IP/ICMP Attacks Lab

1 Overview

The objective of this lab is for students to gain the first-hand experience on various attacks at the IP layer. Some of the attacks may not work anymore, but their underlying techniques are quite generic, and it is important for students to learn these attacking techniques, so when they design or analyze network protocols, they are aware of what attackers can do to protocols. Moreover, due to the complexity of IP fragmentation, spoofing fragmented IP packets is non-trivial. Constructing spoofed IP fragments is a good practice for students to hone their packet spoofing skills, which are essential in network security. We will use Scapy to conduct packet spoofing. This lab covers the following topics:

- The IP and ICMP protocols
- IP Fragmentation and the related attacks
- ICMP redirect attack
- Routing and reverse path filtering

Readings and related topics. Detailed coverage of the IP protocol and the attacks at the IP layer can be found in the following:

- Section 4 of the SEED Lecture, *Internet Security: A Hands-on Approach*, taught by Wenliang Du. See details on this page: `https://www.handsonsecurity.net/video.html`.
- The IP and ICMP protocols are not covered in the current edition of the SEED Book, but they will be added to the next edition.

Lab environment. This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website.

2 Tasks 1: IP Fragmentation

Two VMs are needed for this task. They should be connected to the same network, so they can communicate with each other.

2.1 Task 1.a: Conducting IP Fragmentation

In this task, students need to construct a UDP packet and send it to a UDP server. They can use "`nc -lu 9090`" to start a UDP server. Instead of building one single IP packet, students need to divide the packet into 3 fragments, each containing 32 bytes of data (the first fragment contains 8 bytes of the UDP header plus 32 bytes of data). If everything is done correctly, the server will display 96 bytes of data in total. The following is a sample code for constructing the first fragment.

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

# Construct IP header
ip = IP(src="1.2.3.4", dst="10.0.0.15")
ip.id    = 1000  # Identification
ip.frag  = 0     # Offset of this IP fragment
ip.flags = 1     # Flags

# Construct UDP header
udp = UDP(sport=7070, dport=9090)
udp.len  = 200   # This should be the combined length of all fragments

# Construct payload
payload = ’A’ * 80    # Put 80 bytes in the first fragment

# Construct the entire packet and send it out
pkt = ip/udp/payload  # For other fragments, we should use ip/payload
pkt[UDP].checksum = 0 # Set the checksum field to zero
send(pkt, verbose=0)
```

It should be noted that the UDP checksum field needs to be set correctly. If we do not set this field, Scapy will calculate the checksum for us, but this checksum will only be based on the data in the first fragment, which is incorrect. If we set the checksum field to zero, Scapy will leave it alone. Moreover, the recipient will not validate the UDP checksum if it sees a zero in the checksum field, because in UDP, checksum validation is optional.

If you use Wireshark to observe traffic, it should also be noted that by default, Wireshark will reassemble fragments in the last fragment packet and show it as a complete IP/UDP packet. To change that behavior, we should disable IP fragment reassembly in Wireshark preferences. Click the following menu sequence: `Edit → Preferences`; click the `Protocols` dropdown menu, find and click `IPv4`. Uncheck the `"Reassemble fragmented IPv4 datagrams"` option.

2.2 Task 1.b: IP Fragments with Overlapping Contents

Similar to Task 1.a, students also need to construct 3 fragments to send data to a UDP server. The size of each fragment is up to students. The objective of this task is to create overlapping fragments. In particular, the first two fragments should overlap. Please use experiments to show what will happen when the overlapping occurs. Please try the following overlapping scenarios separately:

- The end of the first fragment and the beginning of the second fragment should have `K` bytes of overlapping, i.e., the last `K` bytes of data in the first fragment should have the same offsets as the first `K` bytes of data in the second fragment. The value of `K` is decided by students (`K` should be greater than zero and smaller than the size of either fragment). In the reports, students should indicate what their `K` values are.

- The second fragment is completely enclosed in the first fragment. The size of the second fragment must be smaller than the first fragment (they cannot be equal).

Please try two different orders: (1) sending the first fragment first, and (2) sending the second fragment first. Please report whether the results will be the same.

2.3 Task 1.c: Sending a Super-Large Packet

As we know, the maximal size for an IP packet is 2^{16} octets, because the length field in the IP header has only 16 bits. However, using the IP fragmentation, we can create an IP packet that exceeds this limit. Please construct such a packet, send it to the UDP server, and see how the server responds to this situation. Please report your observation.

2.4 Task 1.d: Sending Incomplete IP Packet

In this task, we are going to use Machine A to launch a Denial-of-Service attack on Machine B. In the attack, Machine A sends a lot of incomplete IP packets to B, i.e., these packets consist of IP fragments, but some fragments are missing. All these incomplete IP packets will stay in the kernel, until they time out. Potentially, this can cause the kernel to commit a lot of kernel memory. In the past, this had resulted in denial-of-service attacks on the server. Please try this attack and describe your observation.

3 Task 2: ICMP Redirect Attack

An ICMP redirect 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.

In our VM, there is a countermeasure against the ICMP redirect attack. Before doing the task, we need to turn off the countermeasure by configuring the operating system to accept ICMP redirect messages.

```
$ sudo sysctl net.ipv4.conf.all.accept_redirects=1
```

For this task, we should have two VMs, the victim VM (Host A) and the attacker VM (Host M). Students should pick a destination B, which should be a host outside of our local network (e.g., an outside web server). Normally, when A sends a packet to B, the packet will go to the router provided by VirtualBox (usually it is `10.0.2.1` if we use the default IP prefix for NAT Network).

The objective of this task is to launch an ICMP redirect attack on Host A from Host M, such that when Host A sends packets to B, it will use M as the router, and hence sends those packets to M. Since M is controlled by the attacker, the attacker can intercept the packets, make changes, and then send the modified packets out. This is a form of the Man-In-The-Middle (MITM) attack. For the simplicity of this lab, students are not required to conduct the MITM part; they only need to demonstrate that their ICMP redirect attacks can successfully redirect packets from A to B.

Code skeleton. A code skeleton is provided in the following, with some of the essential parameters left out. Students should fill in the proper values in the places marked by `@@@@`.

```
#!/usr/bin/python3

from scapy.all import *

ip = IP(src = @@@@,  dst = @@@@)
icmp = ICMP(type=@@@@, code=@@@@)
icmp.gw = @@@@

# The enclosed IP packet should be the one that
# triggers the redirect message.
ip2 = IP(src = @@@@, dst = @@@@)
```

```
send(ip/icmp/ip2/UDP());
```

Verification. To verify whether the ICMP redirect attack is successful, we can use the "`ip route get`" command to see what router will be used for a packet destination. For example, if we want to know what router will be used for packets going to `8.8.8.8`, we can use the following command:

```
$ ip route get 8.8.8.8
8.8.8.8 via 10.0.2.1 dev enp0s3  src 10.0.2.4
    cache
```

Questions. Please conduct the following experiments, and explain your observations:

1. Can you use ICMP redirect attacks to redirect to a remote machine? Namely, the IP address assigned to `icmp.gw` is a computer not on the local LAN. Please show your experiment result, and explain your observation.
2. Can you use ICMP redirect attacks to redirect to a non-existing machine on the same network? Namely, the IP address assigned to `icmp.gw` is a local computer that is either offline or non-existing. Please show your experiment result, and explain your observation.

4 Task 3: Routing and Reverse Path Filtering

The objective of this task is two-fold: get familiar with routing, and understand a spoof-prevention mechanism called reverse path filtering, which prevents outside from spoofing

4.1 Task 3.a: Network Setup

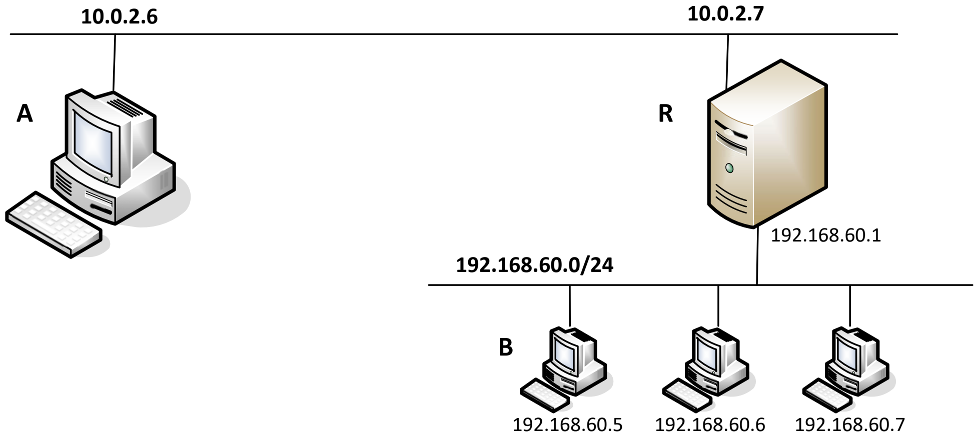


Figure 1: Network setup

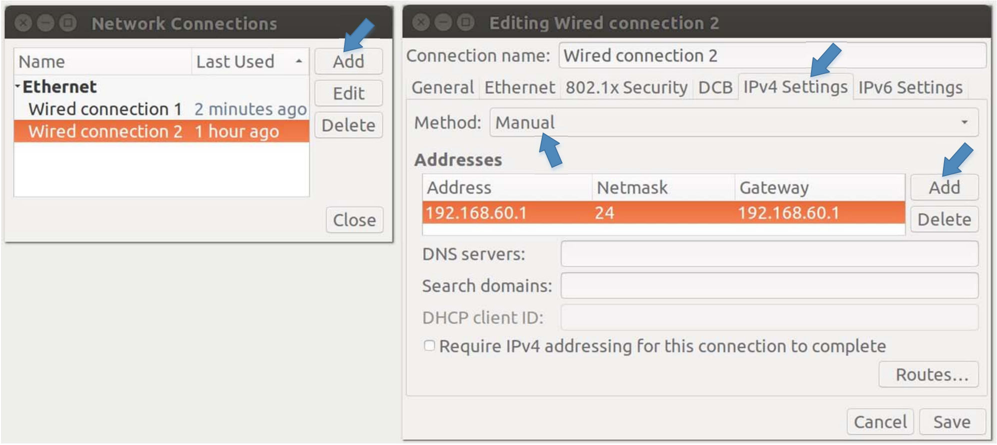
In this task, we will set up two networks and three VMs, `A`, `B`, and `R`. The network setup is depicted in Figure 1. The IP addresses in the figure are only for illustration purposes; students can choose different IP addresses in their setup.

The 10.0.2.0/24 network should use the "NAT Network" adaptor in VirtualBox. Both Machines A and R are attached to this network, so they can directly communicate with each other. The second network 192.168.60.0/24 should use the "Internal Network" adaptor in VirtualBox. Only Machines R and B are attached to this network. Therefore, Machines A and B cannot directly communicate with each other. The figure shows three machines inside the 192.168.60.0/24 network, but students only need to include one VM in their setup.

Since Machine R is attached to both networks, it needs to have two network adaptors, one using the "NAT Network" type and the other using the "Internal Network". In VirtualBox, by default, only one network adaptor is enabled. We can go to the VM Settings to enable the second one, but we need to shutdown the VM first; otherwise, we will find out that the "Adaptor 2" button is grayed out.

Set up static IP addresses. VMs attached to the "NAT Network" network will automatically get their IP addresses from the DHCP server, but for VMs on the "Internal Network", VirtualBox does not provide DHCP, so the VM must be statically configured. To do this, click the network icon on the top-right corner of the desktop, and select "Edit Connections". You will see a list of "Wired connections", one for each of the network adaptors used by the VM. For Machine B, there is only one connection, but for Machine R, we will see two. To make sure that you pick the one attached to the "Internal Network" adapter, You can check the MAC address displayed in the pop-up window after you have picked a connection to edit. Compare this MAC address with the one displayed by the ifconfig command, and you will know whether you have picked the right connection or not.

After selecting the right connection to edit, pick the "ipv4 Settings" tab and select the "Manual" method, instead of the default "Automatic (DHCP)". Click the "Add" button to set up the new IP address for the VM. See Figure 2 for details.



Name	Last Used
Ethernet	
Wired connection 1	2 minutes ago
Wired connection 2	1 hour ago

Connection name: Wired connection 2

Method: Manual

Addresses

Address	Netmask	Gateway
192.168.60.1	24	192.168.60.1

DNS servers:

Search domains:

DHCP client ID:

□ Require IPv4 addressing for this connection to complete

Figure 2: Manually set up the IP address for the "Internal Network" adaptor.

4.2 Task 3.b: Routing Setup

The objective of this task is to configure the routing on Machines A, B and R, so A and B can communicate with each other. Students can use the following commands to set up the routing tables on these three machines.

```
// List all the entries in the routing table
```

```
$ ip route

// Add an entry
$ sudo ip route add <network> dev <interface> via <router ip>

// Delete a routing entry
$ sudo ip route del ...
```

We need to configure R as a router. Unless specifically configured, a computer will only act as a host, not as a gateway. Machine R needs to forward packets, so it needs to function as a gateway. We need to enable the IP forwarding for a computer to behave like a gateway. IP forwarding can be enabled using the following command:

```
$ sudo sysctl net.ipv4.ip_forward=1
```

In your lab report, please demonstrate (using screenshots) that you can successfully `ping` and `telnet` to B from A, and vice versa.

4.3 Task 3.c: Reverse Path Filtering

Linux kernel implements a filtering rule called *reverse path filtering*, which ensures the symmetric routing rule. When a packet with the source IP address X comes from an interface (say I), the OS will check whether the return packet will return from the same interface, i.e., whether the routing for packets going to X is symmetric. To check that, the OS conducts a reverse lookup, finds out which interface will be used to route the return packets back to X. If this interface is not I, i.e., different from where the original packet comes from, the routing path is asymmetric. In this case, the kernel will drop the packet.

The reverse path filtering mechanism can effectively prevent outsiders from spoofing packets with the source IP address that belongs to the internal network. For example, in the Network depicted in Figure 2, Machine A cannot send a spoofed packets to Machine B with a source IP address belonging to the `192.168.60.0/24` network.

In this task, students will conduct an experiment to see the reverse path filtering in action. Students should send three spoofed packets on Machine A. All these packets should be sent to Machine B, but the source IP addresses should use one of the following:

- An IP address belonging to the network `10.0.2.0/24`.
- An IP address belonging to the internal network `192.168.60.0/24`.
- An IP address belonging to the Internet, such as `1.2.3.4`.

During the experiments, students run `Wireshark` on both Machine B and R, and check whether the spoofed packets are forwarded to the internal network or not by R. When running `Wireshark` on machine R, check the traffic on both network interfaces. If the spoofed packet is forwarded, it should appear in both interfaces; otherwise, we will only see the packet from the incoming interface. Please describe and explain your observations.

5 Submission

Students need to submit a detailed lab report to describe what they have done, what they have observed, and how they interpret the results. Reports should include evidences to support the observations. Evidences include packet traces, screenshots, etc. Reports should also list the important code snippets with explanations. Simply attaching code without any explanation will not receive credits.