Python Lab
File Processing

Marc Serra Garcia & David Pina Valero

Enginyera Informàtica | 12/02/2019

Index

Introduction

Our program is distributed in 3 mains scripts: Config Storage, Config Processing and File Processing.

Config Storage is a File Reader of router config files.

Config Processing is a Data Processer in this case specially for Fortigate routers.

File Processing is a Report Maker in PDF made with ReportLab solution.

The main script is the report maker, “File_Processing.py”. To run the script via IDE or command, you must have all the scripts in the same a folder and an extra supplied folder named “/images/” . The script will ask for a config file path to the user then the program will automatically create a pdf file with the extracted information of the input file.

Config Storage

The main function of this object, ConfStorage, is to process the given config file and store all the information, as well as to provide some very useful methods to find the specific information we might need in a quick and easy manner.

To make the information readily accessible, and since it would be necessary to loop through it many times, the config information was stored as blocks, where every main 'config .....' line that is not **indented** in the file is added to a dictionary as a key.

Every line until the end of that specific configuration – All of its edits, sets, and sub-configs – is stored as a list in the same order as it is read, with one string for each line.

This way, looking for a specific *set* within a specific *config block* was limited to that block, instead of having to look through the entire thousands of lines in the file.

Every method is commented within the code, explaining briefly its use, like finding a certain set, a certain edit, or even getting all the edits within a config block.

CODE

```
class ConfStorage:

    def __init__(self, sFileInput):
        with open( sFileInput, 'r' ) as fileInput:
            self.dicConfigs = dict() # Main dictionary with all configs
            self.sCurrentKey = "" # Current Key for the dictionary
            self.listCurrentKey = list() # Holds all the lines of the
current selected config
            self.listEditContent = list() # Holds all the lines of the
current selected edit within the current config

            for sLine in fileInput:
                ## Read initial variables like build number, version etc ##
                sLine.replace('\n', '').replace('\r', '')
                if "#config-version" in sLine:
                    configInfo = sLine.split("-")
                    self.name = configInfo[1].split("=")[1][3:6]
                    self.version = configInfo[2]
                    self.build = configInfo[4][5:8]
                    self.number = configInfo[5].split(":")[0]
                    ## More variables can be added if needed...

                ## Read a config block entirely ##
                elif "config" in sLine:
                    self.sCurrentKey = sLine
                    self.dicConfigs[self.sCurrentKey] = list()
                    self.listCurrentKey =
self.dicConfigs.get(self.sCurrentKey)
```

```
                    while not sLine == 'end':
                        sLine = fileInput.readline().replace('\n',
'').replace('\r', '')
                        self.listCurrentKey.append(sLine)


    ## Sets the current Key to work on ##
    ## Returns true if the key exists ##
    ## Returns false if it is not found ##
    def setCurrentConfig(self, sKey):
        for key in self.dicConfigs.keys():
            if sKey in key:
                self.sCurrentKey = key
                self.listCurrentKey = self.dicConfigs.get(key)
                return True
        return False



    ## Sets the current Edit list ##
    ## Returns false if the edit cannot be found, and leaves the current
edit empty ##
    ## Returns true if the edit is found ##
    def setEdit(self, sEdit):
        found = False
        tabs = 0
        self.listEditContent = list()
        for sLine in self.listCurrentKey:
            if not found:
                if sEdit in sLine:
                    found = True
                    tabs = sLine.count(" ")-1
            elif ("next" in sLine) and (tabs == sLine.count(" ")):
                return True
            else:
                self.listEditContent.append(sLine)
        return False


    ## Returns the sub-values of the CURRENT edit ##
    ## Can be empty if the current Edit has not been set ##
    def getCurrentEdit(self):
        return self.listEditContent

    ## Finds the Set parameter from the CURRENT Edit list ##
    ## Returns the sanitised Set as a string with everything but the
parameter ##
    ## Returns a "-" string if the Set is NOT found ##
    def getSet(self, sSet):
```

```
        found = False
        for line in self.listEditContent:
            if not found:
                if sSet in line:
                    listSet = line.split()
                    found = True
        if found:
            for part in sSet.split():
                if part in listSet:
                    listSet.remove(part)
            return " ".join(listSet)
        else:
            return "-"

    ## Finds all the Sets in the parameter list from CURRENT Edit list ##
    ## Returns a list of strings with all sanitised Sets, in order ##
    ## Includes not found sets as "-" ##
    def getSets(self, listSets):
        listSanitisedSets = list()
        for sSet in listSets:
            listSanitisedSets.append(self.getSet(sSet))
        return listSanitisedSets


    ## Convenience methods using previous ones for ease of processing ##
    ## Ugly, but convenient! ##

    ## Finds the set within the edit in the given config ##
    def getSet1(self, sConfig, sEdit, sSet):
        self.setCurrentConfig(sConfig)
        return self.getSet2(sEdit, sSet)

    ## Finds the set in the given edit ##
    ## Current config only ##
    def getSet2(self, sEdit, sSet):
        self.setEdit(sEdit)
        return self.getSet(sSet)

    ## Finds the set within the entire config ##
    ## getConfig must be called for the config parameter ##
    ## Returns the first set found only ##
    ## Returns '-' if the set is not found ##
    def getSet3(self, sSet, listConfig):
        self.listEditContent = listConfig
        return self.getSet(sSet)

    ## Returns a list with all sets from the given list of sets ##
    ## Searches only in the given edit ##
```

```
    def getSets1(self, listSets, sEdit):
        self.setEdit(sEdit)
        return self.getSets(listSets)

    ## Returns a list with all the unmodified config's content ##
    def getConfig(self, sConfig):
        self.setCurrentConfig(sConfig)
        return self.listCurrentKey

    ## Returns a list with every edit name within the current config ##
    ## Does NOT return their content ##
    def getAllEdits(self):
        result = list()
        for line in self.listCurrentKey:
            if ("edit" in line) and (line.count(' ') == 5):
                result.append(line.replace("edit ", "").replace(" ", ""))
        return result
```

Config Processing

This object's main task is to find whatever information is needed for each snippet to fill of every section, using the previous module. Every snippet is stored in a result list in the very same order as the PDF.

There are no parameters and so every section method always returns the same result for the same files. We will go through some of these section methods to see how the needed information is gathered and treated.

SECTION 2.1

This one is the simplest of all – The ConfStorage object has some handy variables created when it processes the config file, only for the few information that might be used multiple times while redacting the PDF, like the product's name, its version and build number.

```
def section21(self):
        result = [self.conf.name, self.conf.version, self.conf.build,
self.conf.number]
        return sanitize(result)
```

SECTION 2.2

This section is tricky, because there the sets we have to find are not under any edit, and so our default getSet() method would not work.

Two additional methods were made, one to gather a config block's information in a list (*getConfig)*, and another to find the first apparition of a given set on that list (*getSet3)*.

From there on we simply found the needed sets using the method and stored them in the result list.

```
def section22(self):
        result = list()
        temp = self.conf.getConfig("config system global")
        result.append(self.conf.getSet3("set admin-sport", temp))
        self.conf.setCurrentConfig("config system admin")
        self.conf.setEdit('edit "admin"')
        temp = [self.conf.getSet("set trusthost1"), self.conf.getSet("set
trusthost2"), self.conf.getSet("set trusthost3")]
        for item in temp:
            result.append(item)
        return sanitize(result)
```

SECTION 2.3

Much like before, the sets needed here weren't part of any *edit*, so it was necessary to resort to the methods explained on the previous section.

```
def section23(self):
        result = list()
        temp = self.conf.getConfig("config system dns")
        result.append(self.conf.getSet3("set primary", temp))
        result.append(self.conf.getSet3("set secondary", temp))
        result.append(self.conf.getSet3("set domain", temp))
        return sanitize(result)
```

SECTION 2.4

In this section, we needed to find all of the edits within the config block '*config system interface*', and select **only** those with *portN* as a name, where *N* could be any number.

Once we had the list with those edits, we found the needed sets for each and every one of them, storing it all in the result table in its proper order.

```
def section24(self):
        result = list()
        self.conf.setCurrentConfig("config system interface")
        AllEdits = self.conf.getAllEdits()
        edits = list()
        for edit in AllEdits:
            if 'port' in edit:
                edits.append(edit)
        del AllEdits

        for edit in edits:
            result.append(edit)
            result.append(self.conf.getSet2("edit "+edit, "set alias"))
            result.append(self.conf.getSet2("edit "+edit, "set ip"))
            result.append(self.conf.getSet2("edit "+edit, "set dhcp-relay-
ip"))
        return sanitize(result)
```

SECTION 2.5

This section is again two tables, except for the three first snippets which depend on both the number of edits, and their priorities. So first the sets in each edit are found, processed and appended into the result list.

Then each of the edit's gateways are inserted at their corresponding indexes at the beginning of the result list, depending on their priority (from lowest to highest).

The second table has none of this, and only gathers the needed sets for all of its config edits.

```
def section25(self):
        result = list()
        self.conf.setCurrentConfig("config router static")
        edits = self.conf.getAllEdits()
        result.append(len(edits))
        prioritats = list()
        for edit in edits:
            self.conf.setEdit('edit '+edit)
            result.append(self.conf.getSet('set gateway'))
            result.append(self.conf.getSet('set device'))
            sPrio = self.conf.getSet('set priority')
            result.append(sPrio)
            if sPrio.isnumeric():
                prioritats.append(int(sPrio))

        index = prioritats.index(min(prioritats))
        result.insert(index+1, self.conf.getSet2('edit '+edits[index], 'set
gateway'))
        result.insert(index+2, self.conf.getSet2('edit '+edits[index+1],
'set gateway'))
        self.conf.setCurrentConfig('config system link-monitor')
        edits = self.conf.getAllEdits()
        for edit in edits:
            self.conf.setEdit('edit '+edit)
            sets = self.conf.getSets(['set server', 'set gateway-ip', 'set
srcintf', 'set interval', 'set failtime', 'set recoverytime'])
            for item in sets:
                result.append(item)
        return sanitize(result)
```

SECTION 2.6

Yet again, section 2.6 gathers data for another table. This time, though, we don't have to find each set for every edit in the config block, but only for certain given edits. Our method to get all of the edits is no longer useful here, and instead we create the list manually, called *edits*.

Then we loop for each edit, finding its sets and appending them into the result list – The *'set type'* set, or the lack of it, defines part of the table. If the set returns *'iprange'* in a certain edit, we save *'Range'* in the result. If it returns nothing, it means the set is missing and we save *'Subnet'* instead.

```
def section26(self):
        result = list()
        self.conf.setCurrentConfig('config firewall address')
        edits = [
                '"inside_srv"',
                '"inside_wrk"',
                '"cloud1"',
                '"cloud2"',
                '"srv-demeter"',
                '"srv-devrepo"',
                '"srv-nebulaz"',
                '"vpn-net"'
                ]

        for edit in edits:
            self.conf.setEdit('edit '+edit)
            result.append(edit)
            result.append('Address')
            setType = self.conf.getSet('set type')
            if 'iprange' in setType:
                setType = 'Range'
                startIp = self.conf.getSet('set start-ip')
                endIp = self.conf.getSet('set end-ip')
                result.append(startIp+'.'+endIp)
                result.append('Any')
            else:
                setType = 'Subnet'
                ip = self.conf.getSet('set subnet')
                if ip != '-':
                    ip = ip[0:ip.index(' ')]
                result.append(ip)
                result.append('Any')
            result.append(setType)
        return sanitize(result)
```

SECTION 2.7

This section requires looking information for another table. Though this table is specially large, there are no special cases or specific edits to look for. We simply gather every single edit using the *getAllEdits()* method, and loop through them, finding all the sets we need for the table by using *getSets(listOfSets)* and saving them all at the result list, in order as always.

```
def section27(self):
        result = list()
        self.conf.setCurrentConfig('config firewall service custom')
        edits = self.conf.getAllEdits()
        for edit in edits:
            result.append(edit)
            self.conf.setEdit('edit '+edit)
            sets = self.conf.getSets(['set category', 'set tcp-portrange',
'set udp-portrange', 'set protocol'])
            sets[1] = sets[1].replace(' ', '\n')
            sets[2] = sets[2].replace(' ', '\n')
            for sSet in sets:
                result.append(sSet)
        return sanitize(result)
```

SECTION DRAW

We made a specific method like every other section, in this case for the information needed in the drawing. Since the information needed was already found in previous sections, we simply use their methods to get all the information we need.

```
def draw(self):
        result = list()
        temp = self.section24()
        result.append(temp[0]) # draw[0] = Port1
        result.append(temp[2].split(" ")[0]) # draw[1] = IP Port1
        result.append(temp[4]) # draw[2] = Port2
        result.append(temp[6].split(" ")[0]) # draw[3] = IP Port2
        result.append(temp[8]) # draw[4] = Port3
        result.append(temp[10].split(" ")[0]) # draw[5] = IP Port3
        result.append(temp[12]) # draw[6] = Port4
        result.append(temp[14].split(" ")[0]) # draw[7] = IP Port4
        temp = self.section25()
        result.append(temp[1]) # draw[8] = Line from Port1 (IP de sortida
internet principal)
        result.append(temp[2]) # draw[9] = Line from Port4 (IP de Backup +
WIFI)
        temp = self.section26()
        result.append(temp[0]) # draw[10] = Line from Port2 (Text -
inside_srv)
        result.append(temp[2]) # draw[11] = Line from Port2 (IP/DNS under
inside_srv)
```

```
        result.append(temp[5]) # draw[12] = Line from Port3 (Text -
inside_wrk)
        result.append(temp[7]) # draw[13] = Line from Port3 (IP/DNS under
inside_wrk)
        return result
```

OTHER SECTIONS

All the other sections follow pretty much the same process as the ones explained before, using our ConfStorage module's methods to search for specific config blocks, their edits and the sets within those edits. Here's the code of them all:

```
def section28(self):
        result = list()
        self.conf.setCurrentConfig('config firewall vip')
        edits = ['"VIP_srv-01"', '"VIP-srv-02"']
        for edit in edits:
            result.append(edit)
            self.conf.setEdit('edit '+edit)
            sets = self.conf.getSets(['set extintf', 'set extip'])
            result.append(sets[0]+'/'+sets[1])
            sets = self.conf.getSets(['set extport', 'set protocol'])
            result.append(sets[0]+'/'+sets[1])
            result.append(self.conf.getSet('set mappedip'))
            sets = self.conf.getSets(['set mappedport', 'set protocol'])
            result.append(sets[0]+'/'+sets[1])
        return sanitize(result)

    def section29(self):
        result = list()
        self.conf.setCurrentConfig('config firewall policy')
        edits = self.conf.getAllEdits()
        for edit in edits:
            self.conf.setEdit('edit '+edit)
            result.append(edit)
            sets = self.conf.getSets([
                'set srcintf',
                'set srcaddr',
                'set dstintf',
                'set srcaddr',
                'set groups',
                'set dstaddr',
                'set service',
                'set action',
                'set av-profile',
                'set webfilter-profile',
                'set application-list',
                'set ips-sensor',
```

```
                'set ssl-shh-profile',
                'set logtraffic',
                'set nat'
            ])
            result.append(sets[0]+'/'+sets[1])
            result.append(sets[2])
            result.append(sets[3]+'('+sets[4]+')')
            for i in range(5, 15):
                result.append(sets[i])
        return sanitize(result)

    def section210(self):
        self.conf.setCurrentConfig('config antivirus profile')
        edits = self.conf.getAllEdits()
        edits.remove('"default"')
        result = edits
        return sanitize(result)

    def section211(self):
        self.conf.setCurrentConfig('config webfilter profile')
        allEdits = ['"default"', '"web-filter-flow"',  '"monitor-all"',
'"flow-monitor-all"']
        edits = self.conf.getAllEdits()
        for edit in allEdits:
            edits.remove(edit)
        result = edits
        return sanitize(result)

    def section212(self):
        result = list()
        self.conf.setCurrentConfig('config application list')
        result.append(self.conf.getAllEdits()[-1])
        return sanitize(result)

    def section213(self):
        result = list()
        self.conf.setCurrentConfig('config ips sensor')
        edit = self.conf.getAllEdits()[-1]
        result.append(edit)
        self.conf.setEdit(edit)
        sets = self.conf.getSets(['set location', 'set severity', 'set
os'])
        for item in sets:
            result.append(item)
        return sanitize(result)
```

*Please note, the *sanitize* method only removes every " character from the result lists.

File Processing

This script is the main script of the program. It's mandatory to execute this script as the main script of the program, if not it won't work properly. The script is structured in 14 methods, the most of them gets the principal document, with processed information from the config file and adds the information in the proper section of the report.

Footer and Header of report:

```
def add_footer(c):
def add_header(c):
```

Title sheet:

```
def add_Title(c):
```

Index sheet:

```
def add_index(c):
```

Introduction:

```
def add_intro(c,name)
```

Draw Sheet:

```
def add_draw(c,sec):
```

Config sheet 1: sections: 2.1, 2.2, 2.3, 2.4

```
def add_confi_1(c, name, firmware, port, res1, res2, res3, ip1, ip2, domini, dataset):
```

Config sheet 2: sections: 2.5, 2.6

```
def add_confi_2(c, num, ip1, ip2, dt1, dt2, dt3):
```

Config sheet 3,4,5: sections 2.7

```
def add_confi_3(c,dt1):
def add_confi_4(c, dt1):
def add_confi_5(c, dt1, dt2):
```

Config sheet 6: section 2.8, 2.9, 2.10, 2.11

```
def add_confi_6(c,dt1,dt2,utm, utm2):
```

Config sheet 7: section 2.12, 2.13

```
def add_confi_7(c,utm3,utm4,client,sSeverity1,os):
```

Init method with the data process:

```
def __init__(file):
```

Input file and launch program:

```
try:
    nfile = input('Please enter the config file you want to read: ')
    __init__(nfile)
    print("Succesfull file creation.")
```

```
except:
    print('Error file not found')
```

References:

Report Lab file generator:

https://www.reportlab.com/documentation/