Penetration Testing Logbook

Student:
Alessandro Buonerba

Module Leader:
Dr Anatolij Bezemskij

Computer Science (Cybersecurity)
Penetration Testing and Ethical Vulnerability Scanning
COMP-1671

Department of Computing & Mathematical Sciences
Liberal Arts & Sciences

University of Greenwich
London, United Kingdom

December 2021

CONTENTS

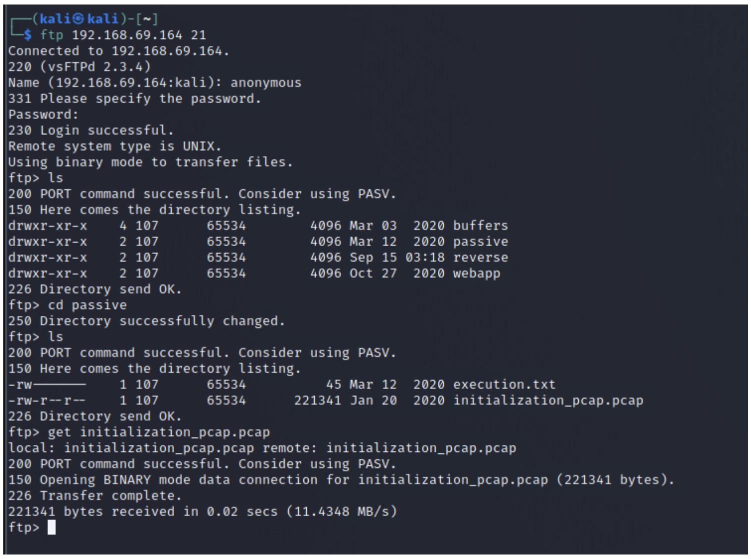
LIST OF FIGURES

1 LAB 1: PASSIVE ENUMERATION

Contrarily from active enumeration, passive enumeration is a technique that does not rely on explicit communication with a target system (Cooper, 2020). To perform a passive enumeration, a network monitor tool such as Wireshark is often used.

1.1 CONNECT TO FTP

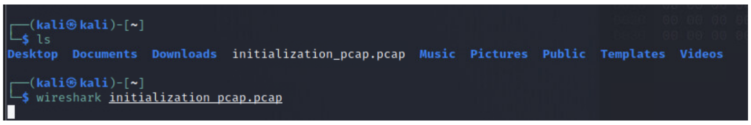
The first part of the task is to connect to the FTP server and download the .pcap file with all the captured network traffic.



```
┌──(kali㉿kali)-[~]
└─$ ftp 192.168.69.164 21
Connected to 192.168.69.164.
220 (vsFTPd 2.3.4)
Name (192.168.69.164:kali): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxr-xr-x    4 107      65534        4096 Mar 03  2020 buffers
drwxr-xr-x    2 107      65534        4096 Mar 12  2020 passive
drwxr-xr-x    2 107      65534        4096 Sep 15 03:18 reverse
drwxr-xr-x    2 107      65534        4096 Oct 27  2020 webapp
226 Directory send OK.
ftp> cd passive
250 Directory successfully changed.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-------    1 107      65534          45 Mar 12  2020 execution.txt
-rw-r--r--    1 107      65534      221341 Jan 20  2020 initialization_pcap.pcap
226 Directory send OK.
ftp> get initialization_pcap.pcap
local: initialization_pcap.pcap remote: initialization_pcap.pcap
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for initialization_pcap.pcap (221341 bytes).
226 Transfer complete.
221341 bytes received in 0.02 secs (11.4348 MB/s)
ftp> █
```

Figure 1.1: Connect to the FTP and get the .pcap file

Now that the file has been downloaded, it can be found in the home directory and we can start the analysis of the network traffic through Wireshark following the tasks assigned to this lab.

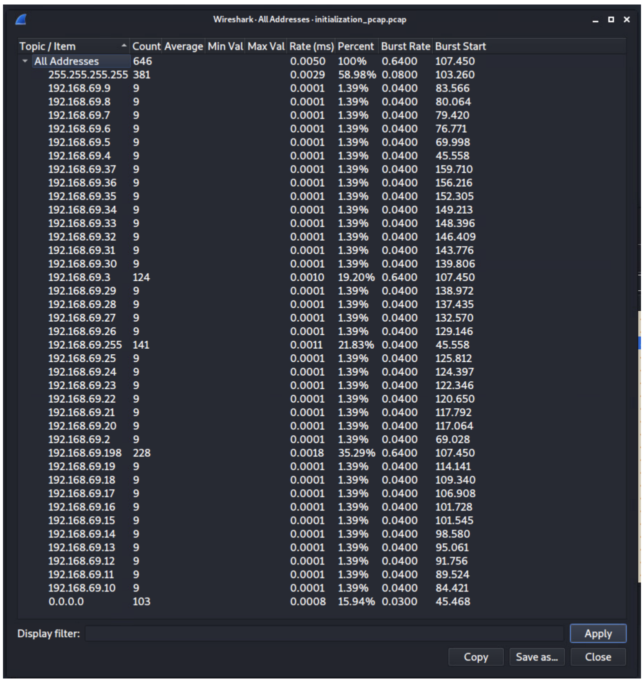


```
┌──(kali㉿kali)-[~]
└─$ ls
Desktop  Documents  Downloads  initialization_pcap.pcap  Music  Pictures  Public  Templates  Videos

┌──(kali㉿kali)-[~]
└─$ wireshark initialization_pcap.pcap
█
```

Figure 1.2: Open .pcap with Wireshark

1.2 find unique ipv4 addresses

The first tasks asks to find the unique IPs that are stored and captured. We can achieve that through the top menu, selecting statistics and IPv4 addresses. The result is shown in the figure below.

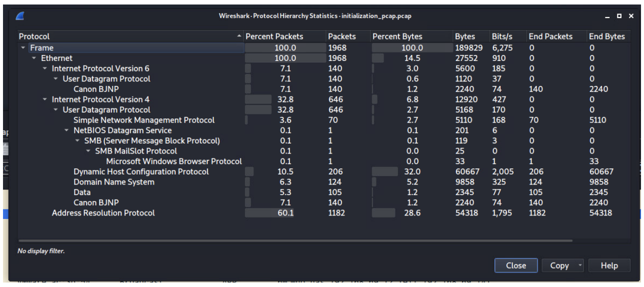


Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
All Addresses	646				0.0050	100%	0.6400	107.450
255.255.255.255	381				0.0029	58.98%	0.0800	103.260
192.168.69.9	9				0.0001	1.39%	0.0400	83.566
192.168.69.8	9				0.0001	1.39%	0.0400	80.064
192.168.69.7	9				0.0001	1.39%	0.0400	79.420
192.168.69.6	9				0.0001	1.39%	0.0400	76.771
192.168.69.5	9				0.0001	1.39%	0.0400	69.998
192.168.69.4	9				0.0001	1.39%	0.0400	45.558
192.168.69.37	9				0.0001	1.39%	0.0400	159.710
192.168.69.36	9				0.0001	1.39%	0.0400	156.216
192.168.69.35	9				0.0001	1.39%	0.0400	152.305
192.168.69.34	9				0.0001	1.39%	0.0400	149.213
192.168.69.33	9				0.0001	1.39%	0.0400	148.396
192.168.69.32	9				0.0001	1.39%	0.0400	146.409
192.168.69.31	9				0.0001	1.39%	0.0400	143.776
192.168.69.30	9				0.0001	1.39%	0.0400	139.806
192.168.69.3	124				0.0010	19.20%	0.6400	107.450
192.168.69.29	9				0.0001	1.39%	0.0400	138.972
192.168.69.28	9				0.0001	1.39%	0.0400	137.435
192.168.69.27	9				0.0001	1.39%	0.0400	132.570
192.168.69.26	9				0.0001	1.39%	0.0400	129.146
192.168.69.255	141				0.0011	21.83%	0.0400	45.558
192.168.69.25	9				0.0001	1.39%	0.0400	125.812
192.168.69.24	9				0.0001	1.39%	0.0400	124.397
192.168.69.23	9				0.0001	1.39%	0.0400	122.346
192.168.69.22	9				0.0001	1.39%	0.0400	120.650
192.168.69.21	9				0.0001	1.39%	0.0400	117.792
192.168.69.20	9				0.0001	1.39%	0.0400	117.064
192.168.69.2	9				0.0001	1.39%	0.0400	69.028
192.168.69.198	228				0.0018	35.29%	0.6400	107.450
192.168.69.19	9				0.0001	1.39%	0.0400	114.141
192.168.69.18	9				0.0001	1.39%	0.0400	109.340
192.168.69.17	9				0.0001	1.39%	0.0400	106.908
192.168.69.16	9				0.0001	1.39%	0.0400	101.728
192.168.69.15	9				0.0001	1.39%	0.0400	101.545
192.168.69.14	9				0.0001	1.39%	0.0400	98.580
192.168.69.13	9				0.0001	1.39%	0.0400	95.061
192.168.69.12	9				0.0001	1.39%	0.0400	91.756
192.168.69.11	9				0.0001	1.39%	0.0400	89.524
192.168.69.10	9				0.0001	1.39%	0.0400	84.421
0.0.0.0	103				0.0008	15.94%	0.0300	45.468

Figure 1.3: Unique IPv4 addresses

1.3 application-layer protocols

The second task asks to find the application-layer protocols that are used in the captured network traffic. This can be displayed using the Protocol Hierarchy command. The result is shown in the figure below.



Wireshark · Protocol Hierarchy Statistics · initialization_pcap.pcap

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes
Frame	100.0	1968	100.0	189829	6,275	0	0
Ethernet	100.0	1968	14.5	27552	910	0	0
Internet Protocol Version 6	7.1	140	3.0	5600	185	0	0
User Datagram Protocol	7.1	140	0.6	1120	37	0	0
Canon BJNP	7.1	140	1.2	2240	74	140	2240
Internet Protocol Version 4	32.8	646	6.8	12920	427	0	0
User Datagram Protocol	32.8	646	2.7	5168	170	0	0
Simple Network Management Protocol	3.6	70	2.7	5110	168	70	5110
NetBIOS Datagram Service	0.1	1	0.1	201	6	0	0
SMB (Server Message Block Protocol)	0.1	1	0.1	119	3	0	0
SMB MailSlot Protocol	0.1	1	0.0	25	0	0	0
Microsoft Windows Browser Protocol	0.1	1	0.0	33	1	1	33
Dynamic Host Configuration Protocol	10.5	206	32.0	60667	2,005	206	60667
Domain Name System	6.3	124	5.2	9858	325	124	9858
Data	5.3	105	1.2	2345	77	105	2345
Canon BJNP	7.1	140	1.2	2240	74	140	2240
Address Resolution Protocol	60.1	1182	28.6	54318	1,795	1182	54318

Figure 1.4: Protocol Hierarchy

1.4 NAME OF THE PROTOCOLS

The application-layer protocol are the following.

- SNMP (Single Network Management Protocol): responsible for the management of network devices, allows the communication between them independently of their spec (Scarpati, 2020).
- DNS (Domain Name System): responsible for the resolution of domain names to IP addresses (Insam, 2020).
- DHCP (Dynamic Host Configuration Protocol): responsible for the dynamic configuration of network devices. This protocol is used to automatically assign IPs to network devices (IBM, 2021).
- SMB (Server Message Block): responsible for the communication between shared devices such as printers on a network (Sheldon and Scarpati, 2020).

1.5 NETWORK DIAGRAM

This task will allow us to have a visual representation of the analysis of the network. Below the diagram with the active protocols and devices.

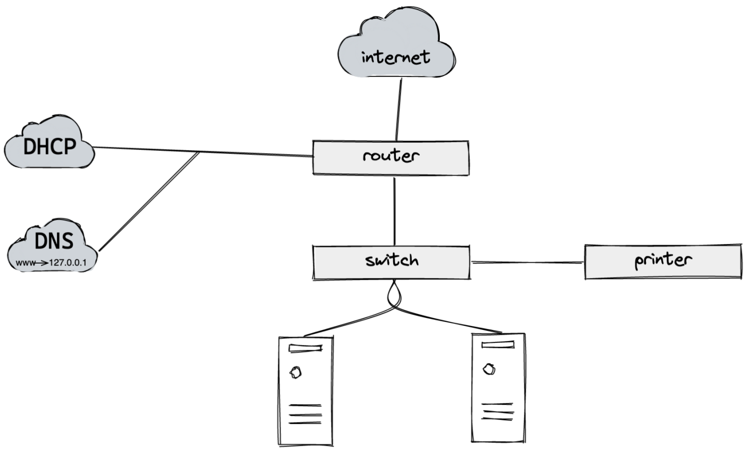


Figure 1.5: Network Diagram

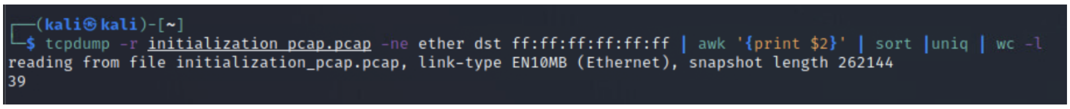
Following the explanation of the protocols, further analysis portrays the use of internet protocol. The protocols in use are UDP, SNMP, DHCP and DNS, meaning computers and shared devices on the network. We can also certify using a BJNP protocol, meaning that the shared device on the network is a Canon printer.

1.6 DISCUSSION

The network traffic analysis suggests that a user uses the shared device since there is a BJNP protocol. There are also ACKs and NAKs portraying active communication between the devices of the network. Some of the UDP packets were broadcasting an std discovery all to find all the services on the network.

1.7 TCP DUMP

Following the instructions and the man page for the tcpdump command, I have been able to reproduce a one liner to output a number of unique MAC addresses in the provided and previously used .pcap file. Below a picture with the result.



```
┌──(kali㉿kali)-[~]
└─$ tcpdump -r initialization_pcap.pcap -ne ether dst ff:ff:ff:ff:ff:ff | awk '{print $2}' | sort |uniq | wc -l
reading from file initialization_pcap.pcap, link-type EN10MB (Ethernet), snapshot length 262144
39
```

Figure 1.6: TCP Dump

The flag `-r` is used to read the file and the flag `-ne` before `ether dst` looks for ethernet destinations with the MAC address format specified right after it. The command `awk` is used to separate them while printing the second argument to get the second column. It will then sort and check for unique entries for then count everything with the last `wc -l` command

1.8 REFLECTION

This has been a very fun lab. I have learned a lot more about Wireshark and how to analyse a .pcap file. Even though I have never used `tcpdump`, there were manyexamples and exhaustive official documentation.

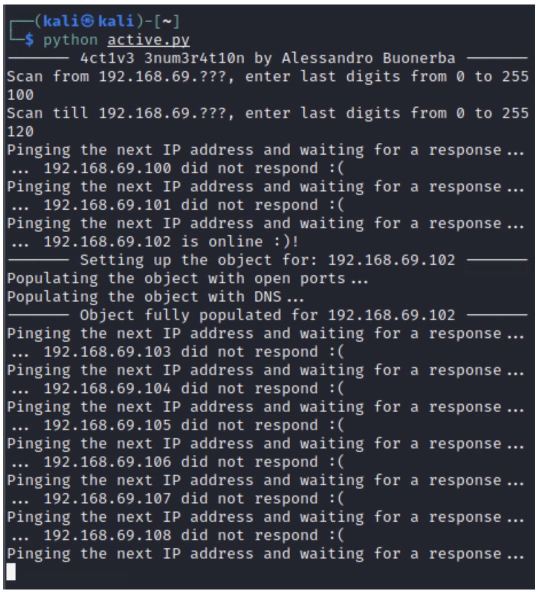
2

LAB 2: ACTIVE ENUMERATION

2.1 INTRODUCTION

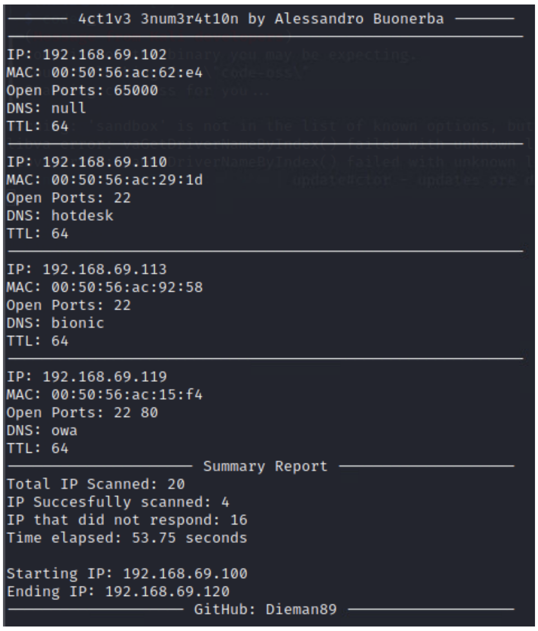
Active enumeration is when a user programmatically gather informations on a system through the use of a set of predefined commands. The most common set of informations that is usually gathered through enumeration are DNS, IPs, ports, and services.

2.2



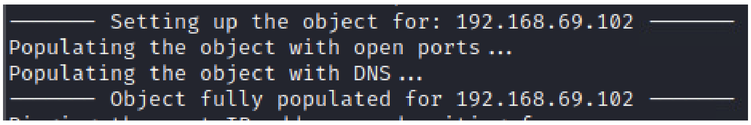
```
┌──(kali㉿kali)-[~]
└─$ python active.py
─────── 4ct1v3 3num3r4t10n by Alessandro Buonerba ───────
Scan from 192.168.69.???, enter last digits from 0 to 255
100
Scan till 192.168.69.???, enter last digits from 0 to 255
120
Pinging the next IP address and waiting for a response...
... 192.168.69.100 did not respond :(
Pinging the next IP address and waiting for a response...
... 192.168.69.101 did not respond :(
Pinging the next IP address and waiting for a response...
... 192.168.69.102 is online :)!
─────── Setting up the object for: 192.168.69.102 ───────
Populating the object with open ports...
Populating the object with DNS...
─────── Object fully populated for 192.168.69.102 ───────
Pinging the next IP address and waiting for a response...
... 192.168.69.103 did not respond :(
Pinging the next IP address and waiting for a response...
... 192.168.69.104 did not respond :(
Pinging the next IP address and waiting for a response...
... 192.168.69.105 did not respond :(
Pinging the next IP address and waiting for a response...
... 192.168.69.106 did not respond :(
Pinging the next IP address and waiting for a response...
... 192.168.69.107 did not respond :(
Pinging the next IP address and waiting for a response...
... 192.168.69.108 did not respond :(
Pinging the next IP address and waiting for a response...
█
```

Figure 2.1: scrolling-text



```
──────── 4ct1v3 3num3r4t10n by Alessandro Buonerba ────────
──────────────────────────────────────────────────────────
IP: 192.168.69.102
MAC: 00:50:56:ac:62:e4
Open Ports: 65000
DNS: null
TTL: 64
──────────────────────────────────────────────────────────
IP: 192.168.69.110
MAC: 00:50:56:ac:29:1d
Open Ports: 22
DNS: hotdesk
TTL: 64
──────────────────────────────────────────────────────────
IP: 192.168.69.113
MAC: 00:50:56:ac:92:58
Open Ports: 22
DNS: bionic
TTL: 64
──────────────────────────────────────────────────────────
IP: 192.168.69.119
MAC: 00:50:56:ac:15:f4
Open Ports: 22 80
DNS: owa
TTL: 64
────────────────────── Summary Report ────────────────────
Total IP Scanned: 20
IP Succesfully scanned: 4
IP that did not respond: 16
Time elapsed: 53.75 seconds

Starting IP: 192.168.69.100
Ending IP: 192.168.69.120
───────────────────── GitHub: Dieman89 ───────────────────
```

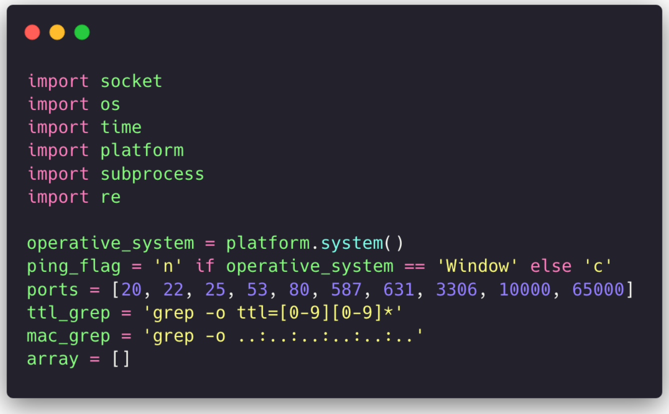
Figure 2.2: result-active-enum



```
──────── Setting up the object for: 192.168.69.102 ────────
Populating the object with open ports ...
Populating the object with DNS ...
──────── Object fully populated for 192.168.69.102 ────────
```

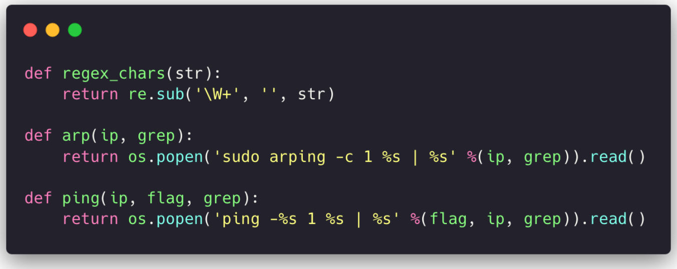
Figure 2.3: object-populating

2.3 PYTHON CODE



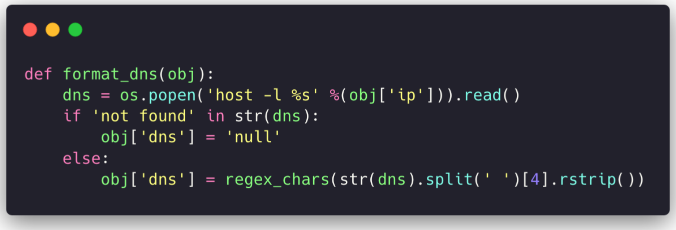
```python
import socket
import os
import time
import platform
import subprocess
import re

operative_system = platform.system()
ping_flag = 'n' if operative_system == 'Window' else 'c'
ports = [20, 22, 25, 53, 80, 587, 631, 3306, 10000, 65000]
ttl_grep = 'grep -o ttl=[0-9][0-9]*'
mac_grep = 'grep -o ..:..:..:..:..:..'
array = []
```

Figure 2.4: imports-declarations



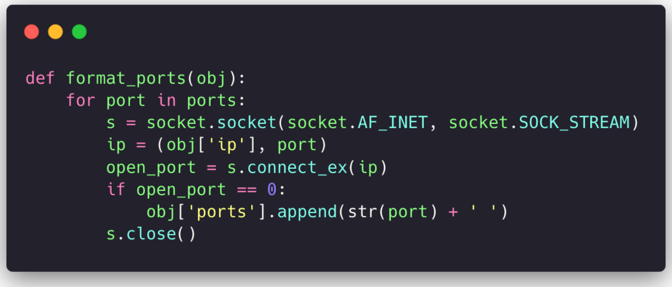
```python
def regex_chars(str):
    return re.sub('\W+', '', str)

def arp(ip, grep):
    return os.popen('sudo arping -c 1 %s | %s' %(ip, grep)).read()

def ping(ip, flag, grep):
    return os.popen('ping -%s 1 %s | %s' %(flag, ip, grep)).read()
```

Figure 2.5: regex-arp-ping



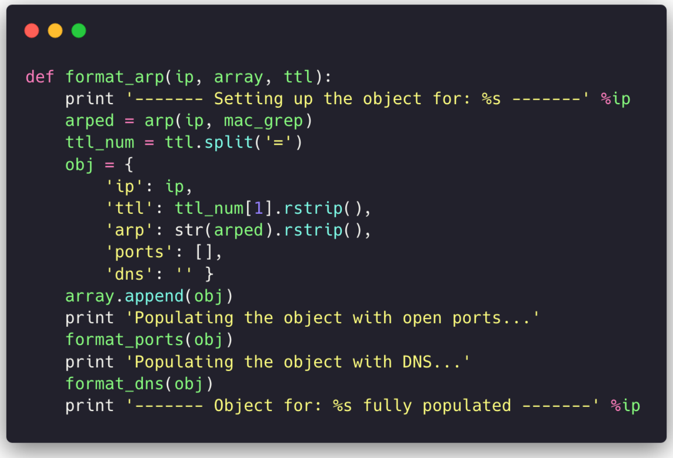
```python
def format_dns(obj):
    dns = os.popen('host -l %s' %(obj['ip'])).read()
    if 'not found' in str(dns):
        obj['dns'] = 'null'
    else:
        obj['dns'] = regex_chars(str(dns).split(' ')[4].rstrip())
```

Figure 2.6: format-dns



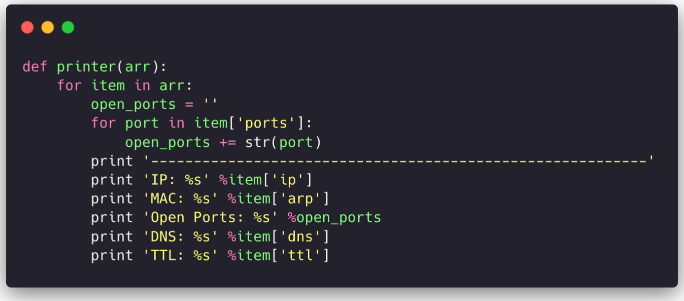
```python
def format_ports(obj):
    for port in ports:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        ip = (obj['ip'], port)
        open_port = s.connect_ex(ip)
        if open_port == 0:
            obj['ports'].append(str(port) + ' ')
        s.close()
```

Figure 2.7: format-ports



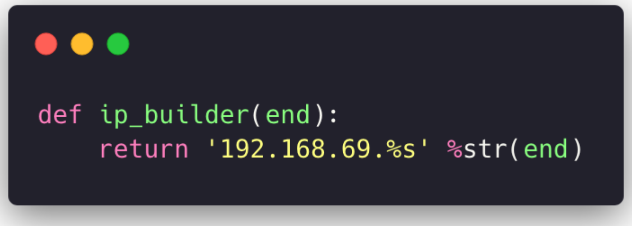
```
def format_arp(ip, array, ttl):
    print '------- Setting up the object for: %s -------' %ip
    arped = arp(ip, mac_grep)
    ttl_num = ttl.split('=')
    obj = {
        'ip': ip,
        'ttl': ttl_num[1].rstrip(),
        'arp': str(arped).rstrip(),
        'ports': [],
        'dns': '' }
    array.append(obj)
    print 'Populating the object with open ports...'
    format_ports(obj)
    print 'Populating the object with DNS...'
    format_dns(obj)
    print '------- Object for: %s fully populated -------' %ip
```

Figure 2.8: format-arp



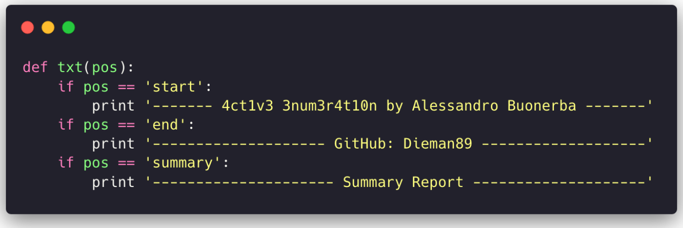
```
def printer(arr):
    for item in arr:
        open_ports = ''
        for port in item['ports']:
            open_ports += str(port)
        print '----------------------------------------------------------'
        print 'IP: %s' %item['ip']
        print 'MAC: %s' %item['arp']
        print 'Open Ports: %s' %open_ports
        print 'DNS: %s' %item['dns']
        print 'TTL: %s' %item['ttl']
```

Figure 2.9: printer



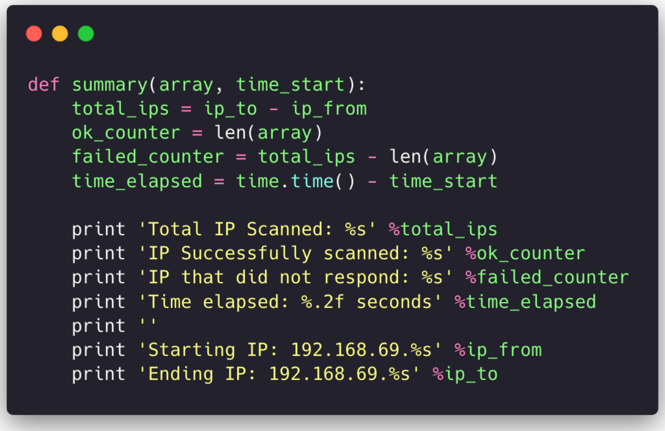
```
def ip_builder(end):
    return '192.168.69.%s' %str(end)
```

Figure 2.10: ip-builder



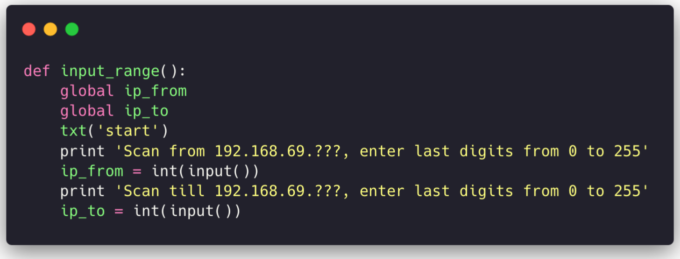
```python
def txt(pos):
    if pos == 'start':
        print '------- 4ct1v3 3num3r4t10n by Alessandro Buonerba -------'
    if pos == 'end':
        print '-------------------- GitHub: Dieman89 ------------------'
    if pos == 'summary':
        print '--------------------- Summary Report -------------------'
```

Figure 2.11: txt



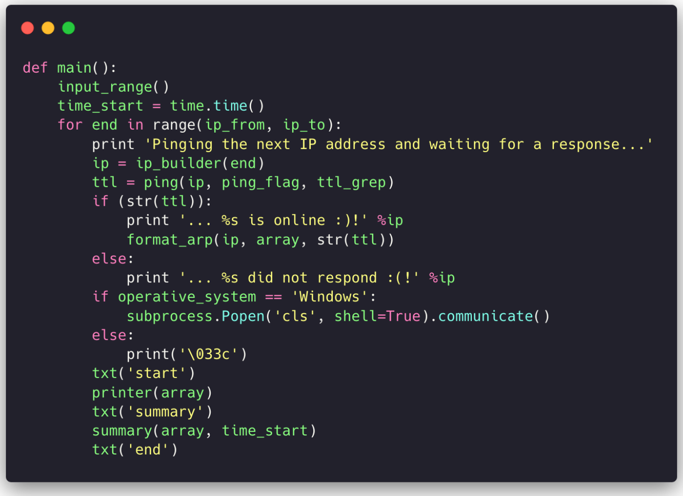
```python
def summary(array, time_start):
    total_ips = ip_to - ip_from
    ok_counter = len(array)
    failed_counter = total_ips - len(array)
    time_elapsed = time.time() - time_start

    print 'Total IP Scanned: %s' %total_ips
    print 'IP Successfully scanned: %s' %ok_counter
    print 'IP that did not respond: %s' %failed_counter
    print 'Time elapsed: %.2f seconds' %time_elapsed
    print ''
    print 'Starting IP: 192.168.69.%s' %ip_from
    print 'Ending IP: 192.168.69.%s' %ip_to
```

Figure 2.12: summary



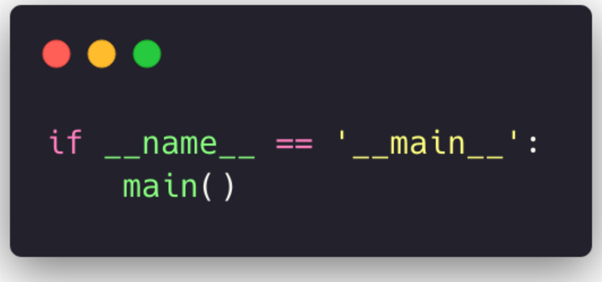
```python
def input_range():
    global ip_from
    global ip_to
    txt('start')
    print 'Scan from 192.168.69.???, enter last digits from 0 to 255'
    ip_from = int(input())
    print 'Scan till 192.168.69.???, enter last digits from 0 to 255'
    ip_to = int(input())
```

Figure 2.13: input-range



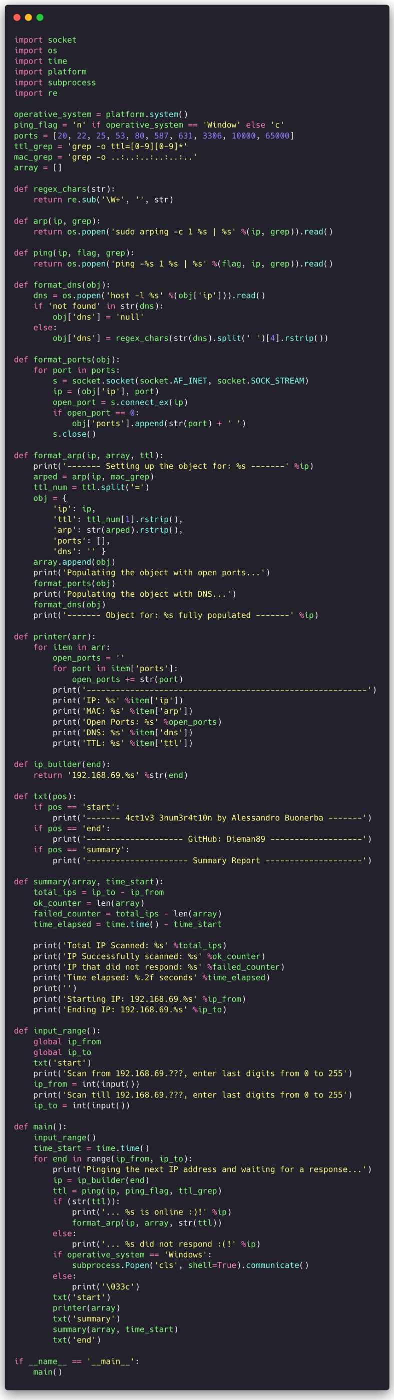
```
def main():
    input_range()
    time_start = time.time()
    for end in range(ip_from, ip_to):
        print 'Pinging the next IP address and waiting for a response...'
        ip = ip_builder(end)
        ttl = ping(ip, ping_flag, ttl_grep)
        if (str(ttl)):
            print '... %s is online :)!' %ip
            format_arp(ip, array, str(ttl))
        else:
            print '... %s did not respond :(!' %ip
        if operative_system == 'Windows':
            subprocess.Popen('cls', shell=True).communicate()
        else:
            print('\033c')
        txt('start')
        printer(array)
        txt('summary')
        summary(array, time_start)
        txt('end')
```

Figure 2.14: main



```
if __name__ == '__main__':
    main()
```

Figure 2.15: namemain



```python
import socket
import os
import time
import platform
import subprocess
import re

operative_system = platform.system()
ping_flag = 'n' if operative_system == 'Window' else 'c'
ports = [20, 22, 25, 53, 80, 587, 631, 3306, 10000, 65000]
ttl_grep = 'grep -o ttl=[0-9][0-9]*'
mac_grep = 'grep -o ..:..:..:..:..:..'
array = []

def regex_chars(str):
    return re.sub('\W+', '', str)

def arp(ip, grep):
    return os.popen('sudo arping -c 1 %s | %s' %(ip, grep)).read()

def ping(ip, flag, grep):
    return os.popen('ping -%s 1 %s | %s' %(flag, ip, grep)).read()

def format_dns(obj):
    dns = os.popen('host -l %s' %(obj['ip'])).read()
    if 'not found' in str(dns):
        obj['dns'] = 'null'
    else:
        obj['dns'] = regex_chars(str(dns).split(' ')[4].rstrip())

def format_ports(obj):
    for port in ports:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        ip = (obj['ip'], port)
        open_port = s.connect_ex(ip)
        if open_port == 0:
            obj['ports'].append(str(port) + ' ')
        s.close()

def format_arp(ip, array, ttl):
    print('------- Setting up the object for: %s -------' %ip)
    arped = arp(ip, mac_grep)
    ttl_num = ttl.split('=')
    obj = {
        'ip': ip,
        'ttl': ttl_num[1].rstrip(),
        'arp': str(arped).rstrip(),
        'ports': [],
        'dns': '' }
    array.append(obj)
    print('Populating the object with open ports...')
    format_ports(obj)
    print('Populating the object with DNS...')
    format_dns(obj)
    print('------- Object for: %s fully populated -------' %ip)

def printer(arr):
    for item in arr:
        open_ports = ''
        for port in item['ports']:
            open_ports += str(port)
        print('----------------------------------------------------------')
        print('IP: %s' %item['ip'])
        print('MAC: %s' %item['arp'])
        print('Open Ports: %s' %open_ports)
        print('DNS: %s' %item['dns'])
        print('TTL: %s' %item['ttl'])

def ip_builder(end):
    return '192.168.69.%s' %str(end)

def txt(pos):
    if pos == 'start':
        print('------- 4ct1v3 3num3r4t10n by Alessandro Buonerba -------')
    if pos == 'end':
        print('-------------------- GitHub: Dieman89 --------------------')
    if pos == 'summary':
        print('--------------------- Summary Report ---------------------')

def summary(array, time_start):
    total_ips = ip_to - ip_from
    ok_counter = len(array)
    failed_counter = total_ips - len(array)
    time_elapsed = time.time() - time_start

    print('Total IP Scanned: %s' %total_ips)
    print('IP Successfully scanned: %s' %ok_counter)
    print('IP that did not respond: %s' %failed_counter)
    print('Time elapsed: %.2f seconds' %time_elapsed)
    print('')
    print('Starting IP: 192.168.69.%s' %ip_from)
    print('Ending IP: 192.168.69.%s' %ip_to)

def input_range():
    global ip_from
    global ip_to
    txt('start')
    print('Scan from 192.168.69.???, enter last digits from 0 to 255')
    ip_from = int(input())
    print('Scan till 192.168.69.???, enter last digits from 0 to 255')
    ip_to = int(input())

def main():
    input_range()
    time_start = time.time()
    for end in range(ip_from, ip_to):
        print('Pinging the next IP address and waiting for a response...')
        ip = ip_builder(end)
        ttl = ping(ip, ping_flag, ttl_grep)
        if (str(ttl)):
            print('... %s is online :)!' %ip)
            format_arp(ip, array, str(ttl))
        else:
            print('... %s did not respond :(!' %ip)
        if operative_system == 'Windows':
            subprocess.Popen('cls', shell=True).communicate()
        else:
            print('\033c')
        txt('start')
        printer(array)
        txt('summary')
        summary(array, time_start)
        txt('end')

if __name__ == '__main__':
    main()
```

Figure 2.16: enum-full-code

BIBLIOGRAPHY

Cooper, Zach (2020). *What's the Difference between Active and Passive Reconnaissance?* URL: https://www.itpro.co.uk/penetration-testing/34465/whats-the-difference-between-active-and-passive-reconnaissance (visited on 10/07/2021).

IBM (2021). *IBM Docs*. URL: https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/aix/7.1?topic=tcpp-tcpip-address-parameter-assignment-dynamic-host-configuration-protocol (visited on 10/07/2021).

Insam, Edward (2020). *Application Layer Protocol - an Overview*. URL: https://www.sciencedirect.com/topics/computer-science/application-layer-protocol (visited on 10/07/2021).

Scarpati, Jessica (2020). *What Is Simple Network Management Protocol (SNMP)? Definition from SearchNetworking*. URL: https://www.techtarget.com/searchnetworking/definition/SNMP (visited on 10/07/2021).

Sheldon, Robert and Jessica Scarpati (2020). *What Is the Server Message Block (SMB) Protocol? How Does It Work?* URL: https://www.techtarget.com/searchnetworking/definition/Server-Message-Block-Protocol (visited on 10/07/2021).