Автоматизация работы сетевого администратора. Часть 3. Знакомство с Paramiko

Предыдущие статьи цикла:

Автоматизация работы сетевого администратора.
Автоматизация работы сетевого администратора, часть 2. Практические кейсы с Telnetlib.

Всех приветствую, продолжаем рассматривать полезные инструменты для работы сетевого инженера или администратора. Для тех, кто только начал изучение темы с текущей статьи, напомню что мы поговорили о концепции SDN и достаточно подробно рассмотрели инструмент TelnetLib с практическими примерами. Буквально после прочтения 2х статей и некоторого навыка в IT, вы уже можете автоматизировать рутинные процессы по настройке вашего сетевого оборудования.

В рамках взаимодействия с сетевыми устройствами, есть различные нюансы, которые плохо реализованы в предустановленных модулях языка Python 3, однако существуют библиотеки от сообщества и энтузиастов, которые подойдут под ваши задачи. Сегодня мы рассмотрим модуль Paramiko.

SSH и Paramiko

Paramiko - это реализация протокола SSHv2 на Python. Paramiko предоставляет функциональность клиента и сервера. В рамках статьи будет рассматриваться лишь функциональность клиента, серверную часть возможно изучить самостоятельно.

Библиотека Paramiko не входит в список стандартных модулей для Python 3, поэтому ее требуется установить:

pip install paramiko

В Python3 есть отличная возможность тестировать ваши скрипты, отправляя одиночные команды, для этого достаточно лишь вызвать сам интерпретатор:

python3

Логика работы Paramiko:

1. создание клиента подключения SSH;
2. настройка клиента подключения SSH;
3. выполнение подключения SSH;
4. переход в интерактивный режим отправки команд для подключения SSH.

Создание клиента SSH:

In [1]: import paramiko
In [2]: client = paramiko.SSHClient()
In [3]: client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
In [4]: client.connect(hostname="10.10.10.1", username="admin", password="admin",look_for_keys=False, allow_agent=False)
In [5]: ssh = client.invoke_shell()

Комментарии к каждой из строк:

In [1]: импорт библиотеки **Paramiko**

In [2]: присваивание переменной client класса **SSHClient** со всеми атрибутами.

In [3]: необязательная строка кода служит для указания какую политику использовать клиенту при подключение к неизвестному серверу в рамках SSH. В примере **paramiko.AutoAddPolicy()** автоматически добавляет новое имя хоста и ключ в локальный объект HostKeys.

In [4]: Метод connect выполняет подключение к SSH-серверу и аутентифицирует подключение. Его параметры: **look_for_keys** - режим аутентификации по ключам (True/False); **allow_agent -** режим подключения через локальный SSH-агент ОС (True/False).

Метод send:

Отправка указанной строки и возврат количества переданных байт, если значение "0" - сессия закрыта и не удалось отправить команду.

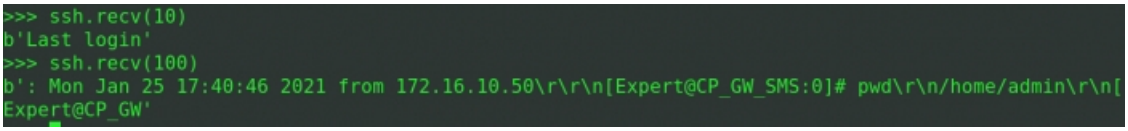
In [6]: ssh.send("pwd\n")
"> 4

В нашем примере мы отправили команду "pwd" с целью узнать текущую директорию и получили ответ в виде 4 переданных байт.

Метод recv:

Получение данных с терминала устройства по построенному каналу SSH. Данные возвращаются в виде строки, вы указывается количество запрашиваемых данных в виде числа байт. Если возвращается число 0, канал SSH будет автоматически закрыт.

In [6]: ssh.recv(100)



Метод close:

Закрытие SSH - сессии

In [7]: ssh.close()

Скрипт с выводом конфигурации на экран

Для закрепления рассмотрим скрипт по выводу "sh run" в Paramiko. После # или двойных кавычек будут следовать комментарии.

"Импорт различных библиотек для скрипта"

```
import paramiko
import time
import socket
from pprint import pprint
import re
```

"Функция по отправке команд "

```
def send_show_command(ip,
username,
password,
enable,
command,
max_bytes=60000,
short_pause=1,
long_pause=5,):
```

```
    cl = paramiko.SSHClient()# создание класса SSH-клиента
    cl.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

"Передача аргументов в метод connect"

```
    cl.connect(
        hostname=ip,
        username=username,
        password=password,
        look_for_keys=False,
        allow_agent=False,
    )
```

"Создание интерактивного подключения и вход в режим enable"

```
    with cl.invoke_shell() as ssh:
        ssh.send("enable\n")
        ssh.send(enable + "\n")
        time.sleep(short_pause)
        ssh.recv(max_bytes)

        result = {}
```

"Отправка команд в цикле"

```
    for command in commands:
      ssh.send(f"{command}\n")
      ssh.settimeout(5)

      output = ""
```

"Вывод за счет цикла всей информации из терминала"

```
       while True:
         try:
            page = ssh.recv(max_bytes).decode("utf-8")
            output += page
            time.sleep(0.5)
         except socket.timeout:
            break
```

"Если требуется перейти в режим More"

```
            if "More" in page:
             ssh.send(" ")
      output = re.sub(" +--More--| +\x08+ +\x08+", "\n", output)
      result[command] = output

   return result # возврат результата вывода
```

"Выполняется, если запускается непосредственно скрипт".

```
if __name__ == "__main__":
   commands = ["sh run"]
   result = send_show_command("192.168.100.1", "admin", "password", "cisco", commands)
   pprint(result, width=120)
```

Конечно, для новичков в Python будет достаточно сложно разобраться со всеми строками кода, здесь используются различные возможности ЯП. По факту вам достаточно лишь изменить конструкцию после **if __name__ == "__main__":**

Где вы будете вызывать фунцию **send_show_command("192.168.100.1", "admin", "password", commands)**

Произойдет соответственно передача аргументов: IP-адрес, логин, пароль, команды.

Вместо заключения:

Сегодня мы достаточно подробно поговорили о модуле для инициализации SSH-соединений Paramiko, рассмотрели практический пример с выводом всей конфигурации маршрутизатора/коммутатора на экран пользователя. Конечно он не является достаточно удобным для многоразового и повседневного использования, потому что требует ввода большого числа операций (создание SSH-агента, настройка будущего соединения и т.д). Сетевые инженеры и специалисты отрасли на основе Paramiko разработали отдельную библиотеку для работы с сетевым оборудованием, который "под капотом" осуществляет большое количество операций и умеет взаимодействовать с большим списков вендоров, но об этом уже в следующей статье, до скорого!