PRACTICA CALIFICADA Nº 1

NOTA: Para cada item de los contenidos, explique con detalle cómo trabajan.

Usando Paramiko para conexión a dispositivos de red a través de SSH

Contenido

- Iniciar una sesión SSH con Paramiko
- Ejecución de un comando a través de SSH
- Leer la salida de un comando ejecutado
- Ejecución de comandos en múltiples dispositivos
- Ejecución de una secuencia de comandos
- Usar claves públicas/privadas para la autenticación
- Cargando la configuración SSH local

Nota: Instalar python3 -m pip install paramiko o python3 -m pip install paramiko==2.7.1

Iniciar una sesión SSH con Paramiko

La base para conectarse a un dispositivo a través de SSH con Python y Paramiko es el objeto SSHClient de la biblioteca. Usaremos este objeto para crear una conexión inicial al servidor SSH y luego usaremos las funciones de este objeto para ejecutar comandos en el dispositivo. Aquí veremos cómo abrir mediante programación una conexión SSH.

Uso del archivo file1.py

Comencemos importando la biblioteca Paramiko y creando un objeto cliente. También especificaremos el host, el nombre de usuario y la contraseña en variables y luego iniciaremos una conexión al host especificado:

1. Importe la biblioteca de Paramiko:

   from paramiko.client import SSHClient

2. Especifique el host, el nombre de usuario y la contraseña. Puede nombrar estas variables como desees. En la comunidad de Python, se ha vuelto estándar poner en mayúsculas estas variables globales. Las tres variables SSH_USER, SSH_PASSWORD y SSH_HOST son variables de tipo cadena y por lo tanto usan comillas dobles para marcarlas. La variable SSH_PORT es un número entero y, por lo tanto, no utiliza comillas dobles:

```
SSH_USER = "developer"                                    # your ssh user
SSH_PASSWORD = "Cisco12345"                               # your ssh password
SSH_HOST = "sandbox-iosxe-recomm-1.cisco.com"   # IP/host of your device/server
SSH_PORT = 22                                  # Change this if your SSH port is different
```

3. Crea un objeto SSHClient, que acabamos de importar de Paramiko:

```
client = SSHClient ()
```

4. Si bien hemos creado el objeto cliente, aún no nos hemos conectado al dispositivo. Usaremos el método *connect* del objeto cliente para hacerlo. Antes de conectarnos realmente, tendremos que asegurarnos de que el cliente conozca las claves de host:

```
client.load_system_host_keys()
try:
    client.connect(SSH_HOST, port=SSH_PORT,
                                    username=SSH_USER,
                                   password=SSH_PASSWORD,
                                   look_for_keys=False)
        print("Connected successfully!")
except Exception:
        print("Failed to establish connection.")
```

5. Finalmente, hemos creado nuestra conexión. Es un buen hábito cerrar las conexiones una vez que hayamos terminado de usarlas. Para hacerlo, podemos usar la función *close ()* del cliente:

```
finally:
        client.close()
```

6. Para ejecutar este script, vaya al terminal y ejecútalo con lo siguiente:

```
python3 file1.py
```

En este ejemplo, confiamos en que el usuario ya había iniciado sesión en el dispositivo desde la línea de comando para que se conociera el host. Si usamos el código anterior para conectarnos a un dispositivo que no se conocía anteriormente, el código fallará con una excepción. La forma en que Paramiko maneja las claves de host desconocidas se puede especificar mediante una política. Una de estas políticas, *AutoAddPolicy*, nos permite simplemente agregar claves de host desconocidas al conjunto de scripts de claves de host:

```
from paramiko.client import SSHClient, AutoAddPolicy
SSH_USER = "<Insert your ssh user here>"
SSH_PASSWORD = "<Insert your ssh password here>"
SSH_HOST = "<Insert the IP/host of your device/server here>"
SSH_PORT = 22 # Change this if your SSH port is different

client = SSHClient()
client.set_missing_host_key_policy(AutoAddPolicy())
client.connect(SSH_HOST, port=SSH_PORT,
        username=SSH_USER,
        password=SSH_PASSWORD)
```

El código anterior agregará automáticamente estas claves de host. Ten en cuenta que esto podría ser un riesgo de seguridad potencial, ya que no se está verificando que el host al que se está conectando sea al que se conectó la última vez. En este ejemplo, pasamos detalles de conexión como nombre de usuario, nombre de host y contraseña directamente como una variable en el script. Si bien esto es excelente para las pruebas, es posible que desees que tu script te solicite una variable al ejecutarse.

Para las variables no secretas como el nombre de usuario, el host y el puerto, podemos usar la función incorporada *input ()*, pero para las contraseñas, es mejor usar una solicitud de contraseña dedicada que oculte lo que ha escrito para que alguien que revise el historial de tu consola no puede recuperar su contraseña. Para este propósito, Python tiene el módulo incorporado *getpass*.

Analiza los pasos para recuperar las variables de configuración necesarias, no como información estática en el script, sino de forma interactiva del usuario mediante una combinación de entrada y el módulo getpass:

```
import getpass
SSH_PASSWORD = getpass.getpass(prompt='Password: ',
stream=None)
SSH_USER = input("Username: ")
SSH_HOST = input("Host: ")
```

```
SSH_PORT = int(input("Port: "))
```

Ejecutando un comando a través de SSH

Ahora, podemos seguir adelante y ejecutar un comando en el dispositivo remoto. De manera similar a la forma en que manejamos la emisión de comandos en un dispositivo remoto a mano, tenemos tres flujos diferentes que regresan a nosotros: la salida estándar (o *stdout*), que es la salida normal, el error estándar (o *stderr*), que es el flujo predeterminado para que el sistema devuelva errores, y el estándar in (o *stdin*), que es el flujo utilizado para enviar texto de vuelta al comando ejecutado. Esto puede resultar útil si, en tu flujo de trabajo, normalmente interactúas con la línea de comandos.

Uso del archivo file2.py

1. Importa la biblioteca de Paramiko:

   ```
   from paramiko.client import SSHClient
   ```

2. Especifique el host, el nombre de usuario y la contraseña. Puede nombrar estas variables como desees. En la comunidad de Python, se ha convertido en un estándar poner en mayúsculas estas variables globales:

   ```
   SSH_USER = "<Insert your ssh user here>"
   SSH_PASSWORD = "<Insert your ssh password here>"
   SSH_HOST = "<Insert the IP/host of your device/server
   here>"
   SSH_PORT = 22 # Change this if your SSH port is different
   ```

3. Crea un objeto SSHClient, que acabamos de importar de Paramiko:

   ```
   client = SSHClient()
   ```

4. Si bien hemos creado el objeto cliente, aún no nos hemos conectado al dispositivo. Usaremos el método de connect del objeto cliente para hacerlo. Antes de conectarnos realmente, tendremos que asegurarnos de que el cliente conozca las claves de host:

   ```
   client.load_system_host_keys()
   client.connect(SSH_HOST, port=SSH_PORT,
           username=SSH_USER,
           password=SSH_PASSWORD)
   ```

5. Finalmente, podemos usar el cliente para ejecutar un comando. La ejecución de un comando nos devolverá tres objetos similares a archivos diferentes que representan stdin, stdout y stderr:

```
CMD = "show ip interface brief" # You can issue any
command you want
stdin, stdout, stderr = client.exec_command(CMD)
client.close()
```

6. Para ejecutar este script, vaya a su terminal y ejecútalo con esto:

```
python3 file2.py
```

Leer la salida de un comando ejecutado

Uso del archivo file3.py

En el código anterior, vimos cómo conectarse primero a un dispositivo y luego ejecutar el comando. Sin embargo, hasta ahora hemos ignorado la salida.

En esta parte, veremos cómo abrir mediante programación una conexión SSH, enviar un comando y luego escribir el resultado de ese comando en un archivo. Usaremos esto para hacer una copia de seguridad de la configuración en ejecución.

1. Importa la biblioteca de Paramiko:

```
from paramiko.client import SSHClient
```

2. Especifica el host, el nombre de usuario y la contraseña. Puede nombrar estas variables como desees. En la comunidad de Python, se ha convertido en un estándar poner en mayúsculas estas variables globales:

```
SSH_USER = "<Insert your ssh user here>"
SSH_PASSWORD = "<Insert your ssh password here>"
SSH_HOST = "<Insert the IP/host of your device/server here>"
SSH_PORT = 22 # Change this if your SSH port is different
```

3. Crea un objeto SSHClient, que acabamos de importar de Paramiko:

```
client = SSHClient()
```

4. Si bien hemos creado el objeto cliente, aún no nos hemos conectado al dispositivo. Usaremos el método de connect del objeto cliente para hacerlo. Antes de conectarnos realmente, tendremos que asegurarnos de que el cliente conozca las claves de host:

   ```
   client.load_system_host_keys()
   client.connect(SSH_HOST, port=SSH_PORT,
   username=SSH_USER,
   password=SSH_PASSWORD)
   ```

5. Finalmente, podemos usar el cliente para ejecutar un comando. La ejecución de un comando nos devolverá tres objetos similares a archivos diferentes que representan *stdin, stdout* y *stderr*:

   ```
   CMD = "show running-config"
   stdin, stdout, stderr = client.exec_command(CMD)
   ```

6. Usaremos el objeto stdout para recuperar lo que ha devuelto el comando:

   ```
   output = stdout.readlines()
   ```

7. A continuación, volvemos a escribir la salida en un archivo:

   ```
   with open("backup.txt", "w") as out_file
           for line in output:
              out_file.write(line)
   ```

8. Para ejecutar este script, vaya a su terminal y ejecútalo con esto:

   python3 file3.py

Ejecutando el mismo comando contra múltiples dispositivos

Uso del archivo file4.py, credencial.json

En esta parte, veremos cómo abrir mediante programación una conexión SSH a varios dispositivos, emitir el mismo comando para todos ellos y luego guardar la salida. Usaremos de nuevo este ejemplo para realizar una copia de seguridad de la configuración en ejecución de varios dispositivos.

Crearemos un archivo llamado ***credentials.json***. Usaremos este archivo para recuperar credenciales como el nombre de usuario y la contraseña de nuestros dispositivos.

Comenzamos creando el archivo de credenciales. Luego leeremos ese archivo de nuestro script de Python, crearemos clientes para cada uno de estos dispositivos y finalmente ejecutaremos el comando mientras también guardamos la salida en nuestro archivo:

1. Importa las bibliotecas necesarias, Paramiko y json

   ```
   import json
   from paramiko.client import SSHClient
   ```

2. Abre el archivo credentials.json y proporcione las credenciales a su (s) dispositivo (s) en el formato que se muestra en el siguiente código. Puede especificar tantos dispositivos como desees:

   ```
   [
   {
           "name": "<insert a unique name of your device>",
           "host": "<insert the host of your device>",
           "username": "<insert the username>",
           "password": "<insert the password",
           "port": 22
   },
   {
           "name": "<insert a unique name of your device>",
           "host": "<insert the host of your device>",
           "username": "<insert the username>",
           "password": "<insert the password",
           "port": 22
   }
   ]
   ```

   En nuestro caso:

   ```
   [
   {
           "name": "sandbox_one",
           "host": "sandbox-iosxe-recomm-1.cisco.com",
           "username": "developer",
   ```

```
        "password": "Cisco12345",
        "port": 22
}
]
```

3. Regresa al archivo file4.py. Ahora abriremos el archivo JSON en nuestro script de Python:

```
credentials = {}
with open("credentials.json") as fh:
    credentials = json.load(fh)
```

4. Crea una variable que contenga el comando que desea ejecutar. Luego, recorreremos todos los dispositivos especificados en el archivo credencial.json y crearemos un objeto de cliente SSH. Además, crearemos un archivo de salida individual para cada uno de nuestros dispositivos según el nombre que especificamos en el archivo JSON:

```
CMD = "show running-config"
for cred in credentials:
 out_file_name = str(cred['name']) + ".txt"
 client = SSHClient()
 client.load_system_host_keys()
 client.connect(cred['host'], port=cred['port'],
                                username=cred['username'],
                                 password=cred['password'])
  stdin, stdout, stderr = client.exec_command(CMD)

  out_file = open(out_file_name, "w")
  output = stdout.readlines()
  for line in output:
      out_file.write(line)
  out_file.close()
  client.close()
  print("Executed command on " + cred['name'])
```

5. Para ejecutar este script, vaya al terminal y ejecútalo con esto: python3 file4.py.

Ejecutando una secuencia de comandos

Uso del archivo file5.py

Aquí veremos cómo abrir mediante programación una conexión SSH a un dispositivo, abrir un shell y luego enviar una lista de comandos al dispositivo antes de cerrar la conexión.

Comenzamos creando el archivo de credenciales. Luego leeremos ese archivo de nuestro script de Python, crearemos clientes para cada uno de estos dispositivos y finalmente ejecutaremos el comando mientras también guardamos la salida en nuestro archivo:

1. Importamos la biblioteca de Paramiko. También necesitaremos la biblioteca de time incorporada:

   ```
   from paramiko.client import SSHClient
   import time
   ```

2. Especifica el host, el nombre de usuario y la contraseña. Puede nombrar estas variables como desee. En la comunidad de Python, se ha convertido en un estándar poner en mayúsculas estas variables globales:

   ```
   SSH_USER = "<Insert your ssh user here>"
   SSH_PASSWORD = "<Insert your ssh password here>"
   SSH_HOST = "<Insert the IP/host of your device/server here>"
   SSH_PORT = 22 # Change this if your SSH port is different
   ```

3. Crea un objeto SSHClient, que acabamos de importar de Paramiko:

   ```
   client = SSHClient()
   ```

4. Si bien hemos creado el objeto cliente, aún no nos hemos conectado al dispositivo. Usaremos el método *connect* del objeto cliente para hacerlo. Antes de conectarnos realmente, tendremos que asegurarnos de que el cliente conozca las claves de host:

   ```
   client.load_system_host_keys()
   client.connect(SSH_HOST, port=SSH_PORT,
                            username=SSH_USER,
                            password=SSH_PASSWORD)
   ```

5. Abre una sesión de shell interactiva y un canal que podamos usar para recuperar la salida:

   ```
   channel = client.get_transport().open_session()
   ```

```
shell = channel.invoke_shell()
```

6. A continuación, especifica la lista de comandos que queremos ejecutar en el dispositivo:

```
commands = [
    "configure terminal",
    "hostname test"
]
```

7. Repite cada uno de los comandos, ejecútelos y luego espera 2 segundos:

```
for cmd in commands:
    shell.send(cmd + "\n")
    out = shell.recv(1024)
    print(out)
    time.sleep(1)
```

8. Finalmente, necesitamos cerrar la conexión:

```
client.close()
```

9. Para ejecutar este script, vaya a su terminal y ejecútalo con esto:

```
python 3 file5.py
```

Usar claves públicas / privadas para la autenticación

Uso del archivo: file6.py

Aquí veremos cómo abrir mediante programación una conexión SSH utilizando una clave privada protegida por contraseña.

Comencemos importando las bibliotecas requeridas, definamos los nuevos detalles de conexión y finalmente abramos una conexión usando autenticación basada en claves:

1. Importa la biblioteca de Paramiko:

```
from paramiko.client import SSHClient
```

2. Especifica el host y el nombre de usuario. Puede nombrar estas variables como desee. En la comunidad de Python, se ha convertido en un estándar poner en mayúsculas estas variables globales. En lugar de la contraseña del dispositivo, ahora necesitaremos dos nuevas variables: la ruta al archivo de clave privada que queremos usar para autenticarnos y la contraseña para ese archivo de clave privada:

   ```
   SSH_USER = "<Insert your ssh user here>"
   SSH_HOST = "<Insert the IP/host of your device/server here>"
   SSH_PORT = 22 # Change this if your SSH port is different
   SSH_KEY = "<Insert the name of your private key here>"
   SSH_KEY_PASSWORD = "<Insert the password here>
   ```

3. Crea un objeto *SSHClient*, que acabamos de importar de Paramiko:

   ```
   client = SSHClient()
   ```

4. Si bien hemos creado nuestro objeto cliente, aún no nos hemos conectado al dispositivo. Usaremos el método *connect* del objeto cliente para hacerlo. Antes de conectarnos realmente, aún tendremos que asegurarnos de que nuestro cliente conozca las claves de host:

   ```
   client.load_system_host_keys()
   client.connect(SSH_HOST, port=SSH_PORT,
                            username=SSH_USER,
                            look_for_keys=True,
                            key_filename=SSH_KEY,
                            passphrase=SSH_KEY_PASSWORD)
   ```

5. Como vimos antes, ahora podemos ejecutar un comando una vez establecida la conexión:

   ```
   stdin, stdout, stderr = client.exec_command('<your command>')
   ```

6. Finalmente, necesitamos cerrar la conexión:
   client.close()

7. Para ejecutar este script, vaya al terminal y ejecútalo con: python 3 file6.py

Cargando la configuración SSH local

En el siguiente código veremos cómo analizar mediante programación el archivo S*SHConfig*, extraer la información relevante basada en un host y almacenarla en un diccionario.
También necesitarás un archivo de configuración SSH para el dispositivo al que está intentando conectarse. En este ejemplo, usaremos un archivo de configuración que tiene el siguiente contenido:

```
Host example
    Host <insert your host address here>
    User <insert your user here>
    Port <insert the port here>
    IdentityFile <insert the path to your private key here>
```

Uso del archivo: file7.py

Comencemos importando las bibliotecas requeridas y definiendo la ruta a nuestra configuración SSH:

1. Importe la biblioteca de Paramiko:

   ```
   from paramiko.client import SSHClient
   from paramiko import SSHConfig
   ```

2. Especifica la ruta a su archivo de configuración SSH y el nombre de su host tal como aparece en su configuración SSH (ejemplo en este fragmento).
   Completaremos todas las demás variables de la configuración que estamos leyendo:

   ```
   SSH_CONFIG = "/Users/yuri/.ssh/config"                    # path to ssh config
   SSH_HOST = "example"
   ```

3. Creamos un objeto SSHConfig, que acabamos de importar de Paramiko, y crea un objeto de archivo local con la ruta a nuestra configuración SSH:

   ```
   config = SSHConfig()
   config_file = open(SSH_CONFIG)
   ```

4. A continuación, necesitamos decirle al objeto SSHConfig que cargue y analice el archivo de configuración:

   config.parse(config_file)

5. Con la configuración analizada, ahora podemos hacer una búsqueda en este objeto de configuración para extraer toda la información almacenada en la propia configuración. La función de búsqueda devolverá un diccionario:

   dev_config = config.lookup(SSH_HOST)

6. Con la configuración de nuestro dispositivo extraída de la configuración SSH, podemos continuar y completar nuestros detalles de conexión con lo que hemos extraído del archivo de configuración SSH:

```
client = SSHClient()
client.load_system_host_keys()

HOST = dev_config['hostname'],
client.connect(HOST, port=int(dev_config['port']),
                        username=dev_config['user'],
                        password =dev_config['password'])
```

7. Con la conexión establecida, podemos hacer todas las cosas diferentes que descubrimos en código anteriores antes de finalmente cerrar la conexión:

   client.close()

8. Para ejecutar este script, vaya al terminal y ejecuta esto: python3 file7.py