Automating License and IDP Package Installation

Important The licenses for this step are not included within the lab due to legal reasons. You will need to place the license files "utm.txt" and "appsecure.txt" into the licenses directory within the repository.

An example license file will look something like this. It may also contain additional comments but the actual license component will look like this.

```
JUNOS604043 xxxqic a6ajc4 a2ob24 bhhqdj yhlqeq 6anha5
            xxx5yb u4dvyc dpagtq oxbmhd smjxgm yecmbq
            xxxtqo brg4ga 4tcbij lwk3tz ouqfu2 dbnztq
            xxxdtn 6mhr4d bldwa7 26bcdg inj3wb p6tjq4
            xxxq4x ms5lt4 5coxjv 52756a n3bark dw64
```

Important

When deploying IPS there are a few steps you must take on a device before it is ready to utilize an IPS policy.

First a license is required to installed on the vSRX. This gives the device the ability to run IPS, AppID, and UTM. Once installed these features are enabled and ready to be used. The IPS and UTM licenses are required to be installed separately.

Once installed you can then download and install the IPS signature packs. Doing this on a single device isn't the most challenging thing to do, but doing it for dozens or hundreds of devices quickly becomes a pain.

In this section we will look at two tools that can be used to simplify the process. Both tools are written in Python and demonstrate two different methods of running commands on the vSRX.

Tools

- tools/licensetool.py
- tools/idpsecpack.py
- Python interpreter

Automating License Installation

Installing a license on an SRX is a bit of a challenge. It requires you to enter a command on the CLI. Unfortunately adding a license is not an available RPC nor can use use the "" RPC to call it. Fortunately with automation we can work around this issue. The tool "licensetool.py" can be found in the "tools" directory in the base of the student repository.

The "licensetool.py" uses a Python library called Paramiko that allows you to automation SSH connections. In fact this tool is used in the Junos PyEZ libraries as well. This tool provides a few different command line flags.

```
usage: licensetool.py [-h] [--host HOST] [--username USERNAME]
                      [--password PASSWORD] [--url URL]

Process user input

optional arguments:
  -h, --help           show this help message and exit
  --host HOST          Specify host to connect to
  --username USERNAME  Specify the username
  --password PASSWORD  Specify the password
  --url URL            Specify the license URL
```

The code for the tool is fairly simple. You can open the file or look at the code below. To see an example of how it is written.

Steps in Script

- 1 Gathers command line options from user
- 2 Connects to the host using the specified username and password
- 3 Executes the command `/usr/sbin/cli -c request system license add URL`
- 4 Prints the output of the result of the command

```
#!/usr/bin/env python

import paramiko
import argparse
import logging

logger = paramiko.util.logging.getLogger()
logger.setLevel(logging.WARN)


parser = argparse.ArgumentParser(description='Process user input')
parser.add_argument("--host", dest="host", default="",metavar="HOST",help="
parser.add_argument("--username", dest="username", metavar="USERNAME",help=
parser.add_argument("--password", dest="password", metavar="PASSWORD",help=
parser.add_argument("--url", dest="url",metavar="URL",help="Specify the lic
args = parser.parse_args()

host = args.host
username = args.username
password = args.password
licenseURL = args.url

client = paramiko.SSHClient()
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
client.connect(host,username=username,password=password,look_for_keys=False
stdin, stdout, stderr = client.exec_command('/usr/sbin/cli -c "request syst

data = stdout.read()
print data
```

You can see that we first take a set of command line arguments using the argparse library. This library was added in Python 2.7 and it is extremely useful for dealing with command line arguments. Once the arguments are received then the script makes an SSH connection to the host on your behalf using the provided credentials to authenticate to the vSRX. Once connected we request SSH to execute a command on the remote device. The command we call is the cli binary. This is the same CLI that you use when you connect into a device. We simply use this with the "-c" flag to run a command. The command chosen is "request system license add URL". This fetches the license from a specific URL and then loads it onto the device. This may take a few moments depending on how busy the vSRX is when you run the command. Once completed it will print out the response from the command. The tool does not have any built in error checking so it may throw errors in the event it runs into any problems.

Using the tool to fetch your licenses

To run the tool change directories to "tools/". From here run the commands specified below. This will fetch your licenses from the proctors server.

```
python licensetool.py  --user root --password Juniper --host 172.16.0.1 --u
```

```
python licensetool.py  --user root --password Juniper --host 172.16.0.1 --u
```

Automating both the IDP Package and Template Installation

Now that we have a license to the device we can now download and install the signature pack. To do this we have created a separate tool. The commands that we need to run can be called using NETCONF RPCs. This means we can utilize the Junos PyEZ library and do not have to use the lower level Paramiko library.

Much like we did in the license tool we have a set of command line flags. In fact it is the same set of command line flags that we used before, minus the "--url" option as that is not required.

```
usage: idpsecpack.py [-h] [--host HOST] [--username USERNAME]
                     [--password PASSWORD]

Process user input

optional arguments:
  -h, --help           show this help message and exit
  --host HOST          Specify host to connect to
  --username USERNAME  Specify the username
  --password PASSWORD  Specify the password
```

The code for the script introduces the usage of loops and regular expressions. The loops are used to periodically request the status of the download and installation of the security package. We use regular expressions to match the output of the command to verify that the download has been completed. While we would hope that the XML response would be descriptive enough for us to look for a completed tag,

sometimes it will not be. In this case the the XML response tag is the same no matter the status of the download or installation. Because of this we must search for the response to validate the output. The particular string that we are looking for is "DONE;". This means that the transaction has completed and we no longer need to poll for the status of the change. For each RPC that it runs it outputs the response to the standard out.

Steps in Script

1. Gathers command line options from user
2. Connects to the host using the specified username and password
3. Requests to download the security package
4. Checks the status of the download
    - Sleeps two seconds at the end of the loop to give the package time to download
    - Once the "DONE;" message is seen then we break out of the loop
5. Requests to install the security package
6. Checks the status of the installation
    - Sleeps two seconds at the end of the loop to give the package time to install
    - Once the "DONE;" message is seen then we break out of the loop
7. Requests to download the policy templates
8. Checks the status of the download
    - Sleeps two seconds at the end of the loop to give the templates time to download
    - Once the "DONE;" message is seen then we break out of the loop
9. Requests to install the policy templates
10. Checks the status of the installation
    - Sleeps two seconds at the end of the loop to give the templates time to install
    - Once the "DONE;" message is seen then we break out of the loop

```
#!/usr/bin/env python

import argparse
import time
import re
import xml.etree.ElementTree as ET

from jnpr.junos import Device

parser = argparse.ArgumentParser(description='Process user input')
parser.add_argument("--host", dest="host", default="",metavar="HOST",help="
```

```
parser.add_argument("--username", dest="username", metavar="USERNAME",help=
parser.add_argument("--password", dest="password", metavar="PASSWORD",help=
args = parser.parse_args()

host = args.host
username = args.username
password = args.password

#first we instantiate an instance of the device
junos_dev = Device(host=host, user=username, password=password)
#now we can connect to the device
junos_dev.open()
#run the RPC to download the security package
download_result = junos_dev.rpc.request_idp_security_package_download()
#output to a sting
print ET.tostring(download_result,encoding="utf8", method="text")

#REGEX to match the done condition
matchDone = re.compile('Done;.*',re.MULTILINE)

#loop through status until OK is seen
while True:
    download_status = junos_dev.rpc.request_idp_security_package_download(s
    print ET.tostring(download_status,encoding="utf8", method="text")
    info = download_status.findtext('secpack-download-status-detail')
    if matchDone.match(info):
        print "Download completed"
        break
    time.sleep(2)

#run the RPC to install the security package
install_result = junos_dev.rpc.request_idp_security_package_install()
#output to a sting
print ET.tostring(install_result,encoding="utf8", method="text")

#loop through status until OK is seen
while True:
    install_status = junos_dev.rpc.request_idp_security_package_install(sta
    print ET.tostring(install_status,encoding="utf8", method="text")
    info = install_status.findtext('secpack-status-detail')
    if matchDone.match(info):
        print "Installation completed"
        break
    time.sleep(2)

#run the RPC to download the download policy templates
download_tmpl_result = junos_dev.rpc.request_idp_security_package_download(
#output to a sting
print ET.tostring(download_tmpl_result,encoding="utf8", method="text")
```

```
#loop through status until OK is seen
matchDone = re.compile('Done;.*',re.MULTILINE)

while True:
    download_status = junos_dev.rpc.request_idp_security_package_download(s
    print ET.tostring(download_status,encoding="utf8", method="text")
    info = download_status.findtext('secpack-download-status-detail')
    if matchDone.match(info):
        print "Download completed"
        break
    time.sleep(2)

#run the RPC to install the download policy templates
install_tmpl_result = junos_dev.rpc.request_idp_security_package_install(po
#output to a sting
print ET.tostring(install_tmpl_result,encoding="utf8", method="text")

#loop through status until OK is seen
while True:
    install_status = junos_dev.rpc.request_idp_security_package_install(sta
    print ET.tostring(install_status,encoding="utf8", method="text")
    info = install_status.findtext('secpack-status-detail')
    if matchDone.match(info):
        print "Installation completed"
        break
    time.sleep(2)
```

Using the tool to download and install the package

To run the tool change directories to "tools/". From here run the commands specified below. This will download and install the packages from the Internet.

```
python idpsecpack.py --user root --password Juniper --host 172.16.0.1
```