LAB 6 SCAPY ATTACKS ON ARP

The steps that follow should only be done on a network segment which is isolated, and over which you are the admin. Your activities in this lab will show up in the logs of any sensors running in the segment. Do not do this lab on a VM that is connected to a bridged network through 0 or more firewalls, since mistakes could result in traffic being generated in the wide area Internet. You can instantiate several Vms on your own segment if you want a variety of targets. Don't take this warning lightly.

Proceeding beyond this point is consent to being solely responsible for the consequences of your actions.

Throughout the lab I am assuming that you are on the segment 192.168.0.0/24 and that your default gateway is 192.168.0.1 – these numbers should be adjusted to reflect your environment. The variable **target** stands for the IP of some target machine (which should be something on your own isolated segment – see note above).

As a warm-up, examine the python code below and the packets generated, to determine what is being achieved. What do you think will happen if they are sent? Send them to a machine within your sandboxed host-only network and examine the traffic in Wireshark to see what actually happens.

```
fragment(IP(dst=target)/ICMP()/("DEADBEEF"*25000))

IP(src=target,dst=target)/TCP(sport=135,dport=135)
```

Part 1: ARP

You need two Ubuntu VMs for this: VM#1 is the victim, and VM#2 is the attacker.

Q1. What are the IP/MAC addresses of each VM and the host? Comment on the three pairs of addresses, their relationship to each other; how might these values have been assigned? What do the VMs and host believe is the default gateway?

In VM#1, open a terminal root shell (or open a regular shell and type 'sudo su' to become root). Learn about the ARP protocol and the 'arp' command.

Q2. Examine the arp table (arp –a –n) on VM#1 and report any entries you find.

Q3. Ping VM#2 from VM#1 and report any changes to the state of the ARP table after the ping.

Q4. Manually delete the entry for the VM#2 machine in VM#1s ARP table (arp –d <ip>). Then, run *tcpdump* **or** *wireshark* **on VM#2 to view the network traffic. Ping VM#2 again from VM#1. Report on the traffic that you see and use it to explain the way in which VM#1 obtains VM#2's MAC address.**

Q5. From VM#1, using everything observed so far, determine the MAC address of the gateway.

Now that you understand how ARP works under normal conditions, you are ready to see how ARP can be subverted. The SCAPY script (*arp.py*) that follows is an incomplete program that waits until it sees an ARP request from the victim machine (for any IP) and then responds to the ARP request with a crafted reply that provides the attacker's MAC address. In doing so, the victim is fooled into believing that the requested IP resolves in the local segment to the attacker's MAC address.

Q6. Complete the program by adding lines to the *arp_callback* **method so as to fully specify the ARP reply; you can list the fields in an ARP packet by doing ls(ARP) in Scapy. Explain the rationale for each of your lines of code.**

Q7. Provide convincing evidence that your completed *arp.py* **works as follows.**

- On VM#1, delete any ARP table entries for the gateway.

- On VM#2, run the *arp.py* **script under python, giving it the victim's (VM#1's) IP and VM#2's MAC address as command-line arguments.**

- On VM#1, ping the gateway.

- Examine the ARP table on VM#1 and provide evidence that VM#1 now erroneously believes that the MAC address of the gateway is the MAC address of VM#2 (i.e. compare your reply with your answer in Q5).

Q6. Describe some strategies to prevent ARP spoofing.

Program Listing: arp.py

```
#!/usr/bin/env python

from scapy import *

#### Adapt the following settings ####
conf.iface = 'eth0'
####

# call this function whenever a new arp packet is sniffed
def arp_callback(pkt):
   # make sure its arp, a request, and the victim is the one asking.
   if ARP in pkt and pkt[ARP].op==1 and pkt[ARP].psrc==victim_ip:
       # get the packet
       arpin = pkt[ARP]
       # show it
       print("Got an arp request")
       arpin.display()

       # make the reply arp packet
       arpout = ARP()

       # you must fill in the fields of arpout here...
       # THIS IS FOR YOU TO DO in Q6!

             # show the arp reply you are about to send
             print("Sending an arp reply")
             arpout.display()

             # now send the packet repeatedly in a loop, once every two seconds
             send(arpout, inter=2, loop=1)

             # return a string as is expected by all sniff callbacks
             return arpout.sprintf("ARP Reply sent!")

   # main program
   # Example usage: ./arp.py 192.168.1.102 00:0C:29:2C:39:2C

   if len(sys.argv) != 3:
      print "Usage: ./arp.py <victim_ip> <impersonating-mac>"
      sys.exit(1)

   victim_ip = sys.argv[1]
   impersonating_mac = sys.argv[2]

   # wait for upto 10 arps and call the arp_callback for each
   sniff(prn=arp_callback, filter="arp", count=10)
```

Q7. Suppose that as a network administrator you wanted to write a SCAPY script that would *detect* an ARP spoofing attack such as the one conducted by *arp.py*. Give a high-level description of how your SCAPY script would work.

Q8* Extra Credit. Implement your solution to Q7 in SCAPY and demonstrate that it works.