

```
Terminal - alumno@debian: ~/Escritorio/SeguridadRedes/TLS
Archivo Editar Ver Terminal Pestañas Ayuda
alumno@debian:~/Escritorio/SeguridadRedes/TLS$ install nvm
install: falta el fichero de destino después de 'nvm'
Pruebe 'install --help' para más información.
alumno@debian:~/Escritorio/SeguridadRedes/TLS$ sudo ./install_nvm.sh
[sudo] password for alumno:
=> Downloading nvm from git to '/root/.nvm'
=> Clonando en '/root/.nvm'...
remote: Enumerating objects: 360, done.
remote: Counting objects: 100% (360/360), done.
remote: Compressing objects: 100% (306/306), done.
remote: Total 360 (delta 40), reused 169 (delta 28), pack-reused 0
Recibiendo objetos: 100% (360/360), 219.95 KiB | 1.18 MiB/s, listo.
Resolviendo deltas: 100% (40/40), listo.
* (HEAD desacoplado en FETCH_HEAD)
  master
=> Compressing and cleaning up git repository

=> Appending nvm source string to /root/.bashrc
=> Appending bash_completion source string to /root/.bashrc
=> Close and reopen your terminal to start using nvm or run the following to use it now:

export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion" # This loads nvm bash_completion
alumno@debian:~/Escritorio/SeguridadRedes/TLS$
```

Npm install

Si no va:

Sudo apt-get install npm

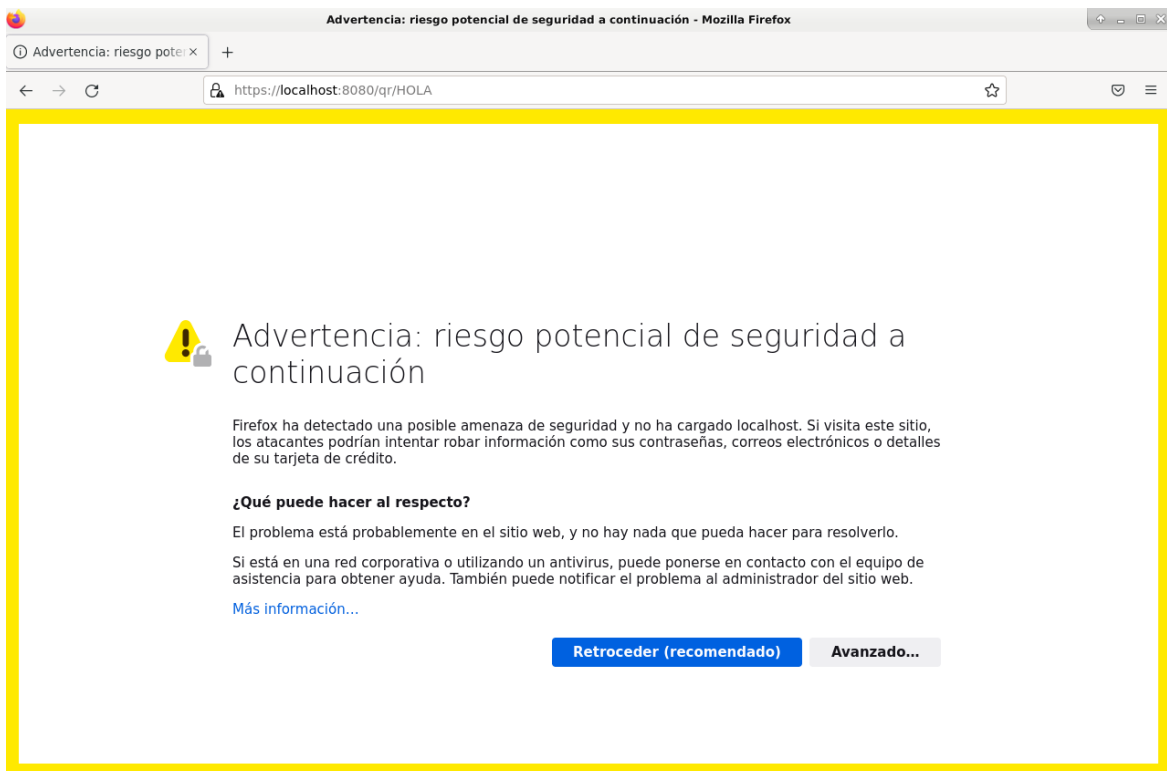
Npm install

```
Terminal - alumno@debian: ~/Escritorio/SeguridadRedes/TLS
Archivo Editar Ver Terminal Pestañas Ayuda
alumno@debian:~/Escritorio/SeguridadRedes/TLS$ sudo node server.js
internal/modules/cjs/loader.js:638
  throw err;
  ^

Error: Cannot find module 'express'
    at Function.Module._resolveFilename (internal/modules/cjs/loader.js:636:15)
    at Function.Module._load (internal/modules/cjs/loader.js:562:25)
    at Module.require (internal/modules/cjs/loader.js:692:17)
    at require (internal/modules/cjs/helpers.js:25:18)
    at Object.<anonymous> (/home/alumno/Escritorio/SeguridadRedes/TLS/server.js:
1:17)
    at Module._compile (internal/modules/cjs/loader.js:778:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:789:10)
    at Module.load (internal/modules/cjs/loader.js:653:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:593:12)
    at Function.Module._load (internal/modules/cjs/loader.js:585:3)
alumno@debian:~/Escritorio/SeguridadRedes/TLS$ sudo wireshark
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
S
```

```
server.js - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

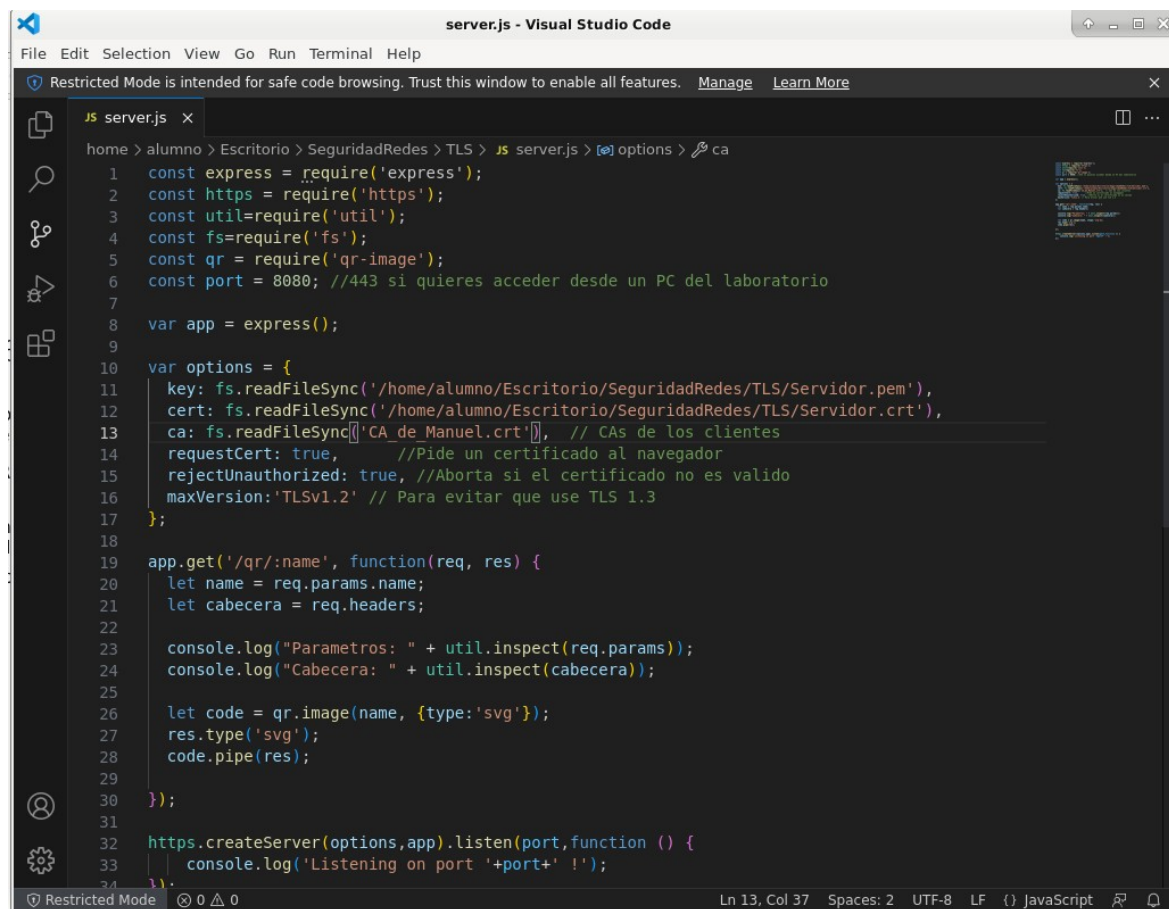
JS server.js x
home > alumno > Escritorio > SeguridadRedes > TLS > JS server.js > ...
1 const express = require('express');
2 const https = require('https');
3 const util=require('util');
4 const fs=require('fs');
5 const qr = require('qr-image');
6 const port = 8080; //443 si quieres acceder desde un PC del laboratorio
7
8 var app = express();
9
10 var options = {
11   key: fs.readFileSync('Servidor.pem'),
12   cert: fs.readFileSync('Servidor.crt'),
13   maxVersion: 'TLSv1.2' // Para evitar que use TLS 1.3
14 };
15
16 app.get('/qr/:name', function(req, res) {
17   let name = req.params.name;
18   let cabecera = req.headers;
19
20   console.log("Parametros: " + util.inspect(req.params));
21   console.log("Cabecera: " + util.inspect(cabecera));
22
23   let code = qr.image(name, {type:'svg'});
24   res.type('svg');
25   code.pipe(res);
26
27 });
28
29 https.createServer(options,app).listen(port,function () {
30   console.log('Listening on port '+port+' !');
31 });
```



HemoS CREADO un servidor con nuestras claves que hicimos para el servidor firmadas por la CA que hicimos. EL cliente que es el navegador se conecta y como no sabe quien es ese servidor NO SE FIA DE SUS CLAVES.

Tenemos que añadir LA CA AL NAVEGADOR PARA QUE ESTE SE FIE DE ESE SERVIDOR QUE FIRMAMOS CON LA CA. SE HACE EN FIREFOX.

AHORA VAMOS A PEDIR UN CERTIFICADO AL NAVEGADOR QUE ESTE FIRMADO POR LA CA DE MANUEL.



```
server.js - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
JS server.js x
home > alumno > Escritorio > SeguridadRedes > TLS > JS server.js > options > ca
1 const express = require('express');
2 const https = require('https');
3 const util=require('util');
4 const fs=require('fs');
5 const qr = require('qr-image');
6 const port = 8080; //443 si quieres acceder desde un PC del laboratorio
7
8 var app = express();
9
10 var options = {
11   key: fs.readFileSync('/home/alumno/Escritorio/SeguridadRedes/TLS/Servidor.pem'),
12   cert: fs.readFileSync('/home/alumno/Escritorio/SeguridadRedes/TLS/Servidor.crt'),
13   ca: fs.readFileSync(['CA_de Manuel.crt']), // CAs de los clientes
14   requestCert: true, //Pide un certificado al navegador
15   rejectUnauthorized: true, //Aborta si el certificado no es valido
16   maxVersion:'TLSv1.2' // Para evitar que use TLS 1.3
17 };
18
19 app.get('/qr/:name', function(req, res) {
20   let name = req.params.name;
21   let cabecera = req.headers;
22
23   console.log("Parametros: " + util.inspect(req.params));
24   console.log("Cabecera: " + util.inspect(cabecera));
25
26   let code = qr.image(name, {type:'svg'});
27   res.type('svg');
28   code.pipe(res);
29
30 });
31
32 https.createServer(options,app).listen(port,function () {
33   console.log('Listening on port '+port+' !');
34 });
```

```
alumno@debian:~/Escritorio/SeguridadRedes/TLS$ sudo node server.js
Listening on port 8080 !
```

# Conexión segura fallida

Ha ocurrido un error al conectar con localhost:8080. El otro extremo de la conexión SSL no ha podido negociar un conjunto aceptable de parámetros de seguridad.

Código de error: SSL\_ERROR\_HANDSHAKE\_FAILURE\_ALERT

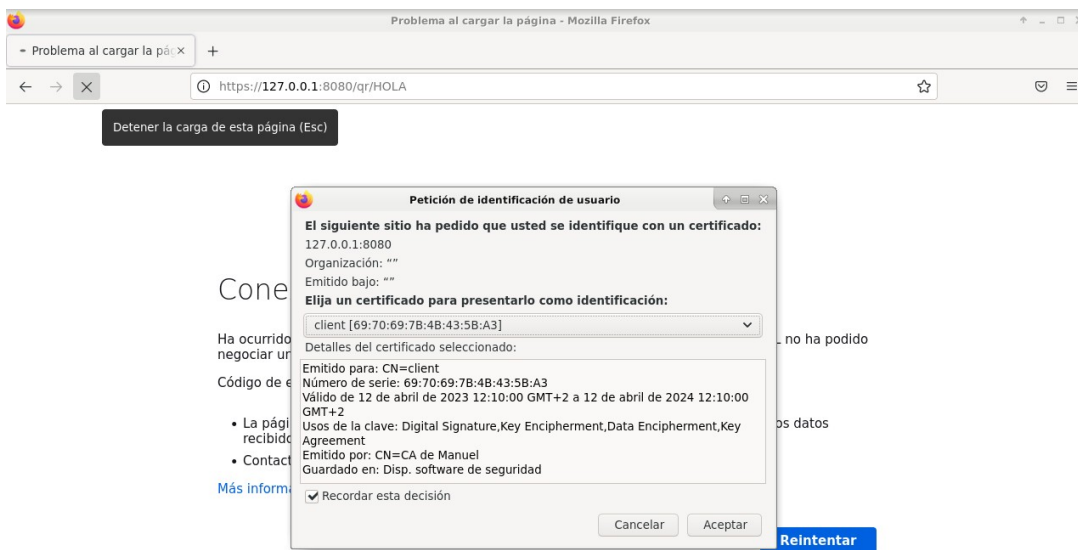
- La página que está intentando ver no se puede mostrar porque la autenticidad de los datos recibidos no ha podido ser verificada.
- Contacte con los propietarios del sitio web para informarles de este problema.

[Más información...](#)

Reintentar

Ahora da otro fallo que indica que se requiere tmb un certificado por parte del cliente(que es el navegador).

Lo instalamos tmb.



127.0.0.1

Y ahora como ya tiene un certificado el cliente, el servidor verifica ese certificado que proviene de esa CA y le permite la conexión.

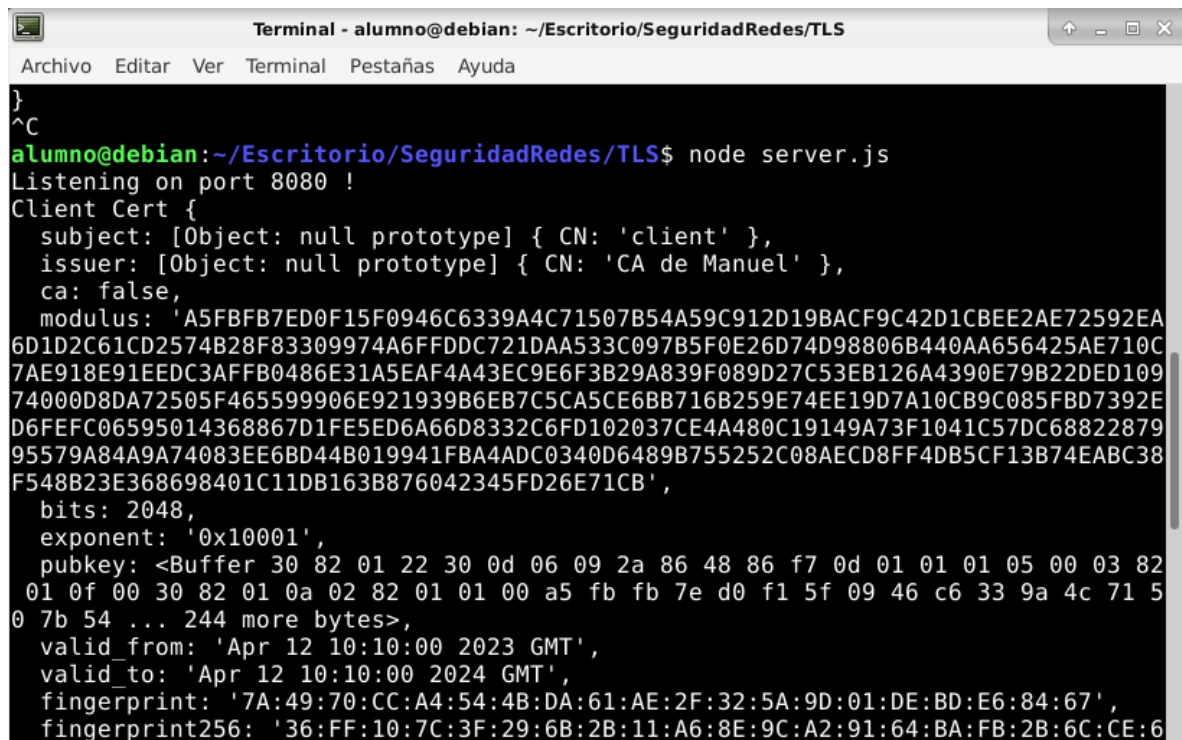
-----

Al añadir en el servidor

```
✓ app.get('/qr/:name', function(req, res) {  
  var certificado= req.connection.getPeerCertificate();  
  console.log("Client Cert", util.inspect(certificado));  
  
  let name = req.params.name;  
  let cabecera = req.headers;  
  
  console.log("Parametros: " + util.inspect(req.params));  
  console.log("Cabecera: " + util.inspect(cabecera));  
  
  let code = qr.image(name, {type:'svg'});  
  res.type('svg');  
  code.pipe(res);  
}
```

Podemos visualizar el certificado al arrancar el servidor y conectarnos con el cliente.

En el servidor:



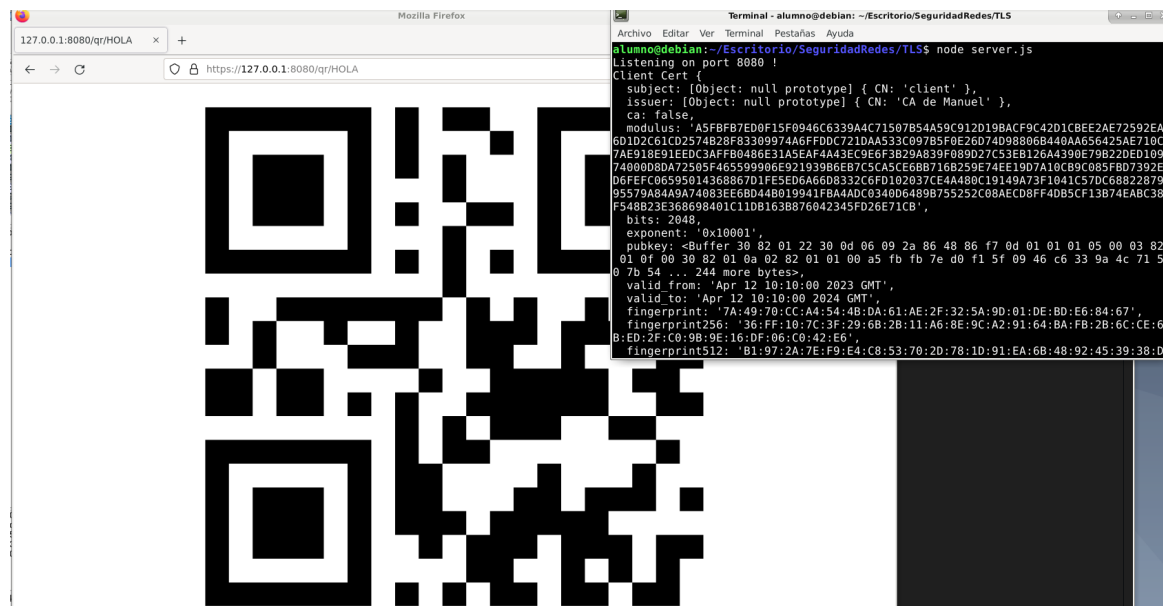
```
Terminal - alumno@debian: ~/Escritorio/SeguridadRedes/TLS  
Archivo Editar Ver Terminal Pestañas Ayuda  
}  
^C  
alumno@debian:~/Escritorio/SeguridadRedes/TLS$ node server.js  
Listening on port 8080 !  
Client Cert {  
  subject: [Object: null prototype] { CN: 'client' },  
  issuer: [Object: null prototype] { CN: 'CA de Manuel' },  
  ca: false,  
  modulus: 'A5FBFB7ED0F15F0946C6339A4C71507B54A59C912D19BACF9C42D1CBEE2AE72592EA  
6D1D2C61CD2574B28F83309974A6FFDDC721DAA533C097B5F0E26D74D98806B440AA656425AE710C  
7AE918E91EEDC3AFFB0486E31A5EAF4A43EC9E6F3B29A839F089D27C53EB126A4390E79B22DED109  
74000D8DA72505F465599906E921939B6EB7C5CA5CE6BB716B259E74EE19D7A10CB9C085FBD7392E  
D6FEFC06595014368867D1FE5ED6A66D8332C6FD102037CE4A480C19149A73F1041C57DC68822879  
95579A84A9A74083EE6BD44B019941FBA4ADC0340D6489B755252C08AEC8FF4DB5CF13B74EABC38  
F548B23E368698401C11DB163B876042345FD26E71CB',  
  bits: 2048,  
  exponent: '0x10001',  
  pubkey: <Buffer 30 82 01 22 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 03 82  
01 0f 00 30 82 01 0a 02 82 01 01 00 a5 fb fb 7e d0 f1 5f 09 46 c6 33 9a 4c 71 5  
0 7b 54 ... 244 more bytes>,  
  valid_from: 'Apr 12 10:10:00 2023 GMT',  
  valid_to: 'Apr 12 10:10:00 2024 GMT',  
  fingerprint: '7A:49:70:CC:A4:54:4B:DA:61:AE:2F:32:5A:9D:01:DE:BD:E6:84:67',  
  fingerprint256: '36:FF:10:7C:3F:29:6B:2B:11:A6:8E:9C:A2:91:64:BA:FB:2B:6C:CE:6
```

Ya lo hemos hecho con PFS que es por defecto que son claves efímeras que utiliza DHE.

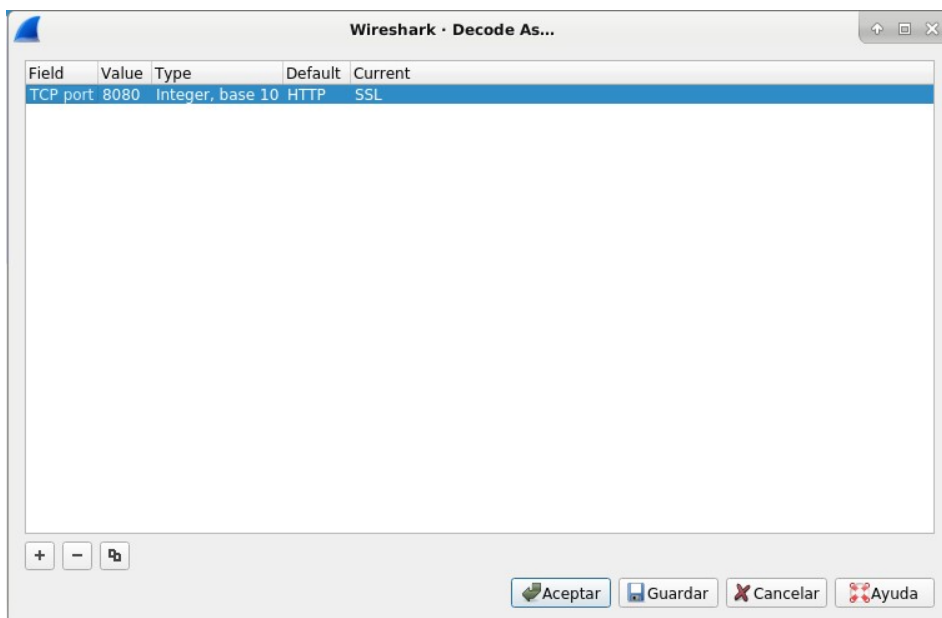
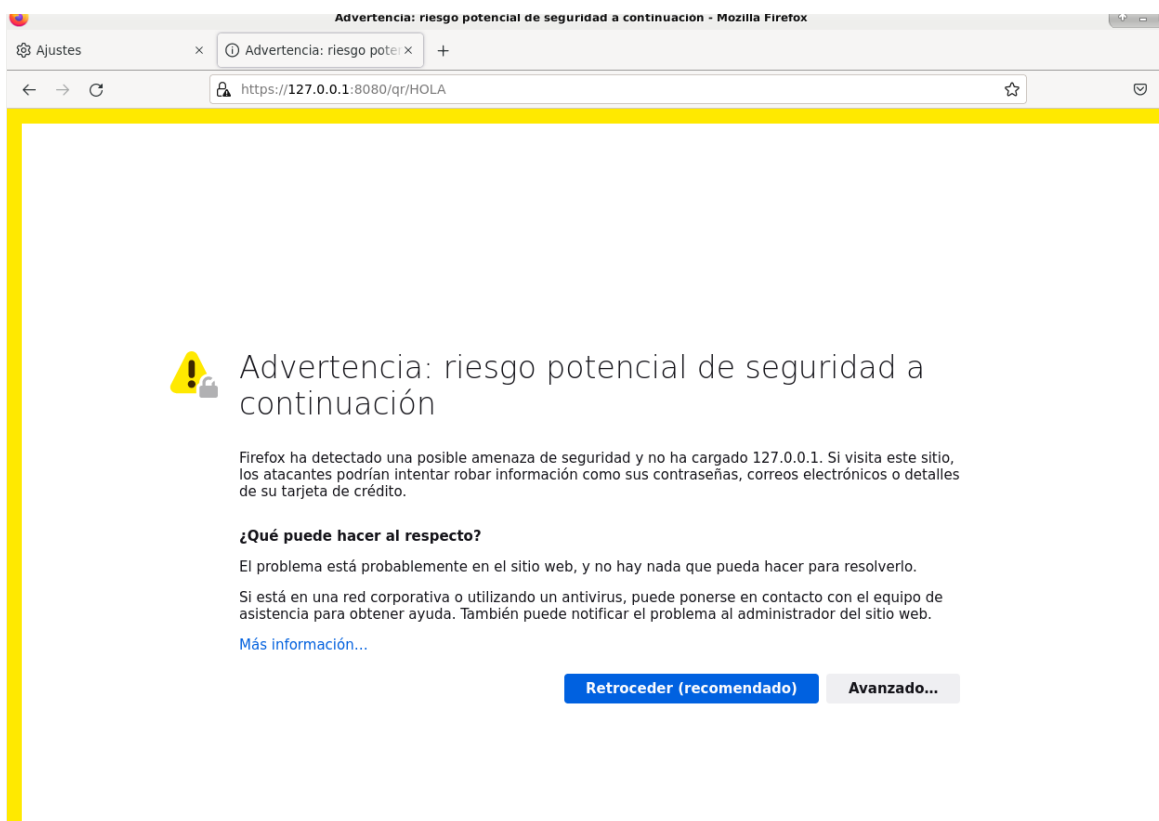
Ahora vamos a forzar que lo haga sin PFS con RSA:

```
var options = {
  key: fs.readFileSync('/home/alumno/Escritorio/SeguridadRedes/TLS/Servidor.pem'),
  cert: fs.readFileSync('/home/alumno/Escritorio/SeguridadRedes/TLS/Servidor.crt'),
  ca: fs.readFileSync('CA_de_Manuel.crt'), // CAs de los clientes
  requestCert: true, //Pide un certificado al navegador
  rejectUnauthorized: true, //Aborta si el certificado no es valido
  maxVersion: 'TLSv1.2', // Para evitar que use TLS 1.3
  ciphers: "AES128-SHA256:AES128-SHA",
  honorCipherOrder: true
};

app.get('/qr/:name', function(req, res) {
  var certificado= req.connection.getPeerCertificate();
```



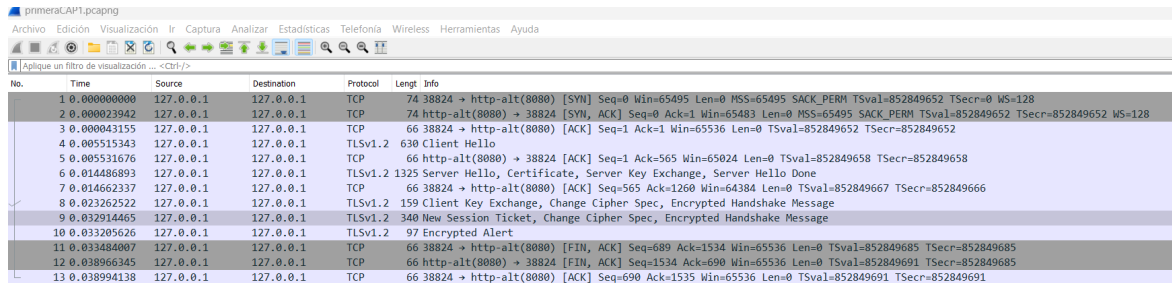
PRIMERO DE TODO: el certificado de la CA no está instalado en el navegador y por tanto el cliente/navegador no se fía de la conexión con el servidor cuyo certificado está firmado por el de la CA.





1. ¿Cómo acaba el handshake cuando el certificado de la CA no está instalado en el navegador? ¿Se usa el protocolo de alerta de SSL?.

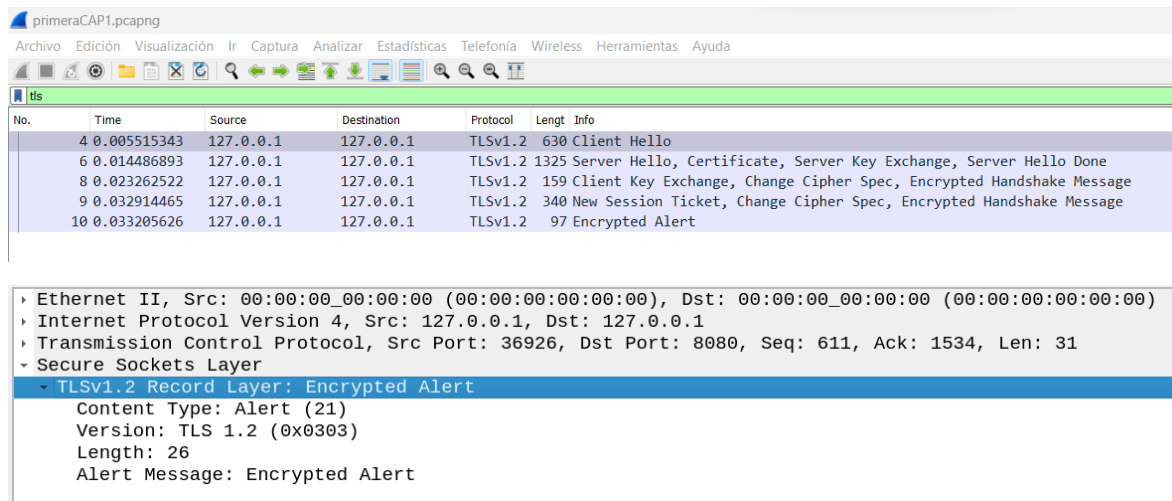
Tras la alerta se cierra la conexión TCP



Wireshark packet capture showing the end of an SSL/TLS handshake. The capture is filtered for 'tls'. The packets are as follows:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	38824 → http-alt(8080) [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=852849652 TSecr=0 WS=128
2	0.000023942	127.0.0.1	127.0.0.1	TCP	74	http-alt(8080) → 38824 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=852849652 TSecr=852849652 WS=128
3	0.000043155	127.0.0.1	127.0.0.1	TCP	66	38824 → http-alt(8080) [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=852849652 TSecr=852849652
4	0.005515343	127.0.0.1	127.0.0.1	TLSv1.2	630	Client Hello
5	0.005531676	127.0.0.1	127.0.0.1	TCP	66	http-alt(8080) → 38824 [ACK] Seq=1 Ack=565 Win=65024 Len=0 TSval=852849658 TSecr=852849658
6	0.014486893	127.0.0.1	127.0.0.1	TLSv1.2	1325	Server Hello, Certificate, Server Key Exchange, Server Hello Done
7	0.014662337	127.0.0.1	127.0.0.1	TCP	66	38824 → http-alt(8080) [ACK] Seq=565 Ack=1260 Win=64384 Len=0 TSval=852849667 TSecr=852849666
8	0.023262522	127.0.0.1	127.0.0.1	TLSv1.2	159	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
9	0.032914465	127.0.0.1	127.0.0.1	TLSv1.2	340	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
10	0.033205626	127.0.0.1	127.0.0.1	TLSv1.2	97	Encrypted Alert
11	0.033484007	127.0.0.1	127.0.0.1	TCP	66	38824 → http-alt(8080) [FIN, ACK] Seq=689 Ack=1534 Win=65536 Len=0 TSval=852849685 TSecr=852849685
12	0.038966345	127.0.0.1	127.0.0.1	TCP	66	http-alt(8080) → 38824 [FIN, ACK] Seq=1534 Ack=690 Win=65536 Len=0 TSval=852849691 TSecr=852849685
13	0.038994138	127.0.0.1	127.0.0.1	TCP	66	38824 → http-alt(8080) [ACK] Seq=690 Ack=1535 Win=65536 Len=0 TSval=852849691 TSecr=852849691

Filtramos TLS



Wireshark packet capture showing the TLS alert packet. The capture is filtered for 'tls'. The packets are as follows:

No.	Time	Source	Destination	Protocol	Length	Info
4	0.005515343	127.0.0.1	127.0.0.1	TLSv1.2	630	Client Hello
6	0.014486893	127.0.0.1	127.0.0.1	TLSv1.2	1325	Server Hello, Certificate, Server Key Exchange, Server Hello Done
8	0.023262522	127.0.0.1	127.0.0.1	TLSv1.2	159	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
9	0.032914465	127.0.0.1	127.0.0.1	TLSv1.2	340	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
10	0.033205626	127.0.0.1	127.0.0.1	TLSv1.2	97	Encrypted Alert

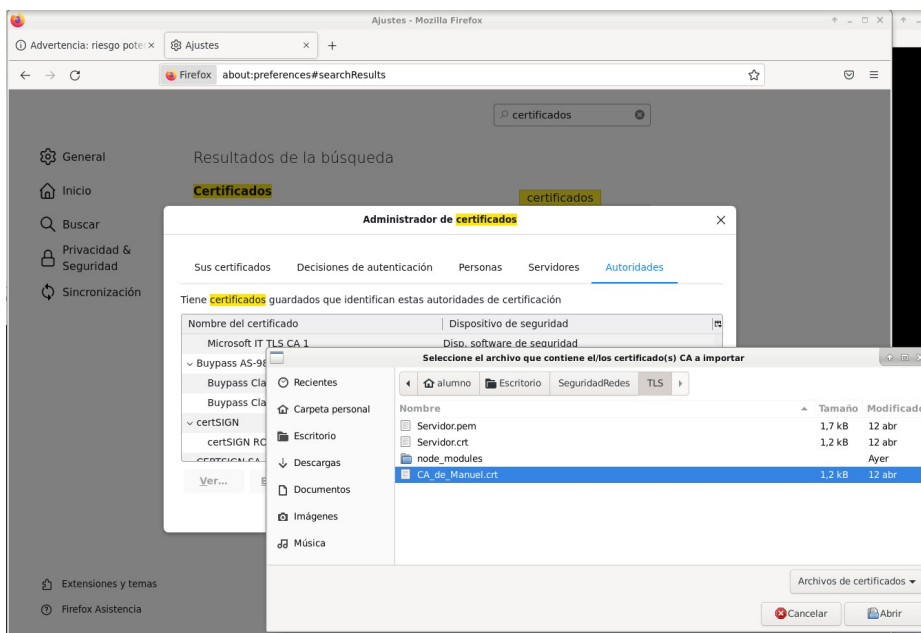
Packet details for the alert (packet 10):

- Ethernet II, Src: 00:00:00\_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00\_00:00:00 (00:00:00:00:00:00)
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
- Transmission Control Protocol, Src Port: 36926, Dst Port: 8080, Seq: 611, Ack: 1534, Len: 31
- Secure Sockets Layer
  - TLSv1.2 Record Layer: Encrypted Alert
    - Content Type: Alert (21)
    - Version: TLS 1.2 (0x0303)
    - Length: 26
    - Alert Message: Encrypted Alert

El primer byte es un 2 que indica que es una alerta fatal.



Ahora añadimos el certificado de la CA que ha firmado el servidor en el navegador para que no vea al servidor como un sospechoso por no ver como confiable la CA que firmó su certificado.

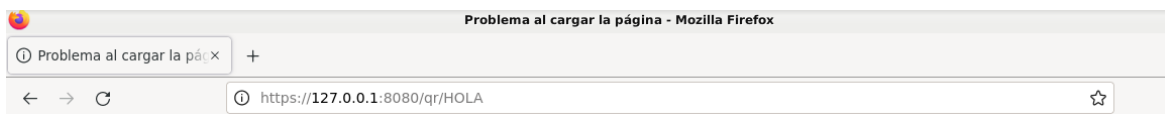


Y añadimos que el servidor le pida al cliente su certificado para la conexión (autenticación mutua):

```
server.js - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

JS server.js x
home > alumno > Escritorio > SeguridadRedes > TLS > JS server.js > app.get('/qr/:name') callback
6 const port = 8080; //443 si quieres acceder desde un PC del laboratorio
7
8 var app = express();
9
10 var options = {
11   key: fs.readFileSync('/home/alumno/Escritorio/SeguridadRedes/TLS/Servidor.pem'),
12   cert: fs.readFileSync('/home/alumno/Escritorio/SeguridadRedes/TLS/Servidor.crt'),
13   maxVersion: 'TLSv1.2', // Para evitar que use TLS 1.3
14
15   //PARA AUTENTICAR AL CLIENTE debe pedirle sus certificados
16   ca: fs.readFileSync('CA_de_Manuel.crt'), // CAs de los clientes
17   requestCert: true, //Pide un certificado al navegador
18   rejectUnauthorized: true //Aborta si el certificado no es valido
19
20   //ciphers: "AES128-SHA256:AES128-SHA",
21   // honorCipherOrder: true
22 };
23
24 app.get('/qr/:name', function(req, res) {
25   // var certificado= req.connection.getPeerCertificate();
26   // console.log("Client Cert", util.inspect(certificado));
27
28   let name = req.params.name;
29   let cabecera = req.headers;
30
31   console.log("Parametros: " + util.inspect(req.params));
32   console.log("Cabecera: " + util.inspect(cabecera));
33
34   let code = qr.image(name, {type: 'svg'});
35   res.type('svg');
36   code.pipe(res);
37 }
```

Ahora nos sale esto:



## Conexión segura fallida

Ha ocurrido un error al conectar con 127.0.0.1:8080. El otro extremo de la conexión SSL no ha podido negociar un conjunto aceptable de parámetros de seguridad.

Código de error: SSL\_ERROR\_HANDSHAKE\_FAILURE\_ALERT

- La página que está intentando ver no se puede mostrar porque la autenticidad de los datos recibidos no ha podido ser verificada.
- Contacte con los propietarios del sitio web para informarles de este problema.

[Más información...](#)

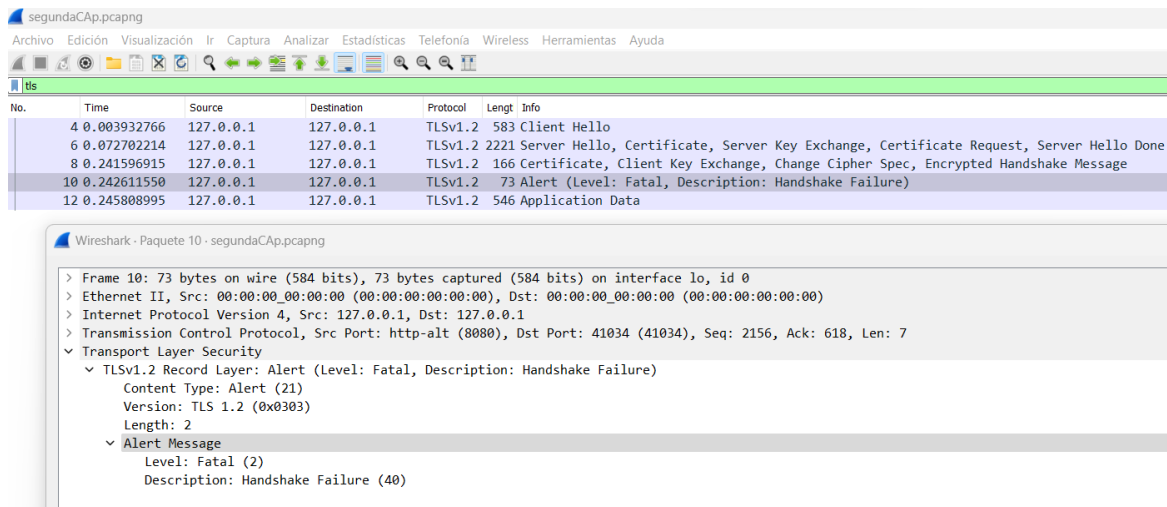
Reintentar

Nos sale que no ha podido autenticar al cliente porque le ha solicitado su certificado y este no está instalado en el navegador.

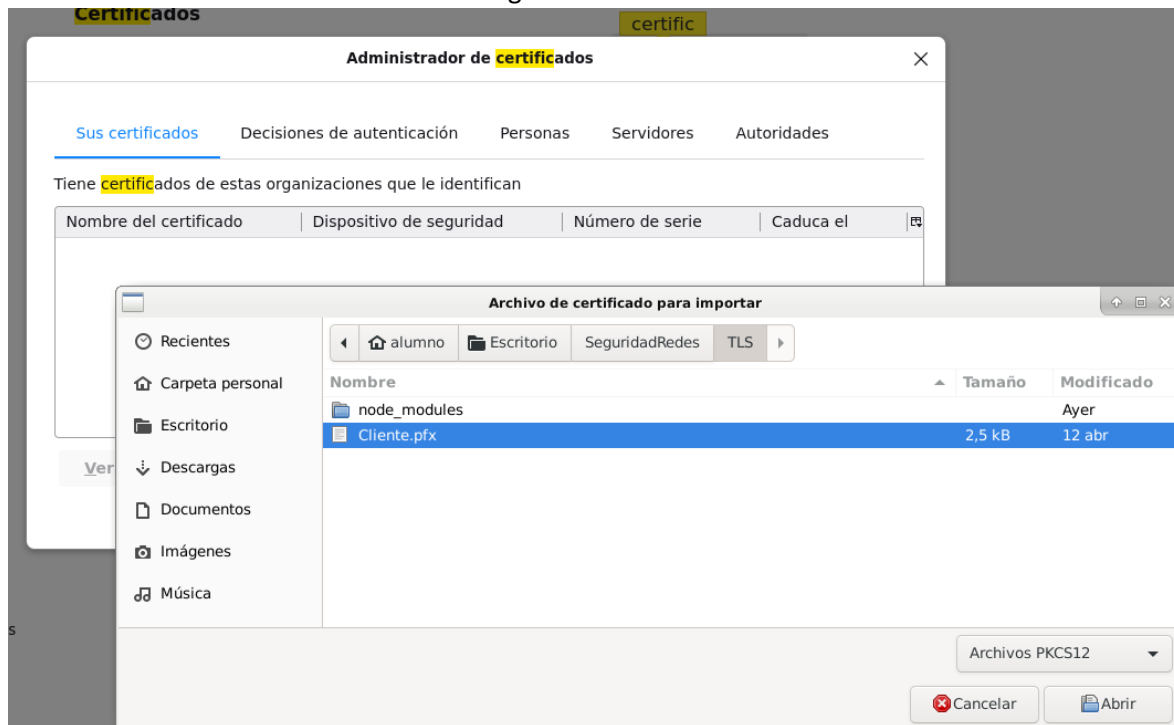
2. ¿Cómo acaba el handshake cuando el certificado del cliente no está instalado en el navegador? ¿Se usa el protocolo de alerta de SSL?

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	41034 → http-alt(8080) [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=853651140 TSecr=0 WS=128
2	0.000020376	127.0.0.1	127.0.0.1	TCP	74	http-alt(8080) → 41034 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=853651140 TSecr=0
3	0.000035022	127.0.0.1	127.0.0.1	TCP	66	41034 → http-alt(8080) [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=853651140 TSecr=853651140
4	0.0003932766	127.0.0.1	127.0.0.1	TLvSv1.2	583	Client Hello
5	0.004019611	127.0.0.1	127.0.0.1	TCP	66	http-alt(8080) → 41034 [ACK] Seq=1 Ack=518 Win=65024 Len=0 TSval=853651144 TSecr=853651144
6	0.072702214	127.0.0.1	127.0.0.1	TLvSv1.2	2221	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
7	0.072827653	127.0.0.1	127.0.0.1	TCP	66	41034 → http-alt(8080) [ACK] Seq=518 Ack=2156 Win=64128 Len=0 TSval=853651212 TSecr=853651212
8	0.241596915	127.0.0.1	127.0.0.1	TLvSv1.2	166	Certificate, Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
9	0.242139563	127.0.0.1	127.0.0.1	TCP	66	http-alt(8080) → 41034 [ACK] Seq=2156 Ack=618 Win=65536 Len=0 TSval=853651381 TSecr=853651381
10	0.242611550	127.0.0.1	127.0.0.1	TLvSv1.2	73	Alert (Level: Fatal, Description: Handshake Failure)
11	0.242620084	127.0.0.1	127.0.0.1	TCP	66	41034 → http-alt(8080) [ACK] Seq=618 Ack=2163 Win=65536 Len=0 TSval=853651382 TSecr=853651382
12	0.245808995	127.0.0.1	127.0.0.1	TLvSv1.2	546	Application Data
13	0.245889261	127.0.0.1	127.0.0.1	TCP	66	http-alt(8080) → 41034 [RST, ACK] Seq=2163 Ack=1098 Win=65536 Len=0 TSval=853651385 TSecr=853651385

Ahora vemos como en el handshake el servidor le hace un certificate request al cliente para autenticarlo.



Añadimos el certificado del cliente al navegador:



Ya no se producen errores en la conexión y visualizamos correctamente el QR.

tercerCAP.pcapng

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

Aplicue un filtro de visualización ... <Ctrl+>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	TCP	74	35050 → http-alt(8080) [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=854109915 TSecr=0 WS=128
2	0.000025701	127.0.0.1	127.0.0.1	TCP	74	http-alt(8080) → 35050 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=854109915 TSecr=0
3	0.000043543	127.0.0.1	127.0.0.1	TCP	66	35050 → http-alt(8080) [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=854109915 TSecr=854109915
4	0.032806975	127.0.0.1	127.0.0.1	TLSv1.2	583	Client Hello
5	0.032835562	127.0.0.1	127.0.0.1	TCP	66	http-alt(8080) → 35050 [ACK] Seq=1 Ack=518 Win=65024 Len=0 TSval=854109947 TSecr=854109947
6	0.069581541	127.0.0.1	127.0.0.1	TLSv1.2	2221	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
7	0.069801331	127.0.0.1	127.0.0.1	TCP	66	35050 → http-alt(8080) [ACK] Seq=518 Ack=2156 Win=64128 Len=0 TSval=854109984 TSecr=854109984
8	0.453829320	127.0.0.1	127.0.0.1	TLSv1.2	1253	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
9	0.453856380	127.0.0.1	127.0.0.1	TCP	66	http-alt(8080) → 35050 [ACK] Seq=2156 Ack=1705 Win=64384 Len=0 TSval=854110368 TSecr=854110368
10	0.453991904	127.0.0.1	127.0.0.1	TLSv1.2	572	Application Data
11	0.454003346	127.0.0.1	127.0.0.1	TCP	66	http-alt(8080) → 35050 [ACK] Seq=2156 Ack=2211 Win=64000 Len=0 TSval=854110368 TSecr=854110368
12	1.289062251	127.0.0.1	127.0.0.1	TLSv1.2	2353	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message, Application Data
13	1.297399410	127.0.0.1	127.0.0.1	TCP	66	35050 → http-alt(8080) [ACK] Seq=2211 Ack=4443 Win=64000 Len=0 TSval=854111203 TSecr=854111203
14	1.329246451	127.0.0.1	127.0.0.1	TLSv1.2	100	Application Data
15	1.329259462	127.0.0.1	127.0.0.1	TCP	66	35050 → http-alt(8080) [ACK] Seq=2211 Ack=4477 Win=64384 Len=0 TSval=854111243 TSecr=854111243
16	2.425151156	127.0.0.1	127.0.0.1	TLSv1.2	484	Application Data
17	2.425173266	127.0.0.1	127.0.0.1	TCP	66	http-alt(8080) → 35050 [ACK] Seq=4477 Ack=2629 Win=65152 Len=0 TSval=854112339 TSecr=854112338
18	2.430324348	127.0.0.1	127.0.0.1	TLSv1.2	97	Encrypted Alert
19	2.430338453	127.0.0.1	127.0.0.1	TCP	66	http-alt(8080) → 35050 [ACK] Seq=4477 Ack=2660 Win=65152 Len=0 TSval=854112344 TSecr=854112344
20	2.430359443	127.0.0.1	127.0.0.1	TCP	66	35050 → http-alt(8080) [FIN, ACK] Seq=2660 Ack=4477 Win=65536 Len=0 TSval=854112344 TSecr=854112344
21	2.473457936	127.0.0.1	127.0.0.1	TLSv1.2	517	Application Data
22	2.473486547	127.0.0.1	127.0.0.1	TCP	54	35050 → http-alt(8080) [RST] Seq=2661 Win=0 Len=0

Se realiza bien el handshake y luego se envían datos. Aunque luego por la cara salta un alerta.

tercerCAP.pcapng

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

fts

No.	Time	Source	Destination	Protocol	Length	Info
4	0.032806975	127.0.0.1	127.0.0.1	TLSv1.2	583	Client Hello
6	0.069581541	127.0.0.1	127.0.0.1	TLSv1.2	2221	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
8	0.453829320	127.0.0.1	127.0.0.1	TLSv1.2	1253	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
10	0.453991904	127.0.0.1	127.0.0.1	TLSv1.2	572	Application Data
12	1.289062251	127.0.0.1	127.0.0.1	TLSv1.2	2353	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message, Application Data
14	1.329246451	127.0.0.1	127.0.0.1	TLSv1.2	100	Application Data
16	2.425151156	127.0.0.1	127.0.0.1	TLSv1.2	484	Application Data
18	2.430324348	127.0.0.1	127.0.0.1	TLSv1.2	97	Encrypted Alert
21	2.473457936	127.0.0.1	127.0.0.1	TLSv1.2	517	Application Data

### 3. ¿Cuántos mensajes se envían en el handshake completo con RSA y autenticación mutua?

Cambiamos la suite de cifrado del servidor:

```
server.js - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. That this window to enable all features. Manage Learn More

server.js x
home > alumno > Escritorio > SeguridadRedes > TLS > server.js > get options
1 const port = 8080; //443 si quieres acceder desde un PC del laboratorio
2
3 var app = express();
4
5
6 var options = {
7   key: fs.readFileSync('/home/alumno/Escritorio/SeguridadRedes/TLS/Server.pem'),
8   cert: fs.readFileSync('/home/alumno/Escritorio/SeguridadRedes/TLS/Server.crt'),
9   maxVersion: 'TLSv1.2', // Para evitar que use TLS 1.3
10
11 //PARA AUTENTICAR AL CLIENTE debe pedirle sus certificados. (Aquí habíamos autenticación mutua)
12 ca: fs.readFileSync('CA de Manuel.crt'), // CAs de los clientes
13 requestCert: true, //Pide un certificado al navegador
14 rejectUnauthorized: true, //Aborta si el certificado no es valido
15
16 //CAMBIAR SUITE DE CIFRADO DEL SERVIDOR
17 //Por defecto hace el handshake con PFS usando DHE.
18 //Así cambiamos para que haga el handshake sin PFS usando RSA:
19 ciphers: 'AES128-SHA256:AES128-SHA',
20 honorCipherOrder: true
21 };
22
23 app.get('/qr/:name', function(req, res) {
24   //Para ver que tiene dentro un certificado
25   var certificado= req.connection.getPeerCertificate(); // Para poder acceder al certificado
26   console.log('Client Cert', util.inspect(certificado)); // Para ver que tiene dentro un cer
27   //usamos util.inspect y nos devuelve un JSON con el conten
28
29   let name = req.params.name;
30   let cabecera = req.headers;
31
32   console.log('Parámetros: ' + util.inspect(req.params));
33   console.log('Cabecera: ' + util.inspect(cabecera));
34
35   let code = qr.image(name, {type: 'svg'});
36   res.type('svg');
37   code.pipe(res);
38 });
```

dosHandshakes.pcapng

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

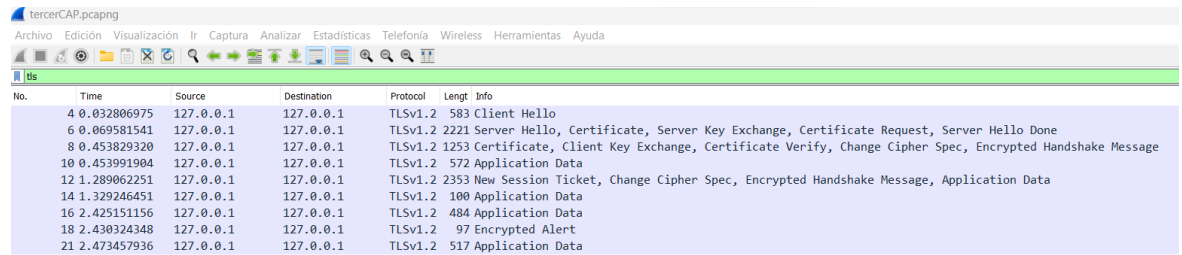
tls

No.	Time	Source	Destination	Protocol	Lengt	Info
4	0.125011850	127.0.0.1	127.0.0.1	TLSv1.2	1462	Client Hello
6	0.143064790	127.0.0.1	127.0.0.1	TLSv1.2	1908	Server Hello, Certificate, Certificate Request, Server Hello Done
8	0.376925386	127.0.0.1	127.0.0.1	TLSv1.2	1502	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Finished
9	0.401481447	127.0.0.1	127.0.0.1	TLSv1.2	1196	New Session Ticket, Change Cipher Spec, Finished
11	0.403060295	127.0.0.1	127.0.0.1	HTTP	599	GET /qr/HOLA HTTP/1.1
13	0.613237221	127.0.0.1	127.0.0.1	TLSv1.2	1271	[TLS segment of a reassembled PDU]
14	0.616913739	127.0.0.1	127.0.0.1	HTTP/X...	119	HTTP/1.1 200 OK
16	5.619619616	127.0.0.1	127.0.0.1	TLSv1.2	119	Alert (Level: Warning, Description: Close Notify)
19	5.634382031	127.0.0.1	127.0.0.1	TLSv1.2	119	Alert (Level: Warning, Description: Close Notify)

Sin PFS usando RSA el handshake es mucho mas corto.

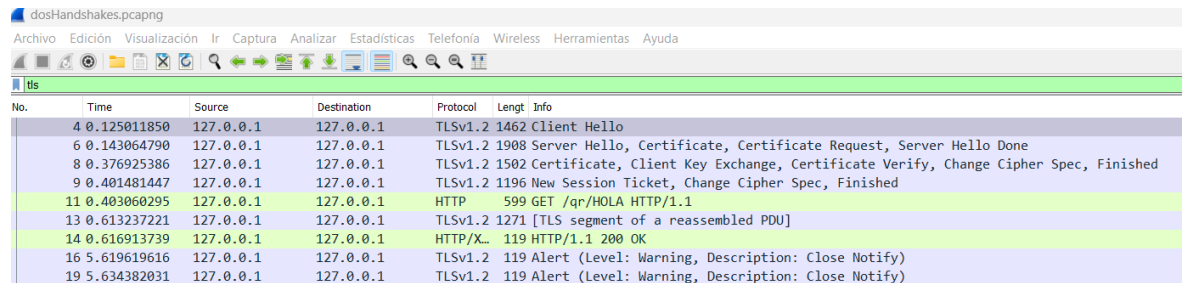
## TENEMOS ESTAS DOS COMPARATIVAS:

### Con PFS usando DHE



No.	Time	Source	Destination	Protocol	Length	Info
4	0.032806975	127.0.0.1	127.0.0.1	TLSv1.2	583	Client Hello
6	0.069581541	127.0.0.1	127.0.0.1	TLSv1.2	2221	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
8	0.453829320	127.0.0.1	127.0.0.1	TLSv1.2	1253	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
10	0.453991904	127.0.0.1	127.0.0.1	TLSv1.2	572	Application Data
12	1.289062251	127.0.0.1	127.0.0.1	TLSv1.2	2353	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message, Application Data
14	1.329246451	127.0.0.1	127.0.0.1	TLSv1.2	100	Application Data
16	2.425151156	127.0.0.1	127.0.0.1	TLSv1.2	484	Application Data
18	2.430324348	127.0.0.1	127.0.0.1	TLSv1.2	97	Encrypted Alert
21	2.473457936	127.0.0.1	127.0.0.1	TLSv1.2	517	Application Data

### Sin PFS usando RSA



No.	Time	Source	Destination	Protocol	Length	Info
4	0.125011850	127.0.0.1	127.0.0.1	TLSv1.2	1462	Client Hello
6	0.143064790	127.0.0.1	127.0.0.1	TLSv1.2	1908	Server Hello, Certificate, Certificate Request, Server Hello Done
8	0.376925386	127.0.0.1	127.0.0.1	TLSv1.2	1502	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Finished
9	0.401481447	127.0.0.1	127.0.0.1	TLSv1.2	1196	New Session Ticket, Change Cipher Spec, Finished
11	0.403060295	127.0.0.1	127.0.0.1	HTTP	599	GET /qr/HOLA HTTP/1.1
13	0.613237221	127.0.0.1	127.0.0.1	TLSv1.2	1271	[TLS segment of a reassembled PDU]
14	0.616913739	127.0.0.1	127.0.0.1	HTTP/X.	119	HTTP/1.1 200 OK
16	5.619619616	127.0.0.1	127.0.0.1	TLSv1.2	119	Alert (Level: Warning, Description: Close Notify)
19	5.634382031	127.0.0.1	127.0.0.1	TLSv1.2	119	Alert (Level: Warning, Description: Close Notify)

4. ¿Cuántos mensajes se envían en el handshake completo con RSA y autenticación mutua?

14 únicamente falta el server key Exchange. Los encrypted handshake message se cambian por mensajes finished.

¿Cuántos mensajes se envían en el handshake completo con ECDHE y autenticación mutua?

15

5. ¿Cuál es la principal diferencia entre los handshakes con RSA y ECDHE?

Con ECDHE el servidor envía un Key Exchange ya que genera unas claves únicamente para esta conexión(PFS) mientras que RSA envía el certificado del servidor que es lo que usa el cliente de vuelta para mandar a clave secreta.

Además, con ECDHE se envía un mensaje encriptado final para comprobar que se ha realizado correctamente el key Exchange.

6. ¿Qué handshake lleva más tiempo, RSA o ECDHE?

ECDHE son 8 segundos mientras que RSA son 5 segundos.

6. ¿Qué handshake consume más datos, RSA o ECDHE?

$ECDHE = 583 + 2221 + 1253 + 2353 = 6410$  este consume más ya que envía las key Exchange en ambos sentidos.

$$\text{RSA} = 1462 + 1908 + 1502 + 1196 = 6068$$

5. ¿Cuál es la principal diferencia entre los handshakes con RSA y ECDHE?

Los handshake con RSA, el servidor le manda su clave pública al cliente, el cliente genera la clave simétrica y se la manda al servidor cifrada con la clave pública del cliente.

En cambio, con ECDHE, el servidor le manda su clave pública al cliente para autenticarse. Después crea una clave privada y una pública única para esta conexión y le manda la pública efímera al cliente. El cliente genera un par de claves efímeras. Genera la clave simétrica usando su clave privada efímera, la clave pública efímera del servidor, y los mensajes random del principio. El servidor genera el mismo número aleatorio usando su clave privada efímera, la clave pública efímera del cliente y los randoms. Ambos llegan al mismo número sin necesidad de enviárselo, es más seguro.

6. ¿Qué handshake lleva más tiempo, RSA o ECDHE?

ECDHE lleva más tiempo ya que tienen que generarse y enviarse el par de claves efímeras por parte del cliente y del servidor.

7. ¿Qué handshake consume más datos, RSA o ECDHE?

ECDHE consume más datos ya que el servidor tiene que enviar además de su clave pública para certificarse, una clave pública única para la conexión particular, y el cliente a su vez tiene que enviar su clave pública efímera al servidor. Esto en RSA no se hace ya que solo se envía la clave simétrica cifrada con la clave pública del servidor