Sockets con Python

Terminal remota UNIX

Autor: González, Agustín

Descripción: Implementación que permite la ejecución de **comandos Unix** (por ejemplo pwd, ls, mv), de forma remota, haciendo uso de encriptación en la comunicación.

Ejecución: Es posible llevar a cabo la ejecución de dos formas: En el primer caso, mediante el archivo *"remote_shell_run.conf"*, que va de la mano con el archivo de configuración *"include/config/remote_shell_run.conf"*, del cual es posible modificar los siguientes parámetros:

Sección **[SERVER]**, configuración de servidor**:**

- **Host** y **Port**: Dirección IP y puerto de servidor. Por omisión: *127.0.0.1* y *8080*, respectivamente.
- **UsersFile**: Archivo de usuarios para autenticación utilizado por el servidor. Por defecto *"include/config/shell_users.txt"*.
- **LogFile**: Archivo log. Por omisión *"output/remote_shell_run/server_log.txt"*.

Sección **[CLIENT]**, opciones de cliente:

- **SvrHost** y **SvrPort**: Dirección IP y puerto del servidor contra el que se realizará la conexión. Por defecto: *127.0.0.1* y *8080*, respectivamente.

La ejecución de este script es muy útil para realizar **pruebas locales**.

Ejemplo:

```
> python remote_shell_run.py include/config/ remote_shell_run.conf
```

Nota: Para la ejecución en esta modalidad, es necesario tener instalado **xterm** (`sudo apt-get install xterm`)

Por otra parte, para la ejecución en **distintos hosts**, es posible utilizar el archivo *"include/remote_shell_initializer.py"*, con los siguientes parámetros:

- **mode**: Parámetro **posicional**. Permite especificar el modo de ejecución del protocolo (server o client).
- **address**: **Posicional**. Host y puerto de servidor de la forma host:puerto.
- **--usersfile**: **Obligatorio** (en caso de mode server). Archivo que contiene usuarios válidos para la autenticación del cliente.
- **--logfile**: **Opcional** (en caso de mode server). Path de log de servidor.

- **--createuser**: **Opcional** (en caso de mode server). Inicia el servidor en modo de creación de usuario.

Ejemplos:

```
Servidor
> python remote_shell_initializer.py server 127.0.0.1:8080 --
usersfile shell_users.txt

Servidor (Modo de creación de usuario)
> python remote_shell_initializer.py server 127.0.0.1:8080 --
usersfile shell_users.txt --createuser

Cliente
> python remote_shell_initializer.py client 127.0.0.1:8080
```

Nota: En ambos casos de ejecución, si el archivo "usersfile", no existe, este se creará y se solicitará un primer usuario.

Notas

1. Para el desarrollo de las funcionalidades de encriptación, se han utilizado metodologías similares a **SSH**, mediante **RSA**, **AES** y **MD5**. Como no se realiza **encriptación** únicamente en la autenticación, sino que a lo largo de **toda la comunicación**, una motivación para el uso de AES, es que no conlleva los largos tiempos de espera de encriptación y desencriptación de RSA.
2. Al iniciarse el programa **servidor**, este verifica la existencia de **usuarios** en el archivo indicado en la terminal o en su defecto en el archivo de configuración. De no encontrarse, el servidor ejecuta el modo de **creación de usuario**, el cual solicita un nuevo nombre y contraseña para que, posteriormente, el cliente pueda realizar con éxito la autenticación. Por otra parte, el uso de **MD5** en el almacenamiento de estos últimos, impide su visualización no deseada por parte de terceros.
3. Respecto a la ejecución de comandos, no es posible utilizar el modo **super-usuario** (**sudo**) desde la terminal cliente.
4. El archivo *"include/config/shell_users.txt"* contiene por defecto el usuario "**admin**" con contraseña "**admin**".
5. Son necesarias las librerías **pycrypto** y **rsa**, las cuales pueden obtenerse mediante los siguientes comandos:

```
> pip install pycrypto
> pip install rsa
```

Funcionamiento de autenticación de usuario

1. Al recibir una **conexión** por parte de un **cliente**, el servidor comienza el **intercambio de claves**, mediante **encriptación asimétrica**. Para esto genera una clave pública y otra privada mediante la utilización de **RSA** de 512 bits. En la única instancia carente de cifrado, el servidor envía su **clave pública** al **cliente**.
2. El **cliente** recibe la **clave pública** del servidor y, a continuación, genera una **clave AES**, para **encriptación simétrica**, la cual es cifrada con la clave pública del servidor y enviada a este.
3. El **servidor** recibe la clave **AES encriptada**, y con su clave privada procede a la **desencriptación** de la misma. Ahora que el servidor y el cliente ya disponen de la clave simétrica, la comunicación se comienza a regir bajo esta última y el servidor queda a la espera de la **autenticación del usuario**.
4. El cliente procede a la autenticación, teniendo **tres instancias** en caso de error o equivocación. En caso de superar la cantidad máxima de intentos fallidos, el servidor procede al cierre de la conexión.
5. Tanto el usuario como la contraseña, son enviados a través de la red, codificados mediante **MD5**, encodeados en **base 64** y encriptados con **AES** (estas dos últimas características se dan a lo largo de toda la comunicación).