



Práctica 3 LSI Nino - Explicada

Lexislación e Seguridade Informática (Universidade da Coruña)

HOW_TO_P3 - DEFENSA

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

¡IMPORTANTE! EN ESTE APARTADO VAMOS A DIFERENCIAR CUANDO EJECUTAMOS COMO
USUARIO LSI (\$)

Y CUANDO EJECUTAMOS COMO ROOT (#)

a. Abra un shell remoto sobre SSH y analice el proceso que se realiza. Configure su
archivo `ssh_known_hosts` para dar soporte a la clave pública del servidor.

¿Como analizar el proceso que se realiza al abrir un shell remoto SSH?

Le daremos verbosidad al proceso de conexión con `-v` (cuántas más `v` más
detalle nos devolverá).

`$ssh -v 10.10.102.Y`

o

```
#ssh -v 10.10.102.Y (si queremos que nos pida contraseña)
```

```
OpenSSH_7.4p1 Debian-10+deb9u7, OpenSSL 1.0.2t 10 Sep 2020
```

```
debug1: Reading configuration data /etc/ssh/ssh_config
```

```
debug1: /etc/ssh/ssh_config line 19: Applying options for *
```

```
debug1: Connecting to 10.10.102.54 [10.10.102.54] port 22.
```

```
debug1: Connection established.
```

//SELECCIONAR VERSIÓN A UTILIZAR Y AUTENTICACION. Ambos extremos conversan para

saber que algoritmo de cifrado conocen ambos y se ponen de acuerdo para utilizar uno,

comprobando el cliente si tiene o no una serie de ficheros. También se selecciona versión de

ssh a utilizar(v1 o v2), en caso de v1 mejor no conectarse.

```
debug1: permanently_set_uid: 0/0
```

```
debug1: key_load_public: No such file or directory
```

```
debug1: identity file /root/.ssh/id_rsa type -1
```

```
debug1: key_load_public: No such file or directory
```

```
debug1: identity file /root/.ssh/id_rsa-cert type -1
```

```
debug1: key_load_public: No such file or directory
```

```
debug1: identity file /root/.ssh/id_dsa type -1
```

```
debug1: key_load_public: No such file or directory
```

```
debug1: identity file /root/.ssh/id_dsa-cert type -1
```

```
debug1: key_load_public: No such file or directory
```

```
debug1: identity file /root/.ssh/id_ecdsa type -1
```

```
debug1: key_load_public: No such file or directory
```

```
debug1: identity file /root/.ssh/id_ecdsa-cert type -1
```

```
debug1: key_load_public: No such file or directory
```

```
debug1: identity file /root/.ssh/id_ed25519 type -1
debug1: key_load_public: No such file or directory
debug1: identity file /root/.ssh/id_ed25519-cert type -1
debug1: Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_7.4p1 Debian-10+deb9u7
debug1: Remote protocol version 2.0, remote software version OpenSSH_7.4p1
Debian-
10+deb9u7
debug1: match: OpenSSH_7.4p1 Debian-10+deb9u7 pat OpenSSH* compat
0x04000000
debug1: Authenticating to 10.10.102.54:22 as 'lsi'
```

//NEGOCIACION DE PARAMETROS DE LA CONEXION(Ofrecidos por el cliente, ya que

normalmente este es el que propone). El cliente recibe la c.pública del servidor, comprueba

con esta la autenticidad de la máquina(en los ssh_known_hosts tanto de etc/ssh como del

\$HOME/.ssh/known_hosts) y cifra tambien con ella la clave de sesión. Esta clave de sesión

cifrada será enviada al servidor más tarde(paso 5) ya que es el único que puede descifrarla al

tener la clave privada. Se negocian los algoritmos que vamos a utilizar en la conexión, se sacan

de los ficheros de configuracion, cliente y servidor miran sus lineas y se ponen de acuerdo.

(algoritmo de autentificacion, algoritmo de intercambio de clave, algoritmo de cifrado ,

argoritmo de MAC para integridad y algoritmo de compresion).

```
debug1: SSH2_MSG_KEXINIT sent
```

```
debug1: SSH2_MSG_KEXINIT received
```

```
debug1: kex: algorithm: curve25519-sha256
```

```
debug1: kex: host key algorithm: ecdsa-sha2-nistp256
```

<implicit> debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC:

compression: none

<implicit> debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC:

compression: none

//AUTENTICACION DEL SERVIDOR. Se inicia el intercambio de la clave de sesión, preguntando

si te fías de la clave en caso de ser la primera vez (utilizando ecdsa algoritmo de curva elíptica)

Para ver el proceso de autenticación por contraseña podemos usar el usuario root, ya que en apartados

posteriores no se configurara autenticación por clave pública para este.

debug1: expecting SSH2_MSG_KEX_ECDH_REPLY

debug1: Server host key: ecdsa-sha2-nistp256

SHA256:9Vdaetzi2SjGJJppDJ+VQh6/exj55jTf138HQIXJBx4

debug1: Host '10.10.102.54' is known and matches the ECDSA host key.

debug1: Found key in /root/.ssh/known_hosts:1

debug1: rekey after 134217728 blocks

Se genera una //ESTABLECIMIENTO DE LA CLAVE DE SESIÓN(utilizando la pública del servidor).

clave de sesión y se cifra con la pública del servidor y se envía. Dicha clave se utiliza en toda la

sesión para cifrar las conexiones y se autentica el servidor, para tener la certeza de que no le

están suplantando)

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: Trying private key: /root/.ssh/id_rsa
debug1: Trying private key: /root/.ssh/id_dsa
debug1: Trying private key: /root/.ssh/id_ecdsa
debug1: Trying private key: /root/.ssh/id_ed25519
debug1: Next authentication method: password
ls@10.10.102.54's password:
debug1: Authentication succeeded (password).
Authenticated to 10.10.102.54 ([10.10.102.54]: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

```

//AUTENTICACIÓN DE USUARIO. Primero este lo intenta mediante la clave pública, es decir, el servidor mira en el fichero `$/home/ssh/authorized_keys(2)` si está registrada la pública del cliente(puede probar con varias: dsa,rsa,ecdsa,ed25519...). En caso de no estar la clave registrada solicita contraseña(esto se puede cambiar en `sshd_config`, haciendo que solo se pueda entrar por clave pública para evitar ataques password guessing).

¿Como configurar el fichero ssh_known_hosts?

Creamos el fichero ssh_known_hosts:

```
#nano /etc/ssh/ssh_known_hosts
```

Metemos la clave pública de nuestro compañero en el fichero:

```
$ssh-keyscan 10.10.102.Y >> /etc/ssh/ssh_known_hosts
```

¿Cómo comprobar que funciona?

Borrar el contenido de .ssh/known_hosts y conectarnos a nuestro compañero

```
$cat /dev/null > .ssh/known_hosts
```

```
$ssh lsi@10.10.102.217
```

```
$cat /etc/ssh/ssh_known_hosts
```

Si no nos sale el mensajito de fingerprinting es que funciona, porque está cogiendo la clave pública del servidor

(nuestro compañero) del /etc/ssh/ssh_known_hosts.

```
*****  
*****
```

b. Haga una copia remota de un fichero utilizando un algoritmo de cifrado determinado. Analice el proceso que se realiza.

TOMÉ

LOCAL -> SERVIDOR (copiar desde tu máquina a otra)

```
$scp -c aes128-ctr tome.txt lsi@10.10.102.217:/home/lsi
```

SERVIDOR -> LOCAL (copiar desde otra máquina a la tuya)

```
$scp -c aes128-ctr lsi@10.10.102.217:/home/lsi/reiner.txt /home/lsi
```

REINER

LOCAL -> SERVIDOR (copiar desde tu máquina a otra)

```
$scp -c aes128-ctr reiner.txt lsi@10.10.102.228:/home/lsi
```

SERVIDOR -> LOCAL (copiar desde otra máquina a la tuya)

```
$scp -c aes128-ctr lsi@10.10.102.228:/home/lsi/tome.txt /home/lsi
```

```
*****  
*****
```

c. Configure su cliente y servidor para permitir conexiones basadas en un esquema de autenticación de usuario de clave pública.

Generamos nuestra clave pública:

```
$ssh-keygen -t rsa
```

Copiamos nuestra clave pública en el fichero `.ssh/authorized_keys` de nuestro compañero:


```
$ssh-copy-id -i $HOME/.ssh/id_rsa lsi@10.10.102.Y
```

o (si dan problemas los permisos)

```
$cat ~/.ssh/id_rsa.pub | ssh lsi@10.10.102.Y "mkdir -p ~/.ssh && chmod 700  
~/.ssh && cat >> ~/.ssh/authorized_keys && chmod 600 ~/.ssh/authorized_keys"
```

TEORÍA INTERESANTE

KEX ---> Key Exchange

SSH_MSG_KEXINIT ---> Algoritmo para crear una clave de sesión (RSA, DIFFIE
HELLMAN, CE Curvas Elípticas)

SSH_KEX_ALG ---> Algoritmo para identificarse y comprobar que el servidor es
quién dice ser

¿Como funciona esto?

El servidor cifra con su privada un token (en el que va su identidad, la del
servidor), el cliente lo descifra con

su pública, si en el token viene la identidad correcta, el único que tiene esa
privada para haberlo cifrado es el

servidor, por tanto comprobamos que es él de verdad.

```
*****  
*****
```

d. Mediante túneles SSH securice algún servicio no seguro.

Supongamos que queremos securizar el puerto 80. Usaremos:

```
$ssh -P -L 10080:10.10.102.217:80 lsi@10.10.102.217
```

Otra forma (sin abrir un shell remoto):

```
$ssh -f -N -L 10080:10.10.102.217:80 lsi@10.10.102.217
```

Mientras la conexión ssh funcione, seremos capaces de hacer peticiones http a nosotros mismos y redireccionar

todo ese tráfico del puerto 80 por nuestra conexión ssh.

¿Cómo comprobar?

Desde otro terminal...

```
lynx http://localhost:10080
```

```
*****  
*****
```

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

```
#apt-get install sshfs
```

El cliente monta el directorio prueba_server de su compañero en su directorio prueba_cliente:

```
Tomé: $sshfs lsi@10.10.102.217:/home/lsi/prueba_server  
/home/lsi/prueba_cliente
```

```
Reiner: $sshfs lsi@10.10.102.228:/home/lsi/prueba_server  
/home/lsi/prueba_cliente
```

Nuestro compañero mete un nuevo archivo en su directorio prueba_server:

Pasar a root para no tener problemas con los permisos...

```
$su
```

Meter el archivo...

```
#nano /prueba_server/prueba2.txt (escribir algo dentro y guardar)
```

El cliente observa si le aparece el archivo:

```
$ls prueba_cliente
```

Para desmontarlo, si queremos borrarlo, el cliente usará:

```
$fusermount -u prueba_cliente
```

f. PARA PLANTEAR DE FORMA TEÓRICA.: Securice su servidor considerando que únicamente dará servicio ssh para sesiones de usuario desde determinadas IPs.

Para permitir solo conexiones desde determinadas IPs editaremos la opción AllowUsers de la configuración del servidor (/etc/ssh/sshd_config), metiendo el usuario e IP de nuestro compañero, de eduroam, de la VPN de la UDC...

```
*****  
*****
```

```
*****  
*****
```

2. Tomando como base de trabajo el servidor Apache2

```
*****  
*****
```


TEORÍA INTERESANTE

Un certificado suele contener la clave pública, privada y algunos datos sobre la entidad certificada.

Una Autoridad Certificadora simplemente se encarga de emitir certificados para distintas entidades.

Con esa clave pública, privada y los datos correspondientes solamente tenemos un certificado SIN FIRMAR.

¿Como se firma este certificado?

La autoridad certificadora también tiene una clave pública y privada.

La autoridad certificadora coge todo el contenido de este certificado y le aplica una función hash, el resultado

de aplicar esa función hash es una huella digital, ahora coge esa huella digital y la cifra con SU CLAVE PRIVADA

(la de la autoridad), obteniendo un certificado cifrado (firmado).

La autoridad certificadora envía a la entidad el certificado firmado.

Ahora la entidad certificada, que tiene su clave pública, privada y datos (su certificado sin firmar) les aplica

una función hash y obtiene una huella digital.

Después busca la clave pública de la autoridad certificadora, normalmente, las autoridades certificadoras ofrecen

su clave pública en distintos sitios de acceso público (navegador, sistema operativo...).

Con esta clave descifra el certificado cifrado que recibió previamente, obteniendo otra huella digital.

Si al comparar estas dos huellas digitales, coinciden, sabemos que esa huella fue certificada por ESA autoridad
certificadora de verdad.

Además al comparar dos hashes, sabemos que la información es íntegra, si cambiase un solo bit de esa información,
el hash ya no sería el mismo.

a. Configure una Autoridad Certificadora en su equipo.

Primero, debemos convertirnos en una autoridad certificadora, ¿Cómo?

Simplemente generaremos una clave pública y una privada para poder firmar nuestros certificados.

```
cd /usr/lib/ssl/misc
```

```
./CA.pl -newca
```

DATOS INTRODUCIDOS:

Frase de paso = mi_web

Country Name (2 letter code) [AU]:ES

State or Province Name (full name) [Some-State]:A Coruña

Locality Name (eg, city) []:Coruña

Organization Name (eg, company) [Internet Widgits Pty Ltd]:UDC

Organizational Unit Name (eg, section) []:FIC

Common Name (e.g. server FQDN or YOUR name) []:web

Email Address []:web@lsi.com

Please enter the following 'extra' attributes
to be sent with your certificate request

A challenge password []:mi_web

An optional company name []:UDC_GAL

Ya somos una autoridad certificadora.

Se nos han generado dos ficheros...

/usr/lib/ssl/misc/demoCA/private/cakey.pem

Este fichero contiene la clave privada de la autoridad certificadora (nosotros), cifrada con la clave de paso introducida (mi_web).

/usr/lib/ssl/misc/demoCA/cacert.pem

Este fichero contiene un certificado con la clave pública de la autoridad certificadora (nosotros) autofirmado.

Detalles del certificado:

Serial Number:

54:aa:e3:ab:d0:b9:61:a0:e9:46:cc:d0:6d:17:48:55:ad:fc:21:ae

Validity

Not Before: Sep 26 23:19:58 2021 GMT

Not After : Sep 25 23:19:58 2024 GMT

Subject:

countryName = ES

stateOrProvinceName = A Coru\C3\B1a

organizationName = UDC

organizationalUnitName = FIC

commonName = web

emailAddress = web@l\08\08\1B\C\1B[Csi.com

X509v3 extensions:

X509v3 Subject Key Identifier:

8F:E9:4C:A5:66:E4:E6:41:EB:34:72:7A:C3:0D:9A:25:DD:AD:6B:C3

X509v3 Authority Key Identifier:

keyid:8F:E9:4C:A5:66:E4:E6:41:EB:34:72:7A:C3:0D:9A:25:DD:AD:6B:C3

X509v3 Basic Constraints: critical

CA:TRUE

Certificate is to be certified until Sep 25 23:19:58 2024 GMT (1095 days)

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

Bueno, y fírmelo.

Ahora genero un certificado ¿Cómo?

Este certificado se lo envío al cliente (mi compañero).

Genero una clave pública y privada para mi cliente (mi compañero):

./CA.pl -newreq-nodes

DATOS INTRODUCIDOS:

Dejar vcía la frase de paso, probablemente ni nos la pida.

Country Name (2 letter code) [AU]:ES

State or Province Name (full name) [Some-State]:A Coruña

Locality Name (eg, city) []:Coruña

Organization Name (eg, company) [Internet Widgits Pty Ltd]:UDC

Organizational Unit Name (eg, section) []:FIC

Common Name (e.g. server FQDN or YOUR name) []:web

Email Address []:web@lsi.com

Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []:mi_web

An optional company name []:UDC_GAL

De nuevo, se nos generan dos ficheros:

/usr/lib/ssl/misc/newreq.pem

Este fichero contiene el certificado y la clave pública

/usr/lib/ssl/misc/newkey.pem

Este fichero contiene la clave privada

Ahora cifro el certificado con mi clave privada (lo estamos firmando):

```
./CA.pl -sign
```

Se nos genera un fichero `/usr/lib/ssl/misc/newcert.pem` que contiene el certificado firmado por la autoridad certificadora.

A nuestro compañero le pasaremos el certificado firmado (`newcert.pem`) y su clave privada (`newkey.pem`).

El compañero hace:

```
chmod 777 /usr/lib/ssl/misc
```

Nosotros hacemos:

```
scp -c aes128-ctr /usr/lib/ssl/misc/newcert.pem  
lsi@10.10.102.cliente:/usr/lib/ssl/misc/newcert_.pem
```

```
scp -c aes128-ctr /usr/lib/ssl/misc/newkey.pem  
lsi@10.10.102.cliente:/usr/lib/ssl/misc/newkey_.pem
```

(Lo guardamos con `_` para que al hacer este apartado en el otro sentido no reescribamos estos archivos al tener el mismo nombre)

El compañero hace:

```
chmod 755 /usr/lib/ssl/misc
```

```
*****  
*****
```

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.

Ahora el cliente, nuestro compañero, cogerá el certificado firmado (newcert.pem) y su clave privada (newkey.pem) que le enviamos y configurará SSL en su apache.

Primero debemos habilitar SSL para apache:

```
a2enmod ssl
a2ensite default-ssl
systemctl restart apache2.service
```

A continuación, configuramos SSL para nuestro apache:

```
nano /etc/apache2/sites-enabled/default-ssl.conf
```

Editamos las líneas...

```
SSLCertificateFile /usr/lib/ssl/misc/newcert_.pem
SSLCertificateKeyFile /usr/lib/ssl/misc/newkey_.pem
```

¿Cómo comprobar que funciona?

Desde una tercera máquina o desde la que está certificando, nos conectamos por https al cliente certificado

```
lynx https://web
```

```
nano /etc/apache2/sites-enabled/default-ssl.conf
```

```
ls /usr/local/share/ca-certificates/
```

Al conectarnos, el cliente nos va a enviar el certificado que le hicimos previamente, el navegador buscará en el repositorio web de autoridades certificadoras, buscará la autoridad correspondiente para obtener su pública y descifrar el certificado.

¿Problema?

Nosotros no vamos a aparecer en un repositorio web como autoridad certificadora.

Solución:

Cogeremos la clave pública, es decir, el certificado de la autoridad y la insertaremos en el repositorio web o el repositorio del sistema:

```
cp /usr/lib/ssl/misc/demoCA/cacert.pem /usr/local/share/ca-certificates/cacert.crt  
update-ca-certificates
```

Hemos pasado el certificado a formato .crt, y lo añadido al conjunto de certificados de autoridades certificadoras del sistema.

Ahora ya podemos realizar `lynx https://10.10.102.cliente` sin problema.

Otra forma de comprobar que el certificado se configuró correctamente es con `openssl`:

```
openssl s_client -connect web:443 -CApath /etc/ssl/certs
```

Si en "Certificate chain" nos aparece nuestro certificado y en el "verify return code" aparece un "0 (ok)" nuestro certificado está configurado correctamente en el servidor apache.

```
*****
*****
```

```
*****
*****
```

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.

```
*****
*****
```

```
*****
*****
```

MÁQUINA X ---> Tomé

MÁQUINA Y ---> Reiner

MÁQUINA X

Instalamos openvpn:

```
apt install openvpn
apt install openssl (ya está instalado)
lsmod | grep tun
```

Si está cargado el módulo tun perfecto (si nos aparece el módulo tun con el comando anterior), si no está cargado introducir:

```
modprobe tun
echo tun >> /etc/modules
```

Generamos una clave en el fichero clave.key:

```
cd /etc/openvpn  
openvpn --genkey --secret clave.key
```

Copiamos la clave en la máquina Y:

```
MÁQUINA Y:chmod 777 /etc/openvpn
```

```
MÁQUINA Y: scp -c aes128-ctr /etc/openvpn/clave.key  
lsi@10.10.102.Y:/etc/openvpn/clave.key
```

```
MÁQUINA Y: chmod 755 /etc/openvpn
```

Ahora creamos el archivo tunel.conf con la siguiente información:

```
local 10.10.102.X  
remote 10.10.102.Y  
dev tun1  
port 5555  
comp-lzo  
user nobody  
ping 15  
ifconfig 172.160.0.1 172.160.0.2  
secret /etc/openvpn/clave.key
```

```
*****  
*****
```

MÁQUINA Y

Instalamos openvpn:

```
apt install openvpn  
apt install openssl (ya está instalado)  
lsmod | grep tun
```

Si está cargado el módulo tun perfecto (si nos aparece el módulo tun con el comando anterior), si no está cargado introducir:

```
modprobe tun  
echo tun >> /etc/modules
```

Ahora creamos el archivo tunnel.conf con la siguiente información:

```
local 10.10.102.Y  
remote 10.10.102.X  
dev tun1  
port 5555  
comp-lzo  
user nobody  
ping 15  
ifconfig 172.160.0.2 172.160.0.1  
secret /etc/openvpn/clave.key
```

Creamos la conexión introduciendo:

```
openvpn --verb 5 --config /etc/openvpn/tunnel.conf
```

```
*****  
*****
```

¿Cómo comprobar que funciona?

MÁQUINA X

Creamos la conexión:

```
openvpn --verb 5 --config /etc/openvpn/tunnel.conf
```

Comprobamos que el tunnel está en nuestra configuración de red:

```
ifconfig -a
```

Probamos si funcionan los ping a nuestra dirección de red configurada:

```
ping 172.160.0.1
```

Al hacer ping a la 172.160.0.1, los paquetes serán redireccionados a través del túnel a 172.160.0.2,

que realmente es la máquina de nuestro compañero (10.10.102.Y), por eso debería funcionar, porque el

compañero nos responde a los ping.

MÁQUINA Y

Creamos la conexión:

```
openvpn --verb 5 --config /etc/openvpn/tunnel.conf
```

Comprobamos que el tunnel está en nuestra configuración de red:

```
ifconfig -a
```

Probamos si funcionan los ping a nuestra dirección de red configurada:

ping 172.160.0.2

```
*****
*****

*****
*****
```

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.

Crearemos un script firewall_script.sh:

```
nano script_firewall.sh
```

(El contenido del script lo copiaremos del fichero de texto donde tenermos el script, quitándole los comentarios)

A continuación damos permisos de ejecución al script y ejecutamos:

```
chmod +x script_firewall.sh
```

```
./script_firewall.sh
```

Desde otro terminal observaremos que las reglas se ha introducido y que todo funciona correctamente.

Para mostrar las reglas usaremos:

```
iptables -L
```


Si queremos ver en que línea está cada regla, para así poder borrar por línea:

```
iptables -L --line-number
```

Probar a hacer ping a nuestro compañero, hacer apt update, buscar en google, ping en IPv6...

```
*****
*****

*****
*****
```