**Práctica III.: Protocolos Seguros y Auditorías de Seguridad**

**1.** Tomando como base de trabajo el SSH pruebe sus diversas utilidades:

CONCEPTOS PREVIOS

Para entender cómo funciona el protocolo ssh debemos comprender los tipos de técnicas de encriptación que usa: (<https://www.hostinger.es/tutoriales/que-es-ssh>)

**- Encriptación simétrica:** Una clave puede ser usada para encriptar un mensaje para el servidor y tb para desencriptar mensajes recibidos por otros participantes. Cualquiera que tenga la clave puede encriptar y desencriptar mensajes para todo el mundo que contenga dicha clave.

Esta técnica es usada para encriptar la **conexión**, mientras que el par clave pública/privada se usa para la encriptación de la autenticación. En resumen, esta técnica protege la autenticación mediante password que se realiza.

**- Encriptación asimétrica:** La diferencia respecto a la encriptación simétrica es que para enviar información en una sóla dirección (servidor al cliente), se necesitan dos claves, una privada y otra pública.

La clave **pública puede compartirse** con cualquier participante. Está **asociada** con la **privada**, pero la privada no puede derivarse a partir de la pública. La relación matemática hace que sólo se pueda desencriptar con la clave privada, mensajes que estén encriptados con la clave pública correspondiente. Es decir, la clave pública no puede desencriptar mensajes que encripta ni mensajes que pueda enviarle la clave privada.

La clave **privada** tiene que mantenerse secreta y **nunca se deberá compartir**. Cualquiera entidad que pueda desencriptar un mensaje encriptado con una clave pública, es conocedor de la clave privada correspondiente.

La encriptación asimétrica es usada en el protocolo en diferentes sitios. Cuando se produce el intercambio de claves (key exchange) para preparar la encriptación simétrica que encriptará la sesión, se usa esta encriptación asimétrica. Los dos extremos producen un par de claves temporales y se intercambian las claves públicas las cuales se utilizarán como las claves secretas para la encriptación simétrica.

Otro uso de la encriptación asimétrica en el protocolo ssh, son las **unique host key**.

Son pares de claves únicas que están asociadas al servidor y al cliente. Las del servidor se utilizan para que el cliente pueda verificarlo (el cliente tiene una lista de claves públicas de los servidores con los que se conecta en $HOME/.ssh/known\_hosts). La de los clientes se utilizan para la **autenticación basada en claves** (en vez de por password). Para esto, el cliente comparte la clave pública al servidor ssh al que quiere acceder. El servidor copia la clave pública en el fichero **$HOME/.ssh/authorized\_keys**. Una vez que se establezca la encriptación simétrica para securizar la comunicación, el cliente se tiene que autenticar. Para esto el servidor usa la clave pública que le compartió el cliente anteriormente para encriptar un mensaje de prueba. El cliente, si tiene la clave privada correspondiente, será **capaz de desencriptarlo** y demostrar que tiene la clave privada correspondiente. Luego, el servidor puede preparar el entorno para el cliente.

**-Hashing:** Las funciones hash son un método de **creación de una firma** para un conjunto de información, son prácticamente únicas.

Usando la misma función hash y un mismo mensaje produce un mismo hash. Modificando una parte del mensaje, el hash que se produce es totalmente diferente. Un usuario no seŕa capaz de determinar el mensaje original a partir de un hash, pero sí que seŕa capaz de determinar el hash a partir del mensaje.

Se usan para comprobar la integración -> integridad? de los datos y para probar las autenticaciones.

Ssh usa los hashes para HMAC(hashed-based message authentication codes), se utiliza para verificar que los mensajes recibidos están **intactos** y no han sido modificados.

Como parte de la encriptación simétrica, un MAC(message authentication code) algoritmo es seleccionado. El **primero de la lista que el servidor soporte será el usado**.

Cada mensaje enviado después de la negociación de la encriptación, debe contener un MAC, para que el otro participante pueda comprobar la integridad del paquete. El MAC es **calculado a partir de la clave compartida simétricamente, el número de secuencia del paquete y el contenido del mensaje**.

**Cómo funciona SSH:**

PASOS:

**1º** - **Cliente pide** una conexión SSH.

**2º** - **Servidor** responde con una **versión** del protocolo preferida

**3º** - Clave **asimétrica exchange** y **negociación de algoritmos**

**4º** - **Autenticación** del cliente.

Cuando el cliente crea la conexión TCP, el servidor responde con la **versión** del protocolo que soporta. Si el cliente la acepta, la conexión continua. El servidor también provee su **clave pública** de host, con la que el cliente comprobará si es el servidor con el que quiere conectarse.

habla de la misma clave arriba y abajo?

El **servidor envía una clave de host única al cliente**. Si el servidor nunca se comunicó con el cliente, el cliente no conocerá esta clave y no se conectará. Para solventar esto, Openssh acepta esta clave de host una vez que el **usuario sea notificado y acepte esta clave**. En las siguientes conexiones, la clave de host del servidor será checkeada con la versión guardada con anterioridad, para comprobar que el cliente está conectándose con el servidor que desea.

Si en un futuro, esta clave de host no coincide, el usuario deberá de eliminar la versión guardada para que la conexión pueda ocurrir.

Una vez que el cliente contacta con el servidor se produce el **key exchange**, para que las dos partes construyan una capa de transporte segura. En este paso **se determina**:

- Algoritmo de **encriptación de clave pública**

- Algoritmo de **encriptación simétrica**

- Algoritmo de **autenticación de los mensajes**

- Algoritmo **hash**

En este punto, los dos extremos negocian la clave de la sesión usando una versión del algoritmo de **Diffie-Hellman**. Esto permitirá que los dos extremos lleguen a tener una clave de sesión secreta idéntica (encriptación simétrica).

**Pasos básicos** de este procedimiento:

* Los dos extremos adoptan un **número largo primo** que servirá como semilla.
* Los dos extremos adoptan un **generador de encriptado** (AES por ejemplo).
* Cada parte, independientemente, genera un **número primo secreto** que será su **clave privada**.
* Con la clave privada, el generador de encriptado y la semilla inicial, cada parte generará su **clave pública**.
* Se **intercambian las claves públicas**.
* Cada parte **usa su propia clave privada**, la clave **pública de la otra parte** y el número primo compartido al inicio (la **semilla**) para generar la **clave secreta compartida**. Esta clave **será idéntica** aunque se compute independientemente por cada parte.
* Esta clave secreta compartida (clave simétrica) será usada para **encriptar toda la comunicación posterior**.

El chiste esta en que aun que se intercepten la información que comparten, como cada uno genera independientemente la shared key, un atacante no podría sacarla dado que no sabe el alg de intercambio de clave no se conoce.

**Comienzo de la autenticación:**

Hay distintas maneras de autenticarse dependiendo de lo que acepte el servidor. La más básica sería mediante **password**, pero esta es poco segura debido a la limitación de la complejidad que pueden tener.

El método más recomendado y popular es el que usa **pares de clave SSH**. Estas pares de claves son asimétricas.

Este método comienza **después** de que se establezca la **encriptación simétrica**.

Los pasos son los siguientes:

* El **cliente envía un ID** representativo del par de claves sobre el que quiere autenticarse al servidor.
* El **servidor checkea el fichero authorized\_keys** en la cuenta del usuario al que el cliente está intentando acceder.
* Si se encuentra la **clave pública que machea con la ID solicitada**, el servidor genera un **número aleatorio lo encripta con esa clave pública**.
* Si el cliente tiene la clave privada asociada, **será capaz de descifrar** el mensaje y obtendrá el número original.
* El **cliente combina** este número junto con la **clave compartida secreta de la sesión que encripta la comunicación y calcula el hash MD5 con ese valor**.
* El cliente **envía este hash** al servidor como respuesta a la anterior pregunta.
* El servidor utiliza la **clave de la sesión** y el **número que generó anteriormente** para obtener el **MD5**. A continuación, **compara** los dos valores, si coincide entonces quiere decir que el cliente posee la clave privada correspondiente y que está autenticado.

EJERCICIO EN SI

a. Abra un shell remoto sobre SSH y analice el proceso que se realiza. Configure su

fichero ssh\_known\_hosts para dar soporte a la clave pública del servidor.

Establecer conexión: Lee ficheros de conf y se conecta por TCP

| OpenSSH\_7.9p1 Debian-10+deb10u2, OpenSSL 1.1.1d 10 Sep 2019 debug1: Reading configuration data /etc/ssh/ssh\_config debug1: /etc/ssh/ssh\_config line 19: Applying options for \* debug1: Connecting to 10.10.102.236 [10.10.102.236] port 22. debug1: Connection established. |
| --- |

Busca claves públicas del cliente y negocian version SSH.

| debug1: identity file /home/lsi/.ssh/id\_rsa type -1 debug1: identity file /home/lsi/.ssh/id\_rsa-cert type -1 debug1: identity file /home/lsi/.ssh/id\_dsa type -1 debug1: identity file /home/lsi/.ssh/id\_dsa-cert type -1 debug1: identity file /home/lsi/.ssh/id\_ecdsa type -1 debug1: identity file /home/lsi/.ssh/id\_ecdsa-cert type -1 debug1: identity file /home/lsi/.ssh/id\_ed25519 type -1 debug1: identity file /home/lsi/.ssh/id\_ed25519-cert type -1 debug1: identity file /home/lsi/.ssh/id\_xmss type -1 debug1: identity file /home/lsi/.ssh/id\_xmss-cert type -1 debug1: Local version string SSH-2.0-OpenSSH\_7.9p1 Debian-10+deb10u2 debug1: Remote protocol version 2.0, remote software version OpenSSH\_7.9p1 Debian-10+deb10u2 debug1: match: OpenSSH\_7.9p1 Debian-10+deb10u2 pat OpenSSH\* compat 0x04000000 debug1: Authenticating to 10.10.102.236:22 as 'lsi' |
| --- |

Se mandan el orden de preferencia de los algoritmos.

kex\_algoritms(rsa,diffie helman,ce) / host key algoritms

curve -> nos genera la clave de sesión para el simétrico

ecds -> para tu autenticar el servidor

chacha -> para el cifrado simétrico

| debug1: SSH2\_MSG\_KEXINIT sent debug1: SSH2\_MSG\_KEXINIT received debug1: kex: algorithm: curve25519-sha256 debug1: kex: host key algorithm: ecdsa-sha2-nistp256 debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none |
| --- |

Si conoce la ip, lo mira en $HOME/.ssh/known\_hosts y le manda un mensaje que el solo sabe descifrar.

Utilizando el algoritmo de cifrado de curva elíptica, claro

| debug1: expecting SSH2\_MSG\_KEX\_ECDH\_REPLY debug1: Server host key: ecdsa-sha2-nistp256 SHA256:GVcakYXLaj/ilPiW6iCXpqrG4eLRVTBqB8RLy4xOpds debug1: Host '10.10.102.236' is known and matches the ECDSA host key. debug1: Found key in /home/lsi/.ssh/known\_hosts:2 debug1: rekey after 134217728 blocks debug1: SSH2\_MSG\_NEWKEYS sent debug1: expecting SSH2\_MSG\_NEWKEYS debug1: SSH2\_MSG\_NEWKEYS received debug1: rekey after 134217728 blocks |
| --- |

chacha20-> supongo que sera la clave simétrica pero npi

| debug1: Will attempt key: /home/lsi/.ssh/id\_rsa debug1: Will attempt key: /home/lsi/.ssh/id\_dsa debug1: Will attempt key: /home/lsi/.ssh/id\_ecdsa debug1: Will attempt key: /home/lsi/.ssh/id\_ed25519 debug1: Will attempt key: /home/lsi/.ssh/id\_xmss debug1: SSH2\_MSG\_EXT\_INFO received debug1: kex\_input\_ext\_info: server-sig-algs=<ssh-ed25519,ssh-rsa,rsa-sha2-256,rsa-sha2-512,ssh-dss,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521> debug1: SSH2\_MSG\_SERVICE\_ACCEPT received |
| --- |

Autentica con clave pública ($/home/ssh/authorized\_keys) o si no con pwd

| debug1: Authentications that can continue: publickey,password debug1: Next authentication method: publickey debug1: Trying private key: /home/lsi/.ssh/id\_rsa debug1: Trying private key: /home/lsi/.ssh/id\_dsa debug1: Trying private key: /home/lsi/.ssh/id\_ecdsa debug1: Trying private key: /home/lsi/.ssh/id\_ed25519 debug1: Trying private key: /home/lsi/.ssh/id\_xmss debug1: Next authentication method: password lsi@10.10.102.236's password: |
| --- |

Problema: es peligroso el proceso de autenticación del servidor por parte del cliente. Si tengo 2000 usuarios en el cliente y cada uno se conecta a un servidor ssh, se realizan 2000 transferencias de la clave del servidor. Es decir, me la pueden liar 2000 veces si se hacen pasar por un servidor. Para esto, configurar manualmente en **/etc/ssh/ssh\_known\_hosts** insertar ahí las claves públicas de los servidores a los que nos queremos conectar. En caso de que no se encuentre en $HOME/.ssh/known\_hosts se buscan en /etc/ssh …

| **nano /$HOME/.ssh/known\_hosts # borrar claves (para un usuario) nano /etc/ssh/ssh\_known\_hosts # meter la clave pública del servidor ssh 10.10.102.233 >> /etc/ssh/ssh\_known\_hosts** |
| --- |

Si no pide lo de figerprinting … está bien.

b. Haga una copia remota de un fichero utilizando un algoritmo de cifrado

determinado. Analice el proceso que se realiza.

**/etc/ssh/ssh\_config** : Cipher -> podemos indicar qué algoritmos de cifrados permitimos y el orden de preferencia. (no lo usamos)

**ssh -Q cipher** : podemos ver los diferentes algoritmos de cifrado que podemos utilizar.

| scp -o Ciphers=aes128-ctr lsi@10.10.102.233:index.html index.html |
| --- |

fuerza a scp a utilizar un determinado cifrado.

Destacar que 1º va lo que se copia y 2º va a donde se copia

| debug1: kex: server->client cipher: aes128-ctr MAC: umac-64-etm@openssh.com compression: none debug1: kex: client->server cipher: aes128-ctr MAC: umac-64-etm@openssh.com compression: none |
| --- |

c. Configure su cliente y servidor para permitir conexiones basadas en un esquema

de autenticación de usuario de clave pública.

**1º** - Crear par clave pública/privada en el cliente, usuario lsi:

| ssh-keygen -t rsa |
| --- |

**-t** -> selecciona el algoritmo de cifrado de la clave, por defecto usa rsa así se puede obviar.

**Cuando pide llave de paso** -> poner en blanco (la llave de paso cifra la clave privada por si nos la roban, además si se inserta una se pedirá cuando se realice la conexión ssh, y en este caso no nos interesa).

**id\_rsa** -> clave privada

**id\_rsa.pub** -> clave pública

**2º** - Meter la clave pública en el servidor:

| ssh-copy-id -i $HOME/.ssh/id\_rsa.pub lsi@10.10.102.servidor |
| --- |

**-i** -> especifica el identity\_file

Copia directamente en el fichero **$HOME/.ssh/authorized\_keys** del servidor la clave pública del cliente. (En algunos sitios sólo hacen ssh-copy-id lsi@10.10.102.servidor pero idk). Pide la contraseña para acceder al servidor.

También se puede hacer manualmente:

**cat ~/.ssh/id\_rsa.pub | ssh username@remote\_host "mkdir -p ~/.ssh && cat >> ~/.ssh/authorized\_keys"**

Igual hay que hacer un cat authorized\_keys >> authorized2\_keys (dijo que igual hacía falta).

d. Mediante túneles SSH securice algún servicio no seguro.

**Redireccionamiento de puertos ssh**: es un mecanismo ssh para tunelizar puertos de aplicaciones desde el cliente al servidor o viceversa. Usado para añadir criptología a aplicaciones, saltarse firewalls...

Tres tipos: local forwarding, remote forwarding, dynamic forwarding.

| ssh -P -L sourcePort:forwardToHost:onPort connectToHost |
| --- |

Significa que se conectará mediante ssh a connectToHost y que se redireccionará todo el tráfico generado en el sourcePort local de la máquina forwardToHost al puerto onPort de la máquina connectToHost. Las dos máquinas tiene que ser alcanzables

**-P** -> XXX

**-L** -> local forwarding.

**sourcePort** -> es el que se quiere securizar

**onPort** -> nuevo puerto securizado (pero ojo, ese puerto no es de esa máquina, es de la que lanza el ssh??)

Securizar conexión a nuestro servidor apache:

**ssh -L 8080:localhost:80 lsi@10.10.102.233**

**pedro, tienes que revisar el forwardtoHost**

Las conexiones realizadas al puerto 8080 de nuestra máquina (tiene que ser alcanzable por 10.10.102.233, conectarse por vpn) serán redireccionadas al puerto 80 de la máquina de prácticas dónde corre nuestro apache. Es decir, en nuestra máquina no está corriendo ningún servidor apache, simplemente se redirecciona a la máquina donde sí está corriendo (máquina de prácticas).

Una vez hecho, ir al navegador -> localhost:8080/index.html

e. “Exporte” un directorio y “móntelo” de forma remota sobre un túnel SSH.

| apt-get install sshfs |
| --- |

1º Crear directorio en el que se montará el directorio remoto:

| mkdir /home/lsi/Compartido\_sshfs |
| --- |

2º Montar directorio remoto:

| sudo sshfs -o allow\_other,default\_permissions lsi@10.10.102.233:/home/lsi /home/lsi/Compartido\_sshfs |
| --- |

* directorio remoto(lsi@10…) - directorio donde quieres montarlo (home/…)

En caso de tener configurado el loggin con clave pública:

| sudo sshfs -o allow\_other,default\_permissions,IdentityFile=~/.ssh/id\_rsa lsi@10.10.102.233:/home/lsi /home/lsi/Compartido\_sshfs |
| --- |

**-o**: opciones :

**allow\_other** : permite acceso a otros usuarios.

**default\_permissions** : permisos por defecto.

3º Si se quiere desmontar:

| sudo umount /home/lsi/Compartido\_sshfs |
| --- |

f. PARA PLANTEAR DE FORMA TEÓRICA.: Securice su servidor considerando que

únicamente dará servicio ssh para sesiones de usuario desde determinadas IPs.

Configurar fichero de configuración:

| cd /etc/ssh/sshd\_config |
| --- |

* Ocultar puerto:

**Port 1111** (otro diferente al 22)

* Restringir acceso al usuario root:

**PermitRootLogin no**

* Restringir protocolos:

**Protocol 2** (el 1 es inseguro)

* Limitar usuarios:

**AllowUsers lsi sergio pedro**

**AllowUsers lsi@10.10.\* sergio@10.\***

* Limitar número de intentos al meter la password:

**MaxAuthTries 3**

* Limitar número de pantallas de loggin simultáneas desde una misma ip:

**MaxStartups 1**

* Establecer máximo tiempo de inactividad:

**ClientAliveInterval 300**

capar todos los puertos y sacar todos los servicios, el firewall viene dpm

**2.** Tomando como base de trabajo el servidor Apache2

Para entender SSL/TLS:

Es un protocolo de la capa de transporte que proporciona cifrado, integridad...

SSL es el predecesor de TLS.

Cómo se establece la sesión SSL entre el cliente y el servidor?

1º El cliente **envía las versiones** del protocolo que soporta, el ID de la sesión, la lista de conjuntos de algoritmo de cifrado que maneja, etc.

2º El **servidor responde con la versión** del protocolo que se va a usar, el ID de la sesión, el conjunto de algoritmo de cifrado que se va a usar, etc.

3º Dependiendo del conjunto de algoritmos de cifrado que se seleccione, el servidor enviará su certificado digital (Por ejemplo, si se elige RSA para el key exchange entonces el cliente necesitará la clave pública del servidor, por lo que el servidor cuando se la envíe, debe de autenticar que la clave pública es de él y para eso envía el certificado).

Si el **servidor necesita el certificado del cliente**, enviará un mensaje de solicitud de su certificado junto a una lista con las autoridades de certificado que acepta.

4º El **cliente enviará** su certificado si se lo pide el servidor.

5º **Cambian el conjunto de algoritmos de cifrado**.

Estos pasos dependen de qué algoritmo de intercambio de claves se escoja (key exchange), así que hay que entenderlo como una aproximación.

Para qué se realiza este handshake?

- Para **negociar** el conjunto de algoritmos de cifrado que se van a utilizar en la transferencia de datos.

- Para establecer y compartir una **clave de sesión** entre el cliente y el servidor.

- Opcionalmente para que el servidor se autentique al cliente.

- Opcionalmente para que el cliente se autentique al servidor.

Cabe destacar que en el conjunto de algoritmos de cifrado se determina:

- Método de intercambio de claves (Key Exchange).

- Cifrado de la transferencia de datos.

- Función hash para integridad.

Https es el protocolo Http pero con SSL/TLS.

Para qué sirven los certificados?

Se utilizan para verificar que un extremo es quien dice ser. Por ejemplo, cuando el servidor le envía su clave pública.

Un **certificado asocia una clave pública con su identidad** (servidor, individual...). También incluye la **identificación de la autoridad certificadora y su firma** (la cifra con su clave privada).

Estos campos a cubrir son especificados en estándares, como el X.509.

Las autoridades certificadoras son las encargadas de certificar una solicitud de firma de certificado CSR. Este CSR contiene la clave pública de la entidad que quiere certificarse e información relacionada de dicha entidad. La CA comprobará su identidad y firma el certificado con su clave privada.

Cuando esta entidad quiera identificarse delante de un tercero, le pasará el certificado al tercero. Este tendrá un conjunto de CA conocidas con sus respectivas claves públicas. Con esto podrá comprobar la firma digital de la CA (por la propiedad de las claves asimétricas, donde algo cifrado por la clave privada puede ser cifrado por la pública correspondiente).

Las CA también pueden certificar otras CA. Esto crea que a veces una cadena de certificados, por lo que el usuario tendrá que recorrer dicha cadena hasta que encuentre un CA en el que confíe.

Quién certifica a los CA de más alto nivel??

En estos casos los CA se certifican a ellos mismos. Los navegadores tienen una lista de CAs conocidos que se saben que son de fiar. A la hora de aceptar un CA que esté certificado por el mismo tenemos que estar muy seguros de que es de fiar.

a. Configure una Autoridad Certificadora en su equipo.

openssl

| cd /usr/lib/ssl/misc/ ./CA.pl -newca  #Enter PEM pass phrase: ninoelprofe  #country: ES  #State: Coruna  #Locality Name: Coruna  #Organization Name: LsiInc  #Organizational Unit Name: ClaseTresCuatro  #Common Name: SergioCA  #Email: lsi@localhost  #Enter pass phrase: ninoelprofe  #el de pedro:  countryName = ES  stateOrProvinceName = Coruna  organizationName = LsiInc  organizationalUnitName = ClaseTresCuatro  commonName = Pedro Guijas Certification Authority  emailAddress = lsi@localhost |
| --- |

Archivos generados (en demoCA):

* cacert.pem -> certificado con clave pública de la CA
* private/cakey.pem -> privada de la CA

b. Cree su propio certificado para ser firmado por la Autoridad Certificadora.

Bueno, y fírmelo.

| ./CA.pl -newreq-nodes  #country: ES  #State: Coruna  #Locality Name: Coruna  #Organization Name: LsiInc  #Organizational Unit Name: ClaseTresCuatro  #Common Name: web  #Email: lsi@localhost  ./CA.pl -sign |
| --- |

node: generar certíficado sin protección con contraseña (a la clave privada)

Archivos generados:

* newcert.pem -> certificado generado y firmado
* newkey.pem -> clave privada del certificado
* newreq.pem -> petición del certificado para firmar

(con scp nos pasamos los certificados previamente de forma que uno genera el del otro)

| cp newkey.pem /etc/ssl/private/xxxxx.key  cp newcert.pem /etc/ssl/certs/xxxxx.crt |
| --- |

c. Configure su Apache para que únicamente proporcione acceso a un

determinado directorio del árbol web bajo la condición del uso de SSL.

Considere que si su la clave privada está cifrada en el proceso de arranque su

máquina le solicitará la correspondiente frase de paso, pudiendo dejarla

inalcanzable para su sesión ssh de trabajo.

/etc/apache2/sites-available/default-ssl

| SSLEngine on SSLCertificateFile /etc/ssl/certs/xxxxxx.crt SSLCertificateKeyFile /etc/ssl/private/xxxxx.key |
| --- |

| a2enmod ssl a2ensite default-ssl systemctl reload apache2 |
| --- |

**En el cliente (el que haga la petición):**

Mete su CA en /**usr/local/share/ca-certificates** EXTENSIÓN .**crt**

Luego:

| **cp cacertsergio.pem /usr/local/share/ca-certificates/cacertsergio.crt update-ca-certificates** |
| --- |

Esto hay que hacerlo para meter el CA en los navegadores como wget, lynx … Porque en **/etc/ssl/certs/ca-certificates.crt** es donde miran los CA que tienen.

Para verificar:

| cp -p /usr/lib/ssl/misc/demoCA/cacert.pem /etc/ssl/certs/xxxx.pem cd /etc/ssl/certs/ openssl x509 -in xxxx.pem -noout -hash #genera hash YYYY ln -s xxxx.pem YYYY.0 #enlace al hash con extensión .0 openssl verify xxxx.pem |
| --- |

Comprobar la conexión:

| openssl s\_client -CApath /etc/ssl/certs/ -connect webpedro:443 |
| --- |

Finalmente denegamos el acceso por el puerto 80 (http) para que solo se pueda acceder a la carpeta onlyhttps por el puerto 443. De esta forma añadimos a /etc/apache2/sites-available/000-default.conf las siguientes líneas **justo antes** del </VirtualHost>

| <Directory /var/www/html/onlyhttps>  Options FollowSymLinks  AllowOverride None  Order deny,allow  Deny from all  </Directory> |
| --- |

Para probar que todo funciona bien:

| w3m https://webpedro openssl verify /etc/ssl/certs/CertParaPedro.crt openssl verify /etc/ssl/certs/CertParaSergio.crt |
| --- |

[Web del Pana Vicente Navarro](http://www.vicente-navarro.com/blog/2009/02/22/crear-los-certificados-ssl-para-nuestro-servidor-web-https-con-apache-openssl-y-debian-lenny)

**3.** Tomando como base de trabajo el openVPN deberá configurar una **VPN** entre dos

equipos virtuales del laboratorio que **garanticen la confidencialidad entre sus**

**comunicaciones**.

Server:

| apt install openvpn apt install openssl lsmod | grep tun # 2 siguientes comandos si no esta cargado modprobe tun echo tun >> /etc/modules cd /etc/openvpn openvpn --genkey --secret clave.key #(crea /etc/oppenvpn/clave.key) nano /etc/openvpn/tunel.conf |
| --- |

tunel.conf:

| local 10.10.102.236 remote 10.10.102.233 dev tun1 port 5555 comp-lzo user nobody ping 15 ifconfig 172.160.0.1 172.160.0.2 secret /etc/openvpn/clave.key |
| --- |

No server:

| apt install openvpn apt install openssl lsmod | grep tun # 2 siguientes comandos si no esta cargado modprobe tun echo tun >> /etc/modules cd /etc/openvpn scp lsi@10.10.102.236:/etc/openvpn/clave.key /etc/openvpn/clave.key  nano /etc/openvpn/tunel.conf |
| --- |

| local 10.10.102.233 remote 10.10.102.236 dev tun1 port 5555 comp-lzo user nobody ping 15 ifconfig 172.160.0.2 172.160.0.1 secret /etc/openvpn/clave.key |
| --- |

Ambos:

| openvpn --verb 5 --config /etc/openvpn/tunel.conf & |
| --- |

lo mismo, compartimos la clave en vez de crearla

el resto es lo mismo, en conf cambiamos las íp, igual con el config (primero la .2)

secularizar el ntp y el van los metemos aquí

**7.** En este punto, cada máquina virtual será servidor y cliente de diversos servicios (NTP,

syslog, ssh, web, etc.). Configure un “firewall stateful” de máquina adecuado a la

situación actual de su máquina.

Usaremos iptables, debemos conocer las siguientes opciones:

* F -> deleting all the rules one by one
* X -> delete every non-builtin chain in the table.
* A -> append. agrega una regla a una cadena.
* P -> policy explica al kernel qué hacer con los paquetes que no coincidan con ninguna regla.
* -t -> tabla(defecto filter)
* -p -> la regla se aplica a un protocolo
* -s -> source ip (origen)
* -d -> destination ip (destino)
* -i -> interface
* -j -> se aplica una poítica

firewall pedro:

| !/bin/bash  MI\_IP\_ENS33="10.10.102.236" MI\_IP\_ENS34="10.10.150.236" MI\_IP\_ENS33\_V6="fe80::250:56ff:fe97:fed8" MI\_IP\_TUN1="172.160.0.1" IP\_COMP\_ENS33="10.10.102.233" IP\_COMP\_ENS34="10.10.150.233" IP\_COMP\_ENS33\_V6="fe80::250:56ff:fe97:c808" IP\_COMP\_TUN1="172.160.0.2" EDUROAM="10.20.32.0/21" VPN="10.30.8.0/21" IP\_DNS="10.10.102.27" TIME=120  echo "" echo "Firewall ON."  echo "Definiendo politica por defecto" iptables -P INPUT DROP iptables -P OUTPUT DROP iptables -P FORWARD DROP ip6tables -P INPUT DROP ip6tables -P OUTPUT DROP  echo "Definiendo reglas de entrada" iptables -A INPUT -i lo -j ACCEPT iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT ip6tables -A INPUT -i lo -j ACCEPT ip6tables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT  iptables -A INPUT -p tcp -s $EDUROAM,$VPN,$IP\_DNS,$IP\_COMP\_ENS33,$IP\_COMP\_ENS34 --dport 22 -d $MI\_IP\_ENS33,$MI\_IP\_ENS34 -m conntrack --ctstate NEW -j ACCEPT ip6tables -A INPUT -p tcp -s $IP\_COMP\_ENS33\_V6 --dport 22 -d $MI\_IP\_ENS33\_V6 -m conntrack --ctstate NEW -j ACCEPT iptables -A INPUT -p tcp -s $IP\_COMP\_ENS33,$IP\_COMP\_ENS34,$IP\_COMP\_TUN1 --dport 80 -m conntrack --ctstate NEW -j ACCEPT iptables -A INPUT -p tcp -s $IP\_COMP\_ENS33,$IP\_COMP\_ENS34 --dport 443 -m conntrack --ctstate NEW -j ACCEPT iptables -A INPUT -p tcp -s $IP\_COMP\_ENS33 --dport 514 -m conntrack --ctstate NEW -j ACCEPT  iptables -A INPUT -p tcp -j REJECT --reject-with tcp-reset iptables -A INPUT -p udp -j REJECT --reject-with icmp-port-unreachable iptables -A INPUT -j REJECT --reject-with icmp-proto-unreachable ip6tables -A INPUT -p tcp -j REJECT --reject-with tcp-reset ip6tables -A INPUT -j REJECT  echo "Definiendo reglas de salida" iptables -A OUTPUT -o lo -j ACCEPT iptables -A OUTPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT  ip6tables -A OUTPUT -o lo -j ACCEPT ip6tables -A OUTPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT  iptables -A OUTPUT -p tcp -s $MI\_IP\_ENS33,$MI\_IP\_ENS34 --dport 22 -m conntrack --ctstate NEW -j ACCEPT ip6tables -A OUTPUT -p tcp -s $MI\_IP\_ENS33\_V6 --dport 22 -m conntrack --ctstate NEW -j ACCEPT iptables -A OUTPUT -p udp -s $MI\_IP\_ENS33 -d $IP\_DNS --dport 53 -m conntrack --ctstate NEW -j ACCEPT iptables -A OUTPUT -p tcp -s $MI\_IP\_ENS33,$MI\_IP\_ENS34,$MI\_IP\_TUN1 -d $IP\_COMP\_ENS33,$IP\_COMP\_ENS34,$IP\_COMP\_TUN1 --dport 80 -m conntrack --ctstate NEW -j ACCEPT iptables -A OUTPUT -p tcp -s $MI\_IP\_ENS33,$MI\_IP\_ENS34 -d $IP\_COMP\_ENS33,$IP\_COMP\_ENS34 --dport 443 -m conntrack --ctstate NEW -j ACCEPT iptables -A OUTPUT -p tcp -s $MI\_IP\_ENS33 -d deb.debian.org,security.debian.org,downloads.metasploit.com --dport 80 -m conntrack --ctstate NEW -j ACCEPT iptables -A OUTPUT -p udp -s $MI\_IP\_ENS33,$MI\_IP\_ENS34 -d $IP\_COMP\_ENS33 --dport 123 -m conntrack --ctstate NEW -j ACCEPT iptables -A OUTPUT -p udp -s $MI\_IP\_ENS33 -d $IP\_COMP\_ENS33 --sport 5555 -m conntrack --ctstate NEW -j ACCEPT  sleep $TIME  echo "Firewall OFF." iptables -F iptables -X iptables -P INPUT ACCEPT iptables -P OUTPUT ACCEPT iptables -P FORWARD ACCEPT ip6tables -F ip6tables -X ip6tables -P INPUT ACCEPT ip6tables -P OUTPUT ACCEPT ip6tables -P FORWARD ACCEPT echo "Eliminandas todas las reglas." |
| --- |

firewall sergio:

| #!/bin/bash  MI\_IP\_ENS33="10.10.102.233" MI\_IP\_ENS34="10.10.150.233" MI\_IP\_TUN1="172.160.0.2" MI\_IP\_ENS33\_V6="fe80::250:56ff:fe97:c808" IP\_COMP\_ENS33="10.10.102.236" IP\_COMP\_ENS34="10.10.150.236" IP\_COMP\_ENS33\_V6="fe80::250:56ff:fe97:fed8" IP\_COMP\_TUN1="172.160.0.1" EDUROAM="10.20.32.0/21" VPN="10.30.8.0/21" IP\_DNS="10.10.102.27" TIME=60  echo "" echo "Firewall ON."   echo "Definiendo politica por defecto" iptables -P INPUT DROP iptables -P OUTPUT DROP iptables -P FORWARD DROP ip6tables -P INPUT DROP ip6tables -P OUTPUT DROP  echo "Definiendo reglas de entrada" iptables -A INPUT -i lo -j ACCEPT iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT ip6tables -A INPUT -i lo -j ACCEPT ip6tables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT  iptables -A INPUT -p tcp -s $EDUROAM,$VPN,$IP\_DNS,$IP\_COMP\_ENS33,$IP\_COMP\_ENS34 -d $MI\_IP\_ENS33,$MI\_IP\_ENS34 --dport 22 -m conntrack --ctstate NEW -j ACCEPT ip6tables -A INPUT -p tcp -s $IP\_COMP\_ENS33\_V6 -d $MI\_IP\_ENS33\_V6 -m conntrack --dport 22 --ctstate NEW -j ACCEPT iptables -A INPUT -p tcp -s $IP\_COMP\_ENS33,$IP\_COMP\_ENS34,$IP\_COMP\_TUN1 -d $MI\_IP\_ENS33,$MI\_IP\_ENS34,$MI\_IP\_TUN1 --dport 80 -m conntrack --ctstate NEW -j ACCEPT iptables -A INPUT -p tcp -s $IP\_COMP\_ENS33,$IP\_COMP\_ENS34 -d $MI\_IP\_ENS33,$MI\_IP\_ENS34 --dport 443 -m conntrack --ctstate NEW -j ACCEPT iptables -A INPUT -p udp -s $IP\_COMP\_ENS33 -d $MI\_IP\_ENS33 --dport 123 -m conntrack --ctstate NEW -j ACCEPT iptables -A INPUT -p tcp -s $IP\_COMP\_ENS33 -d $MI\_IP\_ENS33 --sport 514 -m conntrack --ctstate NEW -j ACCEPT   iptables -A INPUT -p tcp -j REJECT --reject-with tcp-reset iptables -A INPUT -p udp -j REJECT --reject-with icmp-port-unreachable iptables -A INPUT -j REJECT --reject-with icmp-proto-unreachable ip6tables -A INPUT -p tcp -j REJECT --reject-with tcp-reset ip6tables -A INPUT -j REJECT  echo "Definiendo reglas de salida" iptables -A OUTPUT -o lo -j ACCEPT iptables -A OUTPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT  ip6tables -A OUTPUT -o lo -j ACCEPT ip6tables -A OUTPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT  iptables -A OUTPUT -p tcp -s $MI\_IP\_ENS33,$MI\_IP\_ENS34 --dport 22 -m conntrack --ctstate NEW -j ACCEPT ip6tables -A OUTPUT -p tcp -s $MI\_IP\_ENS33\_V6 --dport 22 -m conntrack --ctstate NEW -j ACCEPT iptables -A OUTPUT -p udp -s $MI\_IP\_ENS33 -d $IP\_DNS --dport 53 -j ACCEPT iptables -A OUTPUT -p tcp -s $MI\_IP\_ENS33,$MI\_IP\_ENS34,$MI\_IP\_TUN1 -d $IP\_COMP\_ENS33,$IP\_COMP\_ENS34,$IP\_COMP\_TUN1 --dport 80 -m conntrack --ctstate NEW -j ACCEPT iptables -A OUTPUT -p tcp -s $MI\_IP\_ENS33,$MI\_IP\_ENS34 -d $IP\_COMP\_ENS33,$IP\_COMP\_ENS34 --dport 443 -m conntrack --ctstate NEW -j ACCEPT iptables -A OUTPUT -p tcp -s $MI\_IP\_ENS33 -d deb.debian.org,security.debian.org,downloads.metasploit.com --dport 80 -m conntrack --ctstate NEW -j ACCEPT iptables -A OUTPUT -p tcp -s $MI\_IP\_ENS33 -d $IP\_COMP\_ENS33 --dport 514 -m conntrack --ctstate NEW -j ACCEPT iptables -A OUTPUT -p udp -s $MI\_IP\_ENS33 -d $IP\_COMP\_ENS33 --sport 5555 -m conntrack --ctstate NEW -j ACCEPT  sleep $TIME  echo "Firewall OFF." iptables -F iptables -X iptables -P INPUT ACCEPT iptables -P OUTPUT ACCEPT iptables -P FORWARD ACCEPT ip6tables -F ip6tables -X ip6tables -P INPUT ACCEPT ip6tables -P OUTPUT ACCEPT ip6tables -P FORWARD ACCEPT echo "Eliminadas todas las reglas." |
| --- |