## Seguridad Informática 2020/2021.

## Práctica 2

## MUY IMPORTANTE: Leer el documento con la normativa sobre estas prácticas.

La práctica se realiza en el entorno de virtualización VirtualBox. En esta práctica se utiliza el siguiente sistema, cuya imagen virtual ya está creada en formato **ova**:

- **Seginf-2** -> Credenciales: (**root** / **toor**) , (**user** / **resu**)

Es una máquina Debian 9, con algún añadido (servidor apache2 y sudo). Está en:

https://unizares-my.sharepoint.com/:f:/g/personal/gvalles_unizar_es/EgMaqbww_bJJhjm_bUFOB2YBPy-5eOtQ0U5gtWaRfniHRw?e=X1F0mI

Para crear las máquinas en VirtualBox, seleccionar el menú "Archivo->Importar servicio virtualizado". **MUY IMPORTANTE: En el caso de hacer las prácticas en las máquinas CentOS del laboratorio, importarlas a un USB propio o bien a: /mnt/scratch…….. (crear el directorio), NUNCA a: /home/aXXX…….**

La máquina está configurada con 2 interfaces de red, uno NAT (para descargas y actualizaciones de software, aunque en esta práctica en principio no hace falta), y otro de tipo "Adaptador solo anfitrión" para interaccionar desde el host con la MV, ambos configurados como clientes DHCP.

El enunciado de esta práctica es básicamente una traducción con mínimas variaciones de "SEED Labs – Public-Key Infrastructure Lab", cuyo autor es Wenliang Du (https://seedsecuritylabs.org).

# 1 Introducción

La criptografía de clave pública es hoy en día la base de las comunicaciones seguras, pero está sujeta a los ataques "man-in-the-middle" cuando un lado de la comunicación envía su clave pública al otro lado. El problema fundamental es que no hay una manera fácil de verificar la propiedad de una clave pública, es decir, dada una clave pública y la información sobre el propietario que se reclama como tal, ¿cómo podemos asegurarnos de que la clave pública es efectivamente propiedad del propietario que se reclama como tal?

El objetivo de aprendizaje de este laboratorio es que los estudiantes obtengan una experiencia de primera mano en PKI. Mediante este laboratorio, los estudiantes deben ser capaces de comprender mejor cómo funciona la PKI, cómo se utiliza la PKI para proteger la Web, y cómo los ataques "man-in-the-middle" pueden ser derrotados por la PKI. Además, los estudiantes serán capaces de entender el origen de la confianza en la infraestructura de clave pública, y qué problemas se plantean si la confianza se rompe. Este laboratorio cubre los siguientes temas:

- Cifrado de clave pública
- Criptografía de clave pública (PKI)
- Autoridad de certificación (CA) y CA raíz
- Certificados X.509 y auto-certificados
- Apache, HTTP, HTTPS
- Ataques "man-in-the-middle"

# 2 Tareas del laboratorio

## 2.1 Tarea 1: Cómo convertirse en una Autoridad de Certificación (CA)

Una Autoridad de Certificación (CA) es una entidad de confianza que emite certificados digitales. El certificado digital certifica la propiedad de una clave pública por el sujeto nombrado en el certificado. Existen varias CAs comerciales que se tratan como CA raíz; VeriSign es la CA más grande en el momento de escribir este texto. Los usuarios que quieren obtener cetificados digitales expedidos por las CAs comerciales deben pagar a dichas CAs.

En este laboratorio necesitamos crear certificados digitales, pero no vamos a pagar a ninguna CA comercial. Nos convertiremos en una CA raíz nosotros mismos, y luego utilizaremos esta CA para emitir certificados para otros (por ejemplo, servidores). En esta tarea nos convertiremos en una CA raíz y generaremos un certificado para esta CA. A diferencia de otros certificados, que suelen estar firmados por otra CA, los certificados de la CA raíz son auto-firmados. Los certificados de las CAs raíz están normalmente precargados en la mayoría de los sistemas operativos, navegadores web y otros programas que dependen de PKI. La confianza en los certificados de CAs raíz es completa.

**El archivo de configuración** `openssl.cnf`. Vamos a utilizar OpenSSL para generar certificados. Para ello debe existir un archivo de configuración, `openssl.conf`. Se usa por tres comandos de OpenSSL: `ca`, `req` y `x509`. Podéis obtener una copia del fichero de configuración en `/usr/lib/ssl/openssl.cnf`. Después de copiar este archivo en vuestro directorio actual, necesitáis crear varios subdirectorios como se especifica en el archivo de configuración (ver la sección `[CA_default]`):

```
dir             = ./demoCA      # Where everything is kept
```

```
certs            = $dir/certs      # Where the issued certs are kept
crl_dir          = $dir/crl        # Where the issued crl are kept
new_certs_dir    = $dir/newcerts   # default place for new certs.
database         = $dir/index.txt  # database index file.
serial           = $dir/serial     # The current serial number
```

Para el archivo `index.txt`, simplemente cread un archivo vacío. Para el archivo `serial`, poned un solo número en formato string (por ejemplo, 1000) en el archivo. Una vez que hayáis configurado el archivo de configuración `openssl.cnf`, podréis crear y emitir certificados.

**Autoridad de certificación (CA).** Como hemos descrito anteriormente, necesitamos generar un certificado auto-firmado (un auto-certificado) para nuestra CA. Esto significa que esta CA es totalmente confiable, y su certificado servirá como el certificado raíz. Podéis ejecutar el siguiente comando para generar el certificado auto-firmado para la CA:

```
$ openssl req -new -x509 -keyout ca.key -out ca.crt -config openssl.cnf
```

Se os pedirá información (el único campo que es realmente importante es “Common Name”) y una contraseña. No perdáis esta contraseña, porque tendréis que escribir la frase de contraseña cada vez que queráis usar esta CA para firmar certificados para otros. También se os pedirá que rellenéis algunos datos, como el nombre del país, el nombre común, etc. La salida del comando se almacena en dos archivos: `ca.key` y `ca.crt`. El archivo `ca.key` contiene la clave privada de la CA, mientras que `ca.crt` contiene el certificado de clave pública.

## 2.2 Tarea 2: Creando un certificado para `seginfo.es`

Ahora que nos hemos convertido en una CA raíz, estamos listos para firmar certificados digitales para nuestros clientes. Nuestro primer cliente es una empresa llamada con un nombre de dominio `seginfo.es`. Para que esta empresa obtenga un certificado digital de una CA, necesita pasar por tres pasos.

**Paso 1: Generar un par de claves públicas/privadas.** La empresa necesita crear primero su propio par de claves públicas/privadas. Podemos ejecutar el siguiente comando para generar un par de claves RSA (tanto privadas como públicas). También se os pedirá que proporcionéis una contraseña para cifrar la clave privada (utilizando el algoritmo de cifrado AES-128, tal y como se especifica en la opción de comando). Las claves se almacenarán en el archivo `server.key`:

```
$ openssl genrsa -aes128 -out server.key 1024
```

El server.key es un archivo de texto codificado (también encriptado), por lo que podréis ver el contenido real, como el módulo, exponentes privados, etc. Para verlos, podéis ejecutar el siguiente comando:

```
$ openssl rsa -in server.key -text
```

**Paso 2: Generar una solicitud de firma de certificado (CSR).** Una vez que la empresa tiene el archivo de claves, debe generar una solicitud de firma de certificado (CSR), que básicamente incluye la clave pública de la empresa. La CSR se enviará a la CA, que generará un certificado para la clave (normalmente después de asegurarse de que la información de identidad de la CSR

coincide con la verdadera identidad del servidor). Utilizad `seginfo.es` como nombre común ("Common Name") de la solicitud de certificado:

```
$ openssl req -new -key server.key -out server.csr -config openssl.cnf
```

Debe tenerse en cuenta que el comando anterior es bastante similar al que utilizamos para crear el certificado auto-firmado para la CA. La única diferencia es la opción -x509. Sin él, el comando genera una petición; con él, el comando genera un certificado auto-firmado.

**Paso 3: Generación de certificados.** El archivo CSR debe tener la firma de la CA para formar un certificado. En el mundo real, los archivos CSR suelen enviarse a una CA de confianza para su firma. En este laboratorio, utilizaremos nuestra propia CA de confianza para generar certificados. El siguiente comando convierte la solicitud de firma de certificado (`server.csr`) en un certificado X509 (`server.crt`), utilizando el `ca.crt` y el `ca.key` de la CA:

```
$ openssl ca -in server.csr -out server.crt -cert ca.crt \
-keyfile ca.key -config openssl.cnf
```

Si OpenSSL se niega a generar certificados, es muy probable que los nombres de sus solicitudes no coincidan con los de CA. Las reglas de coincidencia se especifican en el archivo de configuración (consultad la sección `[policy match]`). Podéis cambiar los nombres de vuestras solicitudes para cumplir con la política, o podéis cambiar la política. El archivo de configuración también incluye otra política (llamada "`policy anything`"), que es menos restrictiva. Podéis elegir esa política cambiando la siguiente línea:

```
"policy = policy_match" change to "policy = policy_anything".
```

## 2.3 Tarea 3: Implementación de certificados en un servidor web HTTPS

En este laboratorio, exploraremos cómo los sitios web utilizan los certificados de clave pública para proteger la navegación web. Crearemos un sitio web HTTPS utilizando el servidor web integrado de `openssl`.

**Paso 1: Configuración de DNS.** Elegimos `seginfo.es` como nombre de nuestro sitio web. Para conseguir que nuestras máquinas reconozcan este nombre, agregaremos la siguiente entrada al fichero de traducciones DNS "directas" en el fichero /etc/hosts:

```
127.0.0.1 seginfo.es
```

Esta entrada básicamente mapea el nombre de host `seginfo.es` a la IP local.

**Paso 2: Configuración del servidor web.** Lancemos un servidor web simple con el certificado generado en la tarea anterior. OpenSSL nos permite iniciar un servidor web simple utilizando el comando `s_server`:

```
# Combine the secret key and certificate into one file
% cp server.key server.pem
% cat server.crt >> server.pem
# Launch the web server using server.pem
% openssl s_server -cert server.pem –www
```

Por defecto, el servidor escuchará en el puerto 4433. Podéis modificarlo con la opción `-accept`. Ahora podéis acceder al servidor a través de la siguiente URL: `https://seginfo.es:4433/`. La mayoría de las veces, probablemente, obtendréis un mensaje de error del navegador parecido al siguiente: "*seginfo.es:4433 utiliza un certificado de seguridad no válido. No se confía en el certificado porque el emisor del certificado es desconocido*".

Como utilizamos una máquina sin entorno gráfico, tendremos que ejecutar el servidor de prueba en background, introduciendo el password del certificado del servidor de forma no interactiva:

```
% openssl s_server -cert server.pem –www –pass pass:_mi_password_ &
```

Entonces es posible conectar al servidor mediante curl por ejemplo:

```
% curl https://seginfo.es:4433
```

**Paso 3: Conseguir que el navegador "curl" acepte nuestro certificado de CA.** Si nuestro certificado hubiera sido asignado por VeriSign, no tendríamos tal mensaje de error, ya que es muy probable que el certificado de VeriSign esté precargado en el repositorio de certificados del navegador. Desafortunadamente, el certificado de `seginfo.es` está firmado por nuestro propia CA (es decir, usando ca.crt), y esta CA no es reconocida por ningún navegador. Hay dos maneras de lograr que el navegador acepte el certificado auto-firmado de nuestro CA:

- Podemos solicitar al fabricante del navegador que incluya el certificado de nuestra CA en su software, para que todo el mundo que utilice su navegador puede reconocer nuestra CA. Así es como las CAs reales, como VeriSign, integran sus certificados en el software. Desafortunadamente, nuestra propia CA no tiene un mercado lo suficientemente grande como para que ningún fabricante incluya nuestro certificado, por lo que no seguiremos esta dirección.
- Cargar nuestro certificado `ca.crt` en el navegador. Para ello utilizamos en `curl` la opción `–cacert`.

**Paso 4. Probando nuestro sitio web HTTPS.** Ahora, navegar a https://seginfo.es:4433. Describid y explicad vuestras observaciones. Haced también las siguientes tareas:

1. Modificad un solo byte de `server.pem`, reiniciad el servidor y volved a cargar la URL. ¿Qué es lo que se observa? Aseguraos de restaurar el fichero `server.pem` original después. Nota: el servidor puede no ser capaz de reiniciar si ciertos bytes de `server.pem` están corruptos; en ese caso, debéis elegir otro lugar para modificar.
2. Como no conectamos desde el propio servidor donde está alojada la página web, mediante https://localhost:4433 nos conectaremos al mismo servidor web. Hacedlo y describid y explicad vuestras observaciones.

## 2.4 Tarea 4: Implementación de certificados en un sitio web HTTPS basado en Apache

La configuración del servidor HTTPS usando el comando de servidor de `openssl` es principalmente para propósitos de depuración y demostración. En este laboratorio, creamos un servidor web HTTPS real basado en Apache. El servidor Apache, que ya está instalado en nuestra VM, soporta el protocolo HTTPS. Para crear un sitio web HTTPS, sólo necesitamos configurar el

servidor Apache, para que sepa dónde obtener la clave privada y los certificados. Damos un ejemplo a continuación para mostrar cómo habilitar HTTPS para un sitio web `www.example.com`. Vuestra tarea es hacer lo mismo con `seginfo.es` utilizando el certificado generado en tareas anteriores.

Un servidor Apache puede alojar simultáneamente varios sitios web. Necesita conocer el directorio donde se almacenan los archivos de cada sitio web. Esto se hace a través de su archivo `VirtualHost`, ubicado en el directorio `/etc/apache2/sites-available`. Para añadir un sitio web HTTP, añadimos una entrada de `VirtualHost` al archivo `000-default.conf`. Véase el siguiente ejemplo:

```
<VirtualHost *:80>
      ServerName one.example.com
      DocumentRoot /var/www/Example_One
      DirectoryIndex index.html
</VirtualHost>
```

Para añadir un sitio web HTTPS, necesitamos añadir una entrada de `VirtualHost` al archivo `default-ssl.conf` en el mismo directorio:

```
<VirtualHost *:443>
      ServerName two.example.com
      DocumentRoot /var/www/Example_Two
      DirectoryIndex index.html
      SSLEngine On
      SSLCertificateFile /etc/apache2/ssl/example_cert.pem
      SSLCertificateKeyFile /etc/apache2/ssl/example_key.pem
</VirtualHost>
```

La entrada `ServerName` especifica el nombre del sitio web, mientras que la entrada `DocumentRoot` especifica dónde se almacenan los archivos del sitio web. El ejemplo anterior configura el sitio HTTPS `https://two.example.com` (el puerto 443 es el puerto HTTPS predeterminado). En la configuración, debemos indicar a Apache dónde están almacenados el certificado del servidor (`SSLCertificateFile`) y la clave privada (`SSLCertificateKeyFile`).
Después de modificar el archivo `default-ssl.conf`, necesitamos ejecutar una serie de comandos para habilitar SSL. Apache nos pedirá que escribamos la contraseña utilizada para encriptar la clave privada. Una vez que todo está configurado correctamente, podemos navegar por el sitio web, y todo el tráfico entre el navegador y el servidor estará encriptado.

```
// Enable the SSL module
$ sudo a2enmod ssl
// Enable the site we have just edited
$ sudo a2ensite default-ssl
// Test the Apache configuration file for errors
$ sudo apachectl configtest
// Restart Apache
$ sudo systemctl restart apache2
```

Utilizad el ejemplo anterior como guía para configurar un servidor HTTPS para `seginfo.es`. Describid los pasos que habéis dado, el contenido que habéis añadido al archivo de

configuración de Apache y las capturas de pantalla del resultado final que muestran que podéis navegar con éxito por el sitio HTTPS.

## 2.5 Tarea 5: Lanzando un ataque "man-in-the-middle"

En esta tarea, mostraremos cómo la PKI puede evitar los ataques "man-in-the-middle" (MITM). La figura 1 muestra cómo funcionan los ataques MIT. Suponed que Alice desea visitar `example.com` a través del protocolo HTTPS. Necesita que le den la clave pública del servidor `example.com`; Alice generará un *secreto* y lo encriptará usando la clave pública del servidor. Si un atacante puede interceptar la comunicación entre Alice y el servidor, el atacante puede reemplazar la clave pública del servidor por su propia clave pública. Por lo tanto, el *secreto* de Alice está realmente cifrado con la clave pública del atacante, por lo que el atacante podrá leer el *secreto*. El atacante puede reenviar el *secreto* al servidor usando la clave pública del servidor. El *secreto* se usa para encriptar la comunicación entre Alice y el servidor, o sea que el atacante puede descifrar la comunicación cifrada.

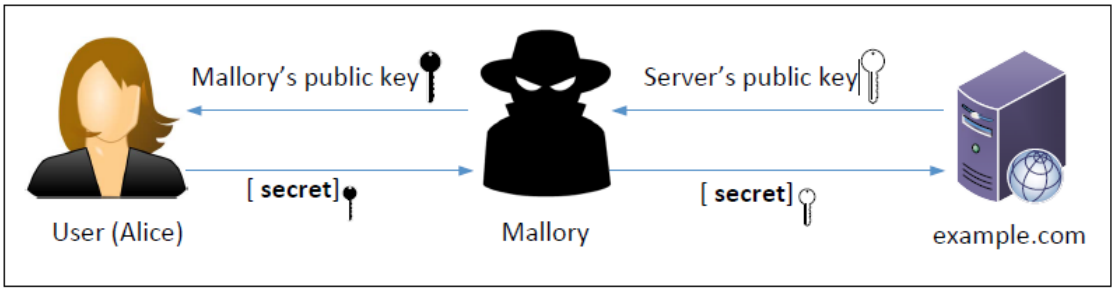


Figura 1. Un ataque "man-in-the-middle"

El objetivo de esta tarea es ayudar a los estudiantes a comprender cómo PKI puede evitar tales ataques MITM. En esta tarea emularemos un ataque MITM, y veremos cómo exactamente PKI puede evitarlo. Primero seleccionaremos un sitio web de destino. En este documento, usamos `example.com` como el sitio web de destino, pero en la tarea, para hacerla más significativa, los estudiantes deben elegir una página web popular, como una página web de un banco o de una red social.

En esta tarea se utilizará el host (donde se ejecuta la VM) como cliente, es decir, el navegador será uno "normal": Firefox, Chrome, Explorer...

**Paso 1: Configuración del sitio web malicioso.** En la Tarea 4, ya hemos creado un sitio web HTTPS para `seginfo.es`. Usaremos el mismo servidor Apache para suplantar a example.com (o el sitio elegido por los estudiantes). Para conseguirlo, seguiremos las instrucciones de la Tarea 4 para añadir una entrada de `VirtualHost` al archivo de configuración SSL de Apache: el `ServerName` debería ser `example.com`, pero el resto de la configuración puede ser la misma que la utilizada en la Tarea 4. Nuestro objetivo es el siguiente: cuando un usuario intenta visitar `example.com`, vamos a hacer que el usuario aterrice en nuestro servidor, que alberga un sitio web falso para `example.com`. Si se tratara de un sitio web de una red social, el sitio falso puede mostrar una página de inicio de sesión similar a la del sitio web de destino. Si los usuarios no pueden notar la diferencia, pueden escribir las credenciales de su cuenta en la página web falsa, revelando esencialmente las credenciales al atacante.

**Paso 2: Convertirse en el "hombre del medio"**. Hay varias formas de conseguir que la petición HTTPS del usuario aterrice en nuestro servidor web. Una forma es atacar el enrutamiento, para

que la petición HTTPS del usuario sea enrutada a nuestro servidor web. Otra forma es atacar DNS, de modo que cuando el equipo de la víctima intenta averiguar la dirección IP del servidor web de destino, obtiene la dirección IP de nuestro servidor web. En esta tarea, utilizamos "ataque DNS". En lugar de lanzar un ataque real de envenenamiento de caché DNS, simplemente modificamos el archivo `/etc/hosts` si nuestro host es Linux (o el equivalente en Windows `C:\Windows\System32\drivers\etc\hosts` si nuestro host es windows) del equipo de la víctima para emular el resultado de un ataque de posición de caché DNS (la dirección IP que aparece a continuación debe ser reemplazada por la dirección IP real del servidor malicioso).

```
<IP_Address> example.com
```

Esta entrada básicamente mapea el nombre de host `seginfo.es` a la IP donde está el servidor web.

**Paso 3: Navegad por el sitio web de destino.** Con todo configurado, visitad el sitio web real de destino y observad lo que diría el navegador. Explicad lo que habéis observado.

**Paso 4: Navegar con el CA raíz en el navegador.** Registrar el certificado raíz generado, `ca.crt`, en el navegador web (la forma de hacerlo exacta depende de cada navegador). Con todo configurado, visitad el sitio web real de destino y observad lo que diría el navegador. Explicad lo que habéis observado.

### 2.6 Tarea 6: Lanzando un ataque "man-in-the-middle" con una CA comprometida

Desgraciadamente la CA raíz que creamos en la Tarea 1 está comprometida por un atacante, y su clave privada ha sido robada. Por lo tanto, el atacante puede generar cualquier certificado arbitrario utilizando la clave privada de esta CA. En esta tarea veremos las consecuencias de este compromiso.
Se pide que diseñéis un experimento para demostrar que el atacante puede lanzar con éxito ataques MITM en cualquier sitio web HTTPS. Podéis utilizar la misma configuración creada en la Tarea 5, pero esta vez debéis demostrar que el ataque MITM tiene éxito, es decir, que el navegador no avisará de ningún riesgo cuando la víctima intente visitar un sitio web, sino que permitirá directamente el acceso al sitio web falso del atacante MITM.

## 3 Presentación del informe

Es necesario presentar un **informe detallado** de laboratorio para describir lo que se ha hecho y lo que se ha observado, incluyendo capturas de pantalla y fragmentos de código. También es necesario dar explicaciones a las observaciones que se consideren interesantes o sorprendentes. Os animo a seguir investigando, más allá de lo requerido en el enunciado del laboratorio.