

NETWORK EXPLOIT

SUBJECT NAME: CRYPTOGRAPHY AND NETWORK SECURITY

SUBJECT CODE: CS6008

MODULE: 4

NAME	BHUVANESHWAR S
REG.NO	2019103513
DATE	05/04/2022

AIM:

Using ARP Poison network and detect using Wireshark.

TOOLS INVOLVED:

- Kali linux VM (2)
- Ettercap
- Wireshark
- Terminal
- python
- Oracle virtual box

PROBLEM DESCRIPTION:

The Address Resolution Protocol (ARP) is a communication protocol used for discovering the link layer address, such as a MAC address, associated with a given internet layer address, typically an IPv4 address. This mapping is a critical function in the Internet protocol suite. The devices of the network peel the header of the data link layer from the protocol data unit (PDU) called frame and transfer the packet to the network layer (layer 3 of OSI) where the network ID of the packet is validated with the destination IP's network ID of the packet and if it's equal then it responds to the source with the MAC address of the destination, else the packet reaches the gateway of the network and broadcasts packet to the devices it is connected with and validates their network ID. The above process continues till the second last network device in the path reaches the destination where it gets validated and ARP, in turn, responds with the destination MAC address.

An ARP spoofing, also known as ARP poisoning, is a Man in the Middle (MitM) attack that allows attackers to intercept communication between network devices. The attack works as follows:

The attacker must have access to the network. They scan the network to determine the IP addresses of at least two devices—let's say these are a workstation and a router.

The attacker uses a spoofing tool, such as Arpspoof or Driftnet, to send out forged ARP responses.

The forged responses advertise that the correct MAC address for both IP addresses, belonging to the router and workstation, is the attacker's MAC address. This fools both router and workstation to connect to the attacker's machine, instead of to each other.

The two devices update their ARP cache entries and from that point onwards, communicate with the attacker instead of directly with each other.

The attacker is now secretly in the middle of all communications.

INPUT:

The user search or visit the insecure website.

OUTPUT:

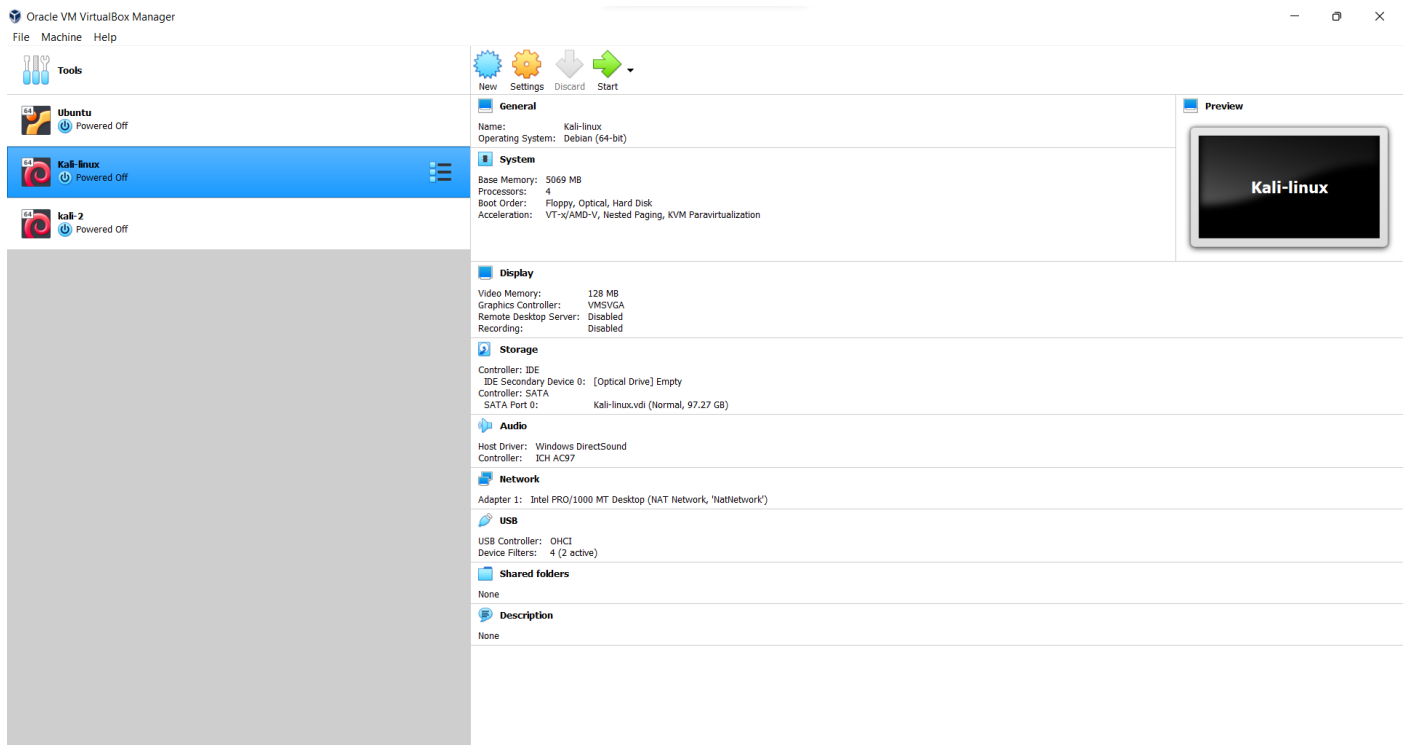
Stole personal information from the user.

SCREENSHOT:

Two kali linux machine are used to do ARP spoofing.

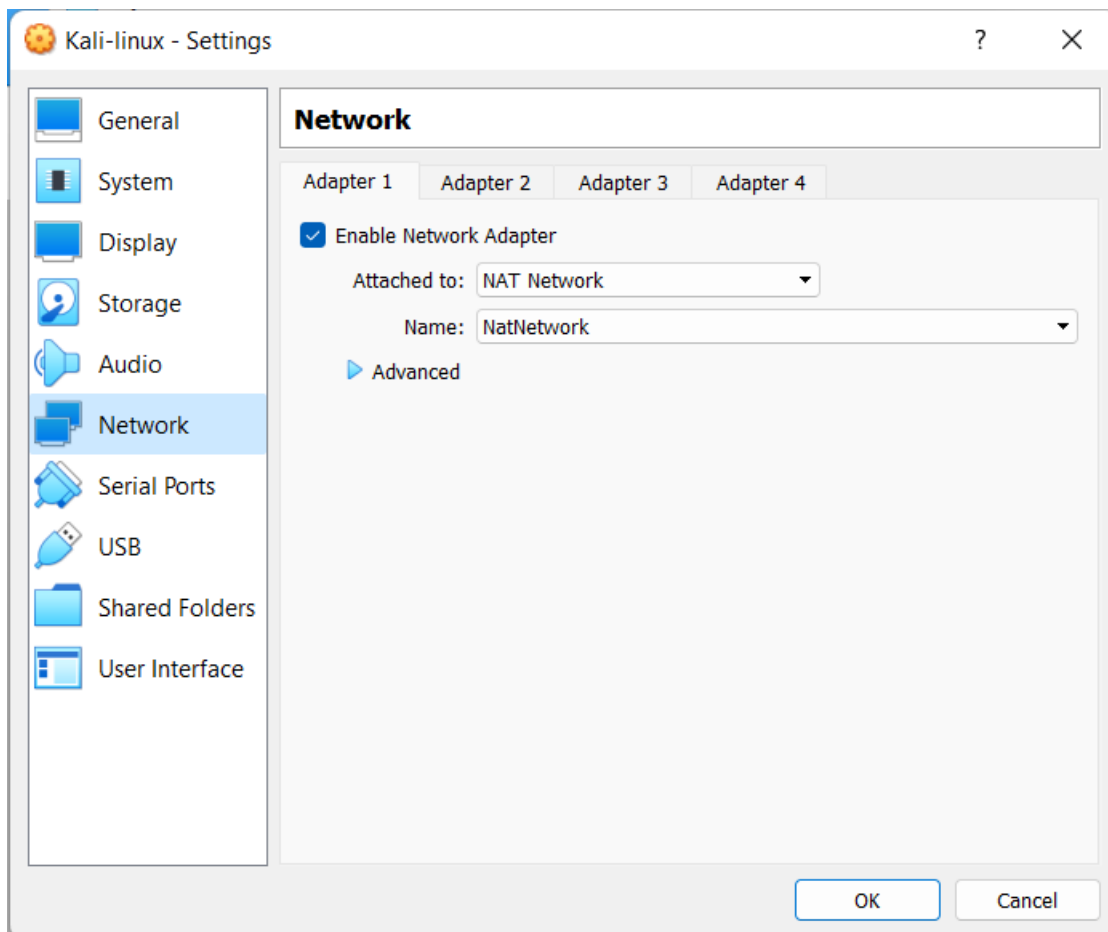
Machine 1 -> attacker

Machine 2 -> user



Here machine 1 is kali-linux and machine 2 is kali 2.

NAT network is used in both virtual machines.



Using the **ifconfig** command, to see the ip address , mac address and default gateway.

MACHINE 1

```
(bhuvan@bhuvan-kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::a00:27ff:feec:29c3 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:ec:29:c3 txqueuelen 1000 (Ethernet)
    RX packets 2 bytes 650 (650.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 25 bytes 3311 (3.2 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

MACHINE 2

```
(kali@kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.4 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fd17:625c:f037:2:9443:9396:54d:9413 prefixlen 64 scopeid 0x0<
global>
    inet6 fd17:625c:f037:2:a00:27ff:fe1d:7431 prefixlen 64 scopeid 0x0<
global>
    inet6 fe80::a00:27ff:fe1d:7431 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:1d:74:31 txqueuelen 1000 (Ethernet)
    RX packets 18 bytes 3890 (3.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 30 bytes 5320 (5.1 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags-73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Ettercap is a comprehensive suite for man in the middle attacks. It features sniffing of live connections, content filtering on the fly and many other interesting tricks. It supports active and passive dissection of many protocols and includes many features for network and host analysis

ETTERCAP – MACHINE 1 (ATTACKER)

The screenshot shows the Ettercap 0.8.3.1 (EB) interface. The Host List on the left shows the following entries:

IP Address	MAC Address	Description
10.0.2.1	52:54:00:12:35:00	
10.0.2.2	52:54:00:12:35:00	
10.0.2.3	08:00:27:88:E5:FC	
10.0.2.4	08:00:27:EAD2:47	

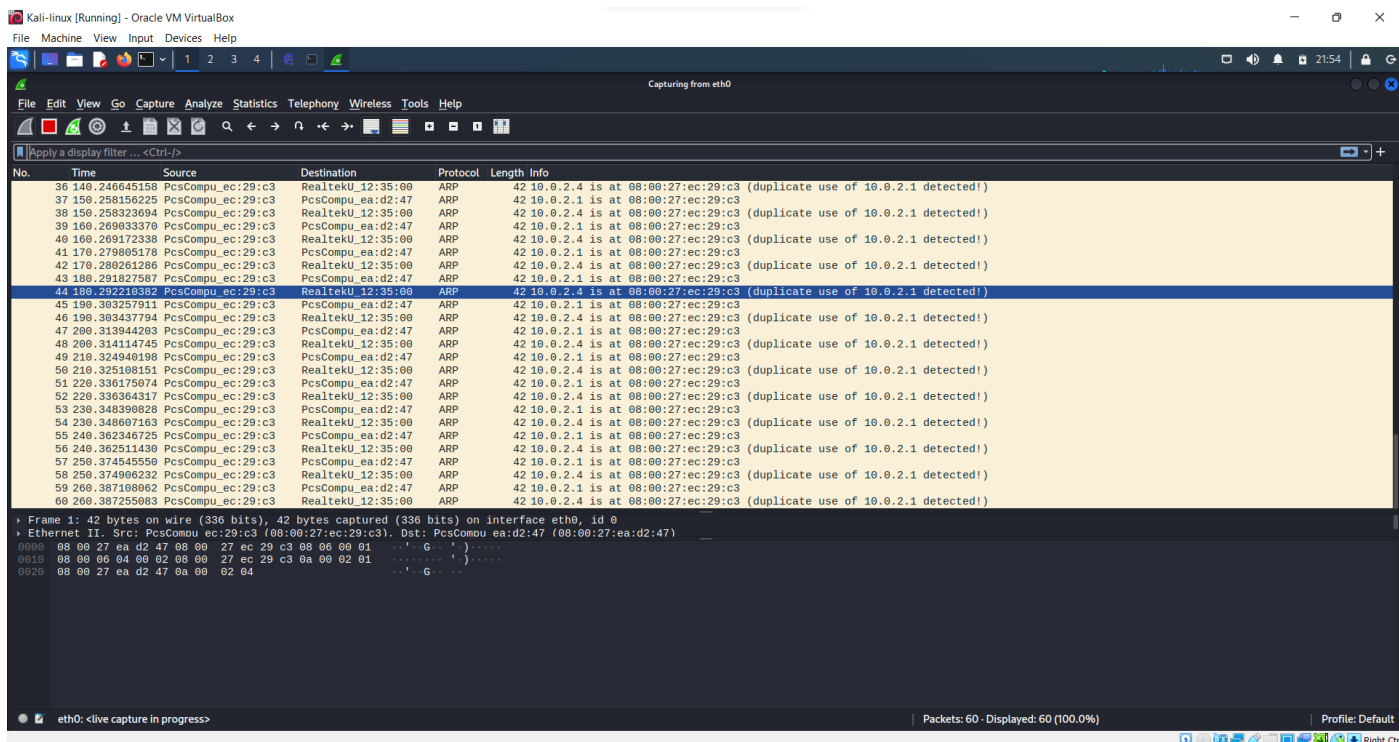
The main window displays the following status information:

- 28330 mac vendor fingerprint
- 1766 tcp OS fingerprint
- 2182 known services
- lua: no scripts were specified, not starting up!
- Starting Unified sniffing...
- Randomizing 255 hosts for scanning...
- Scanning the whole netmask for 255 hosts...
- 4 hosts added to the hosts list...
- Host 10.0.2.4 added to TARGET1
- Host 10.0.2.1 added to TARGET2
- ARP poisoning victims:
- GROUP 1: 10.0.2.4 08:00:27:EAD2:47
- GROUP 2: 10.0.2.1 52:54:00:12:35:00

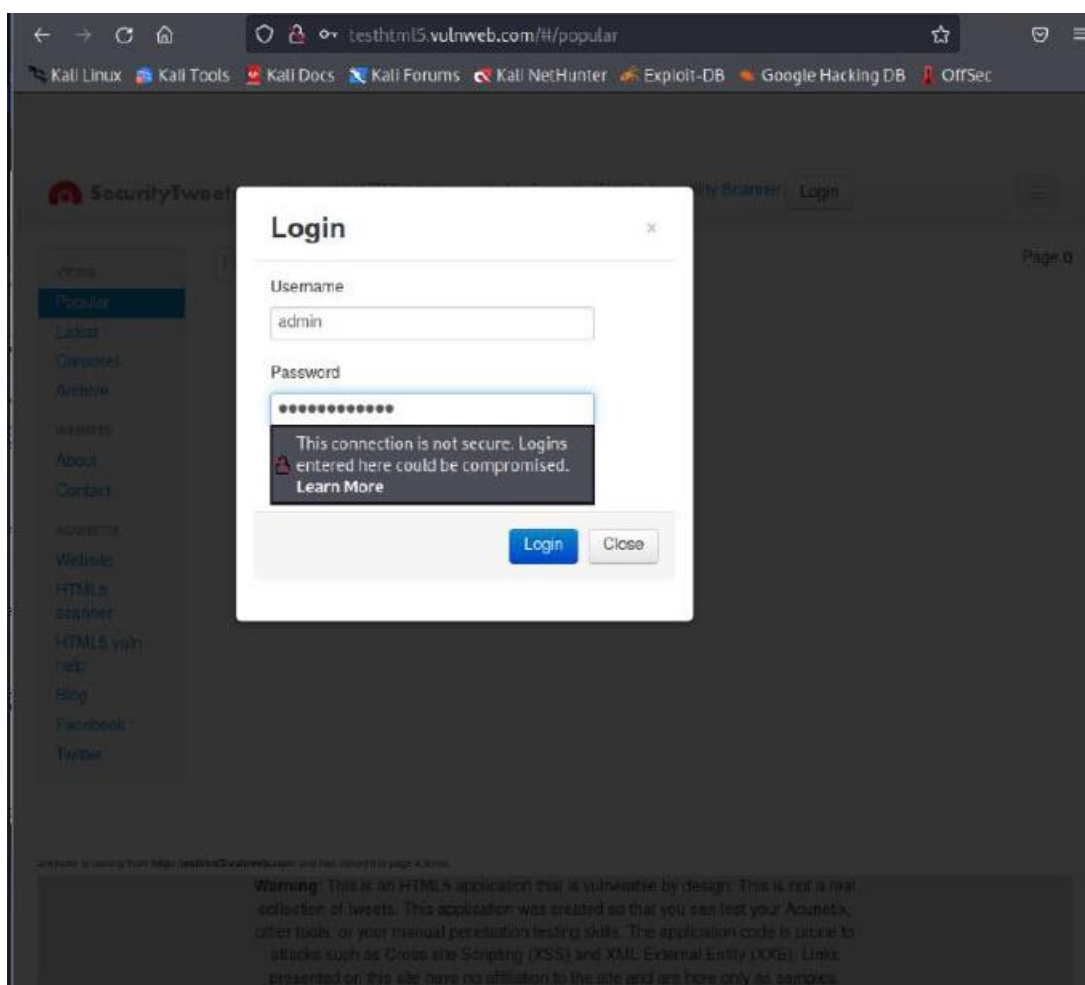
Poisoning the user ip address

WIRESHARK – MACHINE 1 (ATTACKER)

Wireshark capturing of the spoof packets



Client going to testhtml5.vulnweb.com which is a http site



Attacker can see the credential information from the user using wireshark.

*eth0						
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help						
tcp						
No.	Time	Source	Destination	Protocol	Length	Info
330	703.610230420	10.0.2.15	54.149.83.187	TCP	54	47638 → 443 [ACK] Seq=518
331	703.610965931	54.149.83.187	10.0.2.15	TCP	1514	443 → 47638 [PSH, ACK] Seq=518
332	703.610984449	10.0.2.15	54.149.83.187	TCP	54	47638 → 443 [ACK] Seq=518
333	703.611465978	54.149.83.187	10.0.2.15	TLSv1.2	660	Certificate, Server Key E
334	703.611476313	10.0.2.15	54.149.83.187	TCP	54	47638 → 443 [ACK] Seq=518
335	703.637377406	10.0.2.15	54.149.83.187	TLSv1.2	180	Client Key Exchange, Chan
336	703.639511936	10.0.2.15	54.149.83.187	TCP	54	47638 → 443 [FIN, ACK] Seq=350
337	703.640638537	54.149.83.187	10.0.2.15	TCP	60	443 → 47638 [ACK] Seq=350
338	703.678736769	54.149.83.187	10.0.2.15	TLSv1.2	212	Application Data
339	703.678766070	10.0.2.15	54.149.83.187	TCP	54	47636 → 443 [ACK] Seq=126
340	703.817375919	10.0.2.15	54.149.83.187	TLSv1.2	244	Application Data
341	703.892153115	54.149.83.187	10.0.2.15	TLSv1.2	105	Change Cipher Spec, Encry
342	703.892189126	10.0.2.4	54.149.83.187	TCP	54	47638 → 443 [RST] Seq=645
343	703.892796828	54.149.83.187	10.0.2.15	TLSv1.2	85	Encrypted Alert
344	703.892864046	10.0.2.4	54.149.83.187	TCP	54	47638 → 443 [RST] Seq=645
345	703.911197332	54.149.83.187	10.0.2.15	TCP	60	443 → 47636 [ACK] Seq=371
346	704.069875859	54.149.83.187	10.0.2.15	TLSv1.2	253	Application Data
347	704.069904503	10.0.2.15	54.149.83.187	TCP	54	47636 → 443 [ACK] Seq=145
348	704.073304234	10.0.2.15	13.33.146.110	TLSv1.3	478	Application Data
349	704.096637016	13.33.146.110	10.0.2.15	TCP	1494	443 → 58580 [PSH, ACK] Seq=203
350	704.096729941	10.0.2.15	13.33.146.110	TCP	54	58580 → 443 [ACK] Seq=203
351	704.097387610	13.33.146.110	10.0.2.15	TCP	1514	443 → 58580 [ACK] Seq=717
352	704.097387764	13.33.146.110	10.0.2.15	TCP	1514	443 → 58580 [ACK] Seq=863
353	704.097402158	10.0.2.15	13.33.146.110	TCP	54	58580 → 443 [ACK] Seq=203
354	704.097554130	10.0.2.15	13.33.146.110	TCP	54	58580 → 443 [ACK] Seq=203

Frame 154: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0

Ethernet II. Src: PcsCompu ec:29:c3 (08:00:27:ec:29:c3). Dst: RealtekU 12:35:00 (52:54:00:12:35:00)

```

0000  52 54 00 12 35 00 08 00 27 ec 29 c3 08 00 45 00  RT...5...!...)...E.
0010  00 3c d0 0d 40 00 40 06 a6 17 0a 00 02 0f 0d 21  -<...@.@...!
0020  ab 67 b9 34 01 bb 20 86 85 c0 00 00 00 00 a0 02  -g.4...
0030  fa f0 c4 c5 00 00 02 04 05 b4 04 02 08 0a f4 d3  .....
0040  f3 0c 00 00 00 00 01 03 03 07  .....

```

Transmission Control Protocol: Protocol Packets: 11966 · Displayed: 10446 (87.3%) Profile: Default

We can create our own arp spoof detection. So I wrote in python.

```

import scapy.all as scapy

def mac(ipadd):
    arp_request = scapy.ARP(pdst=ipadd)
    br = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    arp_req_br = br / arp_request
    list_1 = scapy.srp(arp_req_br, timeout=5, verbose=False)[0]
    return list_1[0][1].hwsrc

def sniff(interface):
    scapy.sniff(iface=interface, store=False, prn=process_sniffed_packet)

def process_sniffed_packet(packet):

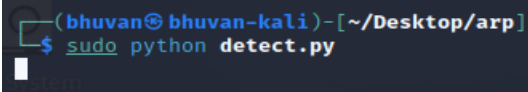
```

```
if packet.haslayer(scapy.ARP) and packet[scapy.ARP].op == 2:
    originalmac = mac(packet[scapy.ARP].psrc)
    responsemac = packet[scapy.ARP].hwsrc

    if originalmac != responsemac:
        print("[*] ALERT!! You are under attack, the ARP table is being poisoned!")

sniff("eth0")
```

OUTPUT



```
(bhuvan@bhuvan-kali)-[~/Desktop/arp]
$ sudo python detect.py
```