**Netmiko Lab 2: Looping through a list of Devices and Error Handling**

**Objectives - Part 1: For Loop through a list of devices**

* Create a list of IPv4 addresses
* Use a **for** loop to access each device in the list
* Use the send\_command() method to display information for each device

**Objectives - Part 2: Error Handling**

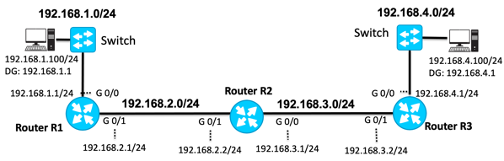
* Add **try** and **except** to capture timeout and authentication exceptions

**Requirements**

* Python: You should be familiar with the basics of Python.
* Using the topology below or your own topology, have access to one or more Cisco IOS devices using SSH.

**Topology**

This lab uses the following topology. You can use a different topology, just be sure to make any necessary modifications to interfaces and addresses. See Netmiko Lab 0: Using a Script for Initial Setup



**Part 1: Using a for loop to access IP addresses in a list**

**Objectives - Part 1**

* Create a list of IPv4 addresses
* Use a **for** loop to access each device in the list
* Use the send\_command() method to display information for each device

**Step 1: Open a new Python file**

Using your Python IDE or equivalent, create a new Python file.

Note: See Step 8 if you wish to see the completed program.

**Step 2: Import Netmiko**

Enter the **import netmiko** command. This command imports the Netmiko library.

**import netmiko**

**Step 3: Create a dictionary of IP addresses**

Assign devices to a dictionary of IP addresses used to SSH into each device. These are three IPv4 addresses, one for each router, that can be used to access the device.

**devices = ['192.168.1.1', '192.168.2.2', '192.168.3.2']**

**Step 4: Create a for loop to access each device in the list of devices**

Create a **for** loop to access each device in the list of devices. This will loop through the IPv4 addresses in the list of devices, each time assigning the IPv4 address to the variable **device**.

**for device in devices:**

**Step 5: Establish an SSH connection - Connect.Handler()**

Within the **for** loop, use the **Connect.Handler()**, to establish the SSH connections to each device. Notice we have all the arguments in the method. However, we have replaced the value of the specific IPv4 address with the value of the current **device** variable.

Be sure this is indented within our for loop.

**connection = netmiko.ConnectHandler(ip=device,**

**device\_type='cisco\_ios',**

**username='admin',**

**password='cisco')**

**Step 6: Send an IOS command - send\_command()**

Within the same for loop, will will do the following:

* Display the value of **device**, the current IPv4 address of the device we used for SSH
* Use the **send\_command** method to retrieve some output from this device. Again, the **print** option is helpful otherwise you get \n instead of line breaks..

**print(device)**

**print('-'\*11)**

**print(connection.send\_command('show ip interface brief'))**

**--------------------------------------------------------------------**

Note: Another option is instead of using the send\_command method directly encapsulated in the print function, is to send the output of the send\_command to a variable. Then print the variable.

**output = connection.send\_command('show running-config')**

**print(output)**

**--------------------------------------------------------------------**

**Step 7: Disconnect from Netmiko - disconnect()**

The disconnect method in Netmiko is used to close the SSH connection to a network device, ensuring proper cleanup and freeing up system resources.

**connection.disconnect()**

**Step 8: Display the program**

Here is the complete program including comments:

import netmiko

devices = ['192.168.1.1', '192.168.2.2', '192.168.3.2']

print('\n')

for device in devices:

# ip = current IP address in list of devices

connection = netmiko.ConnectHandler(ip=device,

device\_type='cisco\_ios',

username='admin',

password='cisco')

print(device)

print('-'\*11)

print(connection.send\_command('show ip interface brief'))

print('\n')

connection.disconnect()

*Rick's program: netmiko\_2a.py*

**Step 6: Run the program**

192.168.1.1

-----------

Interface IP-Address OK? Method Status Protocol

Embedded-Service-Engine0/0 unassigned YES unset administratively down down

GigabitEthernet0/0 192.168.1.1 YES manual up up

GigabitEthernet0/1 192.168.2.1 YES manual up up

Serial0/0/0 unassigned YES unset administratively down down

Serial0/0/1 unassigned YES unset administratively down down

192.168.2.2

-----------

Interface IP-Address OK? Method Status Protocol

Embedded-Service-Engine0/0 unassigned YES NVRAM administratively down down

GigabitEthernet0/0 192.168.3.1 YES NVRAM up up

GigabitEthernet0/1 192.168.2.2 YES NVRAM up up

Serial0/0/0 unassigned YES NVRAM administratively down down

Serial0/0/1 unassigned YES NVRAM administratively down down

192.168.3.2

-----------

Interface IP-Address OK? Method Status Protocol

Embedded-Service-Engine0/0 unassigned YES NVRAM administratively down down

GigabitEthernet0/0 192.168.4.1 YES NVRAM up up

GigabitEthernet0/1 192.168.3.2 YES NVRAM up up

Serial0/0/0 unassigned YES NVRAM administratively down down

Serial0/0/1 unassigned YES NVRAM administratively down down

**Part 2: Netmiko Error and Exception Handling using try and except**

**Objectives - Part 2**

* Add **try** and **except** to capture timeout and authentication exceptions

The **try** and **except** statements in Python are used for error handling and exception handling. They allow you to catch and handle exceptions that may occur during the execution of your code, preventing them from causing the program to terminate abruptly.

If this is new to you, here is how the **try** and **except** statements work:

1. The code that may raise an exception is placed within the **try** block.
2. If an exception occurs within the try block, the code execution immediately jumps to the **except** block.
3. The **except** block specifies the type of exception it can handle. If the raised exception matches the specified type, the code inside the **except** block is executed to handle the exception.
4. If an exception occurs within the **try** block but doesn't match any of the specified **except** blocks, the exception will propagate up the call stack, potentially causing the program to terminate unless it is caught by an outer exception handler.

**Step 1: Edit the ConnectHandler() to have incorrect password and run the program**

Modify the current program with the incorrect authentication **password**, such as **pizza**.

**connection = netmiko.ConnectHandler(ip=device,**

**device\_type='cisco\_ios',**

**username='admin',**

**password='pizza')**

Run the program and notice the program fails.

Traceback (most recent call last):

<Python error messages omitted>

netmiko.exceptions.**NetmikoAuthenticationException**: Authentication to device failed.

Common causes of this problem are:

1. Invalid username and password

2. Incorrect SSH-key file

3. Connecting to the wrong device

Device settings: cisco\_ios 192.168.1.1:22

Authentication failed.

Netmiko raises these exception messages and provides information about the possible causes.

**Step 2: Add try and except for timeout and authentication errors**

To catch and handle these types of issues perform the following tasks. The actual code follows the tasks.

Tasks:

* Add the **try** command immediately after the **for** statement.
* Indent all previous statements below the **try** command.
* Add the two exceptions.
* The first exception will occur if there is a timeout due to being unable to connect to the device.
  + **except netmiko.exceptions.NetmikoTimeoutException:**
* The second exception will occur if the user is unable to SSH to the device.
  + **except netmiko.exceptions.NetMikoAuthenticationException:**
* The second exception will occur if the IOS command times out, many times due to incorrect or missing privileged exec password.
  + **except netmiko.exceptions.NetMikoAuthenticationException:**For the first three exceptions, you will print the possible causes.
* The third exception is a catch-all exception handler that captures any exception that is not explicitly handled by the preceding specific exception handlers. It allows you to catch and handle any exception that occurs during the execution of the code block within the try block. **except Exception** as a catch-all handler can be helpful for gracefully handling unexpected exceptions and preventing the program from abruptly terminating.
  + **except Exception as e:**
* For the first two exceptions, your code will print the possible causes.

Notice that the name of the exception, the exception class, following the except command is the same name that we see in the error message above.

Here is the code with the additions in **bold**:

for device in devices:

**try:**

connection = netmiko.ConnectHandler(ip=device,

device\_type='cisco\_ios',

username='admin',

password='cisco')

print(device)

print('-'\*11)

print(connection.send\_command('show ip interface brief'))

print('\n')

connection.disconnect()

**except netmiko.exceptions.NetmikoTimeoutException:**

**print('Timeout occurred to', device)**

**print('''Common causes of this problem are:**

**1. Incorrect hostname or IP address.**

**2. Wrong TCP port.**

**3. Intermediate firewall blocking access.''')**

**print('\n')**

**except netmiko.exceptions.NetMikoAuthenticationException:**

**print('Authentication error', device)**

**print('''Common causes of this problem are:**

**1. Invalid username and password**

**2. Incorrect SSH-key file**

**3. Connecting to the wrong device''')**

**print('\n')**

**except netmiko.exceptions.ReadTimeout:**

**print('Read timeout. pattern not detected', device)**

**print('''Common causes of this problem are:**

**1. Missing or incorrect secret password in ConnectHandler()**

**2. Adjust the regex pattern to better identify the terminating**

**string. Note, in many situations the pattern is**

**automatically based on the network device's prompt.**

**3. Increase the read\_timeout to a larger value.''')**

**print('\n')**

**except Exception as e:**

**print("An error occurred:", str(e))**

*Rick's program: netmiko\_2b.py*

**Step 3: Test an exception**

Run the program again and notice that we receive the messages from the Authentication exception:

Authentication error 192.168.1.1

Common causes of this problem are:

1. Invalid username and password

2. Incorrect SSH-key file

3. Connecting to the wrong device

Timeout occurred to 192.168.2.2

Common causes of this problem are:

1. Incorrect hostname or IP address.

2. Wrong TCP port.

3. Intermediate firewall blocking access.

Timeout occurred to 192.168.3.2

Common causes of this problem are:

1. Incorrect hostname or IP address.

2. Wrong TCP port.

3. Intermediate firewall blocking access.

Change the **password** argument to include the correct password, **cisco**.

**connection = netmiko.ConnectHandler(ip=device,**

**device\_type='cisco\_ios',**

**username='admin',**

**password='cisco')**

Add a bogus (wrong) IP address to the list of devices (don't forget the quotes and the comma):

devices = ['192.168.1.1', '192.168.10.10', '192.168.2.2', '192.168.3.2']

**Step 4: Display the program**

import netmiko

devices = ['192.168.1.1', '192.168.10.10', '192.168.2.2', '192.168.3.2']

print('\n')

for device in devices:

try:

connection = netmiko.ConnectHandler(ip=device,

device\_type='cisco\_ios',

username='admin',

password='cisco')

print(device)

print('-'\*11)

print(connection.send\_command('show ip interface brief'))

print('\n')

connection.disconnect()

except netmiko.exceptions.NetmikoTimeoutException:

print('Timeout occurred to', device)

print('''Common causes of this problem are:

1. Incorrect hostname or IP address.

2. Wrong TCP port.

3. Intermediate firewall blocking access.''')

print('\n')

except netmiko.exceptions.NetMikoAuthenticationException:

print('Authentication error', device)

print('''Common causes of this problem are:

1. Invalid username and password

2. Incorrect SSH-key file

3. Connecting to the wrong device''')

print('\n')

except netmiko.exceptions.ReadTimeout:

print('Read timeout. pattern not detected', device)

print('''Common causes of this problem are:

1. Missing or incorrect secret password in ConnectHandler()

2. Adjust the regex pattern to better identify the terminating

string. Note, in many situations the pattern is

automatically based on the network device's prompt.

3. Increase the read\_timeout to a larger value.''')

print('\n')

except Exception as e:

print("An error occurred:", str(e))

Note: Don’t be intimidated by the amount of code to handle the exceptions. It is just a way to help us debug any problems we might have when running our code. We will be using (copying) this **for** loop with the **try** and **except** as a template in most of our programs.

**Step 5: Run the program**

Run the program:

192.168.1.1

-----------

Interface IP-Address OK? Method Status Protocol

Embedded-Service-Engine0/0 unassigned YES unset administratively down down

GigabitEthernet0/0 192.168.1.1 YES manual up up

GigabitEthernet0/1 192.168.2.1 YES manual up up

Serial0/0/0 unassigned YES unset administratively down down

Serial0/0/1 unassigned YES unset administratively down down

Timeout occurred to 192.168.10.10

Common causes of this problem are:

1. Incorrect hostname or IP address.

2. Wrong TCP port.

3. Intermediate firewall blocking access.

192.168.2.2

-----------

Interface IP-Address OK? Method Status Protocol

Embedded-Service-Engine0/0 unassigned YES NVRAM administratively down down

GigabitEthernet0/0 192.168.3.1 YES NVRAM up up

GigabitEthernet0/1 192.168.2.2 YES NVRAM up up

Serial0/0/0 unassigned YES NVRAM administratively down down

Serial0/0/1 unassigned YES NVRAM administratively down down

192.168.3.2

-----------

Interface IP-Address OK? Method Status Protocol

Embedded-Service-Engine0/0 unassigned YES NVRAM administratively down down

GigabitEthernet0/0 192.168.4.1 YES NVRAM up up

GigabitEthernet0/1 192.168.3.2 YES NVRAM up up

Serial0/0/0 unassigned YES NVRAM administratively down down

Serial0/0/1 unassigned YES NVRAM administratively down down

Don’t forget to remove the wrong IPv4 address!

devices = ['192.168.1.1', '192.168.10.10', '192.168.2.2', '192.168.3.2']

Congratulations! Now move on to the next lab…