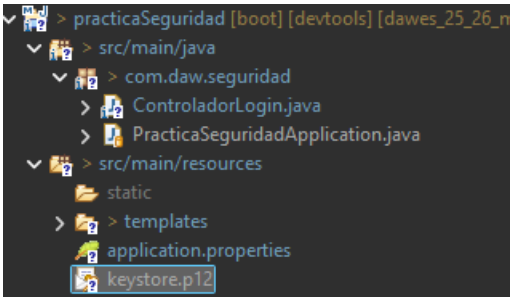
```
C:\Users\adri_\Desktop>keytool -list -v -keystore keystore.p12 -storepass daw2026
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: certificadospringboot
Creation date: 17 ene 2026
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=localhost, OU=DAW, O=Calasanz, L=Salamanca, ST=Castilla y Leon, C=ES
Issuer: CN=localhost, OU=DAW, O=Calasanz, L=Salamanca, ST=Castilla y Leon, C=ES
```

Parte 4: Configurar Spring Boot con HTTPS

- Para usar un certificado en Spring Boot, deberemos mover nuestro archivo `keystore.p12` a la carpeta `resources`



- Y deberemos actualizar `application.properties` con lo siguiente:

```properties
spring.application.name=practicaSeguridad
# Configuración HTTPS
server.port=8443
server.ssl.enabled=true
server.ssl.key-store=classpath:keystore.p12
server.ssl.key-store-password=daw2026
server.ssl.key-store-type=PKCS12
server.ssl.key-alias=certificadoSpringBoot
```

  - **Configuración explicada:**
    - `server.port=8443`: Puerto estándar para HTTPS
    - `server.ssl.enabled=true`: Activa SSL/TLS
    - `server.ssl.key-store`: Ruta al keystore (dentro de resources)
    - `server.ssl.key-store-password`: Contraseña del keystore
    - `server.ssl.key-store-type`: Tipo de almacén (PKCS12)

        - `server.ssl.key-alias`: Alias del certificado a usar
- Si reiniciamos la aplicación, ahora solo estará disponible en **https://localhost:8443**
    - Si da algún error, probad a hacer `Maven` → `Update project`
    - Verás una **advertencia de seguridad** porque el certificado es autofirmado. Esto es por que autofirmado, lo hemos creado nosotros, y no la firmado una Autoridad Certificadora (CA) como Let's Encrypt o DigiCert. Clic en **Avanzado** → **Acceder a localhost (sitio no seguro)**. (Esto para Chrome/Edge, en otros navegadores tendrán algo similar)

Parte 5: Verificación y pruebas

- **Abre el certificado desde la web**, y comprueba que los datos de emisión, y validez son los que hemos puesto cuando lo creamos.
- Vamos ahora a **Captura de Tráfico HTTPS con Wireshark**
    - En Whireshark filtramos por el puerto: `tcp.port == 8443`
    - Realiza un login desde la app de Spring Boot
    - Analiza los paquetes y haz capturas de:
        - El **TLS Handshake** (Client Hello, Server Hello, Certificate, Change Cipher Spec...)
        - Localiza el **Application Data** (datos cifrados del POST) y haz `Seguir` Seguir → `TCP Stream`. ¿Encuentras por algún lado el usuario y la contraseña? ¿O están cifrados?

Parte 6: Preguntas

1. Imagina que desarrollas una aplicación de banca online y por error la despliegas sin HTTPS. Describe **tres tipos de ataques** que un atacante podría realizar interceptando el tráfico HTTP. Para cada ataque, explica qué información podría robar y qué consecuencias tendría para el usuario.
2. En esta práctica hemos usado un certificado autofirmado. Investiga qué es una Autoridad Certificadora (CA) y explica:
    - ¿Por qué los navegadores confían en certificados firmados por CAs reconocidas?
    - ¿Qué diferencias existen entre usar un certificado autofirmado vs uno firmado por Let's Encrypt en producción?
    - ¿Qué pasaría si un sitio de comercio electrónico usara un certificado autofirmado?

Evaluación

- Haz capturas del todo el proceso, comentando/explicando que haces brevemente. Responde también a las preguntas
- Práctica relacionada con RA4
- **Penalización por faltas de ortografía**: -0.25 por falta (máximo -2 en la nota final)

Criterio	Excelente - 3 pts	Bien - 2 pts	Suficiente - 1pt	Insuficiente - 0pt
Captura y Análisis HTTP sin cifrar (Wireshark)	Captura correcta del tráfico HTTP con filtro apropiado (`http` o `http.request.method==POST`), identifica claramente el paquete POST, localiza y muestra las credenciales en texto plano mediante HTTP Stream, capturas bien documentadas y comentadas	Captura correcta del tráfico HTTP, identifica las credenciales en texto plano pero sin usar HTTP Stream o documentación básica, capturas sin comentarios explicativos	Captura realizada pero con filtros incorrectos o incompleta, dificultad para localizar las credenciales, capturas sin contexto o explicación mínima	No presenta capturas válidas del tráfico HTTP o no identifica las credenciales en texto plano
Generación y Configuración del Certificado SSL	Certificado generado correctamente con keytool, todos los parámetros adecuados (CN=localhost, alias, contraseña correctos), keystore verificado con `keytool -list`, archivo movido a `resources`, configuración en `application.properties` completa y correcta, explica cada parámetro	Certificado generado y configurado correctamente, aplicación funciona en HTTPS, pero falta verificación con keytool o explicación superficial de los parámetros	Certificado generado con errores menores (CN incorrecto, problemas con alias/contraseña), configuración incompleta pero funcional, sin verificación ni explicaciones	No consigue generar el certificado correctamente o configuración errónea que impide el funcionamiento HTTPS

Criterio	Excelente - 3 pts	Bien - 2 pts	Suficiente - 1pt	Insuficiente - 0pt
Aplicación HTTPS Funcional y verificación	Aplicación arranca correctamente en puerto 8443 con HTTPS, acepta el certificado autofirmado en el navegador siguiendo los pasos correctos, verifica los datos del certificado desde el navegador (emisión, validez, CN=localhost), documenta el proceso con capturas	Aplicación funciona en HTTPS en puerto 8443, acepta el certificado en navegador, pero no verifica los datos del certificado o documentación básica sin detalles	Aplicación funciona en HTTPS con errores o advertencias no resueltas, acepta el certificado pero sin verificar datos, documentación mínima o confusa	No consigue que la aplicación arranque en HTTPS o no accede correctamente desde el navegador
Captura y Análisis HTTPS cifrado (Wireshark)	Captura correcta con filtro `tcp.port==8443`, identifica y documenta TLS Handshake completo (Client Hello, Server Hello, Certificate, Change Cipher Spec, Finished), localiza Application Data cifrado, realiza TCP Stream y demuestra que credenciales NO son visibles, comparativa visual HTTP vs HTTPS bien documentada	Captura correcta del tráfico HTTPS, identifica handshake básico (al menos 3 fases) y Application Data cifrado, demuestra que credenciales no son visibles, comparativa HTTP vs HTTPS presente pero básica	Captura realizada con filtro correcto, identifica algunos elementos del handshake (1-2 fases) o datos cifrados, comparativa ausente o muy superficial, falta análisis detallado	No presenta capturas válidas de HTTPS, no identifica el handshake TLS, no demuestra el cifrado de datos o no realiza comparativa

Criterio	Excelente - 3 pts	Bien - 2 pts	Suficiente - 1pt	Insuficiente - 0pt
Preguntas de Análisis y Documentación Global	Responde las 2 preguntas con investigación profunda y redacción propia, identifica 3 tipos de ataques específicos con consecuencias detalladas, explica CAs, diferencias certificados y riesgos con ejemplos reales, cita fuentes, todo el proceso documentado ordenadamente con capturas comentadas	Responde las 2 preguntas correctamente con investigación moderada, identifica 3 ataques básicos, explica CAs y diferencias sin ejemplos específicos, documentación ordenada con capturas básicas,	Responde solo 1 pregunta o ambas de forma superficial (<100 palabras), identifica menos de 3 ataques sin detalles, explicación muy básica de CAs sin investigación, documentación desorganizada	No responde o respuestas incorrectas/copiadas textualmente, no identifica ataques o sin sentido técnico, documentación ausente o ilegible, más de 6 faltas ortográficas