INTERNATIONAL
HELLENIC
UNIVERSITY

Network Automation using Python

George Milios

SID: 3307190015

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Cybersecurity

DECEMBER 2020

THESSALONIKI – GREECE

Network Automation using Python

George Milios

SID: 3307190015

Supervisor:	Assistant Prof. Anastasios Politis
Supervising Committee Members:	

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Cybersecurity

DECEMBER 2020

THESSALONIKI – GREECE

Acknowledgements

With many thanks to my supervisor Assistant Prof. Anastasios Politis for his dedicated support and guidance during the running of this project.

I would also like to express my sincere gratitude to my colleague and friend Dimitris Grendas for his support, encouragement, and objective critique throughout this project. His deep knowledge in matters related to VM machines proved invaluable as well as the insights to the development of the application.

Furthermore, special thanks to my colleague and friend Aristidis Dinakus for his tips and tricks regarding the project.

To conclude, I cannot forget to thank my family and friends for all the unconditional support in this very intense academic and especially my wife Aggeliki, for putting up with me being sat in the office for hours on end, and for providing guidance and a sounding board when required.

Abstract

This dissertation was written as a part of the MSc Cybersecurity at the International Hellenic University and aims on researching the fundamental network automation technologies and combine them in a software program that will be developed in python.

This dissertation is basic an application that enables a user to perform basic network automation tasks for instance backup and restore a configuration file on many devices at once but also more advanced operations for example security and configuration settings, through the software's Graphical User Interface (GUI) .In the application's code, as well as in the paper, will be shown and explained various options that are available to the user to connect and configure network devices using Python and its libraries,.

George Milios

04/01/2021

Contents

Figures list

Introduction

Automation at general applies to various technological grounds. As a term according to Wikipedia "Automation is the technology by which a process or procedure is performed with minimal human assistance. Automation or automatic control is the use of various control systems for operating equipment such as machinery, processes in factories, boilers and heat-treating ovens, switching on telephone networks, steering and stabilization of ships, aircraft and other applications and vehicles with minimal or reduced human intervention." [1]

Similarly, Networking automation can be found in different levels, from automating task in a single device to automating processes, for example backup the configurations or configuring a routing protocol on multiple devices and in higher level in the hierarchy as Cross-Domain automation.

A usual "strategy" is to begin building automations at the device level with creating tasks to automate the necessary processes and building it up from there to the Domain level.[2]

Tasks are scripts (or playbooks) that are used to diminish the number of processes that must be performed from people using the console environment.

Every well-defined task that is repeated over time can be automated.

The collection of such tasks is called Device Automation.

Device automation is used for many years for fault management or at service level monitoring but with the growing business needs new challenges as well as a new set of opportunities arise.

One of those opportunities is investing in Network automations by companies.

The rapid development of modern networks in enterprises along with new technologies, such as Internet Of Things (IOT) and cloud computing which are also reliant on the network ,led to the network infrastructure growth necessity resulting in increased workload demands for provisioning , maintenance monitoring and administration by the network engineers.

Methods used by network engineers up until now was not only time consuming but also knowledge about proprietary protocols and technologies was required.

In an effort to reduce cost and create efficiencies network engineers developed Network Automation techniques to automate repeating everyday tasks.[3]

With the support from almost all major networking companies (like Cisco) an open-source community was created which had as a goal to implement automation applications mainly with the use of standard interfaces (SSH, REST) and generic programming languages like python.

Using Python and a collection of modules and functions

A Python library that strengthens the ability of this programs to be vendor neutral is NAPALM.

By using NAPALM, through a set of functions that implements, python applications can interact with different network device Operating Systems using a unified API.

The vendor dependencies were eliminated up to a point by Software Defined Network (SDN) using standard protocols like OpenFlow.

Openflow is a low-level hardware-based protocol and one of the first that started the Software Defined Network revolution.

It allows the network device's control plane to be decomposed from the data plane.

Control plane is used when traffic from a device is directed to another device or when a device has to generate a message, this kind of traffic could be device management traffic, pings, monitoring, routing protocols commands and so on.[4]

Control plane is also involved on how data flows through the network

Data plane, also known as forwarding plane, is defined as the manager that flows traffic through a device.[4]

One of the reasons that we distinguish the kind of traffic that a device handles is that technologies can be developed independently, for instance we can install new IOS in a cisco router to have access to new features because the control plane features are not locked in the hardware of data plane.

Control plane and data plane are abstract logical concepts but SDN separates them into actual devices.

Software Defined Network uses this distinguish that can be made on the control plane and the data plane, removes the control plane from the device and takes over the control of how the network behaves so the network devices focus on forwarding only.

SDN enables IT administrators to manage all network devices and provision network services easily from a SDN application. The ability to automate a network and program the traffic as well as the increased agility are some of the SDN advantages.

On the other hand, unfortunately there is no backwards compatibility with non-SDN network especially with legacy networks.

Dissertations structure

In dissertations next chapters basic technologies of network automation will be explained as well as methods that was used until now from network engineers to automate a network.

Benefits and disadvantages of its method will be briefly described with more in-depth analysis at Network Automation applications that was produced with generic languages like python.

Using python and some of the above methods a GUI application will be developed allowing the user to perform basic network automation tasks.

Particularly in the first chapter the challenges that network managers must confront will be enumerated along with problems that may arise from this challenge presenting some real-life examples.

Later on, the benefits of network automation will be explained in addition to how the preceding issues solved.

Second chapter is all about the technologies and techniques that used in the development of the application. The basic functionality of modules will be explained and the reason that these modules were chosen above others.

In the following chapters the application development will be explained in more details concluding in the demonstration of the applications functionality.

1 Network automation

This chapter briefly describes network automation concepts, the reason that led network engineers and enterprises to network automation and various types of automation.

1.1 Challenges in Network configuration

The main reason for an organization to adopt network automation is to reduce the time that is needed to maintain and deploy changes on the network. Although time is crucial not all organizations choose to move to network automation.[3]

Each network nowadays is unique and the same goes for the network devices, this is something that discourage enterprises on moving to automation on their network because it usually involves upgrade of the current equipment or moving for example on SDN because these unique devices are sometimes difficult if not impossible to be part of an automated network.[4]

Another hitch on moving towards network automation is the lack of a vendor free standardized schema in conjunction with an affordable environment for testing.

Except from time saving network automation helps on troubleshooting problems on a network.

The network administrators had to manually configure each device through CLI and in the case of a change that had to be done in all the devices, such as an addition of a new VLAN, they had to go to its device and set it up.

This was not only time consuming, but it also maximizes the possibility of an error. Moreover, it is dangerous to apply changes on a network on work hours and there are enterprises that works every day and there is a tight window, usually on holidays, for applying changes.

Surveys that have been conducted showed that the most usual reason for a downtime on a network are human errors.

The most common human error on networking is misconfiguration of network devices.

It is common task of a network administrator to apply an update configuration file to a bunch of network devices.

As was mentioned above there is usually a tight time window to complete this task and when tasks like that have to be done by hand from a CLI environment coping and pasting the configuration and applying it without a thorough check is a usual practice.

Here is where network automation comes to assist.

1.2 Automating Network Operations

Network automation does not apply only to configuring devices, on the contrary the most significant part of network automation that helps to reduce human errors is that It gives administrators the ability to automate procedures that runs compliance and validation checks against the current configuration or any configuration that is about to be deployed.

As a result, this reduces delivery times of the network changes and the risk of outage or service distribution also minimizes the possibility of a human error and ensures the alignment with the network polices.

Another procedure that can be automated is troubleshooting. When a problem arises on a network the first step in solving the problem is to collect information's. The collection of the information from each device can be cumbersome and take a lot of time that is essential because usually in the meantime the network, or a part of it, is down.

Be using network automation we can automate all the commands needed to obtain the information's needed for troubleshooting and have real time access to this information.

By obtaining this information's programmatically means that we can also check them in real-time. Checking real time information and choosing what actions to follow if some values, for example MTU, changes is a third aspect of network automation called automated monitoring.

Automated monitoring helps prevent outages caused by hardware failure.

But how does a network automation get that info from the network devices? How does it communicate with them?

1.2.1 Ways of managing network devices

Previously control plane and data plane were mentioned another element of a network device is the management plane [2]. ManagemeSNMPnt plane is a part of control plane

that must be distinguished, it is responsible with the monitoring and configuration of a network device by providing management, monitoring and configuration services.

Along with network automation management plane has evolved from using SNMP to empower a network device to communicate with another application-to-application Programmable Interfaces also known as APIs.[4]

Simple Network Management Protocol (SNMP) is a widely used Layer7 protocol that has been around for decades, it is responsible for exchanging management information's between network devices.

If a network device has the SNMP agent enabled and configured could communicate with another network device, a monitoring tool or a management system.

SNMP has 4 basic components: SNMP Manager, SNMP Agent, Managed Devices and Management Information Base (MIB).

Management Information Base (MIB) describes and models data that are connected from the SNMP agent. MIB is necessary to be able to access and manage network devices to monitor and configure them with SNMP via GET and SET command requests.

Although SNMP is embedded in almost all network devices its major disadvantage is that it is not optimized for real-time monitoring so other methods had to be used, but for simply getting info from network devices and not for real-time monitoring it is used in network automation because as mentioned it is embedded in almost all network devices and there are SNMP libraries in python.

In 2006 Network Configuration protocol (NETCONF) was published. It is a Layer3 protocol, it is compared with SNMP because of its ability to install, manipulate and delete configuration files. Extensible Markup Language (XML) is used for data encoding both for the configuration data and the protocol messages which are exchanged through SSH.[7]

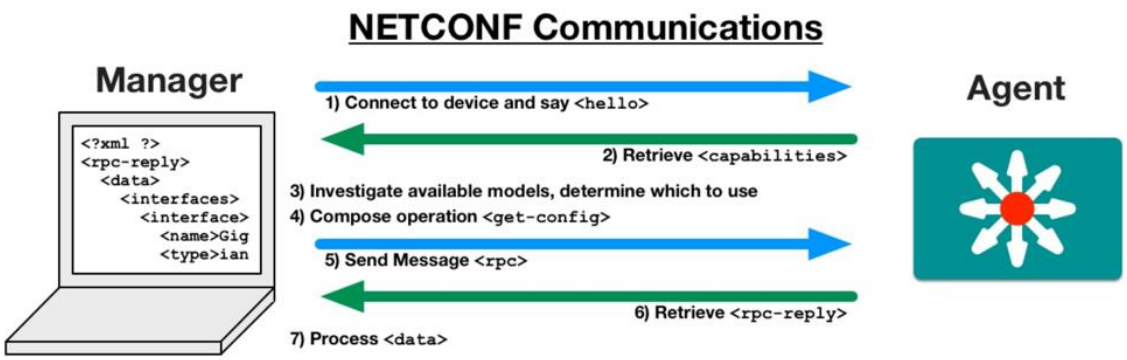


Figure 1: NETCONF Communications

NETCONF uses remote procedure calls (RPCs) encoded in XML files to configure a network device. In addition, more than one RPC can be encoded in an xml but NETFCONF has the option, in contrast with CLI where commands are executed one by one, if a command fails no command from the particular XML applies to the device.

NETCONF'S drawback is that it is usually vendor specific. Even between devices that supports NETCONF there can be incompatibilities depending on how data is modeled.

Another way to manage networks devices was discovered with the contribution of the open networking community that was mentioned in the introduction. In the introduction only OpenFlow was mentioned but this was only one of the many protocols that was created by this community OpenConfig or OpenAPIs and many more. Another achievement of this community is that more and more devices support python box to be able to run automation scripts locally. Furthermore, more sturdy protocols than SNMP and SSH are supported from the network devices like NETCONF that we mentioned above or RESTful APIs.[4]

REST stands for Representational State Transfer, is an architectural style that makes effortless the communication between computer systems on the web.

RESTful is called a system that complains with REST's guideline. The main characteristics of these systems are that they are stateless, meaning that client state is not saved on the server side but client itself is responsible to keep track of the sessions state.

The protocol that are used from the RESTful applications for communication is mostly HTTP. In network automation this can be seen in a network device that is accessible through a URL (for example a router or a switch that we can access their configuration pages through a URL).

The requests are sent with HTTP GET commands and the responses are xml or JSON formats usually. Other HTTP commands that may be used, especially in networking, are HTTP POST, HTTP PUT and HTTP PATCH.

The most current option for managing network devices are Software Defined Network (SDN) controllers. USING SDN controllers an administrator can manage configurations and policies, also SDN controllers take on the role of the control plane.

It is said that SDN controllers are the future on network managing, they simplify the management and visibility of network through their GUI but still if numerous controllers are deployed the troubleshooting and the need to manually apply changes to different controllers still exist so does and the need for Network Automation.

One advantage is that most controllers that exist exposes their APIs so that network administrators could use them and automate processes that despite the use of SDN controllers still have to be done manually.[5][6]

1.2.2 Well known network management platforms

In addition to controller platform solutions released from big networking companies open network community has also released a controller platform as shown in the list below.

- Cisco Open SDN Controller ACI
- Juniper Open Contrail
- VMware NSV
- Big Network Controller by Big Switch
- Network Cloud Service Orchestrator by Amdocs

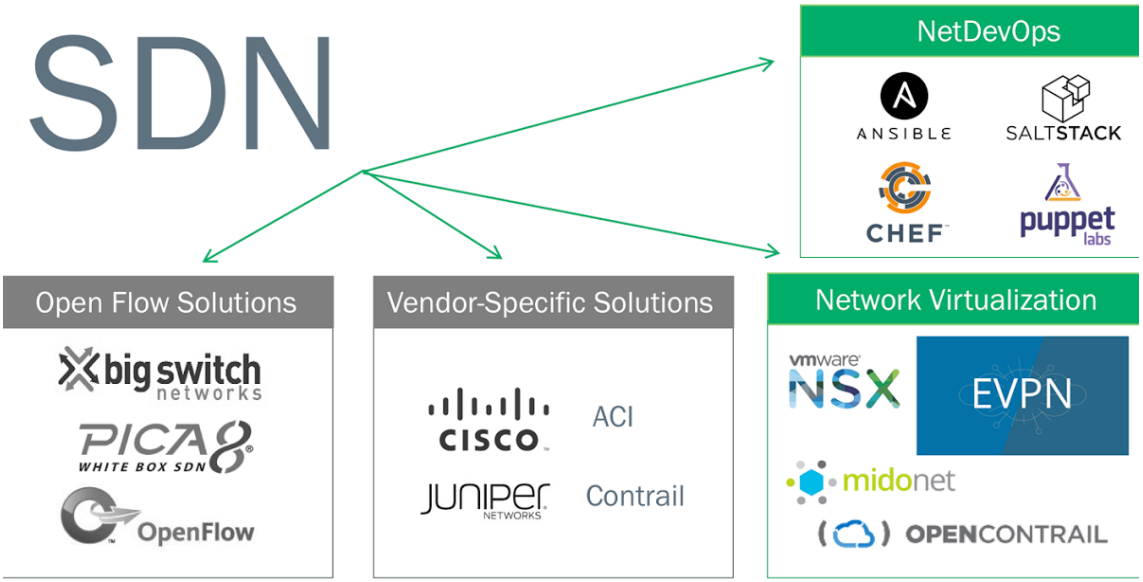


Figure 2:SDN Solutions

1.3 Network automation drawbacks

As explained above there are many reasons why an enterprise should follow the path of network automation, but as everything in life has pros and cons the same applies on network automations.

To automate a network effectively the network engineers must have deep understanding of how network devices work and behave to implement automation to network devices. Automating a network having semi-learning knowledge of the devices that it is consist may lead to major errors and downtime.

Network automation minimizes the human error factor but does not eradicate them. When automating an error, the impact is much bigger than a human making a configuration error in one device.

For example, a misconfiguration caused major outage both in its scope and duration (it lasted about 4 hours) in Google Services. As mentioned in the incident report "Two normally benign misconfigurations, and a specific software bug, combined to initiate the outage: firstly, network control plane jobs and their supporting infrastructure in the impacted regions were configured to be stopped in the face of a maintenance event. Secondly, the multiple instances of cluster management software running the network control plane were marked as eligible for inclusion in a particular, relatively rare maintenance event type. Thirdly, the software initiating maintenance events had a specific bug, allowing it to de-schedule multiple independent software clusters at once, crucially even if those clusters were in different physical locations."

2 Technologies and techniques

2.1 Program development information's

The application's purpose is to demonstrate different ways to connect and configure network devices thus commented code will be present in the code to exhibit an alternative solution.

The implementation will be done in python and the code will follow most of pep 8 instructions. PEP 8 is a style guide for python code that gives coding conventions. Following a style guide when coding an application improves the readability of code, make it consistent and easily maintained. [9]

Some of the recommendations of PEP 8 style guide that was followed are the below.

- Indentation

  4 spaces per indentation level, continuation lines align wrapped elements

- Blank lines

  two lines for top-level functions single line for methods

- Imports

  Each import on separate line except if it is in the format "from library import something". All the imports are at the top of the code.

- snake case naming style

  It refers to the style of variables names writing. Each word starts with lowercase letter and the space is replaced with an underscore.

Secure methods will be used, for example the connectivity with the network devices will be accomplished through SSH instead of Telnet and if there will be a need for data serialization JSON will be used instead of Pickle.

By using methods and functions of the imported modules the application's security feature is enhanced. For instance, the password that the application needs to connect to the network devices are encoded and encrypted and when the user is prompt to input a password the password is masked with asterisks.

Multithreading will be used to speed up the execution of the code.

2.2 Python Modules and libraries

In this chapter important libraries that was used in the applications code will be briefly explained and an explanation will be given why they were chosen over others.

2.2.1 Paramiko

Paramiko is a pure python interface that implements the versions 2 SSH protocol in Python and provides client and Server functionality.

Paramiko can obtain high performance on low level cryptographic concepts.

Any device that can be configured through SSH can also be configured from python scripts using this module.[10]

2.2.2 Netmiko

Netmiko is an open-source multi-vendor library, meaning that by using Netmiko devices from different vendors can be configured from python using Netmiko.

Some of the devices that Netmiko supports are: Cisco IOS, Juniper, Arista, HP and Linux. It also may support and other vendors such as Alcatel, Huawei and Ubiquity but limited testing has done with these vendors.

Netmiko runs in top of Paramiko to make SSH connection to network devices less complex, more versatile, and easiest to use. Although Netmiko is easier to use as mentioned above it supports specific vendors and only a number of their devices. On the other hand, Paramiko can be used to communicate with any device that supports SSH

Both Paramiko and Netmiko are alternative options for devices that do not support APIs[11][12]

2.2.3 JSON

JavaScript Object Notation module is a lightweight interchangeable data format that is used for converting python object, for example list or dictionaries, into a format that can be stored on a text file or a database or to be transmitted across a network connection and then convert the data back to a python object or other environments.

JSON format advantages is its interoperability and security.

It is supported in many environments and it does not allow the execution of arbitrary code.

2.2.4 ipaddress

This module library will be used to check that the addresses that the user inputs into the application are valid IP addresses. Modules like this displays the power of Python. Instead of having to write tenths of lines in order to check if an IP is a valid one, using this module it only takes three to four lines.[13]

2.2.5 Base64 and cryptography

Base64 is used to encode and decode the passwords that are used to connect to networking devices in addition to cryptography and some of its recipes, such as Fernet, so that a high-level symmetric encryption is applied to the passwords. [14][15]

2.2.6 Tkinter

This is the module library that will be used for constructing the Graphical User Interface of the application.[16]

2.2.7 threading

Using threading and its ability to construct high level threading interfaces on the program the application can send Netmiko connection command and configuration commands simultaneously instead of waiting to finish the configuration in one device and then continue with the rest. Using this module, the time that is required for the application to complete shrinks down a lot.[17][18]

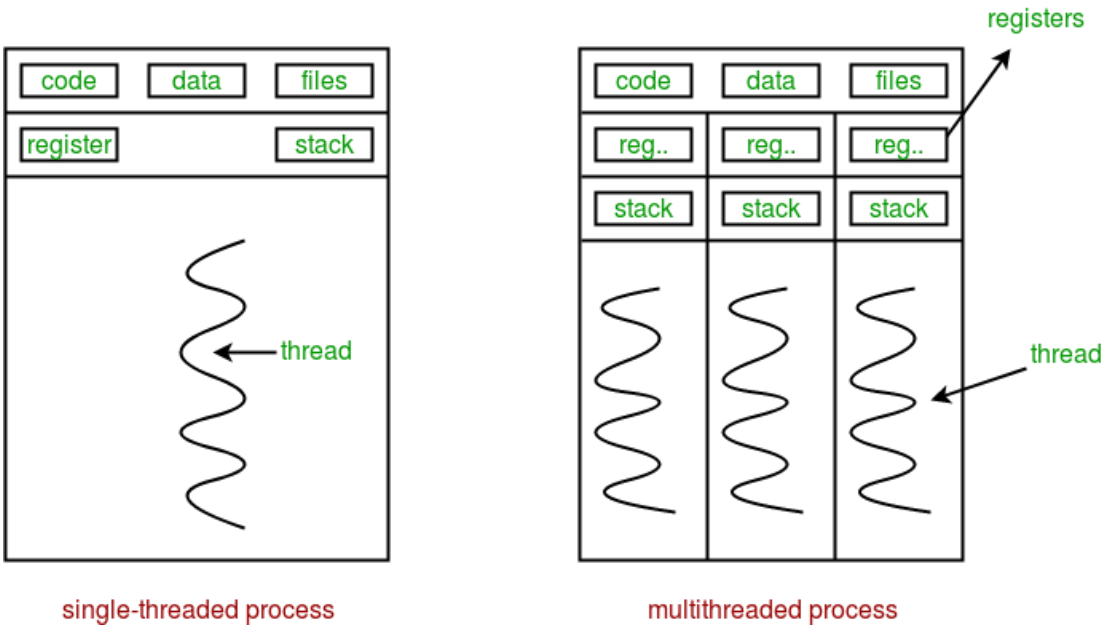


Figure 3:Single threading vs multi-threading

2.3 PyCharm

PyCharm IDE was used to develop the application. IDE is an acronym for integrated development environment. IDE is a software application that provides to programmers comprehensive facilities for developing software. An IDE usually consist of an editor, a debugger and build automation tools.

2.4 Virtual environment

GNS3 was selected as the virtual environment application that will be used to test the application. Another option could be the Cisco's DevNet or as a third option EVE-NG.

2.4.1 GNS3 GUI installation

GNS3 is a software that is used to emulate, configure and test a network environment. It is an open-source free software and can be downloaded from the official web site https://www.gns3.com/ .

GNS3 consist of two components. The all-in-one software (GUI) which is a graphical user interface and the Virtual Machine (VM) which is a server that runs in a virtual environment and provides better topology size and device support.

The installation is straightforward, and the default options should be used.[19]

Figure 4:GNS3 installation

After the installation and booting of the GNS3 GUI on the Servers Summary window the PC's name that the GNS3 is installed must be shown and it should also have a green light on the left.

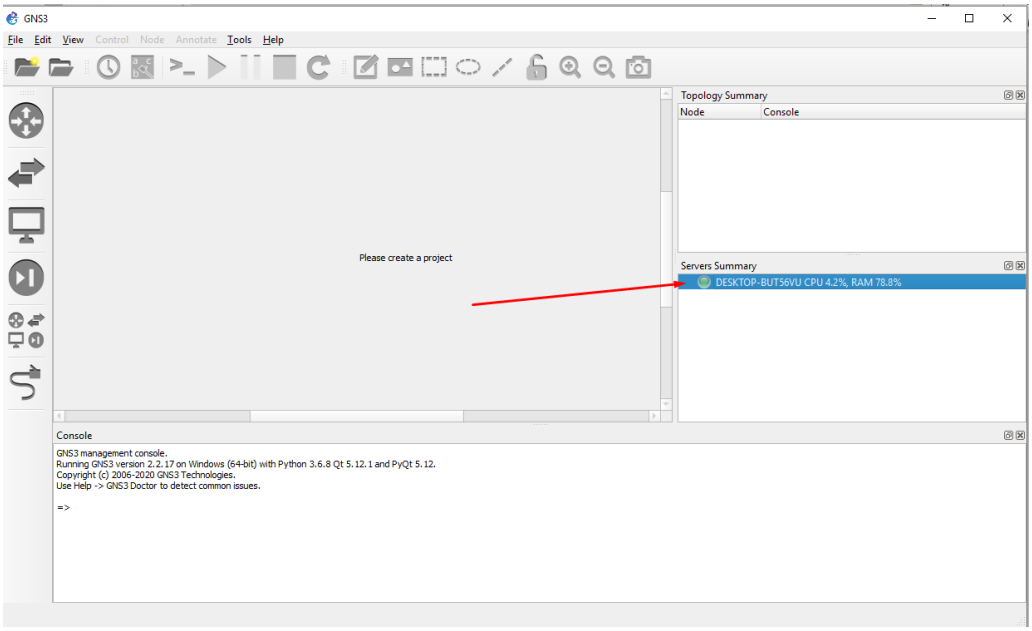


Figure 5:GNS3 Servers

If there is nothing on that window or the light is red try restarting the GNS3, restarting the PC or check if the firewall or antivirus stopped the GNS3 service from running. If it is not running, make sure if necessary that permission was given to GNS3 through firewall and antivirus.

GNS3 doctor (Help -> GNS3 Doctor) is a helpful module to check if there is something that blocks GNS3 from working correctly.

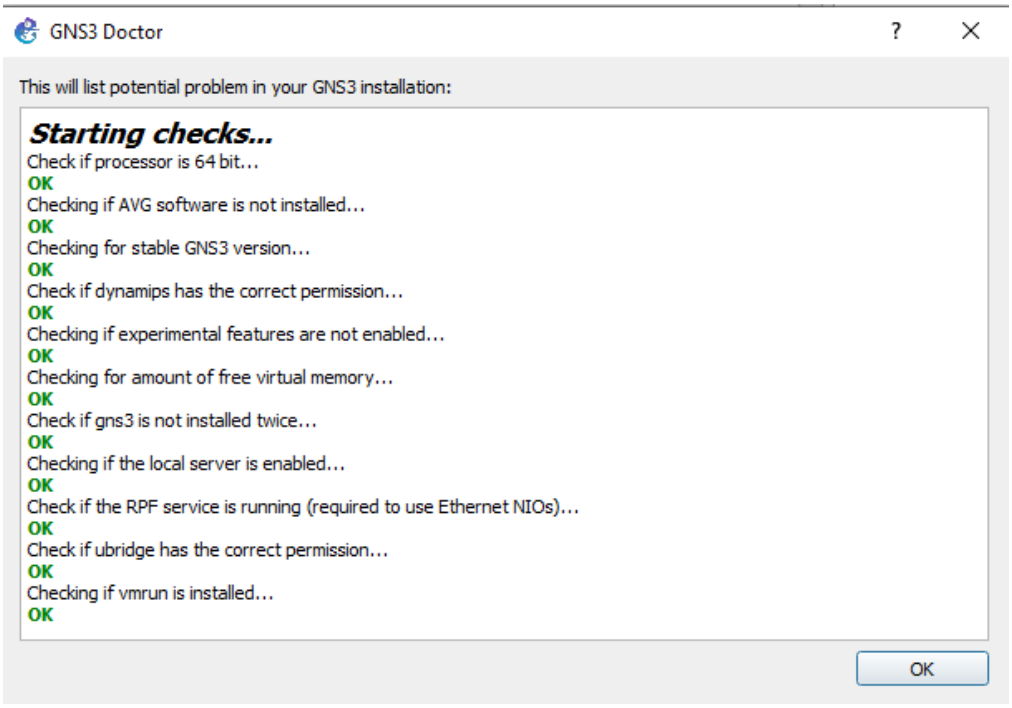


Figure 6:GNS3 Doctor

After the successful installation of GNS3 a basic project can be created to check that GNS3 works.

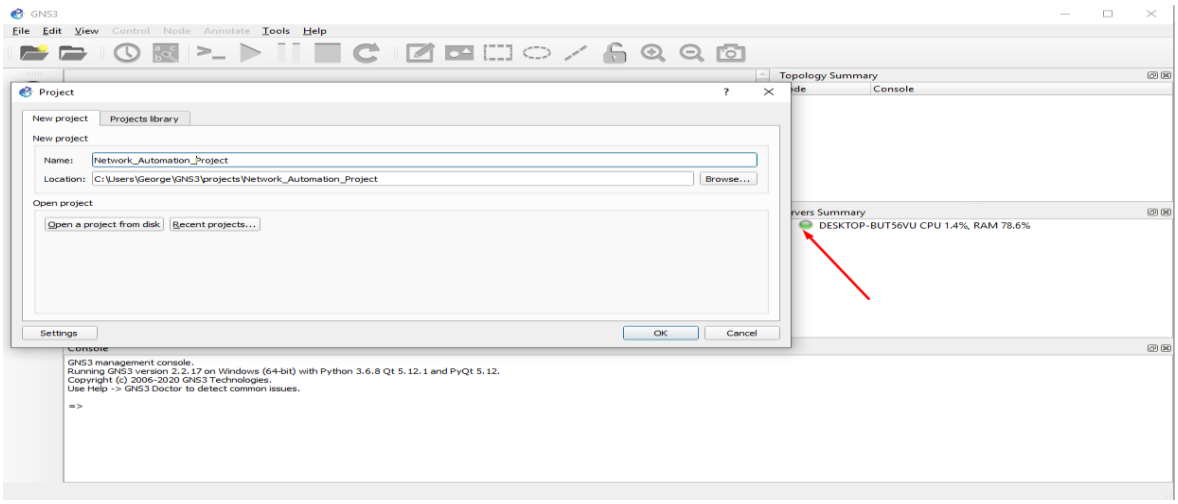


Figure 7:GNS3 Project creation

Network connectivity should be checked by configuring a basic network (a router or a switch and some end-devices) via ping command.

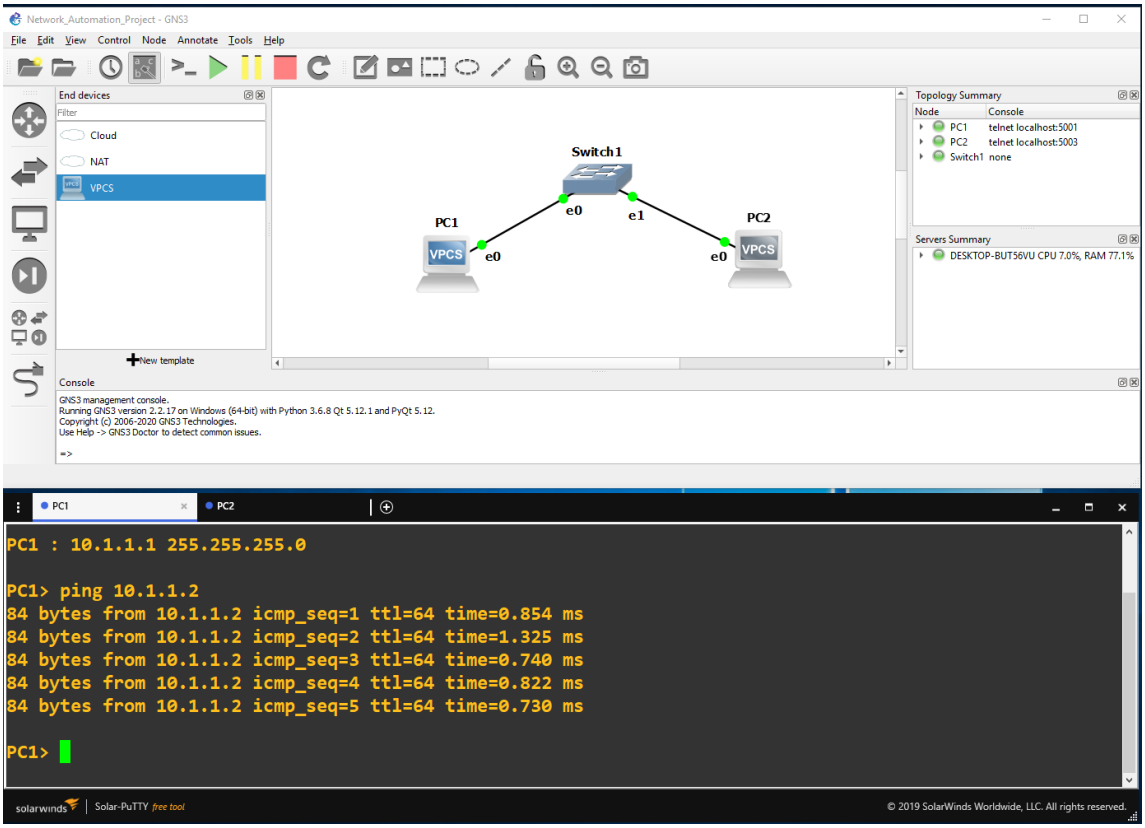


Figure 8:GNS3 Connection testing

After that, GNS3 VM has to be installed in order to be able to run Cisco IOS images.

2.4.2 GNS3 VM

GNS3 VM can be downloaded also from the GNS3 site. To install GNS3 VM a VM player must be used, for this project the VMWare workstation Player will be used but VirtualBox or HyperV could also be used. Attention must be given in the fact that GNS3 GUI and GNS3 VM should be in the same version.

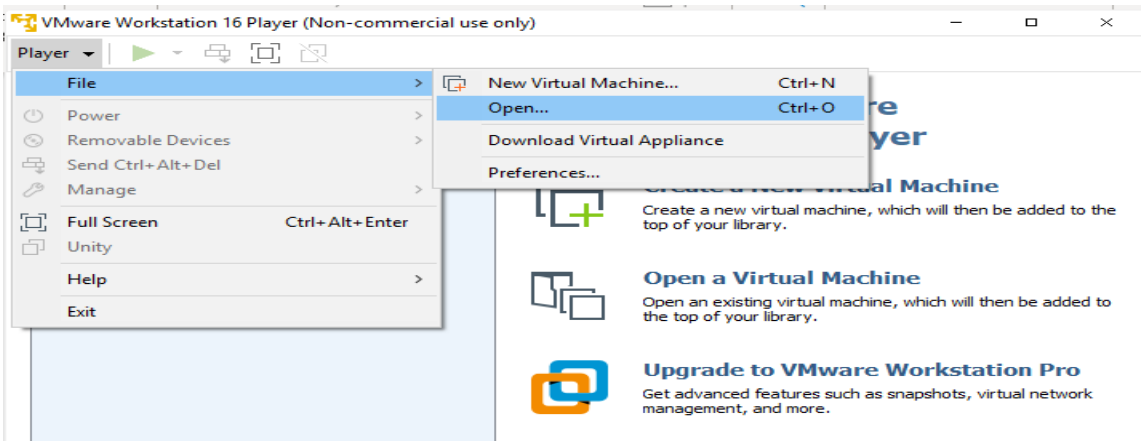


Figure 9:GNS3 VM creation

To install GNS3 VM the OVA file has to be opened in VMWare workstation player, GNS3 GUI should not be running through the process, from player->file->open menu

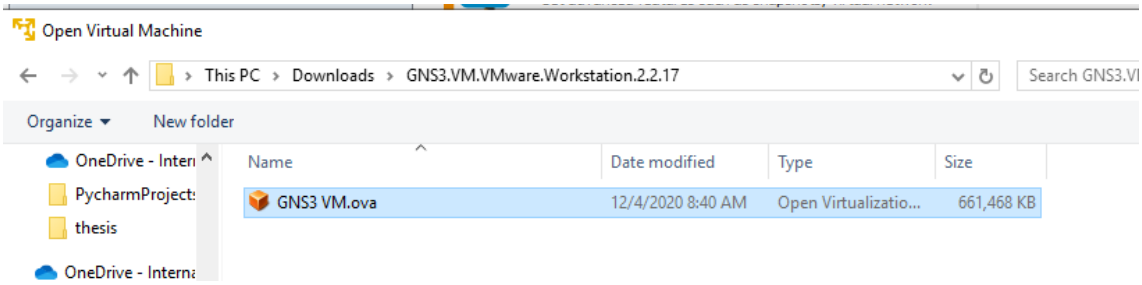


Figure 10:GNS3 VM ova file

It is important to keep the path in the red rectangle at the default.

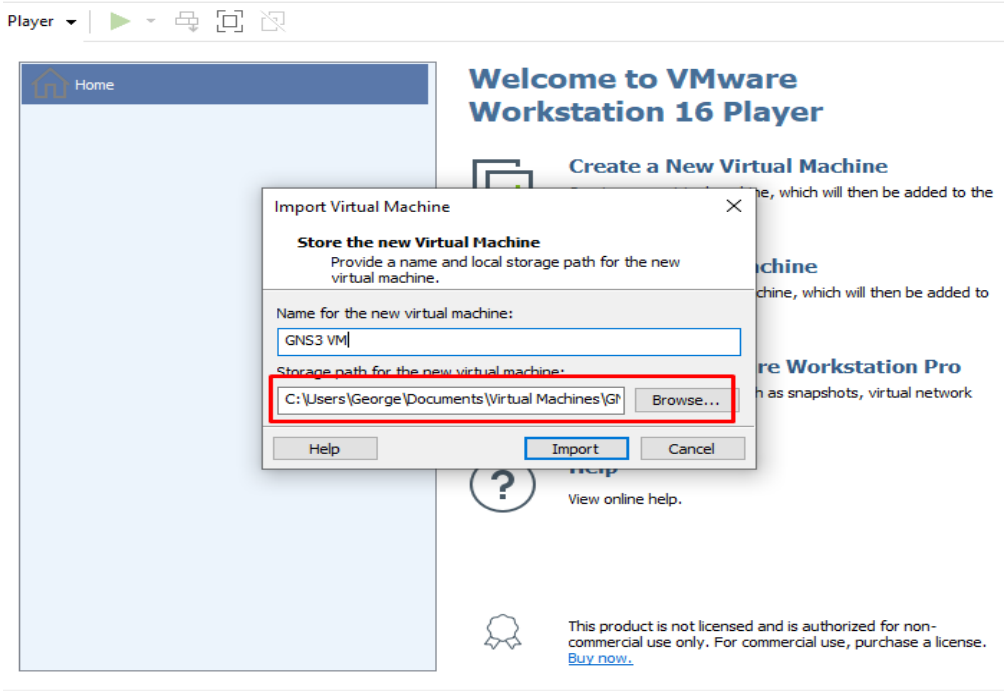


Figure 11:VM installation path

After creating a new blank project (If GNS3 GUI was running it should be restarted) GNS3 VM should be enabled from virtualization engine select which can be found under edit -> preferences menu on VMWare workstation Player.

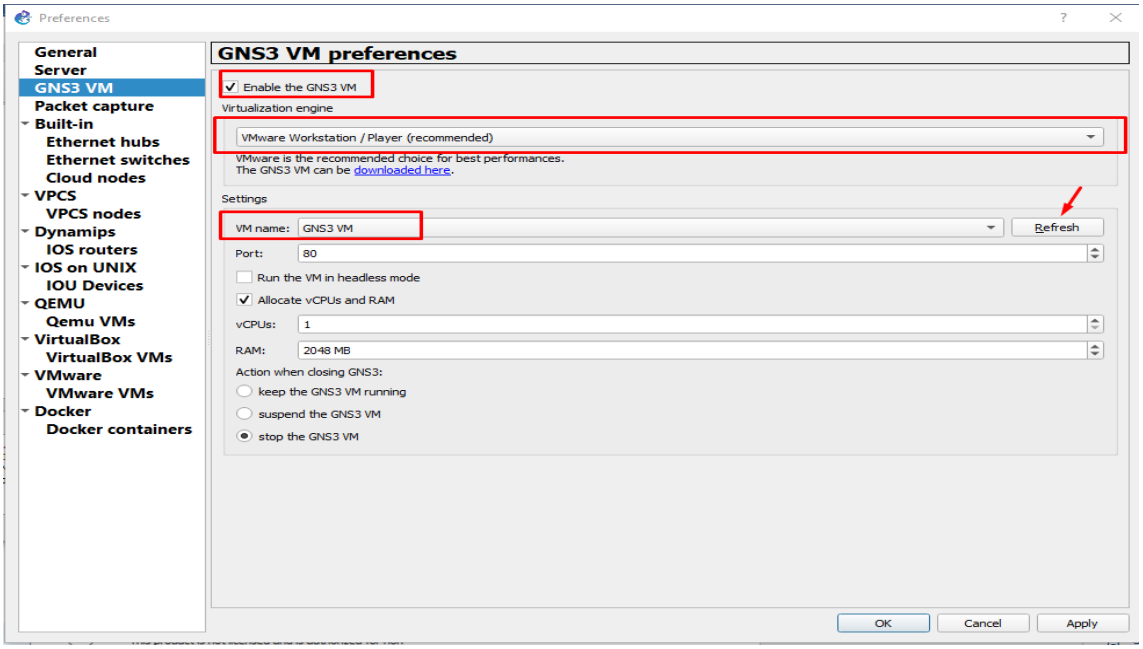


Figure 12:GNS3 VM preferences

On VM name the previously downloaded the OVA GNS3 VM ought to be selected but if nothing is displayed try to restart the GNS3 GUI and go to help->setup wizard. In the wizard choose Run modern IOS and click next until completion. If an error message appears the vix-api probably needs to be installed.

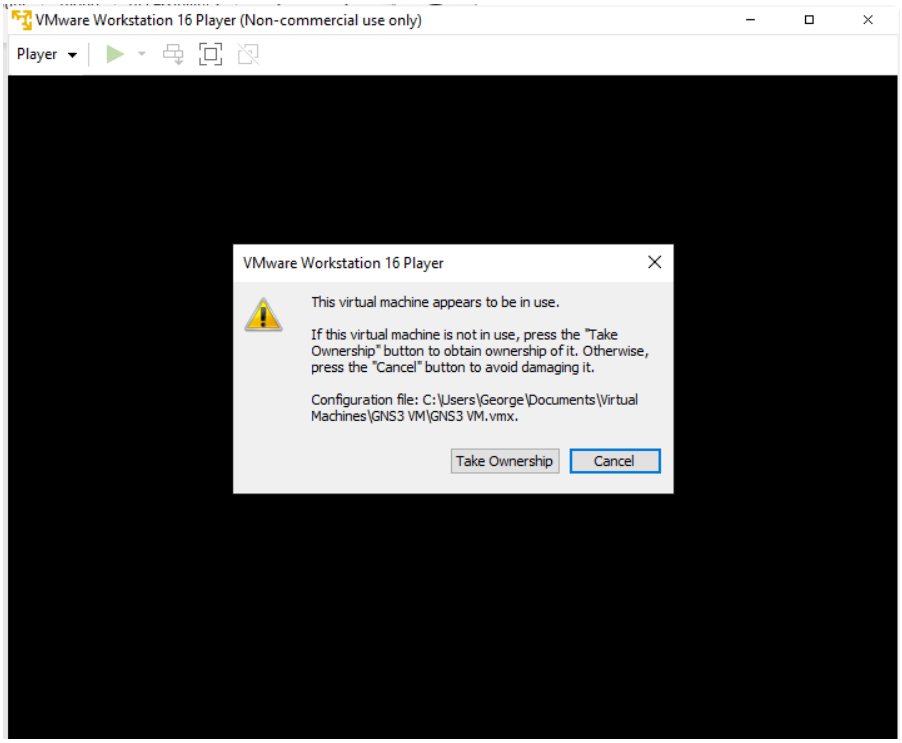


Figure 13:VM ownership

When GNS3 VM boots if the above message appears take Ownership. If at this stage of the process an error message appears restarting both VMWare Player and the GNS3 VM GUI may solve it.

If everything is installed correctly under servers summary there should be now 2 servers, the PC that the GNS3 GUI is installed and the GNS3 VM.

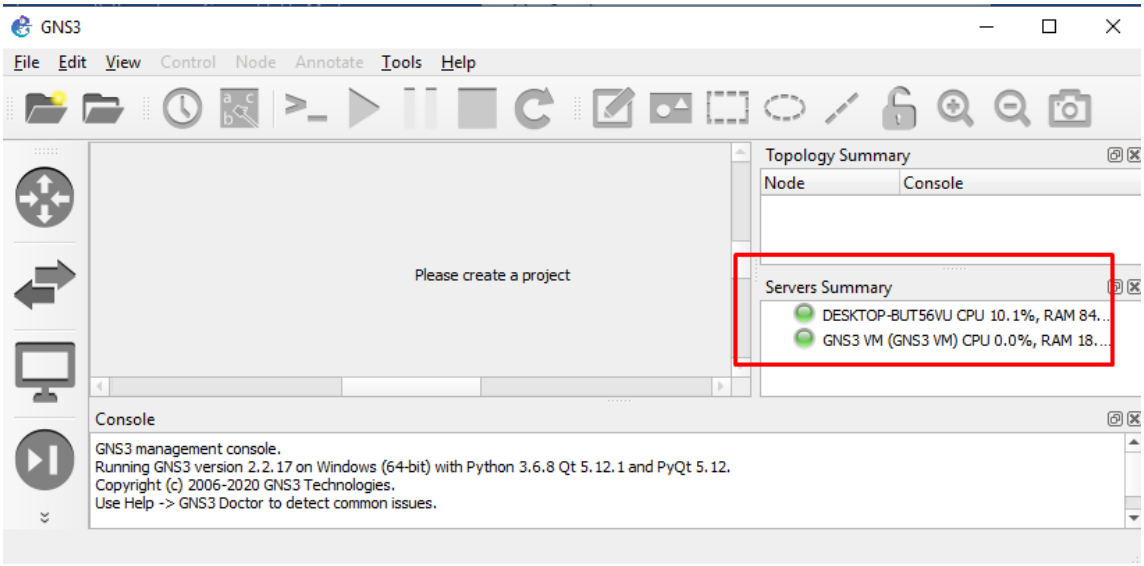


Figure 14:GNS3 running servers

3 Program's design

3.1 Objectives

Through the application's graphical interface, the user will be able to add a device's information (IP, enable password, type, hostname, SSH username, SSH password) which will be stored in a CSV type file. The passwords will be stored in an encrypted format and will be decrypted when used for opening SSH connection with the device.

All the stored devices will be available to select from a list. One or more devices could be selected.

After selecting devices, the user will have to choose which automated procedure will be executed and depending on the procedure some extra parameters must be configured. In

the bottom side of the applications graphical user interface there will be a widow in which the output of the commands will be shown.

3.2 Software and hardware used

- ✓ Python and PyCharm community edition were used for coding and executing the script.
- ✓ GNS3, VMWare player and cisco IOS images, which were provided Department of Computer, Informatics and Telecommunications Engineering Network laboratory, used in order to setup a testing environment and windows 10 laptop from which all the applications were running.

3.3 Development

In the next sections of this chapter important parts of the applications code will be explained in depth, along with alternative methods code parts that has been commented out and explanation of its methods advantages and disadvantages.

3.3.1 Connecting with the devices

As explained in chapter 2 both modules Paramiko and Netmiko can be used to open an SSH connection from the application to the network devices to be configured but Netmiko is less complex than Paramiko. Using Paramiko library to connect to a network device a python object using the **SSHClient** class and the **connect** method has to be created, using keyword arguments to connect to the SSH demon that runs in the networking devices.

Most of the variables should be provided from the user except from the **look_for_keys** and the **allow agent** arguments which ought to be set to false.

Moreover the method **set_missing_host_key_policy** as well as the function **AutoAddPolicy()** are used to automatically accept the servers host key before connecting to the server.

Allow_agent is also has been set to false for security reasons because it keeps decrypted, clear text passwords in ram memory.

After connecting to the network devices SSH daemon a shell object is created by calling the **invoke_shell** method and the **send** method of this object is used to send commands to the network devices.

At the end of the programs code the **get_transport** method is used to see if the connection is open and if it is the **close** method is called to close the connection.

To get the output of the command that is executed the **recv** method of the shell object is used.

The below example of an SSH connection and the code that has to be used to send a command in a network device and get the output of the command can be found commented out in the backup_config function.[10][11][12]

```
host = {'hostname': dev_ip, 'port': '22', 'username': 'ihu',
'password':cipher.decrypt(pwd_to_dec.encode()).decode()}
ssh_client = paramiko.SSHClient()
ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_client.connect(**host, look_for_keys=False, allow_agent=False)
shell = ssh_client.invoke_shell()
shell.send('terminal length 0\n')
shell.send('show run\n')
time.sleep(5)
output = ''
shell.settimeout(3)
   while True:
         try:
             buf = shell.recv(1024).decode('utf-8')
             output = output + buf
          except Exception as e:
              if ssh_client.get_transport().is_active():
                 print('closing connection')
                 ssh_client.close()
              break
```

The above code stores the credential for the connection in the host variable as a dictionary. Then it creates a SSH client using Paramiko library and configures the variable explained above to create an SSH connection. After the connection has been created it creates a shell and uses it to send commands.

The command terminal length must be sent so that the entire command's output will be shown at once if it is long and not having to press a button to show more.

Consequently, the command show running-configuration is send to the network device. In the end the recv command is used to store the output of the command in a variable. The sleep command is used to be sure that the network device has finished the execution of the command before proceeding with reading the output.

The settimeout command and the while loop are used among with the recv command to capture the output of the command. The while loop is needed because the recv command get as argument the number of bytes that has to wait until it returns so in order to get all the output, we have to run the command in a loop and because the recv command always wait's for bytes the settimeout command is used to break out of the loop. Settimeout creates an exception when recv does not get data for 3 seconds.

To achieve the same result with Netmiko the below code has to be executed.

```
netmiko_host = {'host': dev_ip, 'port': '22', 'username': 'ihu',
'password':cipher.decrypt(pwd_to_dec.encode()).decode(), 'de-
vice_type': 'cisco_ios', 'fast_cli': False}
netmiko_connection = Netmiko(**host)
netmiko_connection.enable()
output = netmiko_connection.send_command('show run')
netmiko_connection.disconnect()
```

Although the above example is very basic it is obvious how Netmiko hides the complexity of Paramiko and makes the code more readable and easier to use.

3.3.2 Password encryption

As mentioned on program design passwords for SSH connection and enable passwords are kept on a configuration file. Of course, it would be a huge security flaw if password was kept in a clear text format so with the help of base64 and cryptography libraries the passwords are encrypted, and the program decrypts them when a connection to a device is requested. As an extra security measure, the passwords encryption and decryption key are calculating using a password which the user inputs when a device is added, or a connection is requested.

```
def get_key(master_pwd):
    key_salt = b'(\n\xec\xd9\x1a\xcc\x1e\x86=\xa8\x1b\xd3G\xb9P\xb5'
    kdf = PBKDF2HMAC(algorithm=hashes.SHA256, length=32, salt=key_salt,
    iterations=100000, backend=default_backend())
    enc_key = base64.urlsafe_b64encode(kdf.derive(master_pwd.encode()))
    return enc_key
```

Furthermore, as seen on the code above a salt is used along with the password that the user inputs (master_pwd) to strengthen the encrypted password against brute force in case the user password is weak.

SHA 256 hash function is used and the result from the encryption is encoded with base64. SHA 256 is used instead of other (maybe stronger) hash functions for compatibility reasons.

When a new device is added to the file the encryption key is calculated and it is passes as an argument at the Fernet algorithm.

```
enc_key = get_key(encryption_pwd).decode()
cipher = Fernet(enc_key)
with open('./Config/Devices.csv', 'a+', newline='') as write_obj:
        # Create a writer object from csv module
      csv_writer = writer(write_obj)
        # Add contents of list as last row in the csv file
      new_device_row = [new_device_ip_address.get(),
                        cipher.encrypt(
                        new_device_password.get().encode()).decode(),
                        new_device_selected_type.get(),
                        new_device_name.get(),
                        new_device_ssh_username.get(),
                        cipher.encrypt(
                        new_device_ssh_pass.get().encode()).decode()]
      csv_writer.writerow(new_device_row)
      messagebox.showinfo('Success', 'Device successfully added!')
```

Fernet cipher.encrypt is then called with the users password encoded as an argument and the password is stored in the file in an encrypted format.

IP	Password	Type	Description	ssh_username	ssh_password
192.168.119.201	gAAAAABf69vUN5MKmE	Switch	S1	ihu	gAAAAABf69vUVf2sOdQmqA8hSf1WOW9Puqv_-vVfTIBsT8s5u09xwzt3qQm-SJwc4IVe-OqhZ-yDQIq48JOC-I55KKdIa49Wiw==
192.168.119.202	gAAAAABf7GQfeCpBjspY	Switch	S2	ihu	gAAAAABf7GQfoAQpX_0G-5WWH8X5yLo8wA3QrBUC-zOzGPeZwjZ6tcYaCv-BowWUlgk2Sq2l14Kuja-5OB9ee8P1kGKqWhH8fA==
192.168.119.101	gAAAAABf7ZX7XVYgxAY	Router	R1	ihu	gAAAAABf7ZX7HfiGqnm0z3IGigNUmNvTrFtd8JVoJ8WIiF45Ot2nXbJjiC4AtY6QlATi8-Gbs2n4YQusLwInIJAlWfizkih-bw==
192.168.119.102	gAAAAABf7ZYRmLsJWJo	Router	R2	ihu	gAAAAABf7ZYRx4BaSirPDFeHvYQgicwQ931EhecI95_uTR_mtXhFdN3X9UwThVl6j5eSQktpkBMFuwQKEukUwbUkHLgIFfy6A==
192.168.119.103	gAAAAABf7ZYfdcHDJjMV	Router	R3	ihu	gAAAAABf7ZYfwlKOx8bTs9ZDIFsamrEjqx7NPHqZO22oZroIMoXqV5L9yDciieucNAM9gnIjlcm1lnz4qNoSS-FMNaX1irSXZQ==
192.168.119.104	gAAAAABf7Zc0iK1dlvhv0	Router	R4	ihu	gAAAAABf7Zc0p0a8zLXtTtl9bf1bSvW2N8EmDKiqIMRmtYBBeo9WOVyxsa2p-fzrGN9M-RjWlEdjsbP35hwqeRk4L-xrY6xnkw==

Figure 15:Encrypted passwords

When a connection to a device is requested the application ask's the user's password and pass it as an argument to Fernet cipher.decrypt and stores it in a dictionary that is passed as an argument to the Netmiko connection function

```
device_args = {'device_type': 'cisco_ios', 'ip': dev_ip, 'username':
               'ihu', 'password':
                cipher.decrypt(pwd_to_dec.encode()).decode(),
                              'port': 22, 'verbose': True,
                               'global_delay_factor': 2
                              }
```

This way the real password is only stored in the RAM and great skills are required to retrieve it.

3.3.3 Multi-threading

Without using multithreading when a configuration files has to be restored in multiple devices the program pauses and wait for each configuration to be restored to procced to

the next network device. With multi-threading for every network device a thread is created and stored in a list then outside of the devices loop a new loop runs for all the treads and initiate them.

```
thread_list = list()
for dev_ip in device_to_backup_ip:
    thread_item = threading.Thread(target=restore_config,
                  args=(device_args, restore_filename,))
    thread_list.append(thread_item)
for th in thread_list:
    th.start()
```

Using time module for the restore backup function of the application as the below picture shows without multi-threading each network device needs from twenty to forty seconds to complete the restore and the function needs 136 seconds to complete.

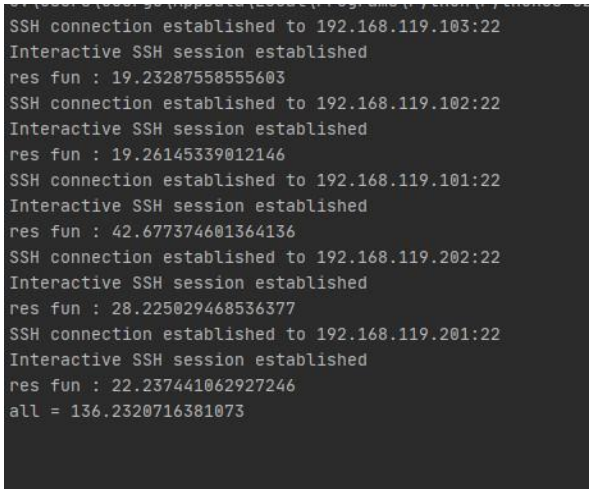


Figure 16:Single thread times

This leads the application to froze for almost two minutes and the user cannot execute another function.

If multi-threading is used every network device needs about the same time but the program continuous to run as it completes threading in almost four seconds.

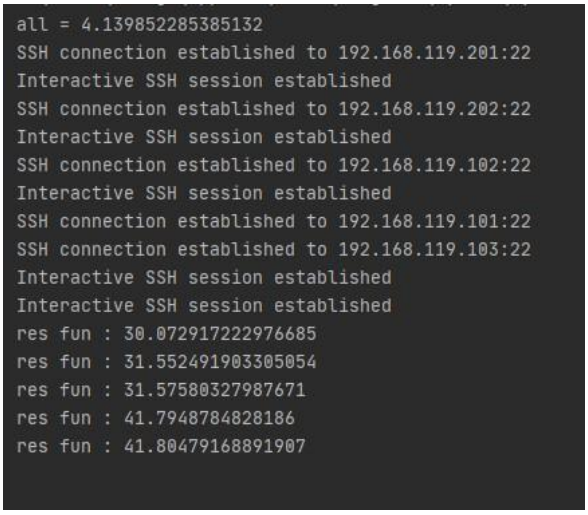


Figure 17:Multi-thread times

4 Application’s Demo

Most of the applications functions are accessible through the main windows that opens with the execution of the code.

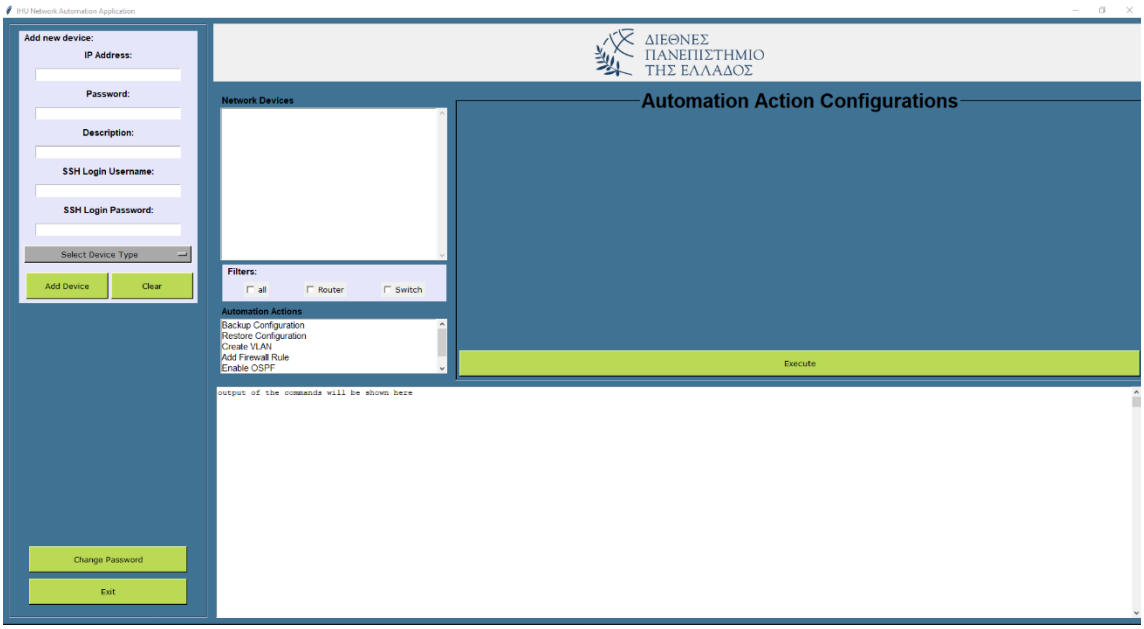


Figure 18:Application’s main screen

4.1 Add device

The first thing that the user has to do when the application is executed for the first time is to add network devices . To add a network device all informations have to be filled(IP address, Enable password, Description, SSH login username and SSH login password), if not a pop-up message appears informing the user.

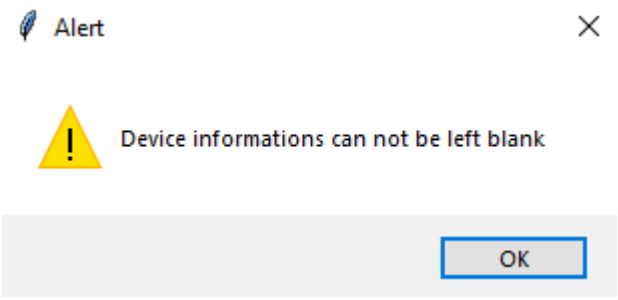


Figure 19:Blank filed error

Another check that takes place is if the IP that the user have entered is a valid IP address if not again a pop-up informs the user that the IP is not valid.

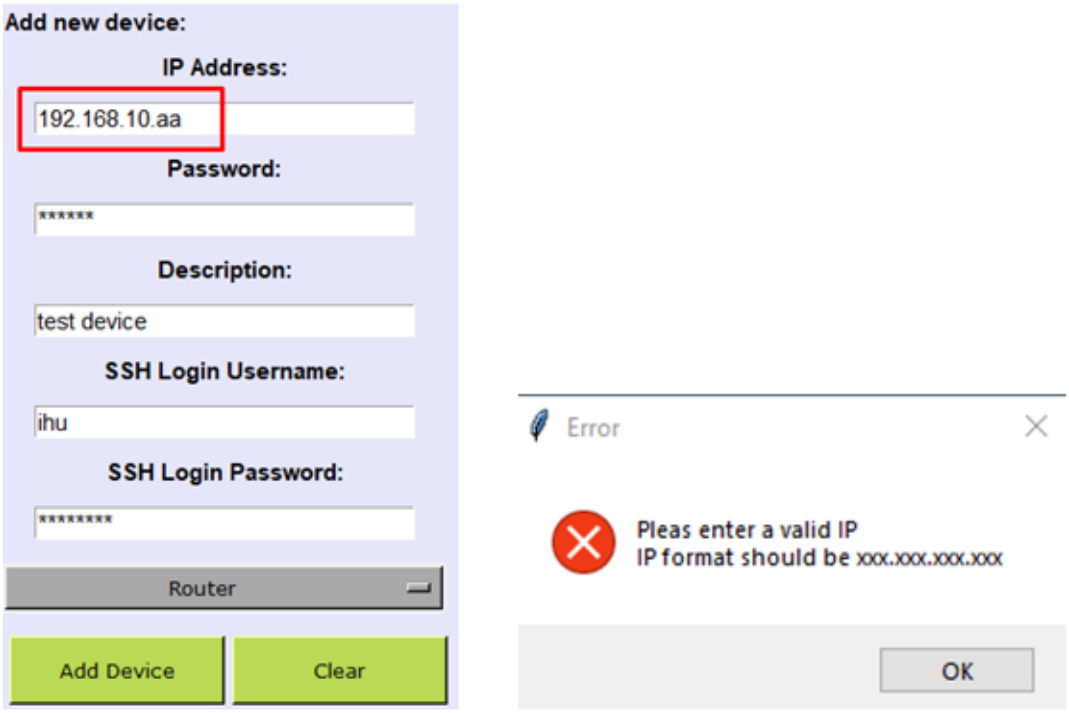


Figure 20:Invalid IP example and error message

When all information's have been entered correctly a window opens asking the user to input a password. This password will be used to encrypt and decrypt the devices password to and from a configuration file. When the user enters the password, a message appears to inform that the devices was added successfully.

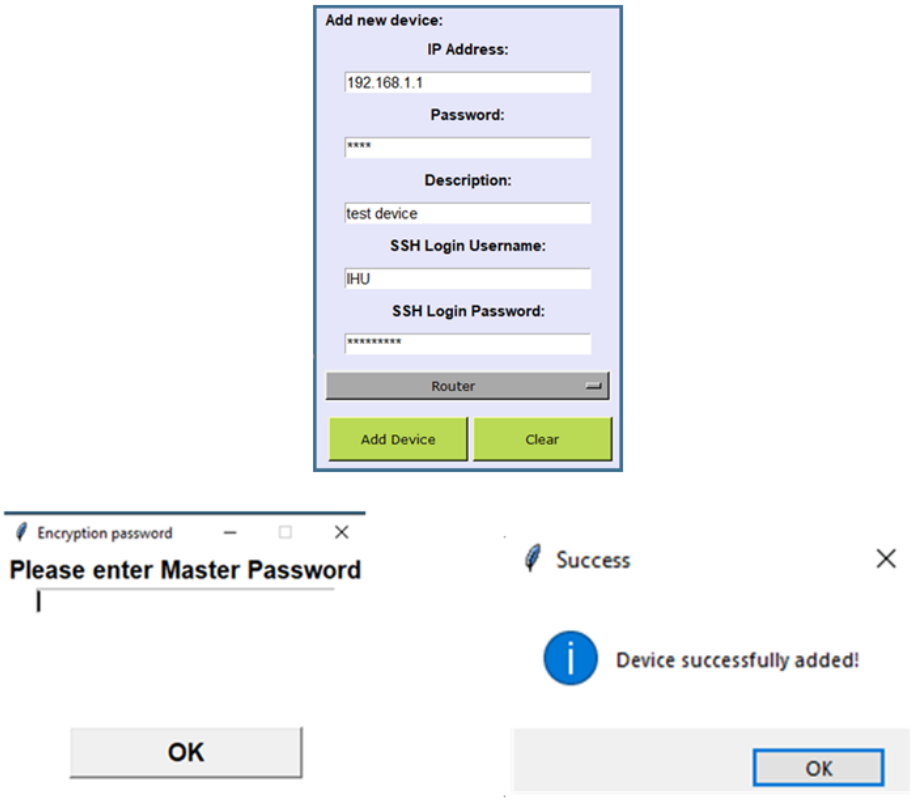


Figure 21:Add device example

Below the add device button there are the change password and exit buttons. The exit button terminates the application, and the change password button changes the password that is used to encrypt and decrypt the devices passwords.

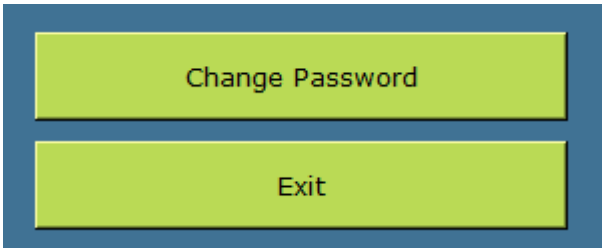


Figure 22:Change password and exit buttons

4.2 Config file

All the devices should be entered using the same password until this password has been changed using this button.

The informations of the device as mentioned above are stored in a configuration file, that can be found at the project directory, with the password in encrypted format.

4.3 Network devices window

The Netword devices window shows all the devices that are stored in the Devices configuration file.

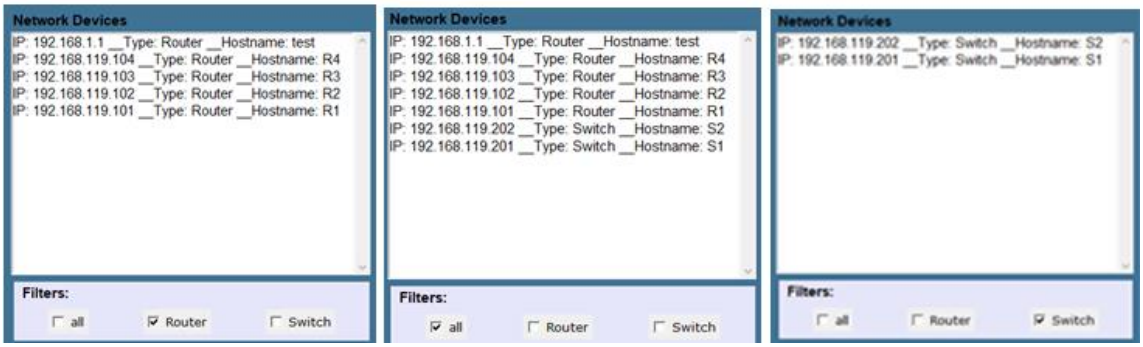


Figure 23:Network devices window

The user can select all devices to be shown, only the routes or only the switches.

In this window the user can select one or multiple devices that he wants to configure.

4.4 Automation actions and configurations

Choosing an action from the automation action list the corresponding options appear at the automation actions configurations window.

4.4.1 Backup configuration

The backup configuration option stores the configuration of the devices that the user chooses at the backup_files directory which is in the projects folder.

The user has 3 options about the format of the files name and should select one of them.

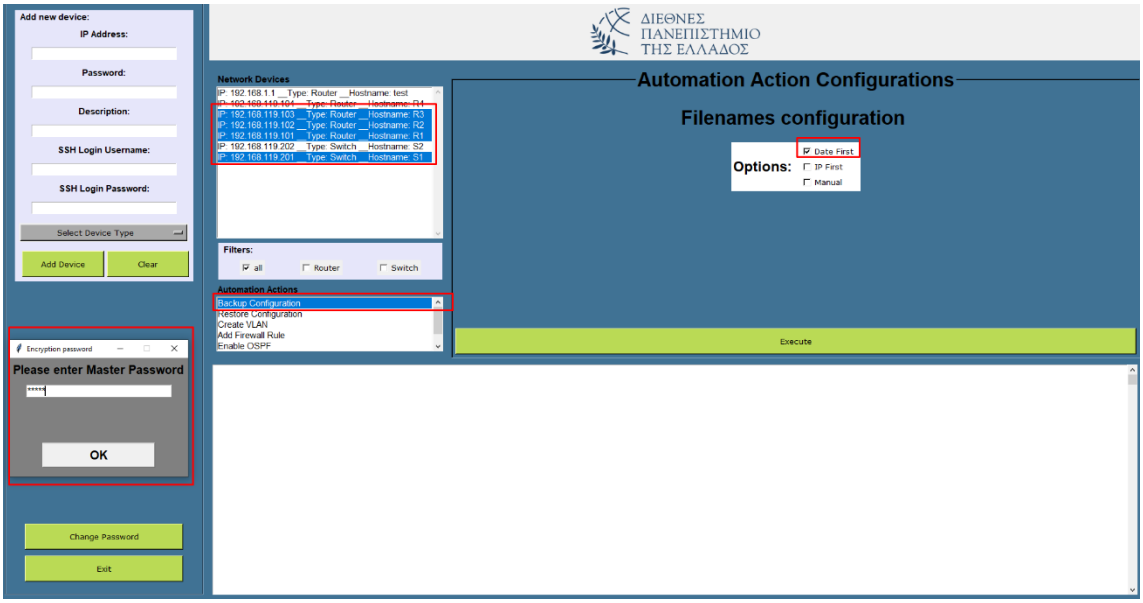


Figure 24:Backup file example

The options are:

- Date first

  The files name starts with the current date with format dd_mm_yyy_hh_mm_ss following by the string backup_of_ and then the IP of the Device

- IP first

  It is the reverse naming format of the Date First option, for example 192.168.10.10_backup_of_10_11_2020_10_15_30

- Manual

  An entry box opens, and the user can manually enter the second half of the files name. The first half is the device's IP.

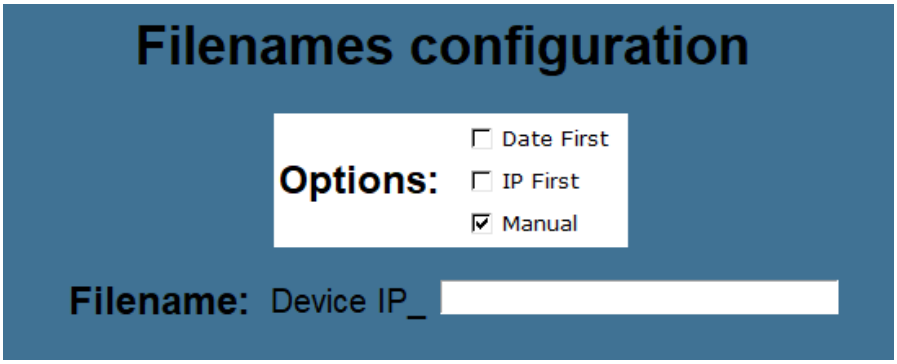


Figure 25:Backup file filename manual option

After finishing the selection for the configuration of the files name the execute button should be pressed and the user will be promed to enter the decryption password.

At the bottom part of the application's main window there is a text area where the user can check if the command completed successfully.

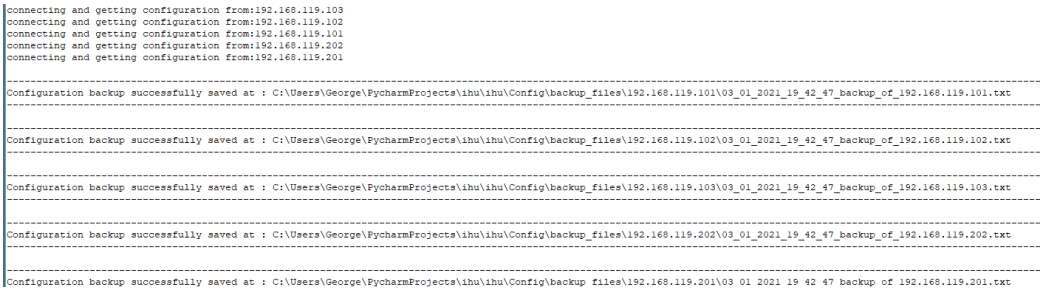


Figure 26:Backup file output example

The files are stored in a separate folder for very devise with name the device's IP.

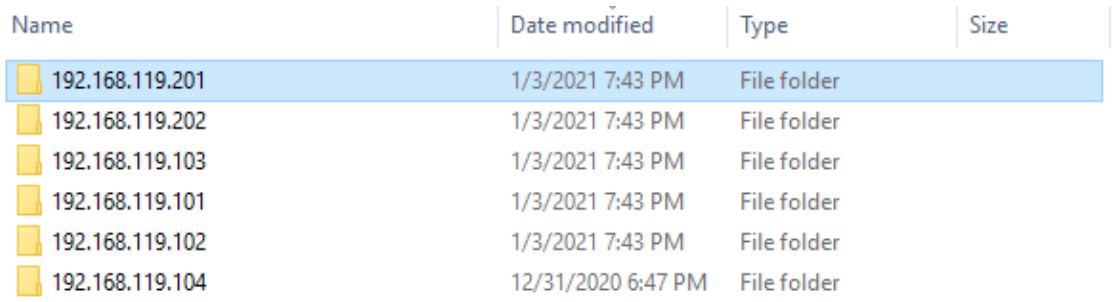


Figure 27:Backup file folder structure

4.4.2 Restore configuration

To restore the configuration file in a device the user has to select one of the two options available.

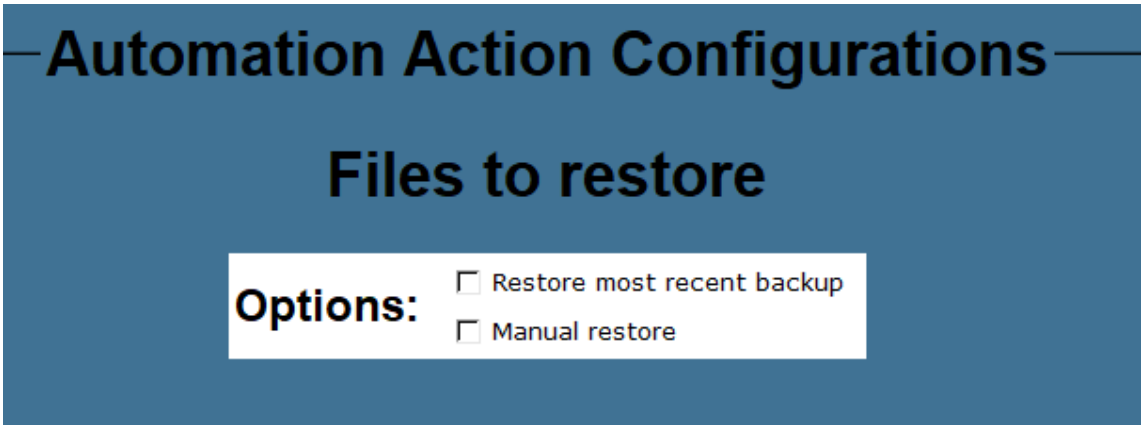


Figure 28:Restore file options

- Restore most recent backup

the application search in each device's that has been selected backup folder, finds the most resent one and restores it to the device.

- Manual restore

  This option is not suitable for a large number of devices because a window will open for each device for the user to choose the file that he wants to restore to the device.

First a popup message informs the user for which device he must choose the file to restore.

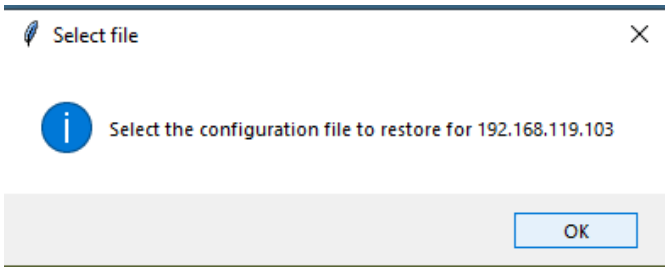


Figure 29:Filename select message

Afterwards an open file dialog window opens to choose the file.

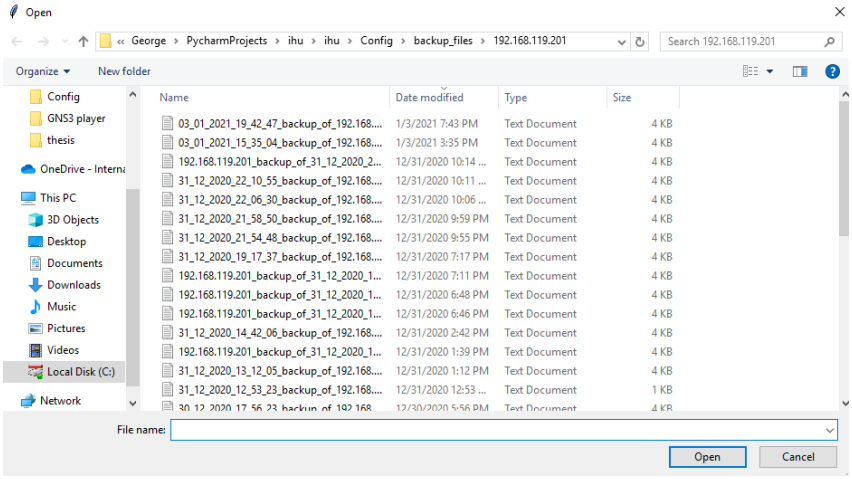


Figure 30:Filename select window

When the execute button is pressed the user is prompt again to insert the password de-cryption password and the results can be found again at the bottom of the application window in the output text window.

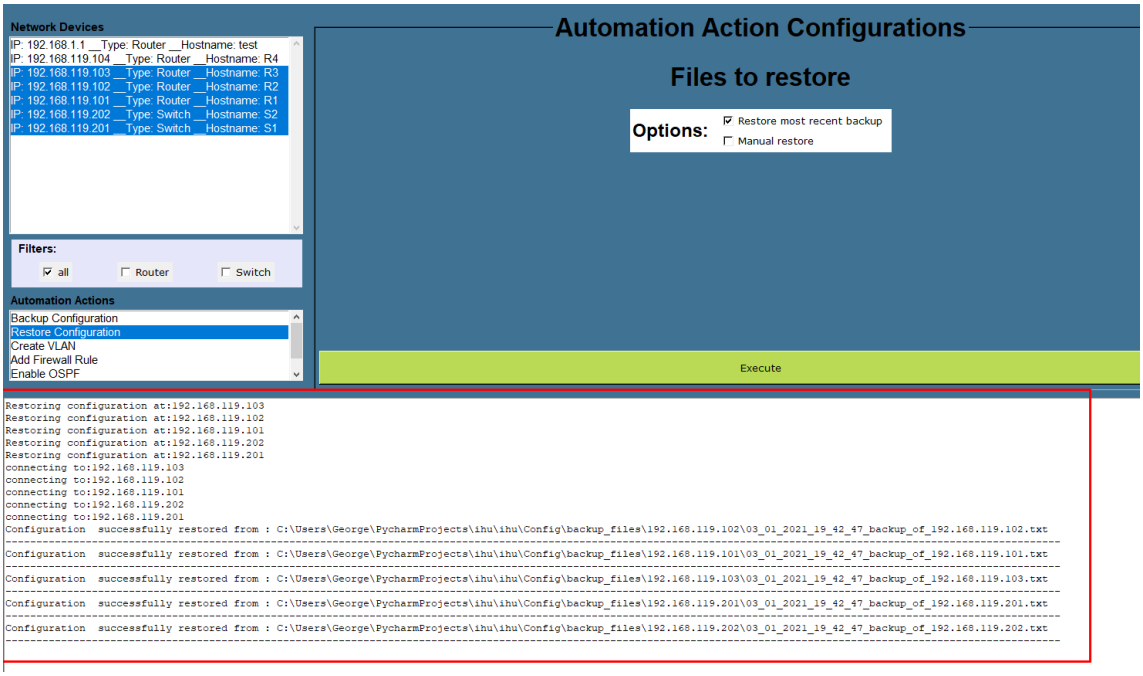


Figure 31:Restore config output

4.4.3 Enable OSPF

To enable OSPF the user should first choose the devices that he wants to enable OSPF, otherwise a warning message appears.

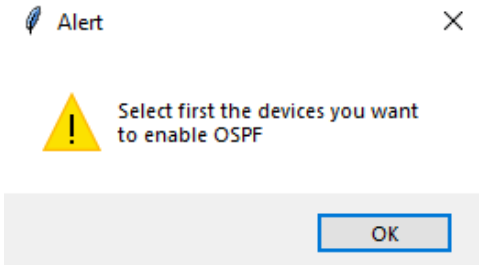


Figure 32:No devise selected error

After selecting them, in the automation configurations section text boxes are generated for each device to complete the variables that a cisco router needs to enable OSPF.

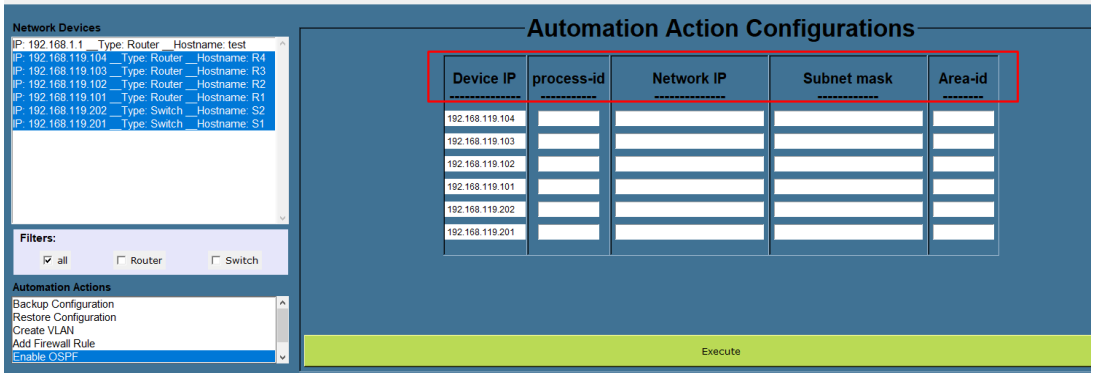


Figure 33:OSPF variables

The variables that user has to complete are: process-id, Network IP (0.0.0.0 for all networks), SubnetMask and Area-ID

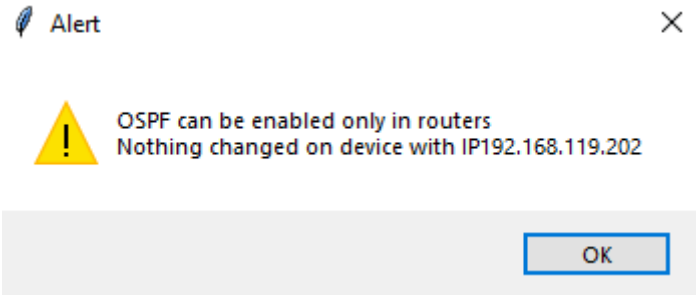


Figure 34:OSPF on switch error

If among the select devices there is a switch a message appears informing the user that OSPF can not be enabled in switches.

In case there is comunication problems with one or more devices an error message is created to inform the user.

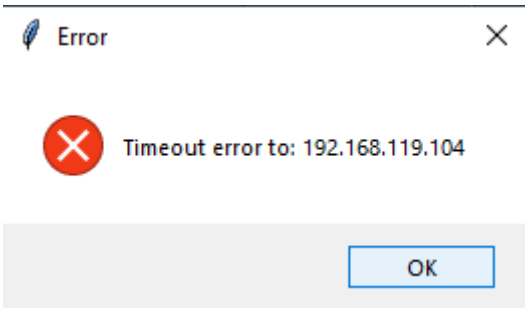


Figure 35:Network error

4.4.4 Add firewall rule

To enter an access list in a network device the user has to select the devices that he wants to enable the access list and then complete all the variables needed.

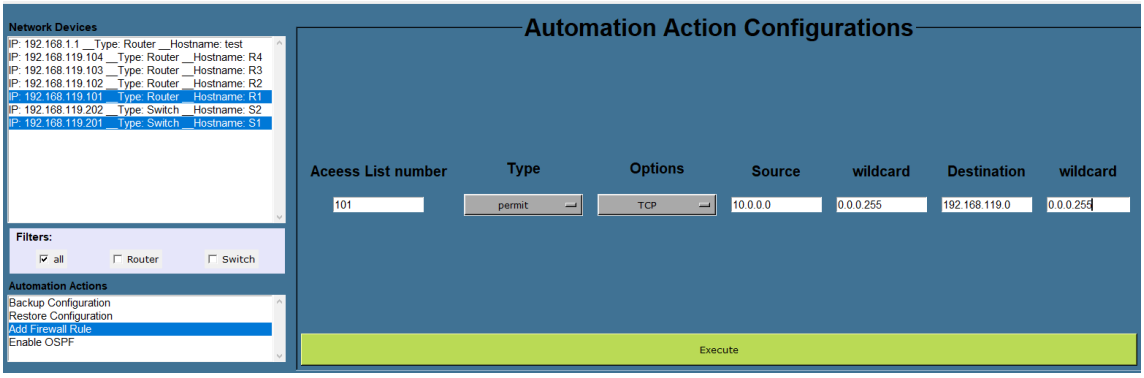


Figure 36:enable access list

when the application has completed the access list creation the user can check the output in the bottom of the application window

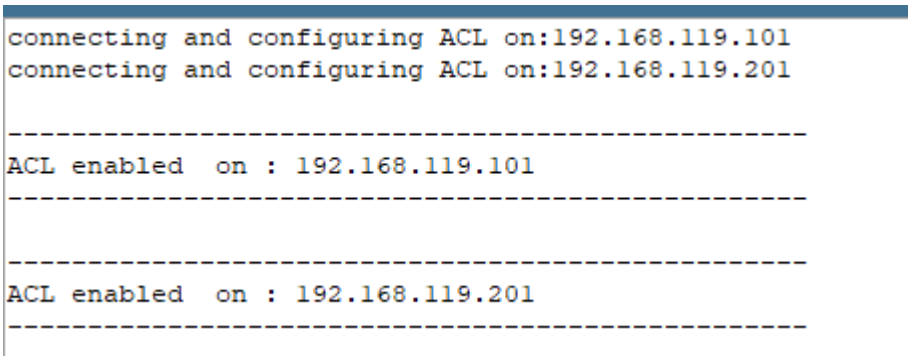
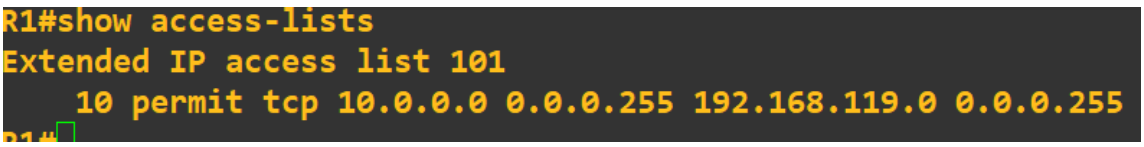


Figure 37:access list successfully enabled

To check if the access list was successfully configured we run to the router a show access-lists command



5 Conclusions

5.1 Future considerations

This application's purpose is to demonstrate the basic idea around network automation with python and as a starting point for an application that can be used in real environment. It needs improvements on the user inputs check and it needs to be customized to fill to the needs of the network environment that it will run. Also because most of the demonstration purpose of the application there are parts of the code that needs to be improved .

5.1.1 Backup and restore from TFTP

The backup of the configuration files is achieved by executing a show running-config command, storing the output in a variable and the writing it in a file.

This technique has a flaw because some configuration commands are not shown in the show running-configuration command, for instance the no shutdown command on an interface.

To restore the device the reverse procedure happens. The commands from the file are sending one by one at a network device.

This can be improved by configuring an TFTP server in the network and sending the command copy running configuration tftp://username:password@ip address and the equivalent of this command for the restoring of the file.

5.1.2 Enable OSPF interfaces

To enable OSPF, user need to input the interfaces IP that he wants OSPF to be enabled. Here a mistake could be made on typing the IP or by entering an IP of an interface that is

not enabled which can be easily avoided by querying its device for active interfaces and list them in a dropbox so the user can securely choose one of them.

5.1.3 Multi-vendor support

The application is written for configuring Cisco devices but in a network, there may be more than one vendor's devices. To be able to automate different vendors devices either the device model has to be given as an input by the user and then code different commands for each vendor and model or NAPALM can be used that has pre-configured commands for different vendors.

NAPALM is an Open-Source Python Library, and it is an acronym for Network Automation and Programmability Abstraction Layer with Multivendor support. It implements a set of functions to interact with different vendors. It provides a Unified API for different vendors such as Cisco, Fortinet, juniper, IBM, and more others. Because there is no API for Cisco IOS devices NAPALM uses Netmiko.

Using NAPALM, it is easier to merge configurations, replace them, configure or even rollback.

As written above for the OSPF Enable function the enabled interfaces should be found on a network device and returned to the user to choose from them in which to enable OSPF. With NAPALM this can be easily achieved with the below code.[20]

```
from napalm import get_network_driver
net_driver = get_network_driver(ios)
device_to_query = [device_list.get(sel_ip) for sel_ip in list(de-
vice_list.curselection())]
for dev in device_to_query:
    host = net_driver(dev, username, password)
    host.open()
    interfaces = host.get_interfaces()
```

The advantage of the NAPALM is that by changing the IOS in this line net_driver = get_network_driver(ios) the same code works for an Arista or any other vendors network device.

Moreover, on a cisco device if Secure Copy (SCP) and archive is enabled the restore function's code can be replaced by the below code.

```
from napalm import get_network_driver
net_driver = get_network_driver(ios)
device_to_query = [device_list.get(sel_ip) for sel_ip in list(de-
vice_list.curselection())]
for dev in device_to_query:
    host = net_driver(dev, username, password)
    host.open()
host.load_replace_candidate(filename='restore_flename.txt')
host.commit_config()
```

This will replace the configuration file with the one that was provided in the filename argument and will create 2 files in the disk that we configured to the device with the archive command, the file that was restored and a rollback file that can be used to restore the configuration that was running before. In addition, NAPALM has the ability using the compare_config command to check if the configuration file has any changes and if show we can commit those changes.

5.1.4 Automated creation of firewall rule

Based on the application's code for the creation of the firewall rule it can be easily created a function that automates the creation of the firewall rule.

For instance, an Intrusion Prevention System (IPS) like snort could be installed.

Snort is an Intrusion Prevention System that identifies malicious network activity and finds packets that match against a set of rules that the user has defined. When a packet matches a rule, an alert is generated.

Snort can be easily installed on windows, Linux or even as a service in a firewall as PfSense.[21][22]

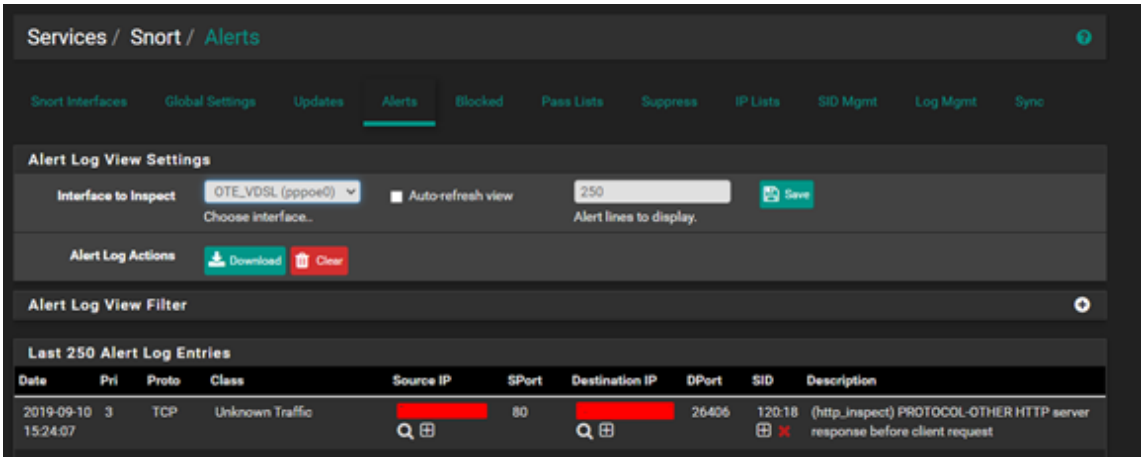


Figure 38:Snort on PFSense

Using spo_csv, a snort plugin that outputs an alert to a csv file, and the code below the application will check in the alert_folder for new files every 10 minutes and when a file is created the code inside if will be executed. The access lists creation code will have as inputs variables from the csv file, for instance the IP address to be blocked.

```
from threading import Timer
import glob
from os.path import splitext

snort_files = (".csv")

def process_files():
    for f in glob.glob('alert_folder/*'):
        if f.splitext()[1] in snort_files:
            # acl creation code here
Timer(10, process_files).start()
```

Anex

```
import threading
from netmiko import ConnectHandler, Netmiko
import time
from tkinter import *
from PIL import ImageTk, Image
from tkinter import messagebox
import os
from functools import partial
import tkinter.font
import csv
from csv import writer
import ipaddress
from datetime import datetime
import base64
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.fernet import Fernet
import glob
from tkinter import filedialog
from netmiko.ssh_exception import NetMikoTimeoutException
from paramiko.ssh_exception import SSHException
from netmiko.ssh_exception import AuthenticationException
from napalm import get_network_driver


def password_change():

    pwd_chng_window = Toplevel(root)
    new_password = StringVar()
    pwd_chng_window.title('Encryption password')
    xpos = 10
    ypos = 600
    wgeo = '300x200+' + str(xpos) + '+' + str(ypos)
    pwd_chng_window.geometry(wgeo)
    pwd_chng_window.configure(background='Grey')
    pwd_chng_window.resizable(False, False)
    pwd_chng_window.grab_set()
    Label(pwd_chng_window, text='New Password', font='Arial 15 bold ',
bg='Grey').pack(side=TOP, pady=5)
    new_password_entry = Entry(pwd_chng_window, textvariable=new_pass-
word, show="*",
                               width=30, font='Helvetica 11')
    new_password_entry.pack(side=TOP, pady=5)
    new_password_entry.focus()
    pwd_chng_ok_btn = Button(pwd_chng_window, text=" OK ", width=15,
height=1,
                             font='Arial 15 bold', com-
mand=pwd_chng_window.destroy)
    pwd_chng_ok_btn.pack(side=BOTTOM, pady=15)
    pwd_chng_window.attributes('-topmost', 1)  # Raising root above
all other windows
```

```
    root.wait_window(pwd_chng_window)
    dec_key = create_key()
    cipher = Fernet(dec_key)
    enc_key = get_key(new_password.get()).decode()
    enc_cipher = Fernet(enc_key)
    new_pass_rows = [['IP', 'Password', 'Type', 'Description',
'ssh_username', 'ssh_password']]
    with open('./Config/Devices.csv', 'rt') as csv_f:
        reader = csv.reader(csv_f, delimiter=',')
        for row in reader:
            if row[1] != 'Password':
                enable_to_dec = row[1]
                ssh_to_dec = row[5]
                dec_enable = cipher.decrypt(enable_to_dec.en-
code()).decode()
                dec_ssh = cipher.decrypt(ssh_to_dec.encode()).decode()
                row[1] = enc_cipher.encrypt(dec_enable.encode()).de-
code()
                row[5] = enc_cipher.encrypt(dec_ssh.encode()).decode()
                new_pass_rows.append(row)
    with open('./Config/Devices.csv', 'w', newline='') as csv_w:
        new_pass_writer = csv.writer(csv_w, delimiter=',')
        for device_row in new_pass_rows:
            new_pass_writer.writerow(device_row)

    messagebox.showinfo('Success', 'Password changed successfully!')


def ip_entered(ip):
    try:
        return ipaddress.ip_address(ip)
    except ValueError:
        messagebox.showerror('Error', 'Pleas enter a valid IP \n' +
'IP format should be xxx.xxx.xxx.xxx')
        return 0


def get_key(master_pwd):
    key_salt = b'(\n\xec\xd9\x1a\xcc\x1e\x86=\xa8\x1b\xd3G\xb9P\xb5'
    kdf = PBKDF2HMAC(algorithm=hashes.SHA256, length=32,
salt=key_salt, iterations=100000, backend=default_backend())
    enc_key = base64.urlsafe_b64encode(kdf.derive(master_pwd.en-
code()))
    return enc_key


def get_config_name(selected_value):
    if automations_list.get(automations_list.curselection()) ==
'Backup Configuration':
        for child in automation_conf_frame.winfo_children():
            if str(child) != '.!labelframe2.!button':
                child.pack_forget()
        backup_name_lbl.pack(side=TOP, pady=20)
        automation_filter_frame.pack(side=TOP)
        cbtn_date_ip.pack(side=TOP, anchor='w')
        cbtn_ip_date.pack(side=TOP, anchor='w')
        cbtn_manual.pack(side=TOP, anchor='w')
    elif automations_list.get(automations_list.curselection()) == 'Re-
store Configuration':
        for child in automation_conf_frame.winfo_children():
            if str(child) != '.!labelframe2.!button':
```

```
                child.pack_forget()
        restore_name_lbl.pack(side=TOP, pady=20)
        automation_restore_options_frame.pack(side=TOP)
        cbtn_last.pack(side=TOP, anchor='w')
        cbtn_manual_ask.pack(side=TOP, anchor='w')
    elif automations_list.get(automations_list.curselection()) == 'En-
able OSPF':
        device_to_enable = [device_list.get(sel_ip) for sel_ip in
list(device_list.curselection())]
        if len(device_to_enable) == 0:
            messagebox.showwarning('Alert', 'Select first the devices
you want \nto enable OSPF')
            automations_list.selection_clear(0, 'end')
        else:
            for child in automation_conf_frame.winfo_children():
                if str(child) != '.!labelframe2.!button':
                    child.pack_forget()
            for child in ospf_main_frm.winfo_children():
                child.pack_forget()
            for child in ospf_device_frm.winfo_children():
                child.pack_forget()
            for child in ospf_process_id_frm.winfo_children():
                child.pack_forget()
            for child in ospf_ip_frm.winfo_children():
                child.pack_forget()
            for child in ospf_mask_frm.winfo_children():
                child.pack_forget()
            for child in ospf_area_id_frm.winfo_children():
                child.pack_forget()
            full_list = [device_list.get(sel_ip) for sel_ip in
list(device_list.curselection())]
            name_list = []
            for backup_dev in full_list:
                name_list.append(backup_dev[1])

            ospf_main_frm.pack(side=TOP)
            ospf_device_frm.pack(side=LEFT, anchor='n')
            ospf_process_id_frm.pack(side=LEFT, anchor='n')
            ospf_ip_frm.pack(side=LEFT, anchor='n')
            ospf_mask_frm.pack(side=LEFT, anchor='n')
            ospf_area_id_frm.pack(side=LEFT, anchor='n')

            ospf_device_lbl.pack(side=TOP)
            ospf_process_id_lbl.pack(side=TOP)
            ospf_ip_lbl.pack(side=TOP)
            ospf_mask_lbl.pack(side=TOP)
            ospf_area_id_lbl.pack(side=TOP)
            global entry
            entry = {}
            for i, n in enumerate(name_list):
                ospf_device_name = str(name_list[i])
                Label(ospf_device_frm, text=ospf_device_name,
font=('Aria', 9), fg='black', width=15, anchor='w',
                      bg='white').pack(side=TOP, pady=5)
                ospf_pid = Entry(ospf_process_id_frm, width=10,
font='Helvetica 11')
                ospf_pid.pack(side=TOP, pady=5)
                ospf_ip = Entry(ospf_ip_frm, width=25, font='Helvetica
11')
                ospf_ip.pack(side=TOP, pady=5)
```

```
                ospf_mask = Entry(ospf_mask_frm, width=25, font='Hel-
vetica 11')
                ospf_mask.pack(side=TOP, pady=5)
                ospf_area = Entry(ospf_area_id_frm, width=10,
font='Helvetica 11')
                ospf_area.pack(side=TOP, pady=5)
                entry[n] = [ospf_pid, ospf_ip, ospf_mask, ospf_area]
    elif automations_list.get(automations_list.curselection()) == 'Add
Firewall Rule':
        for child in automation_conf_frame.winfo_children():
            if str(child) != '.!labelframe2.!button':
                child.pack_forget()
        acl_number_frm.pack(side=LEFT, padx=5, pady=10)
        acl_type_frm.pack(side=LEFT, padx=5, pady=10)
        acl_options_frm.pack(side=LEFT, padx=5, pady=10)
        acl_source_frm.pack(side=LEFT, padx=5, pady=10)
        acl_swildcard_frm.pack(side=LEFT, padx=5, pady=10)
        acl_des_frm.pack(side=LEFT, padx=5, pady=10)
        acl_dwildcard_frm.pack(side=LEFT, padx=5, pady=10)

        acl_number_lbl.pack(side=TOP, anchor='n', padx=5, pady=10)
        acl_type_lbl.pack(side=TOP, anchor='n', padx=5, pady=10)
        acl_protocol_lbl.pack(side=TOP, anchor='n', padx=5, pady=10)
        acl_source_lbl.pack(side=TOP, anchor='n', padx=5, pady=10)
        acl_swildcard_lbl.pack(side=TOP, anchor='n', padx=5, pady=10)
        acl_des_lbl.pack(side=TOP, anchor='n', padx=5, pady=10)
        acl_dwildcard_lbl.pack(side=TOP, anchor='n', padx=5, pady=10)

        acl_nmb_entry.pack(side=TOP, anchor='n', padx=5, pady=10)
        acl_types.pack(side=TOP, anchor='n', padx=5, pady=10)
        acl_options.pack(side=TOP, anchor='n', padx=5, pady=10)
        source_ip_entry.pack(side=TOP, anchor='n', padx=5, pady=10)
        source_wildcard_entry.pack(side=TOP, anchor='n', padx=5,
pady=10)
        destination_ip_entry.pack(side=TOP, anchor='n', padx=5,
pady=10)
        destination_wildcard_entry.pack(side=TOP, anchor='n', padx=5,
pady=10)


def create_key():
    pwd_window = Toplevel(root)
    master_password = StringVar()
    pwd_window.title('Encryption password')
    xpos = 10
    ypos = 600
    wgeo = '300x200+' + str(xpos) + '+' + str(ypos)
    pwd_window.geometry(wgeo)
    pwd_window.configure(background='Grey')
    pwd_window.resizable(False, False)
    pwd_window.grab_set()
    Label(pwd_window, text='Please enter Master Password', font='Arial
15 bold ', bg='Grey').pack(side=TOP, pady=5)
    master_password_entry = Entry(pwd_window, textvariable=mas-
ter_password, show="*",
                                  width=30, font='Helvetica 11')
    master_password_entry.pack(side=TOP, pady=5)
    master_password_entry.focus()
    pwd_ok_btn = Button(pwd_window, text=" OK ", width=15, height=1,
                        font='Arial 15 bold', command=pwd_window.de-
stroy)
```

```
    pwd_ok_btn.pack(side=BOTTOM, pady=15)
    pwd_window.attributes('-topmost', 1)  # Raising root above all
other windows
    root.wait_window(pwd_window)
    decryption_pwd = master_password.get()
    ret_dec_key = get_key(decryption_pwd).decode()
    return ret_dec_key


def backup_config(host, path):
    output_description.insert(END, f'connecting and getting configura-
tion from:{host["host"]}\n')
    output_description.update_idletasks()
    try:
        netmiko_connection = Netmiko(**host)
    except NetMikoTimeoutException:
        messagebox.showerror('Error', ('Timeout error to: ' +
host["host"]))
    except AuthenticationException:
        messagebox.showerror('Error', 'Authentication failure error
to: ' + host["host"])
    except SSHException:
        messagebox.showerror('Error', 'SSH Error. Check if SSH is ena-
bled ' + host["host"])
    except EOFError:
        messagebox.showerror('Error', 'End of file while attempting
device' + host["host"])
    except Exception as unknown_error:
        messagebox.showerror('Error', 'Unknown error to: ' +
host["host"] + str(unknown_error))
    netmiko_connection.enable()
    output = netmiko_connection.send_command('show run')

    output = output[output.find('version'):]
    output = output.rsplit("end", 1)[0]

    with open(path, "w") as text_file:
        text_file.write(output)
    output_description.insert(END, '\n' + ('-' * (len(path) + 50)) +
'\n')
    output_description.insert(END, 'Configuration backup successfully
saved at : ' + path)
    output_description.insert(END, '\n' + ('-' * (len(path) + 50)) +
'\n')
    output_description.update_idletasks()
    netmiko_connection.disconnect()


def restore_config(r_host, r_filename):
    s1 = time.time()
    output_description.insert(END, f'Restoring configuration
at:{r_host["ip"]}\n')
    output_description.update_idletasks()
    net_connection = ConnectHandler(**r_host)
    output_description.insert(END, f'connecting to:{r_host["ip"]}\n')
    output_description.update_idletasks()
    net_connection.send_config_set(r_filename, cmd_verify=False)
    output_description.insert(END, 'Configuration  successfully re-
stored from : ' + r_filename)
    output_description.insert(END, '\n-------------------------------
-------------------------------'
```

```
                                    '---------------------------------
--------------------------------'
                                    '---------------------------------
--------\n')
    output_description.update_idletasks()
    net_connection.send_command('wr')
    net_connection.disconnect()
    e1 = time.time()
    print('res fun :', e1 - s1)


def ospf_config(host, pid, ip_for_ospf, mask_for_ospf, area):
    output_description.insert(END, f'connecting and enabling OSPF
on:{host["ip"]}\n')
    output_description.update_idletasks()
    # try:
    ospf_connection = Netmiko(**host)
    # except NetMikoTimeoutException:
    #     messagebox.showerror('Error', ('Timeout error to: ' +
host["ip"]))
    # except AuthenticationException:
    #     messagebox.showerror('Error', 'Authentication failure error
to: ' + host["ip"])
    # except SSHException:
    #     messagebox.showerror('Error', 'SSH Error. Check if SSH is
enabled ' + host["ip"])
    # except EOFError:
    #     messagebox.showerror('Error', 'End of file while attempting
device' + host["ip"])
    # except Exception as unknown_error:
    #     messagebox.showerror('Error', 'Unknown error to: ' +
host["ip"] + str(unknown_error))
    ospf_connection.enable()

    ospf_commands = ['enable', 'configure terminal', 'router ospf ' +
pid, 'network ' + ip_for_ospf + ' ' +
                     mask_for_ospf + ' area ' + area, 'end']

    output = ospf_connection.send_config_set(ospf_commands)
    print(output)
    output_description.insert(END, '\n' + ('-' * (len(ip_for_ospf) +
50)) + '\n')
    output_description.insert(END, 'OSPF enabled at : ' + host["ip"] +
' for ' + ip_for_ospf + ' network')
    output_description.insert(END, '\n' + ('-' * (len(ip_for_ospf) +
50)) + '\n')
    output_description.update_idletasks()
    ospf_connection.disconnect()

def acl_config(ac_host, ac_num, ac_action, ac_prot, ac_sip, ac_sw,
ac_dip, ac_dw):
    output_description.insert(END, f'connecting and configuring ACL
on:{ac_host["ip"]}\n')
    output_description.update_idletasks()
    # try:
    acl_connection = Netmiko(**ac_host)
    # except NetMikoTimeoutException:
    #     messagebox.showerror('Error', ('Timeout error to: ' +
host["ip"]))
    # except AuthenticationException:
```

```
    #     messagebox.showerror('Error', 'Authentication failure error
to: ' + host["ip"])
    # except SSHException:
    #     messagebox.showerror('Error', 'SSH Error. Check if SSH is
enabled ' + host["ip"])
    # except EOFError:
    #     messagebox.showerror('Error', 'End of file while attempting
device' + host["ip"])
    # except Exception as unknown_error:
    #     messagebox.showerror('Error', 'Unknown error to: ' +
host["ip"] + str(unknown_error))
    acl_connection.enable()

    acl_commands = ['access-list ' + ac_num + ' ' + ac_action + ' ' +
ac_prot + ' ' +
                    ac_sip + ' ' + ac_sw + ' ' + ac_dip + ' ' + ac_dw
]
    output = acl_connection.send_config_set(acl_commands)
    output_description.insert(END, '\n' + ('-' *  50) + '\n')
    output_description.insert(END, 'ACL enabled  on : ' +
ac_host["ip"])
    output_description.insert(END, '\n' + ('-' *  50) + '\n')
    output_description.update_idletasks()
    acl_connection.disconnect()
    print(output)


def execute_automations():
    output_description.delete('1.0', END)
    if automations_list.get(automations_list.curselection()) ==
'Backup Configuration':
        if c_date_ip.get() == 1 or c_ip_date.get() == 1 or c_man-
ual.get() == 1:
            if c_manual.get() == 1 and manual_config_name.get() == '':
                messagebox.showerror('Error', 'Filename can not be
empty')
            else:
                device_to_backup = [device_list.get(sel_ip) for sel_ip
in list(device_list.curselection())]
                device_to_backup_ip = []
                for backup_dev in device_to_backup:
                    device_to_backup_ip.append(backup_dev[1])
                dec_key = create_key()
                cipher = Fernet(dec_key)
                thread_list = list()
                for dev_ip in device_to_backup_ip:
                    with open('./Config/Devices.csv', 'rt') as csv_f:
                        reader = csv.reader(csv_f, delimiter=',')
                        for row in reader:
                            if row[0] == dev_ip:
                                pwd_to_dec = row[5]

                    # host_dev = {'hostname': dev_ip, 'port': '22',
'username': 'ihu', 'password':
                    #             cipher.decrypt(pwd_to_dec.en-
code()).decode()}
                    netmiko_host = {'host': dev_ip, 'port': '22',
'username': 'ihu', 'password':
                                    cipher.decrypt(pwd_to_dec.en-
code()).decode(), 'device_type': 'cisco_ios',
                                    'fast_cli': False}
```

```
                    b_working_folder = os.getcwd()
                    if c_date_ip.get() == 1:
                        if not os.path.exists(b_working_folder +
'\\Config\\backup_files\\' + netmiko_host['host']):
                            os.makedirs(b_working_folder + '\\Con-
fig\\backup_files\\' + netmiko_host['host'])
                        backup_filename = str(
                            b_working_folder + '\\Con-
fig\\backup_files\\' + netmiko_host['host'] + '\\' +

datetime.now().strftime('%d_%m_%Y_%H_%M_%S') + '_' + 'backup_of_' +
                            netmiko_host['host'] + '.txt')
                    elif c_ip_date.get() == 1:
                        if not os.path.exists(b_working_folder +
'\\Config\\backup_files\\' + netmiko_host['host']):
                            os.makedirs(b_working_folder + '\\Con-
fig\\backup_files\\' + netmiko_host['host'])
                        backup_filename = str(
                            b_working_folder + '\\Con-
fig\\backup_files\\' + netmiko_host['host'] + '\\' +
                            netmiko_host['host'] + '_' + 'backup_of_'
+

datetime.now().strftime('%d_%m_%Y_%H_%M_%S') + '.txt')
                    else:
                        if not os.path.exists(b_working_folder +
'\\Config\\backup_files\\' + netmiko_host['host']):
                            os.makedirs(b_working_folder + '\\Con-
fig\\backup_files\\' + netmiko_host['host'])
                        backup_filename = str(
                            b_working_folder + '\\Con-
fig\\backup_files\\' + netmiko_host['host'] + '\\' +
                            netmiko_host['host'] + '_' + manual_con-
fig_name.get() + '.txt')

                    thread_item = threading.Thread(target=backup_con-
fig, args=(netmiko_host, backup_filename,))
                    thread_list.append(thread_item)
                for th in thread_list:
                    th.start()

        else:
            messagebox.showerror('Error', 'Select on of the below op-
tions\n Date First\n IP First\n Manual')
            manual_entry.delete(0, 'end')

    elif automations_list.get(automations_list.curselection()) == 'Re-
store Configuration':
        start = time.time()

        if c_last.get() == 1 or c_manual_ask.get() == 1:
            device_to_backup = [device_list.get(sel_ip) for sel_ip in
list(device_list.curselection())]
            device_to_backup_ip = []
            for backup_dev in device_to_backup:
                device_to_backup_ip.append(backup_dev[1])
            dec_key = create_key()
            cipher = Fernet(dec_key)
            thread_list = list()
            for dev_ip in device_to_backup_ip:
                with open('./Config/Devices.csv', 'rt') as csv_f:
```

```
                    reader = csv.reader(csv_f, delimiter=',')
                    for row in reader:
                        if row[0] == dev_ip:
                            pwd_to_dec = row[5]
                device_args = {'device_type': 'cisco_ios', 'ip':
dev_ip, 'username': 'ihu', 'password':
                               cipher.decrypt(pwd_to_dec.encode()).de-
code(), 'port': 22, 'verbose': True,
                               'global_delay_factor': 2}
                working_folder = os.getcwd()
                if c_last.get() == 1:
                    device_folder = glob.glob(working_folder + '\\Con-
fig\\backup_files\\' + device_args['ip'] + '\\*')
                    restore_filename = max(device_folder,
key=os.path.getctime)

                else:
                    messagebox.showinfo('Select file', 'Select the
configuration file to restore for ' +
                                        device_args["ip"])
                    restore_filename = filedialog.askopenfilename()
                thread_item = threading.Thread(target=restore_config,
args=(device_args, restore_filename,))
                thread_list.append(thread_item)

            for th in thread_list:
                th.start()

        else:
            messagebox.showerror('Error', 'Select on of the below op-
tions\n '
                                          'Restore most recent backup
\n Manual restore')
        end = time.time()
        print('all =', end - start)
    elif automations_list.get(automations_list.curselection()) == 'En-
able OSPF':
        device_to_enable = [device_list.get(sel_ip) for sel_ip in
list(device_list.curselection())]
        device_to_enable_ip = []
        for backup_dev in device_to_enable:
            device_to_enable_ip.append(backup_dev[1])
        dec_key = create_key()
        cipher = Fernet(dec_key)
        thread_list = []
        for dev_ip in device_to_enable_ip:
            with open('./Config/Devices.csv', 'rt') as csv_f:
                reader = csv.reader(csv_f, delimiter=',')
                for row in reader:
                    if row[0] == dev_ip:
                        pwd_to_dec = row[5]
                        dev_type = row[2]
            ospf_device_args = {'device_type': 'cisco_ios', 'ip':
dev_ip, 'username': 'ihu', 'password':
                                cipher.decrypt(pwd_to_dec.en-
code()).decode(), 'port': 22, 'verbose': True,
                                'global_delay_factor': 2}

            if (len(entry[dev_ip][0].get()) == 0 or len(en-
try[dev_ip][1].get()) == 0 or
```

```
                    len(entry[dev_ip][2].get()) == 0 or len(en-
try[dev_ip][3].get()) == 0):
                messagebox.showwarning('Alert', 'All OSPF parameters
should be filled')
            elif dev_type == 'Switch':
                messagebox.showwarning('Alert', 'OSPF can be enabled
only in routers\nNothing changed on'
                                                ' device with IP' +
dev_ip)
            else:
                ospf_pid = entry[dev_ip][0].get()
                ospf_ip = entry[dev_ip][1].get()
                ospf_mask = entry[dev_ip][2].get()
                ospf_area = entry[dev_ip][3].get()
                thread_item = threading.Thread(target=ospf_config,
                                               args=(ospf_device_args,
ospf_pid, ospf_ip, ospf_mask, ospf_area,))
                thread_list.append(thread_item)
        for th in thread_list:
            th.start()
    elif automations_list.get(automations_list.curselection()) == 'Add
Firewall Rule':
        device_to_enable = [device_list.get(sel_ip) for sel_ip in
list(device_list.curselection())]
        if len(device_to_enable) == 0:
            messagebox.showwarning('Alert', 'Select first the devices
you want \nto enable Add a firewall rule')
            automations_list.selection_clear(0, 'end')
        else:
            device_to_enable = [device_list.get(sel_ip) for sel_ip in
list(device_list.curselection())]
            device_to_enable_ip = []
            for backup_dev in device_to_enable:
                device_to_enable_ip.append(backup_dev[1])
            dec_key = create_key()
            cipher = Fernet(dec_key)
            thread_list = []
            for dev_ip in device_to_enable_ip:
                with open('./Config/Devices.csv', 'rt') as csv_f:
                    reader = csv.reader(csv_f, delimiter=',')
                    for row in reader:
                        if row[0] == dev_ip:
                            pwd_to_dec = row[5]
                            dev_type = row[2]
                acl_device_args = {'device_type': 'cisco_ios', 'ip':
dev_ip, 'username': 'ihu', 'password':
                                       cipher.decrypt(pwd_to_dec.en-
code()).decode(), 'port': 22, 'verbose': True,
                                        'global_delay_factor': 2}
                if (len(acl_nmb.get()) == 0 or len(source_ip.get()) ==
0 or len(source_wildcard.get()) == 0 or
                    len(destination_ip.get()) == 0 or len(destina-
tion_wildcard.get()) == 0 or\
                        acl_selected_type.get() == 'Select Action'):
                    messagebox.showwarning('Alert', 'All ACL parame-
ters are necessary except protocol')
                else:
                    acl_number = acl_nmb.get()
                    acl_action = acl_selected_type.get()
                    acl_protocol = acl_selected_options.get()
                    acl_sourceip = source_ip.get()
```

```
                    acl_sorce_wild = source_wildcard.get()
                    acl_destip = destination_ip.get()
                    acl_dest_wild = destination_wildcard.get()
                    thread_item = threading.Thread(target=acl_config,
args=(acl_device_args, acl_number, acl_action,

acl_protocol, acl_sourceip,acl_sorce_wild,

acl_destip,acl_dest_wild,))
                    thread_list.append(thread_item)
            for th in thread_list:
                th.start()

# function to add devices to the Device.csv file


def add_device_to_file():
    if (len(new_device_ip_address.get()) == 0 or len(new_device_pass-
word.get()) == 0 or
            new_device_selected_type.get() == 'Select Device Type' or
len(new_device_name.get()) == 0 or
            len(new_device_ssh_username.get()) == 0 or len(new_de-
vice_ssh_pass.get()) == 0):
        messagebox.showwarning('Alert', 'Device informations can not
be left blank  ')
        print(len(new_device_ip_address.get()), len(new_device_pass-
word.get()), new_device_selected_type.get(),
              len(new_device_name.get()), len(new_de-
vice_ssh_username.get()), len(new_device_ssh_pass.get()))

    else:
        if ip_entered(new_device_ip_address_entry.get()) == 0:
            pass
        else:
            pwd_window = Toplevel(root)
            master_password = StringVar()
            pwd_window.title('Encryption password')
            xpos = 10
            ypos = 600
            wgeo = '300x200+' + str(xpos) + '+' + str(ypos)
            pwd_window.geometry(wgeo)
            pwd_window.configure(background='white')
            pwd_window.resizable(False, False)
            pwd_window.grab_set()
            Label(pwd_window, text='Please enter Master Password',
font='Arial 15 bold ', bg='white').pack(side=TOP)
            master_password_entry = Entry(pwd_window, textvaria-
ble=master_password, show="*",
                                          width=30, font='Helvetica
11')
            master_password_entry.pack(side=TOP)
            master_password_entry.focus()
            pwd_ok_btn = Button(pwd_window, text=" OK ", width=15,
height=1,
                                font='Arial 15 bold', command=pwd_win-
dow.destroy)
            pwd_ok_btn.pack(side=BOTTOM, pady=15)
            # messagebox.showwarning('Alert', 'Please check the
weight!')
            pwd_window.attributes('-topmost', 1)  # Raising root above
all other windows
```

```
            root.wait_window(pwd_window)
            encryption_pwd = master_password.get()
            enc_key = get_key(encryption_pwd).decode()
            cipher = Fernet(enc_key)
            with open('./Config/Devices.csv', 'a+', newline='') as
write_obj:
                # Create a writer object from csv module
                csv_writer = writer(write_obj)
                # Add contents of list as last row in the csv file
                new_device_row = [new_device_ip_address.get(),
                                  cipher.encrypt(new_device_pass-
word.get().encode()).decode(),
                                  new_device_selected_type.get(),
new_device_name.get(),
                                  new_device_ssh_username.get(),
                                  cipher.encrypt(new_de-
vice_ssh_pass.get().encode()).decode()]
                csv_writer.writerow(new_device_row)
                messagebox.showinfo('Success', 'Device successfully
added!')

# this function is called when
# the user clicks the clear button on the add device frame
# and clears alla the inputs of the user


def clear_add_device():
    new_device_ip_address_entry.delete(0, 'end')
    new_device_password_entry.delete(0, 'end')
    new_device_name_entry.delete(0, 'end')
    new_device_ssh_username_entry.delete(0, 'end')
    new_device_ssh_pass_entry.delete(0, 'end')
    new_device_selected_type.set('Select Device Type')

# this function is used to have only one checkbox checked
# and also to create the manual filename entrybox gfor the  backup
function
# of the application


def get_backup_options(selected_type):
    global manual_config_name
    if selected_type == 'Date_First':
        c_ip_date.set(0)
        c_manual.set(0)
        for child in manual_file_frame.winfo_children():
            child.pack_forget()
    elif selected_type == 'IP_First':
        c_date_ip.set(0)
        c_manual.set(0)
        for child in manual_file_frame.winfo_children():
            child.pack_forget()
    elif selected_type == 'Manual':
        c_date_ip.set(0)
        c_ip_date.set(0)
        filename_lbl.pack(side=LEFT)
        manual_file_frame.pack(side=TOP)
        dev_ip_lbl.pack(side=LEFT)
        manual_entry.pack(pady=5)
    elif selected_type == 'Recent':
        c_manual_ask.set(0)
```

```
    elif selected_type == 'Ask':
        c_last.set(0)

# this function retrieve the devices from Devices.csv and filters them
# when user chooses all,Routers or switch


def getdevices(selected_type):
    device_list.delete(0, END)
    if selected_type == 'All':
        c_router.set(0)
        c_switch.set(0)
    elif selected_type == 'Router':
        c_all.set(0)
        c_switch.set(0)
    elif selected_type == 'Switch':
        c_all.set(0)
        c_router.set(0)

    if selected_type == 'All':
        with open('./Config/Devices.csv', newline='') as f:
            reader = csv.reader(f)
            devices_to_list = [list(row) for row in reader]
            devices_to_list.pop(0)

        for device_to_add in devices_to_list:
            device_to_add.pop(5)
            device_to_add.pop(4)
            device_to_add.pop(1)
            device_to_add.insert(0, 'IP:')
            device_to_add.insert(2, '__Type:')
            device_to_add.insert(4, '__Hostname:')
            device_list.insert(0, device_to_add)
    elif selected_type == 'Router':
        with open('./Config/Devices.csv', newline='') as f:
            reader = csv.reader(f)
            devices_to_list = [list(row) for row in reader]
            devices_to_list.pop(0)

        for device_to_add in devices_to_list:
            if device_to_add[2] == 'Router':
                device_to_add.pop(5)
                device_to_add.pop(4)
                device_to_add.pop(1)
                device_to_add.insert(0, 'IP:')
                device_to_add.insert(2, '__Type:')
                device_to_add.insert(4, '__Hostname:')
                device_list.insert(0, device_to_add)
    elif selected_type == 'Switch':
        with open('./Config/Devices.csv', newline='') as f:
            reader = csv.reader(f)
            devices_to_list = [list(row) for row in reader]
            devices_to_list.pop(0)

        for device_to_add in devices_to_list:
            if device_to_add[2] == 'Switch':
                device_to_add.pop(5)
                device_to_add.pop(4)
                device_to_add.pop(1)
                device_to_add.insert(0, 'IP:')
                device_to_add.insert(2, '__Type:')
```

```
                device_to_add.insert(4, '__Hostname:')
                device_list.insert(0, device_to_add)


# create the main window

root = Tk()
root.title('IHU Network Automation Application ')
def_font = tkinter.font.nametofont("TkDefaultFont")
def_font.config(size=10, family='Verdana')
# root.iconbitmap('c:/Users/g.milios/PycharmProjects/QA Reports/im-
ages/rtgr_logo_w_symbol.ico')
w, h = root.winfo_screenwidth(), root.winfo_screenheight()
root.state('zoomed')
root.config(bg='#407294')

# frame labels
device_add_frame_lbl = Label(text="Add new device:", font='Helvetica
11 bold', bg='#e6e6fa')
device_filter_frame_lbl = Label(text="Filters:", font='Helvetica 11
bold', bg='#e6e6fa')
devices_frame_lbl = Label(text="Network Devices", font='Helvetica 11
bold', bg='#407294')
automations_frame_lbl = Label(text="Automation Actions", font='Helvet-
ica 11 bold', bg='#407294')
automation_conf_frame_lbl = Label(text="Automation Action Configura-
tions", font='Helvetica 25 bold', bg='#407294')

# create frames
info_frame = LabelFrame(root, padx=5, pady=5, bg='#407294')
left_sub_frame = Frame(root, padx=5, relief=FLAT, bg='#407294')
device_filter_frame = LabelFrame(left_sub_frame, labelwidget=de-
vice_filter_frame_lbl, padx=5, bg='#e6e6fa', relief=FLAT)
devices_frame = LabelFrame(left_sub_frame, labelwidget=de-
vices_frame_lbl, padx=5, relief=FLAT, bg='#407294')
automations_frame = LabelFrame(left_sub_frame, labelwidget=automa-
tions_frame_lbl, padx=5, relief=FLAT, bg='#407294')
buttons_frame = Frame(info_frame, padx=5, relief=FLAT, bg='#407294')
output_frame = Frame(root, padx=5, bg='#407294', relief=SUNKEN)
device_add_frame = LabelFrame(info_frame, labelwidget=de-
vice_add_frame_lbl, padx=5, bg='#e6e6fa', relief=FLAT)
img_frame = Frame(root, padx=5, relief=FLAT, bg='#f0f0f0')
automation_conf_frame = LabelFrame(root, labelwidget=automa-
tion_conf_frame_lbl, padx=5, relief=SUNKEN, bg='#407294',
                                   labelanchor='n')
manual_file_frame = Frame(automation_conf_frame, padx=5, pady=15, re-
lief=FLAT, bg='#407294')

# build OSPF enable options widgets

ospf_main_frm = Frame(automation_conf_frame, relief=FLAT,
bg='#407294', pady=15)
ospf_device_frm = LabelFrame(ospf_main_frm, relief=SUNKEN,
bg='#407294', pady=15)
ospf_process_id_frm = LabelFrame(ospf_main_frm, padx=2, relief=SUNKEN,
bg='#407294', pady=15)
ospf_ip_frm = LabelFrame(ospf_main_frm, padx=5, relief=SUNKEN,
bg='#407294', pady=15)
ospf_mask_frm = LabelFrame(ospf_main_frm, padx=5, relief=SUNKEN,
bg='#407294', pady=15)
```

```
ospf_area_id_frm = LabelFrame(ospf_main_frm, padx=5, relief=SUNKEN,
bg='#407294', pady=15)

ospf_device_lbl = Label(ospf_device_frm, text='Device IP\n----------
--', font=('Aria', 15, 'bold'), fg='black',
                        anchor='w', bg='#407294')
ospf_process_id_lbl = Label(ospf_process_id_frm, text='process-id\n---
--------', font=('Aria', 15, 'bold'),
                           fg='black', anchor='w', bg='#407294')
ospf_ip_lbl = Label(ospf_ip_frm, text='Network IP\n--------------',
font=('Aria', 15, 'bold'), fg='black', anchor='w',
                    bg='#407294')
ospf_mask_lbl = Label(ospf_mask_frm, text='Subnet mask\n------------',
font=('Aria', 15, 'bold'), fg='black',
                      anchor='w', bg='#407294')
ospf_area_id_lbl = Label(ospf_area_id_frm, text='Area-id\n--------',
font=('Aria', 15, 'bold'), fg='black', anchor='w',
                         bg='#407294')

# build backup configuration options widgets
automation_filter_frame_lbl = Label(text="Options: ", font='Helvetica
18 bold',
                                    bg='white')
automation_filter_frame = LabelFrame(automation_conf_frame, label-
widget=automation_filter_frame_lbl,
                                     padx=5, relief=FLAT, bg='white',
labelanchor='w')
backup_name_lbl = Label(automation_conf_frame, text="Filenames config-
uration ",
                        font='Helvetica 25 bold', bg='#407294')

c_date_ip, c_ip_date, c_manual = IntVar(), IntVar(), IntVar()
cbtn_date_ip = Checkbutton(automation_filter_frame, text="Date First",
variable=c_date_ip,
                           onvalue=1, offvalue=0, state=NORMAL)
cbtn_ip_date = Checkbutton(automation_filter_frame, text="IP First",
variable=c_ip_date,
                           onvalue=1, offvalue=0, state=NORMAL)
cbtn_manual = Checkbutton(automation_filter_frame, text="Manual", var-
iable=c_manual,
                          onvalue=1, offvalue=0, state=NORMAL)
cbtn_date_ip.configure(command=partial(get_backup_options,
'Date_First'), bg='white')
cbtn_ip_date.configure(command=partial(get_backup_options,
'IP_First'), bg='white')
cbtn_manual.configure(command=partial(get_backup_options, 'Manual'),
bg='white')

# widgets in manual backup file frame
manual_config_name = StringVar()
manual_entry = Entry(manual_file_frame, textvariable=manual_con-
fig_name,
                     width=30, font='Helvetica 11')
filename_lbl = Label(manual_file_frame, text="Filename: ",
                     font='Helvetica 18 bold', bg='#407294')
dev_ip_lbl = Label(manual_file_frame, text="Device IP_ ",
                   font='Helvetica 15', bg='#407294')

# build restore configuration options widgets
automation_restore_options_frame_lbl = Label(text="Options: ",
font='Helvetica 18 bold',
```

```
                                                    bg='white')
automation_restore_options_frame = LabelFrame(automation_conf_frame,
labelwidget=automation_restore_options_frame_lbl,
                                                     padx=5, relief=FLAT,
bg='white', labelanchor='w')
restore_name_lbl = Label(automation_conf_frame, text="Files to re-
store",
                         font='Helvetica 25 bold', bg='#407294')

c_last, c_manual_ask = IntVar(), IntVar()
cbtn_last = Checkbutton(automation_restore_options_frame, text="Re-
store most recent backup", variable=c_last,
                        onvalue=1, offvalue=0, state=NORMAL)
cbtn_manual_ask = Checkbutton(automation_restore_options_frame,
text="Manual restore", variable=c_manual_ask,
                              onvalue=1, offvalue=0, state=NORMAL)

cbtn_last.configure(command=partial(get_backup_options, 'Recent'),
bg='white')
cbtn_manual_ask.configure(command=partial(get_backup_options, 'Ask'),
bg='white')

# build Add firewall options widgets

acl_number_frm = Frame(automation_conf_frame, relief=FLAT,
bg='#407294', pady=15)
acl_type_frm = Frame(automation_conf_frame, relief=FLAT, bg='#407294',
pady=15)
acl_options_frm = Frame(automation_conf_frame, relief=FLAT,
bg='#407294', pady=15)
acl_source_frm = Frame(automation_conf_frame, relief=FLAT,
bg='#407294', pady=15)
acl_swildcard_frm = Frame(automation_conf_frame, relief=FLAT,
bg='#407294', pady=15)
acl_des_frm = Frame(automation_conf_frame, relief=FLAT, bg='#407294',
pady=15)
acl_dwildcard_frm = Frame(automation_conf_frame, relief=FLAT,
bg='#407294', pady=15)

acl_number_lbl = Label(acl_number_frm, text='Aceess List number',
font=('Aria', 15, 'bold'), fg='black',
                       anchor='w', bg='#407294')
acl_type_lbl = Label(acl_type_frm, text='Type', font=('Aria', 15,
'bold'), fg='black',
                     anchor='w', bg='#407294')
acl_protocol_lbl = Label(acl_options_frm, text='Options',
font=('Aria', 15, 'bold'), fg='black',
                         anchor='w', bg='#407294')
acl_source_lbl = Label(acl_source_frm, text='Source', font=('Aria',
15, 'bold'), fg='black',
                       anchor='w', bg='#407294')
acl_swildcard_lbl = Label(acl_swildcard_frm, text='wildcard',
font=('Aria', 15, 'bold'), fg='black',
                          anchor='w', bg='#407294')
acl_des_lbl = Label(acl_des_frm, text='Destination', font=('Aria', 15,
'bold'), fg='black',
                    anchor='w', bg='#407294')
acl_dwildcard_lbl = Label(acl_dwildcard_frm, text='wildcard',
font=('Aria', 15, 'bold'), fg='black',
                          anchor='w', bg='#407294')
```

```
acl_selected_type = StringVar()
acl_selected_type.set('Select Action')
acl_selected_type_options = ['permit', 'deny']
acl_types = OptionMenu(acl_type_frm, acl_selected_type, *acl_se-
lected_type_options)
acl_types.config(width=15, bg='dark grey')
acl_types["highlightthickness"] = 1

acl_selected_options = StringVar()
acl_selected_options.set('Select Protocol')
acl_selected_options_options = ['ICMP', 'TCP', 'UDP']
acl_options = OptionMenu(acl_options_frm, acl_selected_options,
*acl_selected_options_options)
acl_options.config(width=15, bg='dark grey')
acl_options["highlightthickness"] = 1

acl_nmb, source_ip, source_wildcard, destination_ip, destination_wild-
card = StringVar(), StringVar(), StringVar(),\

StringVar(), StringVar()

acl_nmb_entry = Entry(acl_number_frm, textvariable=acl_nmb,
                      width=15, font='Helvetica 11')
source_ip_entry = Entry(acl_source_frm, textvariable=source_ip,
                        width=15, font='Helvetica 11')
source_wildcard_entry = Entry(acl_swildcard_frm, textvaria-
ble=source_wildcard,
                              width=15, font='Helvetica 11')
destination_ip_entry = Entry(acl_des_frm, textvariable=destination_ip,
                             width=15, font='Helvetica 11')
destination_wildcard_entry = Entry(acl_dwildcard_frm, textvaria-
ble=destination_wildcard,
                                   width=15, font='Helvetica 11')


# create image widget
bgimg = ImageTk.PhotoImage(Image.open('./im-
ages/ihu_logo.png').resize((300, 93)), Image.ANTIALIAS)
bg_lbl = Label(img_frame, image=bgimg)

# create  text box
output_description = Text(output_frame)
output_description.insert('1.0', 'output of the commands will be shown
here')

# create scrollbars
v_device_list = Scrollbar(devices_frame, orient=VERTICAL,
bg='#ffffff')
v_output = Scrollbar(output_frame, orient=VERTICAL, bg='#ffffff')
v_automation_actions = Scrollbar(automations_frame, orient=VERTICAL,
bg='#ffffff')

# create stored device listbox
device_list = Listbox(devices_frame, selectmode="multiple", ex-
portselection=False, yscrollcommand=v_device_list.set)
device_list.config(width=45, height=14, font='Arial 11', ac-
tivestyle='none')
automations_list = Listbox(automations_frame, selectmode="single", ex-
portselection=False,
                           yscrollcommand=v_automation_actions.set)
```

```
automations_list.config(width=45, height=8, font='Arial 11', ac-
tivestyle='none')
automations_list.bind("<<ListboxSelect>>", get_config_name)
# configure scrollbars
v_device_list.config(command=device_list.yview)
v_output.config(command=output_description.yview)
v_automation_actions.config(command=automations_list.yview)

# add items to automation actions
action_options = ['Enable OSPF', 'Add Firewall Rule', 'Restore Config-
uration', 'Backup Configuration']
for action_to_add in action_options:
    automations_list.insert(0, action_to_add)

# create device type options
c_all, c_router, c_switch = IntVar(), IntVar(), IntVar()
cbtn_all = Checkbutton(device_filter_frame, text="all", varia-
ble=c_all, onvalue=1, offvalue=0, state=NORMAL)
cbtn_router = Checkbutton(device_filter_frame, text="Router", varia-
ble=c_router, onvalue=1, offvalue=0, state=NORMAL)
cbtn_switch = Checkbutton(device_filter_frame, text="Switch", varia-
ble=c_switch, onvalue=1, offvalue=0, state=NORMAL)

cbtn_all.configure(command=partial(getdevices, 'All'))
cbtn_router.configure(command=partial(getdevices, 'Router'))
cbtn_switch.configure(command=partial(getdevices, 'Switch'))

# create buttons

exit_btn = Button(buttons_frame, text="Exit", width=32, height=2,
                  command=root.destroy, bg='#bada55')
change_pass_btn = Button(buttons_frame, text="Change Password",
width=32, height=2,
                         command=password_change, bg='#bada55')
execute_btn = Button(automation_conf_frame, text="Execute", width=147,
height=2,
                     bg='#bada55', command=execute_automations,
state='normal')

add_device_ok_btn = Button(device_add_frame, text="Add Device",
width=16, height=2, command=add_device_to_file,
                           bg='#bada55', state='normal')
add_device_clear_btn = Button(device_add_frame, text="Clear",
width=16, height=2,
                              command=clear_add_device, bg='#bada55')

# create entryboxes
new_device_ip_address, new_device_password, new_device_name, new_de-
vice_ssh_username, new_device_ssh_pass = \
    StringVar(), StringVar(), StringVar(), StringVar(), StringVar()

# entrybox labels

new_device_ip_address_lbl = Label(device_add_frame, text="IP Ad-
dress:", font='Helvetica 11 bold', bg='#e6e6fa')
new_device_password_lbl = Label(device_add_frame, text="Password:",
font='Helvetica 11 bold', bg='#e6e6fa')
new_device_name_lbl = Label(device_add_frame, text="Description:",
font='Helvetica 11 bold', bg='#e6e6fa')
new_device_ssh_username_lbl = Label(device_add_frame, text="SSH Login
Username:",
```

```
                                        font='Helvetica 11 bold',
bg='#e6e6fa')
new_device_ssh_pass_lbl = Label(device_add_frame, text="SSH Login
Password:",
                                    font='Helvetica 11 bold',
bg='#e6e6fa')

# entryboxes
new_device_ip_address_entry = Entry(device_add_frame, textvaria-
ble=new_device_ip_address,
                                        width=30, font='Helvetica 11')
new_device_password_entry = Entry(device_add_frame, textvaria-
ble=new_device_password, show="*",
                                    width=30, font='Helvetica 11')
new_device_name_entry = Entry(device_add_frame, textvariable=new_de-
vice_name, width=30, font='Helvetica 11')
new_device_ssh_username_entry = Entry(device_add_frame, textvaria-
ble=new_device_ssh_username,
                                      width=30, font='Helvetica 11')
new_device_ssh_pass_entry = Entry(device_add_frame, textvaria-
ble=new_device_ssh_pass, show="*",
                                    width=30, font='Helvetica 11')

# create new device type optionmenu
new_device_selected_type = StringVar(device_add_frame)
new_device_selected_type.set('Select Device Type')
new_device_type_options = ['Router', 'Switch']
new_device_types = OptionMenu(device_add_frame, new_device_se-
lected_type, *new_device_type_options)
new_device_types.config(width=30, bg='dark grey')


# packs
new_device_ip_address_lbl.pack(pady=5)
new_device_ip_address_entry.pack(pady=5)
new_device_password_lbl.pack(pady=5)
new_device_password_entry.pack(pady=5)
new_device_name_lbl.pack(pady=5)
new_device_name_entry.pack(pady=5)
new_device_ssh_username_lbl.pack(pady=5)
new_device_ssh_username_entry.pack(pady=5)
new_device_ssh_pass_lbl.pack(pady=5)
new_device_ssh_pass_entry.pack(pady=5)
new_device_types.pack(pady=10)
add_device_ok_btn.pack(side=LEFT, padx=5, pady=5)
add_device_clear_btn.pack(side=LEFT, pady=5)

# packs
info_frame.pack(side=LEFT, fill=BOTH, padx=10, pady=10)
buttons_frame.pack(side=BOTTOM, anchor="n", padx=10, pady=10)
output_frame.pack(fill=BOTH, side=BOTTOM, pady=10)
img_frame.pack(fill=BOTH, side=TOP, pady=10)
left_sub_frame.pack(side=LEFT, fill=BOTH, pady=10)
bg_lbl.pack()
devices_frame.pack(anchor="nw")
device_filter_frame.pack(anchor="nw", padx=10, pady=5)
automations_frame.pack(anchor="nw")
device_add_frame.pack(anchor="nw", padx=10, pady=5)
automation_conf_frame.pack(side=LEFT, fill=BOTH)
v_device_list.pack(side=RIGHT, fill=Y)
device_list.pack(side=LEFT, anchor="n")
```

```
cbtn_all.pack(side=LEFT, anchor="n", padx=29, pady=5)
cbtn_router.pack(side=LEFT, anchor="n", padx=29, pady=5)
cbtn_switch.pack(side=LEFT, anchor="n", padx=29, pady=5)
v_automation_actions.pack(side=RIGHT, fill=Y)
automations_list.pack(side=LEFT, anchor="n")

exit_btn.pack(anchor="s", side=BOTTOM, pady=5)
change_pass_btn.pack(anchor="s", side=BOTTOM, pady=5)
execute_btn.pack(side=BOTTOM, pady=5)
v_output.pack(side=RIGHT, fill=Y)
output_description.pack(fill=BOTH, side=BOTTOM)

root.mainloop()
```

Bibliography

[1] En.wikipedia.org. 2020. Automation. [online] Available at: <https://en.wikipedia.org/wiki/Automation> [Accessed 10 May 2020].

[2] Cisco. 2020. What Is Network Automation? [online] Available at: <https://www.cisco.com/c/en/us/solutions/automation/network-automation.html> [Accessed 11 September 2020].

[3] Juniper Networks. 2020. What Is Network Automation? - Juniper Networks. [online] Available at: <https://www.juniper.net/us/en/products-services/what-is/network-automation/> [Accessed 23 May 2020].

[4] Edelman, Jason, Scott S. Lowe, and Matt Oswalt. 2018. *Network Programmability And Automation_ Skills For The Next-Generation Network Engineer*. 1st ed. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

[5] Software-Defined Networking: The New Norm For Networks. 2012. Ebook. OPEN NETWORK FOUNTATION.

https://pdfs.semanticscholar.org/a3f6/9f6181a0b4d481073a21eafbcc434a800db6.pdf.

[6] Feamster, Nick, Jennifer Rexford, and Ellen Zegura. 2013. "The Road To SDN: An Intellectual History Of Programmable Networks". Cs.Princeton.Edu. https://www.cs.princeton.edu/courses/archive/fall13/cos597E/papers/sdnhistory.pdf.

[7] ManageEngine, communications@manageengine.com. 2020. "Network Monitoring Software By Manageengine Opmanager". Manageengine Opmanager. https://www.manageengine.com/network-monitoring/what-is-snmp.html.

[8] "Google Cloud Status Dashboard". 2020. Status.Cloud.Google.Com. https://status.cloud.google.com/incident/cloud-networking/19009.

[9] van Rossum, Guido, Barry Warsaw, and Nick Coghlan. 2001. "PEP 8 -- Style Guide For Python Code". Python.Org. https://www.python.org/dev/peps/pep-0008/.

[10] "Paramiko Documentation". 2013. Paramiko.Org. http://www.paramiko.org/.

[11] "Ktbyers/Netmiko". 2014. Github. https://github.com/ktbyers/netmiko.

[12] "Cisco Devnet Code Exchange: Discover Code Repositories Related To Cisco Technologies". 2015. Developer.Cisco.Com.

https://developer.cisco.com/codeexchange/github/repo/NetworkGirlDebi/Netmiko/.

[13] "Ipaddress — Ipv4/Ipv6 Manipulation Library — Python 3.9.1 Documentation". 2001. *Docs.Python.Org*. https://docs.python.org/3/library/ipaddress.html.

[14] "Cryptography". 2014. *Pypi*. https://pypi.org/project/cryptography/.

[15] "Cryptography 3.4.Dev1 Documentation". 2014. *Cryptography.Io*. https://cryptography.io/en/latest/.

[16] "Tkinter — Python Interface To Tcl/Tk — Python 3.9.1 Documentation". 2014. *Docs.Python.Org*. https://docs.python.org/3/library/tkinter.html.

[17] "Threading — Thread-Based Parallelism — Python 3.9.1 Documentation". 2019. *Docs.Python.Org*. https://docs.python.org/3/library/threading.html.

[18] "Multithreading In Python | Set 1 - Geeksforgeeks". 2019. *Geeksforgeeks*. https://www.geeksforgeeks.org/multithreading-python-set-1/.

[19] 2020. *Gns3.Com*. https://www.gns3.com/.

[20] "Napalm Network Automation". 2020. *Napalm Network Automation*. https://napalm-automation.net/.

[21] "Snort - Network Intrusion Detection & Prevention System". 2020. *Snort.Org*. https://www.snort.org/.

[22] "Threatstream/Snort". 2020. *Github*. https://github.com/threatstream/snort/blob/master/src/output-plugins/spo_csv.c.