

Assignment 33

GUI and SSH with paramiko



LILLEBAELT ACADEMY OF
PROFESSIONAL HIGHER EDUCATION

Authors

Martin Grønholdt
mart80c7@edu.eal.dk

Rickie Ljungberg,
rick0118@edu.eal.dk

Kasper Soelberg
kasp062e@edu.eal.dk

Saturday 11 March 2017

Table of Contents

Introduction.....1

Juniper configuration snatcher.....2

Conclusion.....12

Appendix A.....13

Introduction.....14

Instructions.....14

Errors.....17

Introduction

The programs in this hand-in is an example of using libraries to make simple GUI applications in python using SSH and paramiko to login to a Juniper device and extract the configuration.

All files for this hand in are available at:

https://github.com/deadbok/eal_programming/tree/master/ass33

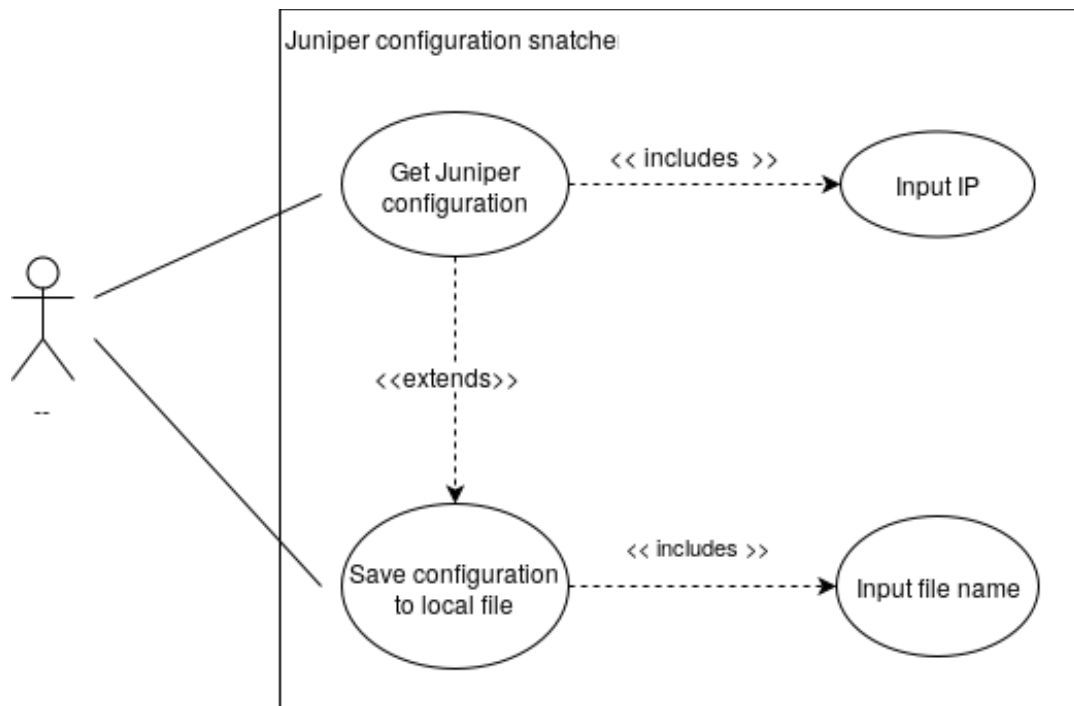
Error handling

This program will show a dialog box with a description of the error that occurred.

Juniper configuration snatcher

Specifications:

- GUI interface
- Fetching Juniper configurations
- Displaying Juniper configurations
- Save Juniper configuration to local file

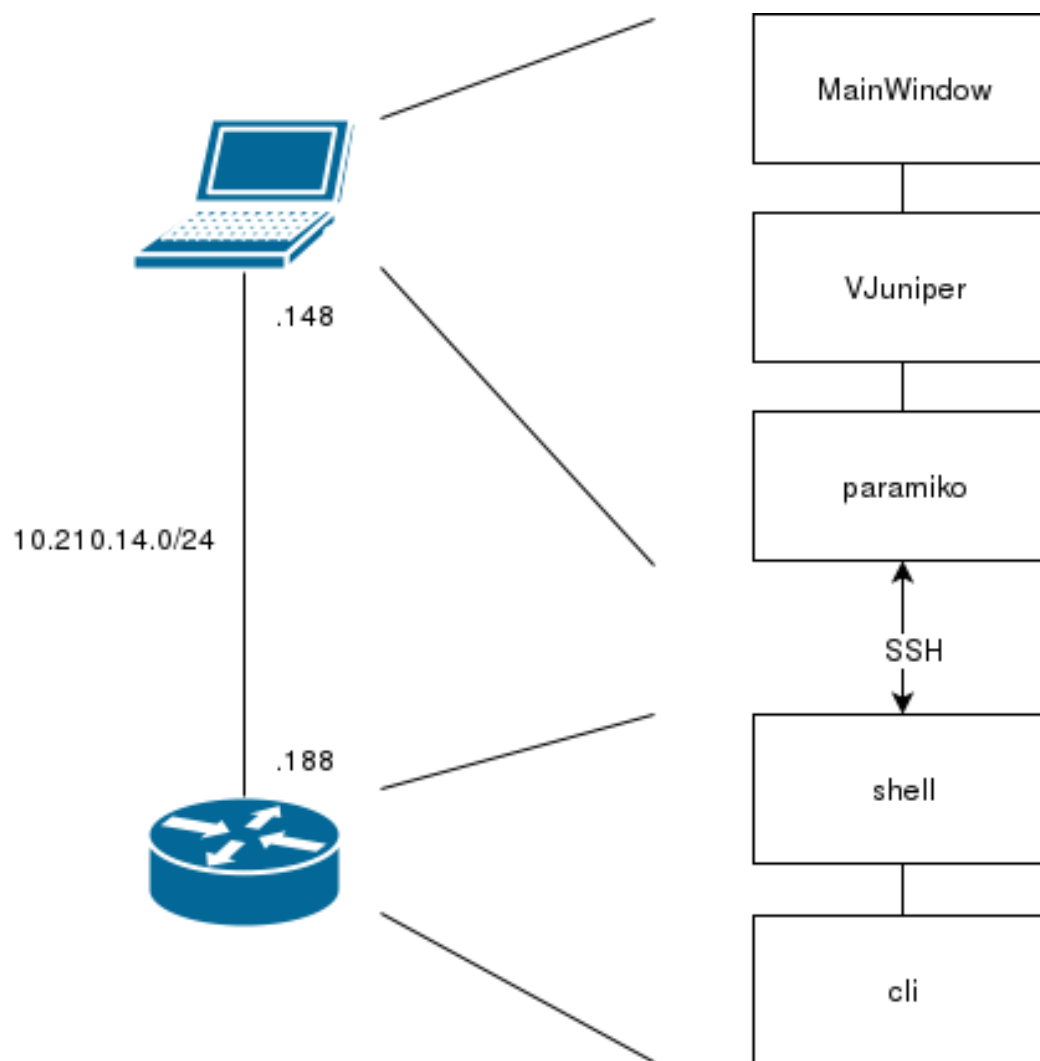


Use case diagram

The program fetches the configuration of a Juniper device, and is therefore expected to be connected to one such device. The illustration on the following page shows the client connected to the a Juniper router. In the client, communication with the router is accomplished using the paramiko SSH library. The actual commands used for controlling the router are all encapsulated in the VJuniper class, along with some logic to get the output back from the router through paramiko. The instantiated MainWindow object uses a VJuniper instance to get the configuration from the router when the user clicks the “Get configuration” button.

On the router the program logs into a shell using SSH, starts the cli, and shows the confugation.

Assignment 33



Network and block diagram

Assignment 33

jpgetconf.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Name: Juniper configuration snatcher
Author: Martin Bo Kristensen Grønholdt, Rickie Ljungberg, Kasper Soelberg.
Version: 1.0 (2017-03-09)

Program with a nice GUI to get the configuration from a Juniper device.
"""

import sys

from PyQt5.QtWidgets import QApplication
from mainwindow import MainWindow

def main():
    """
    Main program
    """
    # Instantiate the QT application class
    app = QApplication(sys.argv)
    # Create out window
    ui = MainWindow()
    # Exit when done.
    sys.exit(app.exec_())

# Run this when invoked directly
if __name__ == '__main__':
    main()
```

Assignment 33

mainwindow.py

```
# -*- coding: utf-8 -*-
"""
Name: Junpier configuration snatcher
Author: Martin Bo Kristensen Grønholdt, Rickie Ljungberg, Kasper Soelberg.
Version: 1.0 (2017-03-09)

Main window.
"""

import socket
import sys
from PyQt5.QtWidgets import QWidget, QPushButton, QGridLayout, QApplication, \
    QLabel, QLineEdit, QPlainTextEdit, QInputDialog, QMessageBox
from paramiko.ssh_exception import AuthenticationException, \
    BadHostKeyException

from vjuniper import VJuniper

class MainWindow(QWidget):
    """
    Class encapsulates the main window.
    """

    def __init__(self):
        """
        Constructor, creates the UI.
        """
        # Call the parent constructor.
        super().__init__()

        # Create the labels.
        ip_label = QLabel('Juniper IP address:')
        file_name_label = QLabel('Configuration file name:')
        config_label = QLabel('Configuration:')

        # Create the IP and file name edits.
        self.__ip_edit = QLineEdit()
        # self.ip_edit.setInputMask('000.000.000.000')
        self.__ip_edit.setPlaceholderText('127.0.0.1:22')
        self.__file_name_edit = QLineEdit()
        self.__file_name_edit.setPlaceholderText(
            'Leave empty to only view the config')

        # Create the configuration file view.
        self.__config_edit = QPlainTextEdit()
        # Do not allow editing the configuration.
        self.__config_edit.setReadOnly(True)

        # Create the buttons
        get_button = QPushButton("Get configuration")
        # Connect the get config button to the handler.
        get_button.clicked.connect(self.getConfigClicked)

        quit_button = QPushButton("Quit")
        # Close the window on clicking "Quit"
```

Assignment 33

```
quit_button.clicked.connect(self.close)

# Create a grid layout
grid = QGridLayout()
grid.setSpacing(10)

# Place the labels in the top row.
grid.addWidget(ip_label, 0, 0, 1, 1)
grid.addWidget(file_name_label, 0, 2, 1, 1)

# Place the edit fields on the next line.
grid.addWidget(self.__ip_edit, 1, 0, 1, 2)
grid.addWidget(self.__file_name_edit, 1, 2, 1, 4)

# Place the configuration view label on the next line.
grid.addWidget(config_label, 2, 0)

# Place the configuration view.
grid.addWidget(self.__config_edit, 3, 0, 1, 6)

# Place the buttons at the second row, with some cells between them for
# spacing.
grid.addWidget(get_button, 4, 3)
grid.addWidget(quit_button, 4, 5)

# Set the layout of this widget
self.setLayout(grid)

# Set title
self.setWindowTitle('Juniper configuration snatcher')
# Show window
self.show()

# Create the VSRX instance used to talk to the Juniper device.
self.__vjuniper = VJuniper()

def error(self, msg):
    """
    Open a message box for errors.
    :param msg: The error message.
    """
    # Create an instance.
    mb = QMessageBox()
    # Set the window title.
    mb.setWindowTitle('Error')
    # Set the window content text
    mb.setText(msg)
    # Show the message box.
    mb.exec()

def getConfigClicked(self):
    """
    Handler that is called when the get config button is clicked
    """
    try:
        # Get the contents of the IP edit field.
        ip = self.__ip_edit.text()
        # Split in port number and IP if : is in the input.
        if ':' in ip:
```


Assignment 33

```
# Isolate the port.
port = int(ip.split(':')[1])
# Isolate the IP address
ip = ip.split(':')[0]
else:
    port = 22

# Default value if the edit field is empty.
if ip == '':
    ip = '127.0.0.1'

# Handle wrong data.
except ValueError:
    self.error('Error in IP address')
    return

# Show a dialog to get the login user.
username, ok = QDialog.getText(self, 'Enter user name',
                                'Enter user name:')

# Get out if the user pressed cancel.
if not ok:
    return

# Show dialog to get the login password.
password, ok = QDialog.getText(self, 'Enter password',
                                'Enter passwod:',
                                QLineEdit.Password)

# Get out if the user pressed cancel.
if not ok:
    return

# Everything has checked out so far, lets talk to the juniper device.
try:
    # Connect to the juniper device.
    self.__vjuniper.connect(ip, port, username=username,
                             password=password)
    # Run "show configuration" on the Juniper device and return the
    # output.
    config = self.__vjuniper.showConfiguration()
# Handling of various communication errors.
except AuthenticationException:
    self.error('Could not authenticate with the router')
    return
except BadHostKeyException:
    self.error('The IP address entered was invalid')
    return
except socket.error:
    self.error('Connection error or time out')
    return

# Put the configuration file in the edit component in the GUI.
self.__config_edit.setPlainText(config)

# Save the configurtion to a file, if a file name was entered.
try:
    file_name = self.__file_name_edit.text()
    if file_name != '':
        with open(file_name, 'w') as config_file:
            config_file.write(config)
```

Assignment 33

```
except IOError:
    self.error('Could not save the configuration file')
```

vjuniper.py

```
"""
Name: Juniper configuration snatcher
Author: Martin Bo Kristensen Grønholdt, Rickie Ljungberg, Kasper Soelberg.
Version: 1.0 (2017-03-09)

Class that encapsulates the finer details of communicating with a Juniper
device using paramiko.
"""

import sys, paramiko, time

class VJuniper():
    """
    Class that encapsulates the finer details of communicating with a Juniper
    device using paramiko.
    """
    # Constant used to tell that the Juniper device is in operational mode.
    OPERATIONAL = 0
    # Constant used to tell that the Juniper device is in configuration mode.
    CONFIGURATION = 1
    # Constant used to tell that the Juniper device is in shall mode.
    SHELL = 2

    def __init__(self):
        """
        Constructor..
        """
        # Used for the channel that is opened using paramiko.
        self.__channel = None
        # Used to keep track of the mode that the Juniper device is in.
        self.__mode = None

        # Create the paramiko SSH client object.
        self.__client = paramiko.SSHClient()
        # Allow unknown hosts to be added to the host keys
        self.__client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    def __getOutput(self, wait_interval=0.1, wait_period=1):
        """
        Empty the paramiko in buffer and return the contents.

        :param wait_interval: The interval between checking for new output.
        :param wait_period: The maximum amount of time that this method is
                           allowed to wait for output.
        :return: The output of the Juniper device as a string.
        """
        # Start of with
        # an empty return value.
        ret = ''
        # We haven't done any waiting yet.
        current_wait = 0
```

Assignment 33

```
# We are not done.
done = False

# Instead of waiting blindly crossing our fingers that it is enough,
# this code block tries to be smarter.
# It uses the fact that whenever the Juniper device is done running a
# command, it will show a prompt. The prompt is different in each mode
# which is why we keep track of the mode in other places. This will
# loop until the maximum amount of time allowed, has passed, checking
# at wait_interval periods for the prompt at the end of the output.
#
# To things will break this:
# * Always run commands using no-more to make sure that the Juniper
#   device will not wait for a keypress that will never happen.
# * There is a possibility that the end of the output from the router
#   could be the start of a comment, a hashtag followed by a space,
#   which is handled as a prompt in configuration mode. To fix this
#   a regular expression including the user@hostname part would be
#   better.
while not done:
    # Is there any new output?
    if self.__channel.recv_ready():
        # Add it to our return variable.
        ret += self.__channel.recv(10000).decode()
        # Reset the wait period.
        current_wait = 0
    else:
        # No new output, wait some more.
        time.sleep(wait_interval)
        current_wait += wait_interval

    # Have we reached the time out?
    if current_wait == wait_period:
        done = True

    # Check for a prompt which indicates the end of the output, and
    # get out if we find one..
    if self.__mode == self.SHELL:
        if ret.endswith('% '):
            done = True
    elif self.__mode == self.OPERATIONAL:
        if ret.endswith('> '):
            done = True
    elif self.__mode == self.CONFIGURATION:
        if ret.endswith('# '):
            done = True
# Return the output.
return (ret)

def startCLI(self):
    """
    Start the CLI on the Juniper device, entering operational mode.
    """
    # Invoke a shell on the Juniper device.
    self.__channel = self.__client.invoke_shell()
    # The Juniper device is now in shell mode.
    self.__mode = self.SHELL

    # Enter the cli.
```

Assignment 33

```
self.__channel.send('cli\n')
# We are now in operational mode
self.__mode = self.OPERATIONAL

# Empty the paramiko input buffer, and discard the data.
self.__getOutput()

def connect(self, ip, port='22', username='root', password='TestTest'):
    """
    Connect to a Junpier device using paramiko SSH, and start the cli.

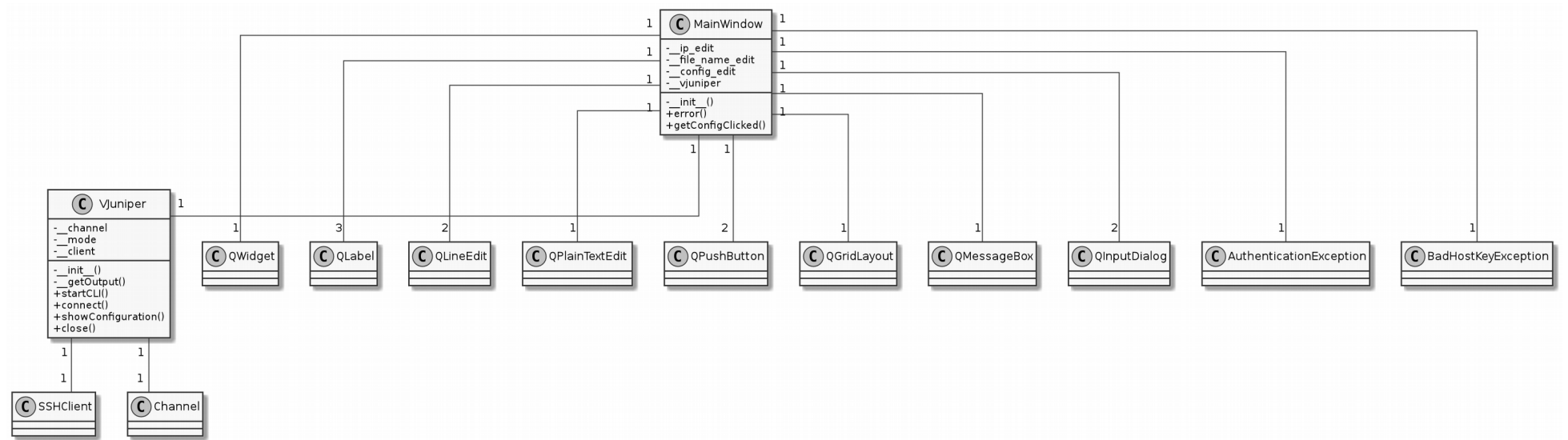
    :param ip: IP address of the Juniper device.
    :param port: The port that SSH is listening on.
    :param username: The user name used to log in to the Juniper device,
    :param password: The password used to log in to the Juniper device,
    """
    # Use paramiko to connect.
    self.__client.connect(ip, port=port, username=username,
                          password=password, timeout=10)
    # Start the cli.
    self.startCLI()

def showConfiguration(self):
    """
    Run the show configuration command on the Juniper device and return
    the output.

    :return: The Juniper device configuration.
    """
    # Start of with an empty return value,
    ret = ['', '', '']
    # Check that we have a connection.
    if self.__channel is not None:
        # Send the show configuration command.
        self.__channel.send('show configuration | no-more\n')
        # Get the output from the Juniper device, make a list by splitting
        # the string at each new line..
        ret = self.__getOutput().split('\n')

    # Return the list as a string, but remove the first and last line,
    # which is the command that we ran, at the top line, and the prompt at
    # the last line.
    return ('\n'.join(ret[1:-1]))

def close(self):
    """
    Close the connection to the Juniper device.
    """
    # If the channel is open, exit the cli for good measure.
    if self.__channel is not None:
        self.__channel.send('exit')
    # Close the connection.
    self.__channel.close()
```



Class diagram

Conclusion

GUI programming in Python is fairly simple and understandable, paramiko is a powerful tool for automation and fairly easy to work with.

Appendix A

Juniper configuration snatcher

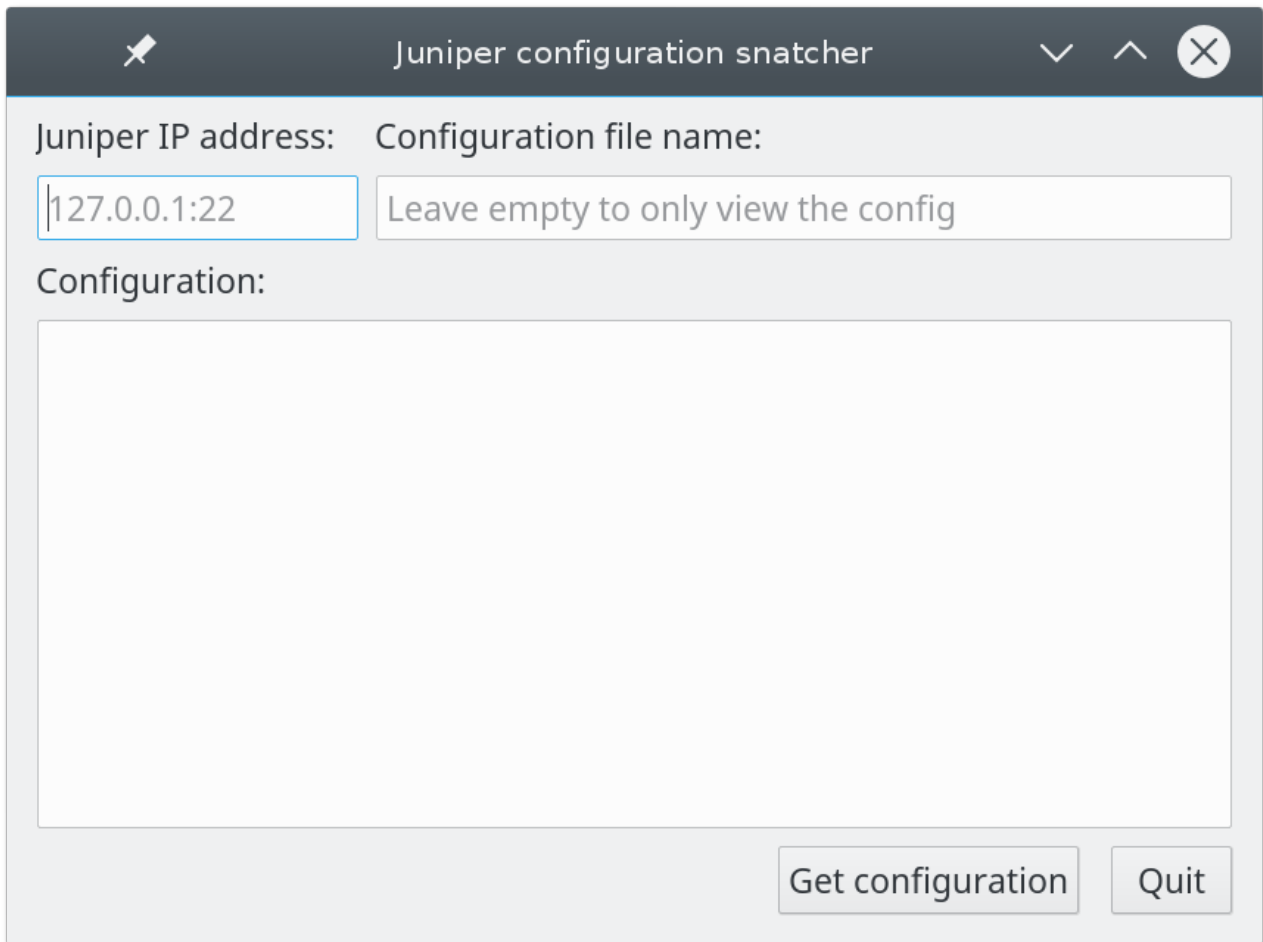
User manual

v1.0

Introduction

This manual describes the use of the “Juniper configuration snatcher” program. The purpose of this program is to download the configuration from a Juniper device to a local file.

Instructions

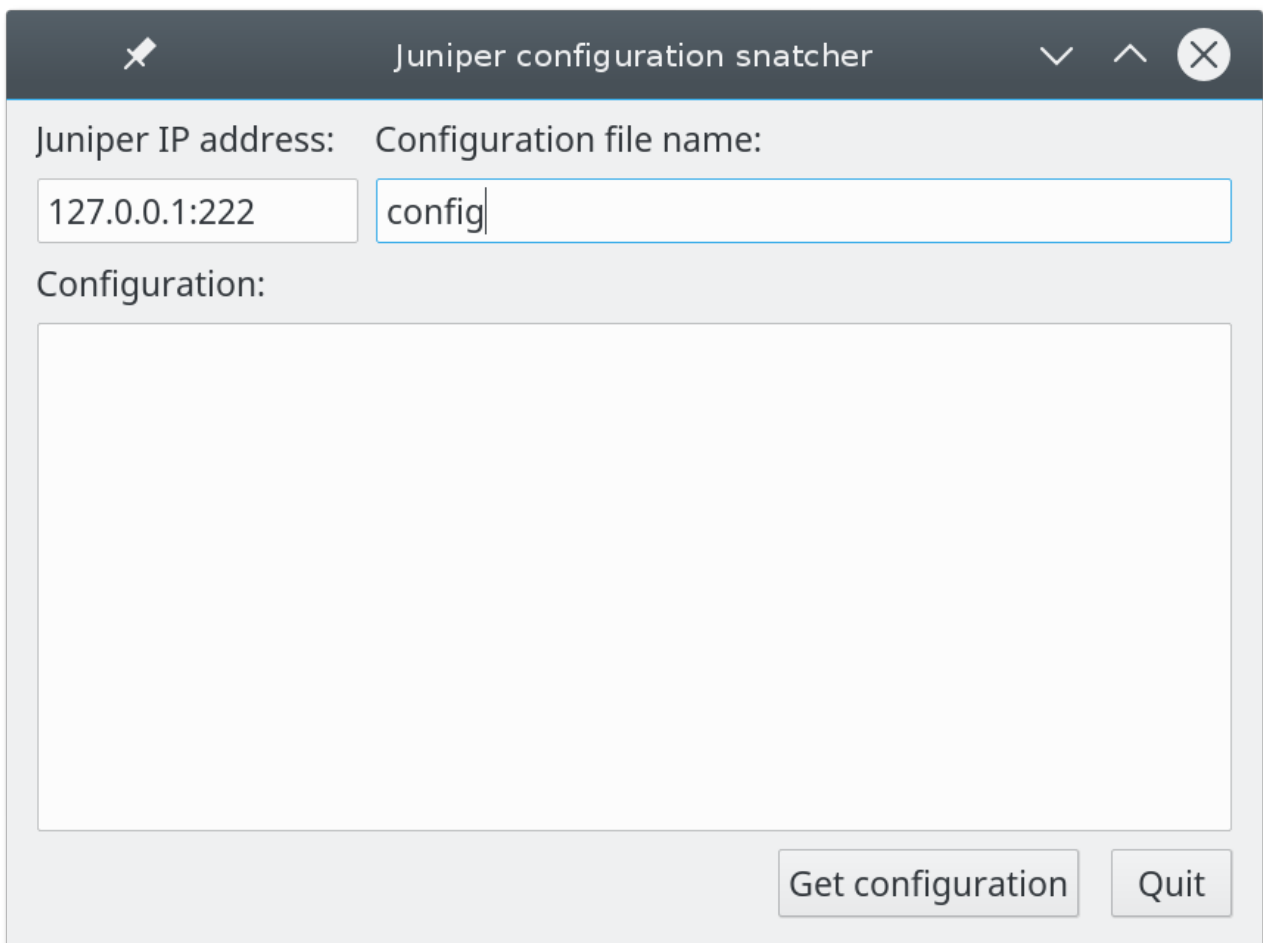


The screenshot shows a window titled "Juniper configuration snatcher". It has a dark title bar with a close button (X) and window controls (minimize, maximize). The main area contains two input fields. The first is labeled "Juniper IP address:" and contains the text "127.0.0.1:22". The second is labeled "Configuration file name:" and contains the placeholder text "Leave empty to only view the config". Below these fields is a large text area labeled "Configuration:". At the bottom right of the window are two buttons: "Get configuration" and "Quit".

Illustration 1: The main window at program start.

The program is quite simple it works by inserting the IP address of the router, possibly followed by the port number assigned to the SSH service in the first text box (referring to Illustration 1). A default port of 22 is assumed if no port is entered. The next text box is for entering the file name used to save the configuration file locally. If no file name is entered, the program will still show the configuration, but not save it.

Assignment 33



Juniper configuration snatcher

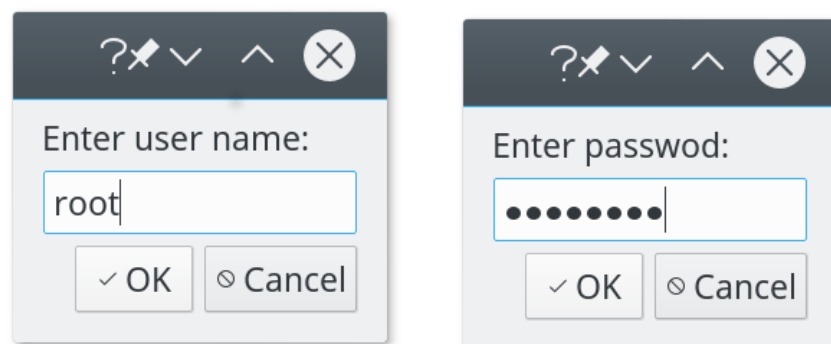
Juniper IP address: 127.0.0.1:222 Configuration file name: config

Configuration:

Get configuration Quit

Illustration 2: Main window with the values entered.

When the information has been entered, as seen in Illustration 2, pressing the button “Get configuration” will access the router via SSH and ask for the login information as shown in Illustration 3.



Enter user name: root OK Cancel

Enter passwod: OK Cancel

Illustration 3: Entering the user name and password.

Assignment 33

After that the program will pull the configuration file from the juniper device and save it to a file if a name was given earlier, in any case the configuration is also be shown in the box in the program.

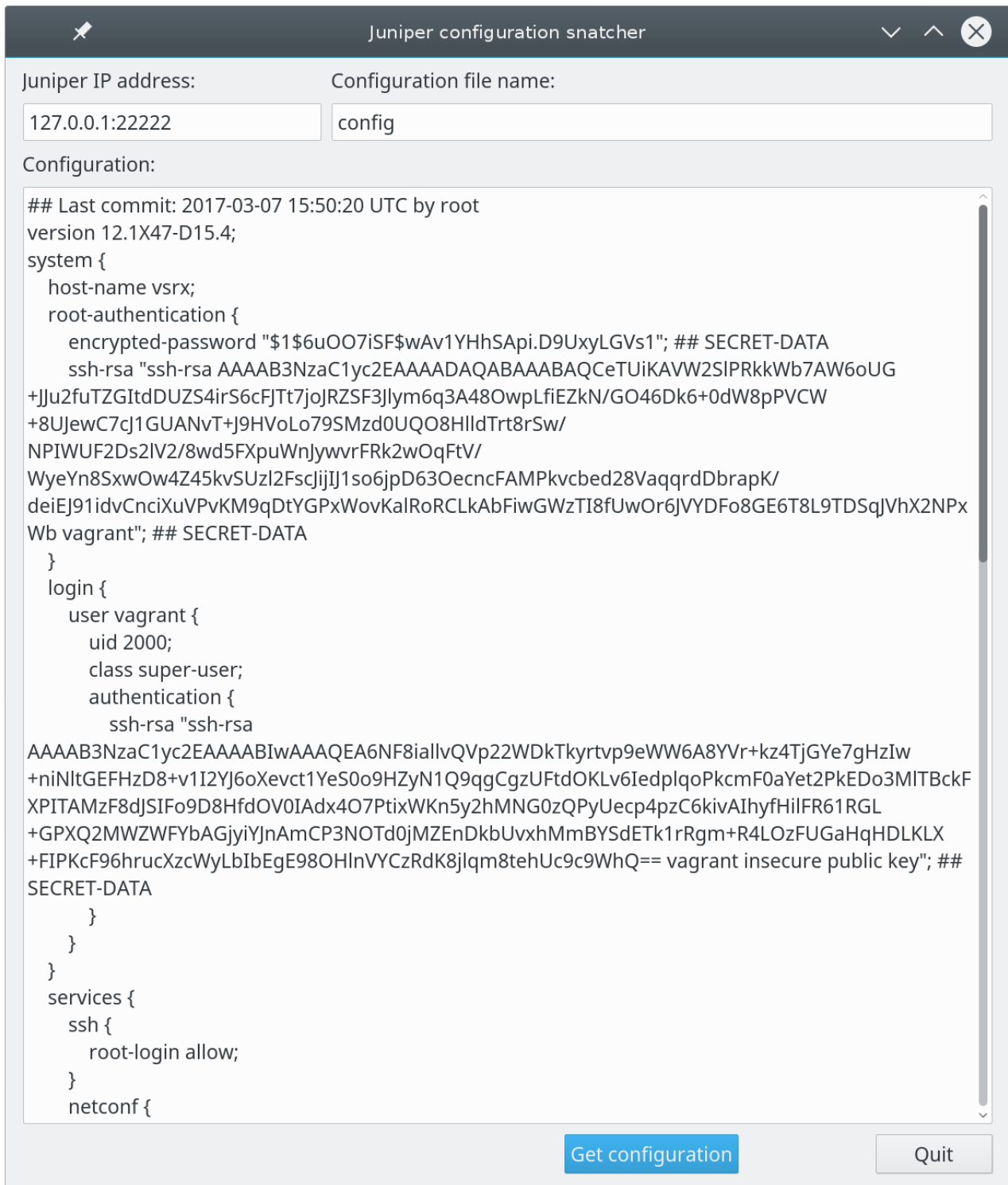


Illustration 4: The main window after the configuration is fetched

Errors

If there is a problem while running the program you will see one of the following error boxes which will specify what went wrong.

