

Práctica 2

Seguridad Informática

Pedro Allué Tamargo (758267)
Juan José Tambo Tambo (755742)

23 de octubre de 2020

Índice

1. Tarea 1: Crear CA	1
2. Tarea 2: Crear certificado para <i>seginfo.es</i>	1
3. Tarea 3: Implementación de certificados en un servidor web <i>HTTPS</i>	2
4. Tarea 4: Implementación de certificados en un servidor web <i>HTTPS</i> basado en <i>Apache</i>	5
5. Tarea 5: Lanzando un ataque <i>MITM</i>	6
6. Tarea 6: Lanzando un ataque <i>MITM</i> con una CA comprometida	8

1. Tarea 1: Crear CA

Se va a proceder a crear una *CA* (*Certification Authority*). Para ello se va a crear un certificado y se va a firmar por nosotros mismos. Para esta tarea, se utilizará la herramienta `openssl`.

Lo primero será obtener una copia del fichero de configuración de `openssl` disponible en `/usr/lib/ssl/openssl.cnf`. Se va a modificar el fichero de configuración de tal forma que la variable `dir` almacene el valor `./ourCA`.

Tras la modificación se van a crear los subdirectorios `certs`, `crl_dir`, `new_certs_dir` y los ficheros `database` y `serial`.

Ahora se podrá crear el *certificado auto-firmado* para la *CA*. Para ello se ejecutará el comando:

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

La opción `-x509` indica que el certificado es *auto-firmado*.

2. Tarea 2: Crear certificado para *seginfo.es*

Para crear un certificado para *seginfo.es* debemos generar un par de claves públicas/privadas. Para ello utilizaremos el siguiente comando:

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

El programa pedirá una contraseña para cifrar el fichero *server.key*. Ahora debe ser la entidad certificadora la que firme el certificado. Para ello se debe crear una solicitud de firma de certificado (*CSR*). Para generar el fichero *server.csr* se utilizará el comando:

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

Durante la creación de la petición de firma el programa pedirá datos como el *Common Name*. A este campo se le dará el valor de *seginfo.es*.

Una vez obtenido el fichero de petición de firma del certificado se pedirá a la *CA* que firme el certificado. Para ello la *CA* ejecutará el siguiente comando.

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

Ahora el fichero *server.crt* contendrá el certificado firmado por nuestra *CA* que demuestra su identidad frente a la entidad certificadora.

3. Tarea 3: Implementación de certificados en un servidor web *HTTPS*

Tras la generación del certificado para el servidor *seginfo.es* se va a proceder a implementar el certificado con un servidor web *HTTPS*. Para este paso se va a realizar una modificación sobre el *DNS* de la máquina para acceder al servidor web de *seginfo.es* (ubicado en la propia máquina). Para ello se modificará el fichero */etc/hosts* y se añadirá el siguiente contenido:

```
127.0.0.1 seginfo.es
```

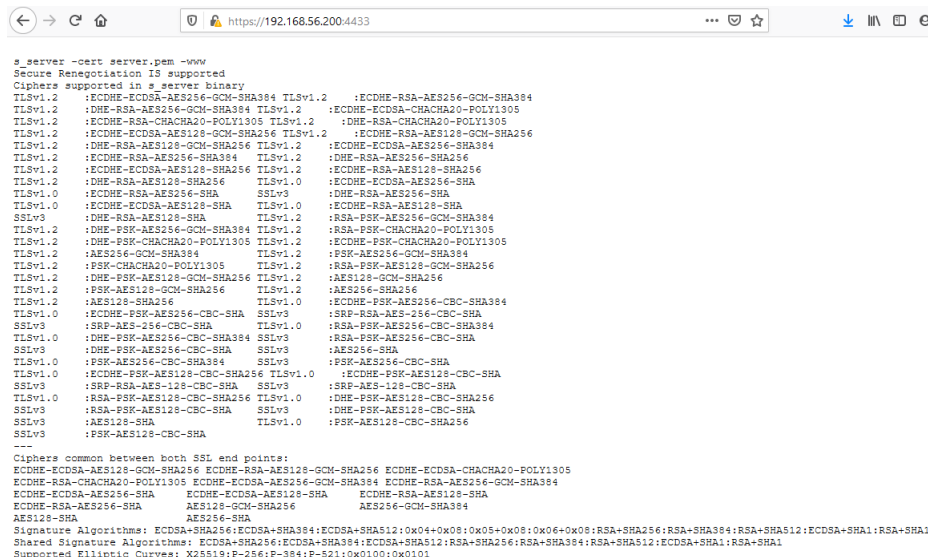
Tras la modificación del *DNS* se va a proceder a configurar el servidor web. Para ello se va a combinar el fichero de clave privada con el certificado en un solo fichero. Para ello se van a ejecutar los siguientes comandos:

```
cp server.key server.pem
cat server.crt >> server.pem
```

Tras la combinación de claves procedemos a lanzar el servidor web integrado en la herramienta *openssl* utilizando el comando:

```
openssl s_server -cert server.pem -www
# Si se quiere ejecutar el comando en
# segundo plano se utilizara:
# openssl s_server -cert server.pem -www \
# -pass pass:_password_ &
```

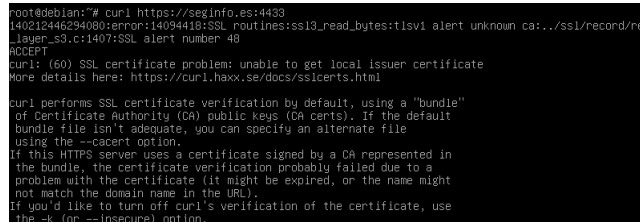
Si se accede al servidor web utilizando el navegador de la máquina *host*. Para ello se utilizará la IP de la máquina virtual (Figura 1). Se puede observar que el navegador no trata a la conexión como segura ya que no dispone del certificado de nuestra *CA*.



```
s_server -cert server.pem -www
Secure Renegotiation IS supported
Ciphers supported in s_server binary
TLSv1.2 : ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 : ECDHE-RSA-AES256-GCM-SHA384
TLSv1.2 : DHE-RSA-AES256-GCM-SHA384 TLSv1.2 : ECDHE-ECDSA-CHACHA20-POLY1305
TLSv1.2 : ECDHE-RSA-CHACHA20-POLY1305 TLSv1.2 : DHE-RSA-CHACHA20-POLY1305
TLSv1.2 : ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2 : ECDHE-RSA-AES128-GCM-SHA256
TLSv1.2 : DHE-RSA-AES128-GCM-SHA256 TLSv1.2 : ECDHE-ECDSA-AES256-SHA384
TLSv1.2 : ECDHE-RSA-AES256-SHA384 TLSv1.2 : DHE-RSA-AES256-SHA256
TLSv1.2 : ECDHE-ECDSA-AES128-SHA256 TLSv1.2 : ECDHE-RSA-AES128-SHA256
TLSv1.2 : DHE-RSA-AES128-SHA256 TLSv1.0 : ECDHE-ECDSA-AES256-SHA
TLSv1.0 : ECDHE-RSA-AES256-SHA SSLv3 : DHE-RSA-AES256-SHA
TLSv1.0 : ECDHE-ECDSA-AES128-SHA TLSv1.0 : ECDHE-RSA-AES128-SHA
SSLv3 : DHE-RSA-AES128-SHA TLSv1.2 : RSA-PSK-AES256-GCM-SHA384
TLSv1.2 : DHE-PSK-AES256-GCM-SHA384 TLSv1.2 : RSA-PSK-CHACHA20-POLY1305
TLSv1.2 : AES256-GCM-SHA384 TLSv1.2 : ECDHE-PSK-CHACHA20-POLY1305
TLSv1.2 : PSK-CHACHA20-POLY1305 TLSv1.2 : PSK-AES256-GCM-SHA384
TLSv1.2 : DHE-PSK-AES128-GCM-SHA256 TLSv1.2 : RSA-PSK-AES128-GCM-SHA256
TLSv1.2 : PSK-AES128-GCM-SHA256 TLSv1.2 : AES128-GCM-SHA256
TLSv1.2 : PSK-AES128-SHA256 TLSv1.2 : AES256-SHA256
TLSv1.0 : AES128-SHA256 TLSv1.0 : ECDHE-PSK-AES256-CBC-SHA384
SSLv3 : ECDHE-PSK-AES256-CBC-SHA SSLv3 : SRP-RSA-AES-256-CBC-SHA
SSLv3 : SRP-AES-256-CBC-SHA TLSv1.0 : RSA-PSK-AES256-CBC-SHA384
TLSv1.0 : DHE-PSK-AES256-CBC-SHA384 SSLv3 : RSA-PSK-AES256-CBC-SHA
SSLv3 : DHE-PSK-AES256-CBC-SHA SSLv3 : AES256-SHA
TLSv1.0 : PSK-AES256-CBC-SHA384 SSLv3 : PSK-AES256-CBC-SHA
TLSv1.0 : ECDHE-PSK-AES128-CBC-SHA256 TLSv1.0 : ECDHE-PSK-AES128-CBC-SHA
SSLv3 : SRP-RSA-AES-128-CBC-SHA SSLv3 : SRP-AES-128-CBC-SHA
TLSv1.0 : RSA-PSK-AES128-CBC-SHA256 TLSv1.0 : DHE-PSK-AES128-CBC-SHA256
SSLv3 : RSA-PSK-AES128-CBC-SHA SSLv3 : DHE-PSK-AES128-CBC-SHA
SSLv3 : AES128-SHA TLSv1.0 : PSK-AES128-CBC-SHA256
SSLv3 : PSK-AES128-CBC-SHA
---
Ciphers common between both SSL end points:
ECDHE-ECDSA-AES128-GCM-SHA256 ECDHE-RSA-AES128-GCM-SHA256 ECDHE-ECDSA-CHACHA20-POLY1305
ECDHE-RSA-CHACHA20-POLY1305 ECDHE-ECDSA-AES256-GCM-SHA384 ECDHE-RSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES256-SHA ECDHE-ECDSA-AES128-SHA ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES256-SHA AES128-GCM-SHA256 AES256-GCM-SHA384
AES128-SHA AES256-SHA
Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:0x04+0x08:0x05+0x08:0x06+0x08:RSA+SHA256:RSA+SHA384:RSA+SHA512:ECDSA+SHA1:RSA+SHA1
Shared Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA512:ECDSA+SHA1:RSA+SHA1
Supported Elliptic Curves: X25519:P-256:P-384:P-521:0x010:0x0101
```

Figura 1: Captura de pantalla del acceso desde el navegador web

Si se intenta acceder utilizando *curl* se obtendrá un error por conexión no segura ya que se desconoce el proveedor del certificado (Figura 2).



```
root@debian:~# curl https://seginfo.es:4433
140212446294080:error:14094418:SSL routines:ssl3_read_bytes:tlsv1 alert unknown ca:../ssl/record/record_layer_s3.c:1407:SSL alert number 48
ACCEPT
curl: (60) SSL certificate problem: unable to get local issuer certificate
More details here: https://curl.haxx.se/docs/sslcerts.html

curl performs SSL certificate verification by default, using a "bundle"
of Certificate Authority (CA) public keys (CA certs). If the default
bundle file isn't adequate, you can specify an alternate file
using the --cacert option.
If this HTTPS server uses a certificate signed by a CA represented in
the bundle, the certificate verification probably failed due to a
problem with the certificate (it might be expired, or the name might
not match the domain name in the URL).
If you'd like to turn off curl's verification of the certificate, use
the -k (or --insecure) option.
```

Figura 2: Fallo al acceder mediante curl

Para conseguir que *curl* acepte la veracidad del servidor *seginfo.es* se debe añadir un argumento a la llamada:

```
curl --cacert ca.crt https://seginfo.es:4433
```

Tras esto se observaría que el resultado es el mostrado en la Figura 3.
¿Qué pasaría si modificasemos un solo byte del fichero *server.pem*? Depende.
Si se modifica un byte de la zona *Certificate* el servidor no se inicia (Figura 4).

Si se modifica un byte de la zona *signature-algorithm* el servidor se inicia sin problemas.

¿Qué ocurre si nos intentamos conectar mediante `https://localhost:4433`?
Que no podremos conectar ya que el certificado ha sido emitido para *seginfo.es*

```

root@debian:~# curl --cacert ca.crt https://seginfo.es:4433
ACCEPT
<HTML><BODY BGCOLOR="#ffffff">
<pre>

s_server -cert server.pem -www -pass pass:seguridad
Secure Renegotiation IS supported
Ciphers supported in s_server binary
TLSv1.2      :ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2      :ECDHE-RSA-AES256-GCM-SHA384
TLSv1.2      :DHE-RSA-AES256-GCM-SHA384 TLSv1.2      :ECDHE-ECDSA-CHACHA20-POLY1305
TLSv1.2      :ECDHE-RSA-CHACHA20-POLY1305 TLSv1.2      :DHE-RSA-CHACHA20-POLY1305
TLSv1.2      :ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2      :ECDHE-RSA-AES128-GCM-SHA256
TLSv1.2      :DHE-RSA-AES128-GCM-SHA256 TLSv1.2      :ECDHE-ECDSA-AES256-SHA384
TLSv1.2      :ECDHE-RSA-AES256-SHA384 TLSv1.2      :DHE-RSA-AES256-SHA256
TLSv1.2      :ECDHE-ECDSA-AES128-SHA256 TLSv1.2      :ECDHE-RSA-AES128-SHA256
TLSv1.2      :DHE-RSA-AES128-SHA256 TLSv1.0      :ECDHE-ECDSA-AES256-SHA

```

Figura 3: Captura de pantalla del acceso desde curl con certificado de CA

```

root@debian:~# openssl s_server --cert server.pem -www -pass pass:seguridad &
[1] 927
root@debian:~# unable to load certificate
139780374564928:error:0D0680A8:asn1 encoding routines:asn1_check_tlen:wrong tag:../crypto/asn1/tasn_dec.c:1129:
139780374564928:error:0D07803A:asn1 encoding routines:asn1_item_embed_d2i:nested asn1 error:../crypto/asn1/tasn_dec.c:289:Type=X509
139780374564928:error:0906700D:PEM routines:PEM_ASN1_read_bio:ASN1 lib:../crypto/pem/pem_oth.c:33:

```

Figura 4: Captura de pantalla de fallo de inicio de server

y no para *localhost* (Figura 5).

```

root@debian:~# curl --cacert ca.crt https://localhost:4433
curl: (51) SSL: certificate subject name 'seginfo.es' does not match target host name 'localhost'
ACCEPT

```

Figura 5: Captura de pantalla de fallo de inicio de server

4. Tarea 4: Implementación de certificados en un servidor web *HTTPS* basado en *Apache*

Para configurar el servidor *Apache*, se deben modificar los ficheros *000-default* (para servicios *HTTP*) y *default-ssl* (servicios *HTTPS*) alojados en el directorio */etc/apache2/sites-available*. Se debe de crear una nueva entrada que indique el nombre del servidor y el directorio raíz donde se encuentran los archivos relacionados con el mismo (Figura 6).

```
<VirtualHost *:80>
    ServerName seginfo.es
    DocumentRoot /var/www/seginfo.es
    DirectoryIndex index.html
</VirtualHost>
```

Figura 6: Modificación de fichero *000-default*

Para un sitio web *HTTPS*, se debe añadir un par de campos más relacionados con el certificado (Figura 7).

```
<VirtualHost *:443>
    ServerName seginfo.es
    DocumentRoot /var/www/seginfo.es
    DirectoryIndex index.html
    SSLEngine On
    SSLCertificateFile /etc/apache2/ssl/seguridad_cert.pem
    SSLCertificateKeyFile /etc/apache2/ssl/seguridad_key.pem
</VirtualHost>
```

Figura 7: Modificación de fichero *default-ssl*

Posteriormente se debe crear directorio */var/www/seginfo.es* y el fichero *index.html* incorporando el texto que queramos que se vea al acceder al servicio. A continuación, se deben ejecutar una serie de comandos para habilitar *SSL* y reiniciar el servicio *Apache*. Al hacerlo, se pide la contraseña utilizada para cifrar la clave privada.

```
// Habilitar modulo SSL
$ sudo a2enmod ssl
// Habilitar la web anadida
$ sudo a2ensite default-ssl
// Comprobar si la configuracion de Apache contiene errores
$ sudo apachectl configtest
// Reiniciar Apache
$ sudo systemctl restart apache2
```

Por último, se ejecuta el comando `curl --cacert ca.crt https://seginfo.es` para acceder al servicio desde la máquina y se muestra el contenido de *index.html*, como se muestra en la Figura 8.

```

root@debian:~# echo Hello World > /var/www/seginfo.es/index.html
root@debian:~# sudo systemctl restart apache2
Enter passphrase for SSL/TLS keys for seginfo.es:443 (RSA): ****
root@debian:~# curl --cacert ca.crt https://seginfo.es/
Hello World
root@debian:~# _

```

Figura 8: Modificación de fichero *index.html* y acceso al servicio

5. Tarea 5: Lanzando un ataque *MITM*

En este apartado se va a proceder a realizar un ataque *Man in The Middle* (*MITM*). Para ello, se ha decidido suplantar la web <https://example.com>. Se deben seguir los pasos descritos en los puntos anteriores para poder crear el certificado del sitio a suplantar. Al finalizar se deben obtener los ficheros *example.csr*, *example.crt* y *example.key* (Figura 9).

```

Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Zaragoza
Locality Name (eg, city) []:Sadaba
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Ecotambo
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:example.com
Email Address []:

```

Figura 9: Creación de certificado para *example.com*

También debe modificarse las entradas *VirtualHost* de *Apache*, indicando como *ServerName* “*example.com*”. También se debe crear el directorio */var/www/example.com* junto al archivo *index.html*, el cual debe contener el código *HTML* del servicio a suplantar. En este caso se ha modificado parte del código para observar que se accede al servicio suplantado.

Para lanzar un *MITM* hay que modificar el *DNS* de la víctima para que en vez de acceder al servicio original, acceda a la página suplantada alojada en la máquina virtual. Para ello, se modifica el archivo *C:/Windows/System32/drivers/etc/hosts* (si la víctima es un servidor con Sistema Operativo *Windows*) añadiendo la siguiente entrada: “*__IP_MV__ example.com*”.

Si se accede a la página desde un navegador, se obtiene lo mostrado en la Figura 10.

Se puede observar que el navegador indica que la conexión no es segura ya que se desconoce el certificado del emisor.

Para obtener una conexión segura, se debe añadir el certificado *ca.crt* al navegador (en este caso *Firefox*) (Figura 11).

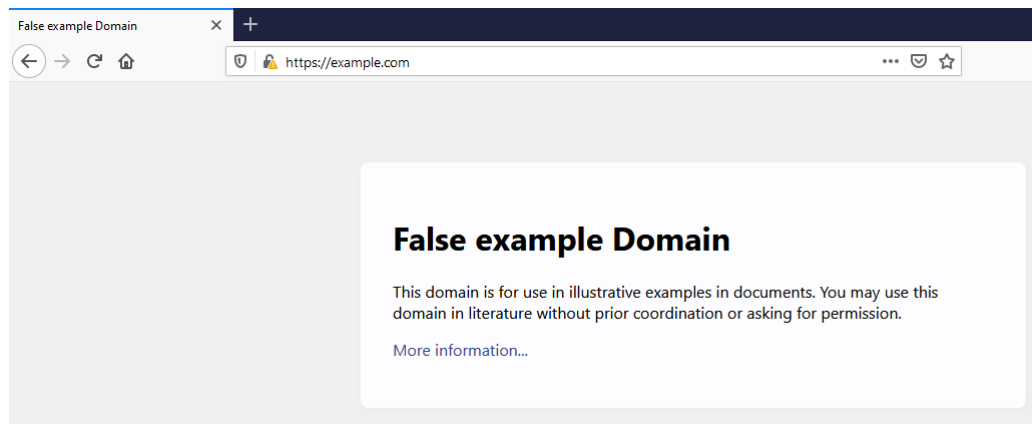


Figura 10: Acceso a *example.com* sin certificado

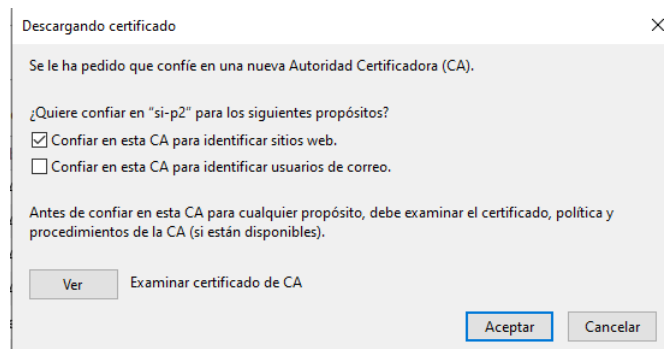


Figura 11: Adición de certificado a *Firefox*

Tras añadir el certificado, se vuelve a acceder a la página y esta vez no se muestra el aviso de “conexión no segura” (Figura 12).

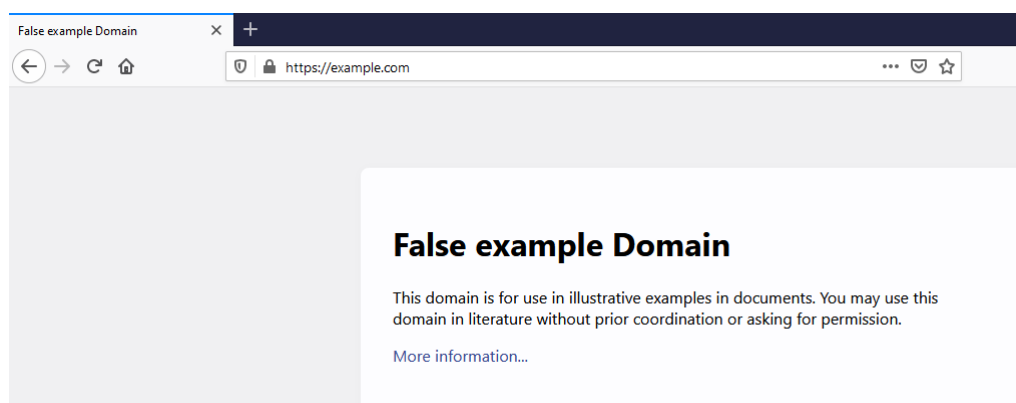


Figura 12: Acceso a *example.com* con certificado

6. Tarea 6: Lanzando un ataque *MITM* con una *CA* comprometida

En este apartado, se supone un escenario en el que un atacante ha obtenido la clave privada de nuestra *CA* y puede crear cualquier certificado a partir de esta clave.

Siguiendo el escenario anterior, la víctima que había añadido el certificado a su navegador está expuesta a un ataque *MITM*, ya que basta con crear un nuevo certificado para un servicio *HTTPS* y realizar un ataque *DNS* para redirigir al servicio suplantado. De esta manera, el navegador confiaría en la página sin mostrar ninguna advertencia.

Se procede a realizar un nuevo ataque *MITM* con una nueva página, <https://www.pcbox.com>. Para ello, se siguen los pasos descritos en puntos anteriores (creación de certificado, nuevo servicio en *Apache* y “ataque” *DNS*).

Si la víctima accede a este sitio web, se le redirigirá al servicio suplantado y el navegador no informará de ningún riesgo, tal y como se muestra en la Figura 13.

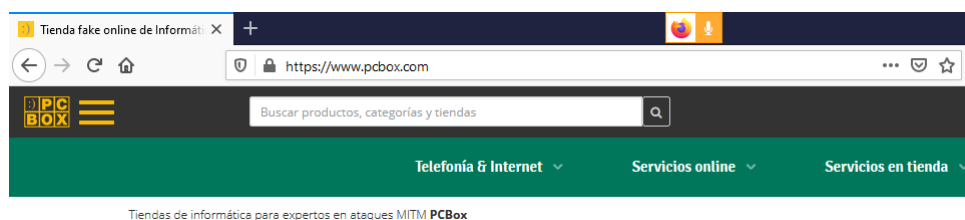


Figura 13: Acceso a *www.pcbox.com* falso