3ª Practica de LSI

Lexislación e Seguridade Informática (Universidade da Coruña)

1) Explicacion SSH

Punto 1: Abrir conexión (Establecer conexión: el cliente lee los ficheros de configuración y establece la conexión con el servidor remoto. Además, se comprueba la versión del protocolo SSH que utilizan ambos por temas de compatibilidad, se abre la conexión entre el cliente y el servidor. El cliente lee su fichero de configuración, el ssh_config; el sshd_config es el del servidor. Lo mira y abre la conexión al puerto 22):

OpenSSH_7.2p2 Ubuntu-4ubuntu2.6, OpenSSL 1.0.2g 1 Mar 2016

*Lectura del fichero de configuración

debug1: Reading configuration data /etc/ssh/ssh_config
debug1: /etc/ssh/ssh_config line 19: Applying options for *
debug1: Connecting to 10.10.102.84 [10.10.102.84] port 22.
debug1: Connection established.

Punto 2: Seleccionar la versión a utilizar y autenticación (Negociación del algoritmo de cifrado: ambos extremos conversan para saber qué algoritmos conocen y se ponen de acuerdo para utilizar uno de ellos, el usuario que se conecta mira si tiene una serie de ficheros. Se selecciona qué versión de ssh va a utilizarse, hay versión 1 y versión 2; nosotros tenemos versión 2. Lo mejor, en caso de que sea la versión 1, es no conectarse):

debug1: key_load_public: No such file or directory
debug1: identity file /home/alex/.ssh/id_rsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/alex/.ssh/id_rsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/alex/.ssh/id_dsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/alex/.ssh/id_dsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/alex/.ssh/id_ecdsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/alex/.ssh/id_ecdsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/alex/.ssh/id_ed25519 type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/alex/.ssh/id_ed25519-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.6
debug1: Remote protocol version 2.0, remote software version OpenSSH_7.4p1 Debian-10+deb9u4
debug1: match: OpenSSH_7.4p1 Debian-10+deb9u4 pat OpenSSH* compat 0x04000000
debug1: Authenticating to 10.10.102.84:22 as 'lsi'

Punto 3: Negociación de parámetros (ofrecidos por el cliente) (Intercambio de clave de sesión: el cliente recibe la clave pública del servidor, comprueba con ella la autenticidad de la máquina (mira si está en el fichero /etc/ssh/ssh_known_hosts, o sino en $HOME/ssh/known_hosts) y cifra, también con ella, la clave de sesión. Después se la envía (cifrada) al servidor que es el único que puede descifrarla con su clave privada. Aquí lo que se negocia es qué algoritmos vamos a utilizar en la conexión; normalmente es el cliente el que propone. Se sacan los algoritmos del fichero de configuración. Cliente y servidor miran sus líneas de configuración y se ponen de acuerdo en los comunes):

debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: curve25519-sha256@libssh.org
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

Punto 4: Se inicia el intercambio de la clave de sesión (Se realiza la autenticación del servidor. Pregunta si te fías de esa clave si es la primera vez, en nuestro caso ya no nos pregunta):

debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: Server host key: ecdsa-sha2-nistp256 SHA256:uKJ5IlbkROry6HwRrwcYVlUxGwpGNkmJorMRDUNXnFc
debug1: Host '10.10.102.84' is known and matches the ECDSA host key.
debug1: Found key in /home/alex/.ssh/known_hosts:3
debug1: rekey after 134217728 blocks

Punto 5: Se establece la clave de sesión utilizando la pública del servidor (Se genera una clave de sesión y se cifra con la pública del servidor y se envía. ¿Quién puede descifrar eso? El servidor, que tiene la privada. Y esa clave de sesión es la que se utiliza en toda la sesión. Con esto se cifran las conexiones y se autentica el servidor, para tener la certeza de que es él y no otro):

debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_EXT_INFO received
debug1: kex_input_ext_info: server-sig-algs=<ssh-ed25519,ssh-rsa,ssh-dss,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521>
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey

debug1: Offering RSA public key: alejandro.garcia.rodriguez@udc.es
debug1: Authentications that can continue: publickey,password
debug1: Trying private key: /home/alex/.ssh/id_rsa
debug1: Trying private key: /home/alex/.ssh/id_dsa
debug1: Trying private key: /home/alex/.ssh/id_ecdsa
debug1: Trying private key: /home/alex/.ssh/id_ed25519

Punto 6: Autenticación de usuario (primero lo intenta mediante clave pública, es decir, el servidor mira en el fichero $HOME/.ssh/authorized_keys (o authorized_keys2, dependiendo de la distribución) si está registrada la clave pública del cliente (puede probar con varias claves si no lo consigue a la primera: rsa, ecdsa, ed25519...), si ninguna de las claves registradas coincide, pregunta la contraseña del usuario (en /etc/ssh/sshd_config se puede indicar que no pida contraseña nunca y sólo admita autenticación por clave pública, para evitar ataques de password guessing)):

debug1: Next authentication method: password
lsi@10.10.102.84's password: x.x.x.x
debug1: Authentication succeeded (password).
Authenticated to 10.10.102.84 ([10.10.102.84]:22).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: pledge: network
debug1: client_input_global_request: rtype hostkeys-00@openssh.com want_reply 0
debug1: Sending environment.
debug1: Sending env LANG = es_ES.UTF-8

***Aquí se acaba el debug.**

A continuación haremos la segunda parte del apartado a)

Configurar el fichero **ssh_known_hosts** (el cual por defecto no existe sino que hay que crearlo) para dar soporte a la clave pública del servidor.

En el archivo **/etc/ssh/ssh_known_hosts** guardamos las claves públicas a nivel de máquina (**/home/lsi/.ssh/known_hosts** las guarda a nivel de usuario, siendo lsi el usuario en este caso).
Para crear **/etc/ssh/ssh_known_hosts** ejecutamos:

```
cp /home/lsi/.ssh/known_hosts /etc/ssh/ssh_known_hosts
```

El fichero contiene 3 valores: el hash, el tipo de cifrado y la clave en sí.
Podemos sustituir el hash del nombre del equipo por la IP, para hacer las cosas más fácil. Ejemplo con el nombre sustituido por la IP:

```
root@debian:/home/lsi/.ssh# cat /home/lsi/.ssh/known_hosts


10.10.102.189 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNT
YAAABBBD88+qJBmnruVaATtz1GOCaFFziDOY6LI1IRSoigL0zIow+DVU0NyhNWBJyWBd8Gm4YL+0fXM+
vxIEn0zfywihk=
```

También se puede hacer el siguiente comando:
>ssh-keyscan 10.10.102.43 >> /etc/ssh/ssh_known_hosts (Este comando devuelve las claves públicas del host indicado)

En **/etc/ssh** están las claves de servidor:
ssh_host_dsa_key
ssh_host_dsa_key.pub
ssh_host_rsa_key
ssh_host_rsa_key.pub
ssh_host_ecdsa_key
ssh_host_ecdsa_key.pub
ssh_host_ed25519_key
ssh_host_ed25519_key.pub

Información disponible en https://www.ssh.com/ssh/sshd_config/

Ficheros:

Los ficheros que se encuentran en /etc pertenecen a todos los usuarios, son ficheros del sistema, así que no pertenecen a un usuario en concreto, si no a todos.

/etc/ssh/ssh_config (configuración cliente)
/etc/ssh/sshd_config (configuración servidor)

Los siguientes 4 ficheros pertenecen al usuario en concreto (fijarse que usamos $HOME)

$HOME/.ssh/identity (claves públicas y privadas de usuario)
$HOME/.ssh/id_rsa
$HOME/.ssh/id_dsa
$HOME/.ssh/known_hosts (claves públicas de los servicios a los que nos conectamos, las cuales se incluyen de forma automática)

/etc/ssh/ssh_known_hosts (claves públicas de los servidores a los que nos conectamos, las cuales se mantienen de forma manual)

En nuestro caso, cogemos del servidor su **/etc/ssh/ssh_host_rsa_key.pub** y

lo metemos en nuestros clientes en **/etc/ssh/ssh_known_hosts**. Iniciando la línea con la IP del servidor y un espacio.
Si utilizamos curvas elípticas, ecdsa, entonces meteremos en el fichero **/etc/ssh/ssh_host_ecdsa_key.pub** en el **/etc/ssh_known_hosts**

O meter todas las públicas para que, aunque se utilice el asimétrico que se utilice, disponer de la clave pública.

También podemos hacer, en lugar de los anterior, e introducir todas directamente (rsa, ecdsa, ed25519):
>ssh-keyscan 10.10.102.servidor-ssh >> /etc/ssh/ssh_known_hosts

De esta forma, utilizamos estas claves públicas para el intercambio de la clave de sesión, en lugar de que nos la remita. Así evitamos inyecciones de clave.

Comprobar borrando el contenido del **$HOME/.ssh/known_hosts** del usuario y conectado.

Lanzar una sesión X sobre una conexión ssh
>ssh -X lsi@10.10.102.servidor-ssh (estando como lsi)

Se debe tener en el servidor, en concreto el fichero **/etc/ssh/sshd_config**

X11forwarding yes

2) Configuración de las claves públicas y privadas del servidor /etc/ssh-known-host qué son las claves (servidor para intercambiar claves de sesión y autenticar servidor)

La autenticación con clave pública para conectarse a un servidor remoto usando el protocolo SSH funciona con dos claves: una pública y otra privada. Para entender el funcionamiento se suele recurrir a la metáfora del candado y la llave. La clave pública funciona como un candado y la privada como la llave. El candado se colocará en el servidor remoto al que se quiere acceder; cuando se intenta acceder se comprobará que la máquina que intenta conectar tiene la llave, la clave privada.

/etc/ssh/sshd_config debe estar a yes
RSAAuthentication yes
PubkeyAuthentication yes

ssh-keygen -t rsa
ssh-keygen -t ecdsa
ssh-keygen -t ed25519

scp /root/.ssh/*.pub lsi@10.10.102.55:.ssh/

cd .ssh
cat id_rsa.pub >> authorized_keys

rm id_rsa.pub

ssh root@10.10.102.55

3) Securizar servicio por ssh (redireccion de trafico a traves de conexion ssh con lynx o alguno de esos)

Con este comando creamos un tunel SSH local que nos permite securizar servicio de manera que al acceder al puerto 10080 del compañero, nos redirige al nuestro que en este caso es el del apache de forma segura.

---

ssh -P -L 10080:10.10.102.52:80 lsi@10.10.102.52

4) Autoridad certificadora -> qué hizo para ser AC, qué tienen los ficheros. Organization name, common name y todo eso. Site de configuración de ssl. El AC inserta el certificado en el navegador y hace conexión ssl contra el compañero.

Una Autoridad Certificadora señala una entidad de confianza, responsable de emitir y revocar los certificados, utilizando en ellos la firma electronica, para lo cual se emplea la criptografia de clave publica. Se utilizan para garantizar la seguridad de las comunicaciones digitales via el protocolo TLS utilizados en las comunicaciones web (HTTPS) o email (SMTP, POP3, IMAP).

Mi compañero genera un certificado firmado mediante los ficheros de clave publica y privada y me lo pasa. Esto lo hace cogiendo la clave publica, hasheandola y aplicandole la privada, y posteriormente añade el resultado de esto, que se considera la firma, y lo pega eal fichero de la clave publica.Yo monto mi servidor apache en https y lo configuro utilizando estos ficheros. Luego, un alumno se conecta a mi pagina y mi servidor le devuelve el certificado con la clave publica. Cifra la clave de sesion con la publica que le acaba de llegar y ya estaria cifrada la conexion.

Nano /etc/apache2/sites-available/000-default.conf
Nano /etc/apache2/sites-available/default-ssl.conf
cd /var/www/
openssl x509 -noout -text -in /etc/ssl/certs/Mi_Certificado_Digital.crt
openssl rsa -noout -text -in /etc/ssl/private/Mi_Clave_Privada.key

5) OpenVPN -> ifconfig -a . Hacer ping.

Las redes VPN, cuyas siglas significan Virtual Private Network o Red Privada Virtual, en español, son un tipo de red en el que se crea una extensión de una red privada para su acceso desde Internet, es como la red local que tienes en casa o en la oficina pero sobre Internet.
Gracias a una conexión VPN, podemos establecer contacto con máquinas que estén alojadas en nuestra red local -u otras redes locales- de forma totalmente segura, ya que la

conexión que se establece entre ambas máquinas viaja totalmente cifrada, es como si desde nuestro equipo conectado a Internet estableciésemos un túnel privado y seguro hasta nuestro hogar o hasta nuestra oficina, con el que podremos comunicarnos sin temer que nuestros datos sean vulnerables.

Nano /etc/openvpn/tunnel.conf
openvpn --verb 5 --config /etc/openvpn/tunnel.conf
ping 172.160.0.1
ping 172.160.0.2
ifconfig -a

6) Ntp -> ntp q -pn. Enseñar fichero de configuración para ver que hay keys.

En el servidor creamos un fichero con claves (borramos las de SHA1, nos quedamos con MD5) ---> hashes

A cada cliente se le da una clave de este fichero.

Cuando el cliente se vaya a conectar al servidor para sincronizarse tiene un paquete de red y la clave 1. Coge el paquete y le pega la clave 1 y le aplica un hash al conjunto, dando una huella digital del paquete+clave. Después coge el paquete y le quita la clave y le pega la huella, enviándolo al servidor así.

El servidor recibe el paquete con la huella, la quita, coge el paquete, le une la clave del cliente, le aplica el hash y mira si las huellas coinciden. Si coincide funciona, si no no.

Los dos hacen lo mismo (el cliente también mira cuando recibe paquetes si coinciden)

Nano /etc/ntp.conf
ntpq -pn
watch ntpq -c as

7) OpenVAS -> Analizar una máquina y enseñarle informe. También se le puede dar usuario y que haga pruebas desde un usuario raso (no lo pide).

64-bit block cipher 3des vulnerable to sweet32 attack
Protocolos criptográficos como TLS, SSH, Ipsec y OpenVPN usan algoritmos de bloques cifrados (como AES, Triple-DES y Blowfish) para encriptar los datos entre servidor y cliente. Para usar estos algoritmos, los datos se rompen en pedazos de tamaño fijo llamados bloques, y cada bloque es encriptado de forma separada. Un tamaño pequeño de bloque es vulnerable a lo que se conoce como un birthday attack. Este ataque se basa en la paradoja del cumpleaños.

---

https://localhost:4000
User: admin
Pass: admin

8) Abrir firewall (enseñarle el script) -> pregunta qué cosas hace y tal y si tienes bien las lineas de cliente servidor de ntp. Repositorios o no se qué. Ejecutar script.

Nano firewall.sh
sh firewall.sh start

sh firewall.sh stop