Project Assignment
& Solution

Cybersecurity Professional Program

Introduction to Python
for Security

Final Project

PY-07-LS1
ARP Spoof Detector

Note: Solutions for the instructor are shown inside the green box.

Project Objective

Implement learned skills to create an automated program that can be used in real-life scenarios.

Project Mission

Create a program that can detect active ARP spoofing attacks on host machines.

Project Duration

2-4 hours.

Requirements

- Working knowledge of Python.
- Working knowledge of Python's OS module.

Resources

- Environment & Tools
    - Windows, MacOS, and Linux
        - PyCharm
        - Python 3
    - Kali-linux-2019.3-amd64.iso

Project Scenario

The HackRS company has been experiencing network connection issues, such as low bandwidth and disconnections, in the past few days. The IT team couldn't find the cause for the issue, and the company decided to ask for your assistance.

You suspect that the network is under an MiTM attack.

As a Python scripts expert, you need to create a script that is able to automatically identify ARP spoofing behaviour on workstations.

In the first stage, the script should read a station's ARP table, and extract the address from it.

In the next stage, the script should search for MAC duplications in table entries, and log every ARP spoofing event.

Project Task 1: ARP Table Extraction

The first part of the program must handle the extraction of the ARP table data. The data must be saved in a dynamic structure in which the IP address corresponds to the MAC address. The structure is required for later comparison operations, to identify duplicated addresses.

1. Plan a function that will extract the ARP table data from the machine. How can Python access this type of data? How should the data be saved for later use?

Since the ARP table is an operating system component, the os module must be imported, to enable ARP table data printing.

The ARP table data should be saved in a variable and then divided in list form. The list structure must iterate over the table's rows, and treat each one individually.

The list should be filtered from unnecessary data and saved in a dictionary that lists IP addresses corresponding to MAC addresses, throughout iteration over ARP data lines.

2. Import the required modules for the program.

```
import os
```

3. Define a function that will handle the ARP table data extraction.

```
import os

def arp_table_extraction():
```

4. Create three variables, one to store the ARP table data, another to store a list of the separated lines, and the third to store the final filtered data.

```
import os

def arp_table_extraction():
    arp_table = os.popen("arp -a").read()
    arp_table_lines = arp_table.splitlines()
    addresses = {}
```

5 Iterate over the lines and save the IP addresses and their corresponding MAC addresses after data filtration.
Only IP and MAC addresses should be saved in the third variable.
Filter the rest of the data, such as the interface's IP address or broadcast data.

```
import os

def arp_table_extraction():
    arp_table = os.popen("arp -a").read()
    arp_table_lines = arp_table.splitlines()
    addresses = {}
    for line in arp_table_lines:
        if "ff-ff-ff-ff-ff-ff" in line or "ff:ff:ff:ff:ff:ff" in line:
            break
        if arp_table_lines.index(line) > 3:
            ip, mac, _type = line.split()
            addresses[ip] = mac
    return addresses
```

Project Task 2: ARP Table Extraction

The second part of the program must take the extracted data and identify if an ARP Spoofing attack is underway.This is done by locating MAC address duplications in the ARP table entries.

1. Plan a function that will identify MAC address duplication.

The function should accept the dictionary generated by the previous steps and iterate over it.

Every iterated MAC address should be saved to a temporary list. The list will then be used to find the MAC address duplications.

2. Define a function to identify duplication in MAC addresses. The function should accept a parameter.

```
def identify_duplication(addresses):
```

3. Create a variable to store iterated MAC addresses for later comparison.

```
def identify_duplication(addresses):
    tmp_mac_lst = []
```

4. Iterate over the recorded MAC addresses and compare them to the saved ones to identify duplications. Print a message that notifies when a duplication is identified.

In the following code, the sleep method is used to perform the calculation. To use this method, the time module must be imported.

```
def identify_duplication(addresses):
    tmp_mac_lst = []
    print("Scanning...")
    time.sleep(3)
    for mac in addresses.values():
        if mac in tmp_mac_lst:
            print("Arp Spoofed!", "\nThe address is:", mac)
            break
        tmp_mac_lst.append(mac)
```

5 In the ARP extracting function, pass the filtered data to the current function.

```
def arp_table_extraction():
    arp_table = os.popen("arp -a").read()
    arp_table_lines = arp_table.splitlines()
    addresses = {}
    for line in arp_table_lines:
        if "ff-ff-ff-ff-ff-ff" in line or "ff:ff:ff:ff:ff:ff" in line:
            break
        if arp_table_lines.index(line) > 2:
            ip, mac, _type = line.split()
            addresses[ip] = mac

    identify_duplication(addresses)
```

Project Task 3: Logging Events

The third part of the program will handle the logging of the Arp Spoof activity.

1. Plan a function that will log every ARP Spoofing event and save it to a file.

The function should accept a message from the **identify_duplication** function.

To act as a legitimage log record, it should also add the event's time to the record, which will require importing the datetime module.

The function must also create a file and append to it all data regarding the event.

2. Define a function to handle ARP Spoofing event logging. The function should accept data regarding the event.

```
from datetime import datetime

def create_log(message):
```

3. Create a variable to store the date and time of the event.

```
from datetime import datetime

def create_log(message):
     date = datetime.now()
```

4 Save the logged data to a file.

```
from datetime import datetime

def create_log(message):
    print("Generating logs...")
    time.sleep(3)
    date = datetime.now()
    with open("log.txt", "a") as log:
        log.write(message + "\nDate: {}\n\n".format(date))
    print("The event is logged in log.txt")
```

5 In the ARP Spoof identification function, pass a message to the log creator function.

```
def identify_duplication(addresses):
    tmp_mac_lst = []
    print("Scanning...")
    time.sleep(3)
    for mac in addresses.values():
        if mac in tmp_mac_lst:
            print("Finished scanning")
            create_log("Arp Spoofed!\nThe address is:" + mac)
            break
        tmp_mac_lst.append(mac)
```

6 Add execution control to make sure the program is executed only if its file is directly executed.

```
import os
import time
from datetime import datetime


def arp_table_extraction():
    arp_table = os.popen("arp -a").read()
    arp_table_lines = arp_table.splitlines()
    addresses = {}
    for line in arp_table_lines:
        if "ff-ff-ff-ff-ff-ff" in line or "ff:ff:ff:ff:ff:ff" in line:
            break
        if arp_table_lines.index(line) > 2:
            ip, mac, _type = line.split()
            addresses[ip] = mac

    identify_duplication(addresses)


def identify_duplication(addresses):
    tmp_mac_lst = []
    print("Scanning...")
    time.sleep(3)
    for mac in addresses.values():
        if mac in tmp_mac_lst:
            print("Finished scanning")
            create_log("Arp Spoofed!\nThe address is:" + mac)
            break
        tmp_mac_lst.append(mac)


def create_log(message):
    print("Generating logs...")
    time.sleep(3)
    date = datetime.now()
    with open("log.txt", "a") as log:
        log.write(message + "\nDate: {}\n\n".format(date))
    print("The event is logged in log.txt")


if __name__ == "__main__":
    arp_table_extraction()
```

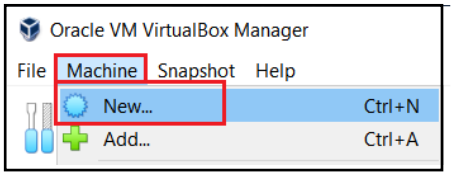
Project Task 4: Testing

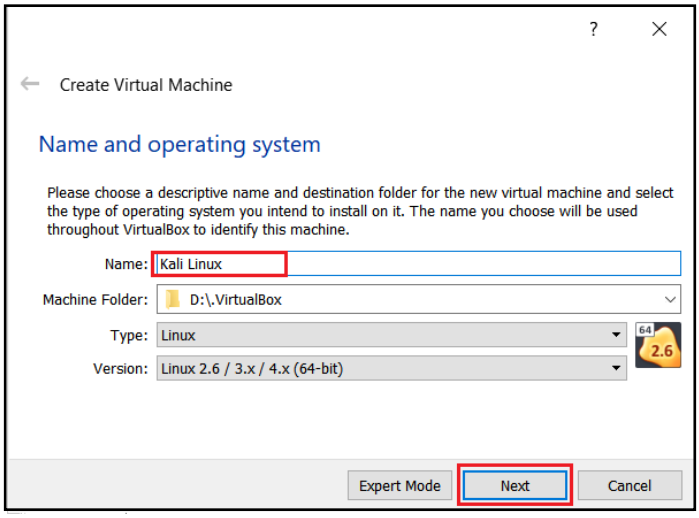
This task is the final step in the project. The program to identify ARP Spoofing activity in the network is complete and the only thing left to do, is to test its functionality.

To check the script, an ARP Spoof attack must be launched. This can be done from a Kali Linux VM against a Windows host or VM.

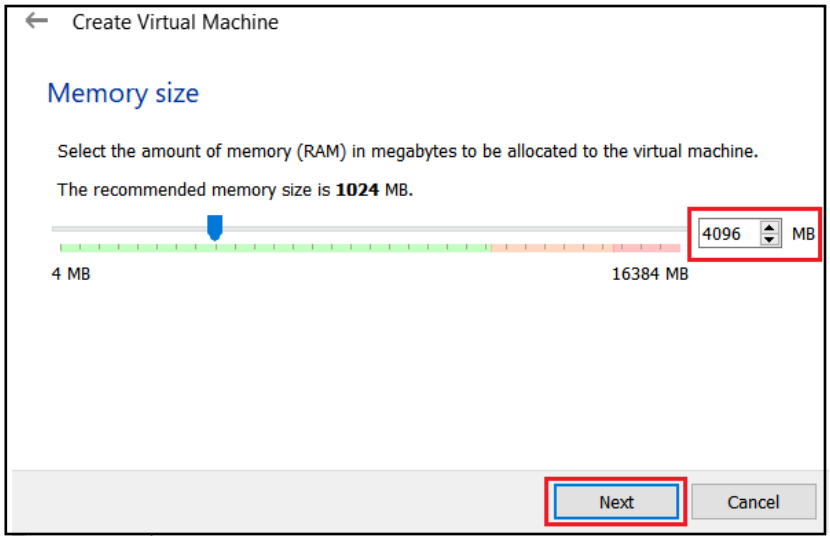
The OS's should be configured to use the Bridge Adapter network configuration, for direct communicate with the router.

1 Open VirtualBox, click the **Machine** tab, click **New**, and name the VM "kali linux".

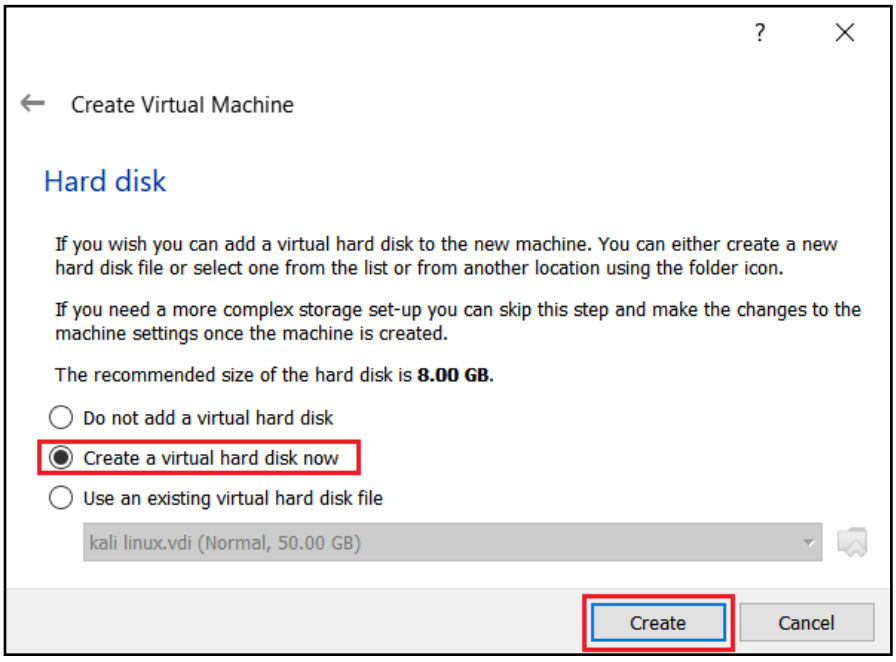




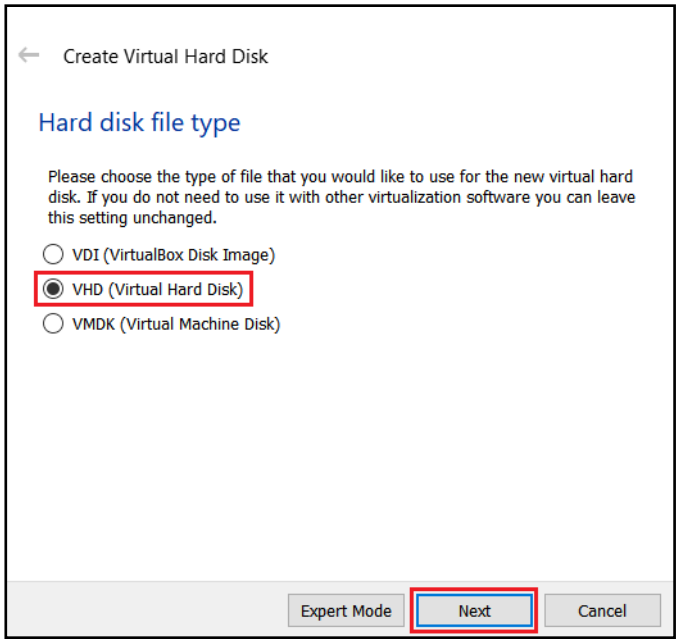
2 Set the memory to **2048 MB** for proper functionality.
You can use a larger setting (in accordance with the computer's available resources) for enhanced functionality.



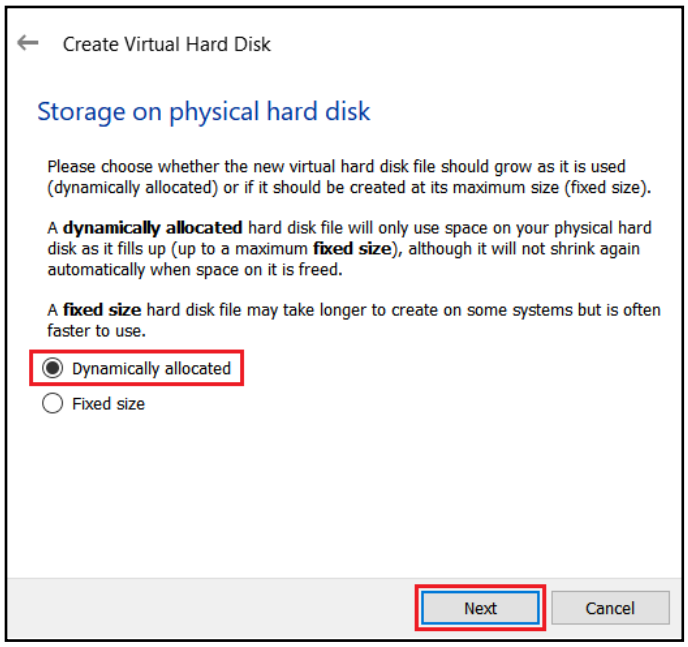
3 In the next window, select **Create a virtual hard disk now**, and click **Create**.



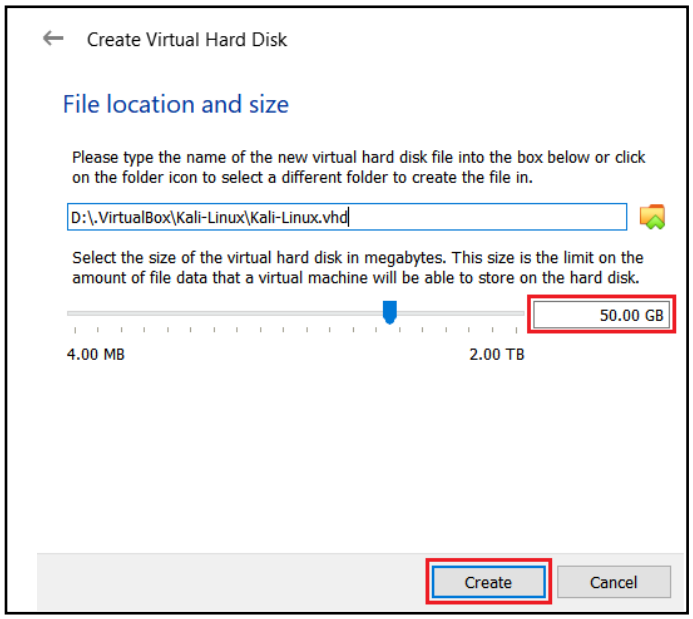
4 Select **VHD** for the Hard disk file type, and click **Next**.



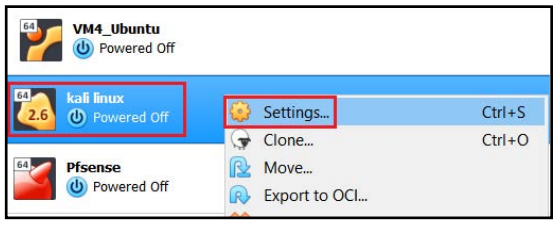
5 Select **Dynamically allocated** and click **Next**.

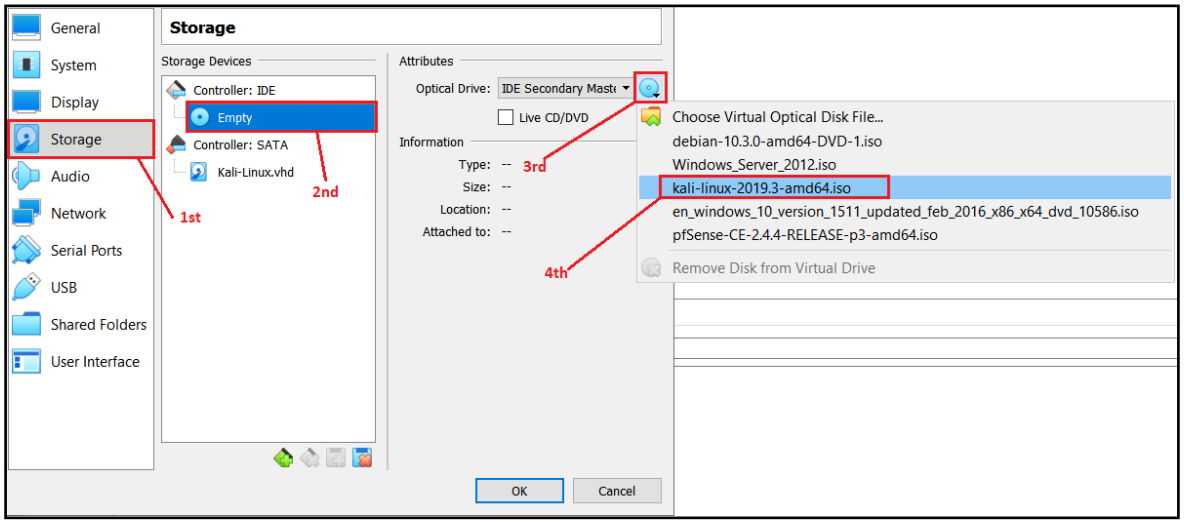


6 Select the file location and set the storage size to **50 GB**.
If your computer does not have enough storage, you can set it to **20 GB**.



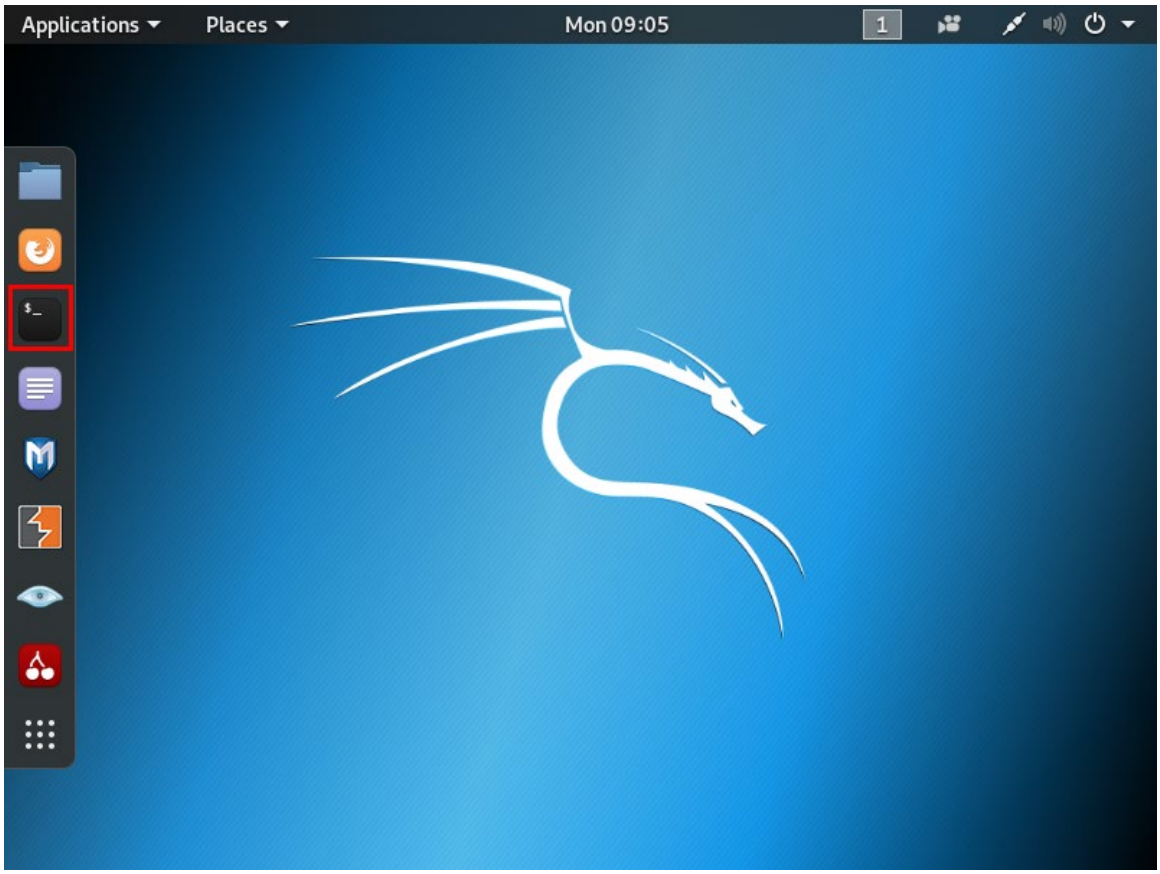
7 Right-click the Kali VM, go to **Settings**, and insert the **Kali-linux-2019.3-amd64.iso** in the drive.



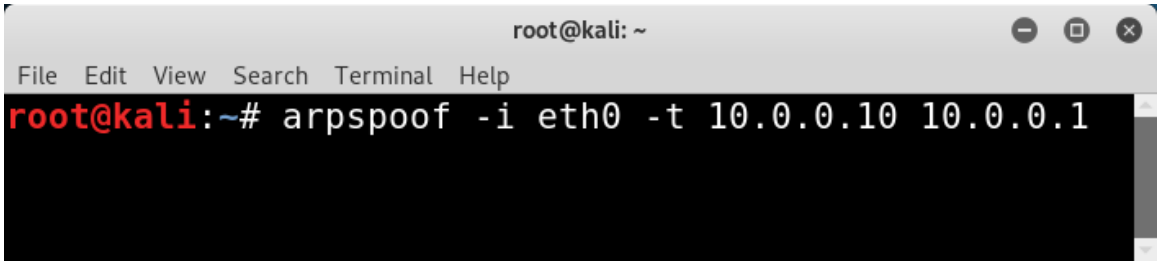


8 Run the virtual machine and select the live version.

9 Open the terminal by clicking on it, in the options on the left.



10 Run the command **arpspoof -i eth0 -t <target IP> <spoofed IP>**



11 Execute the script and note that a log was created.