ETHICAL HACKING
LAB SERIES

Lab 10: Packet Crafting with Scapy

Material in this Lab Aligns to the Following Certification Domains/Objectives
SANS GPEN Objective
5: Exploitation Fundamentals

Document Version: 2016-03-09

Contents

Introduction

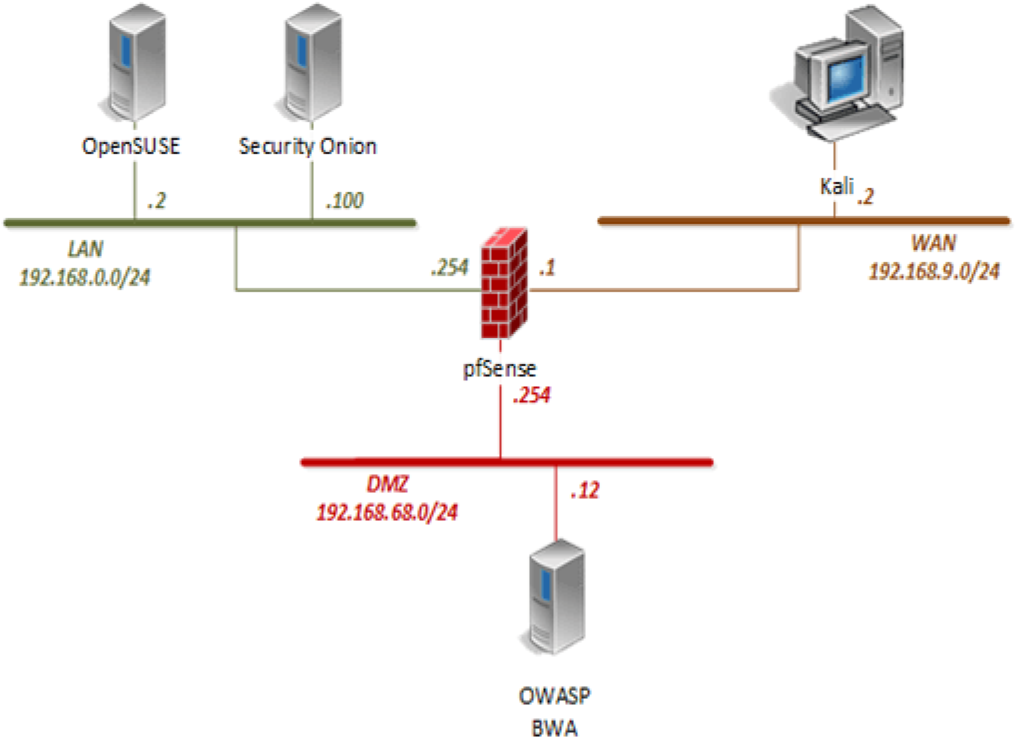
Building a packet field-by-field demonstrates how someone could manipulate the packet traffic entering or leaving a network. This lab shows how to build packets layer-by-layer using Scapy, a packet manipulation tool and then implementing the finished packets to perform various network functions.

Objective

In this lab, you will be conducting ethical hacking practices using various tools. You will be performing the following tasks:

1. Creating Packets with Scapy
2. Sending Crafted Packets

Pod Topology



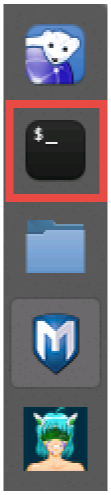
Lab Settings

The information in the table below will be needed in order to complete the lab. The task sections below provide details on the use of this information.

Virtual Machine	IP Address	Account (if needed)	Password (if needed)
Kali Linux	192.168.9.2	root	toor
pfSense	192.168.0.254 192.168.68.254 192.168.9.1	admin	pfsense
OWASP Broken Web App	192.168.68.12	root	owaspbwa
OpenSUSE	192.168.0.2	osboxes	osboxes.org
Security Onion	192.168.0.100	ndg	password123

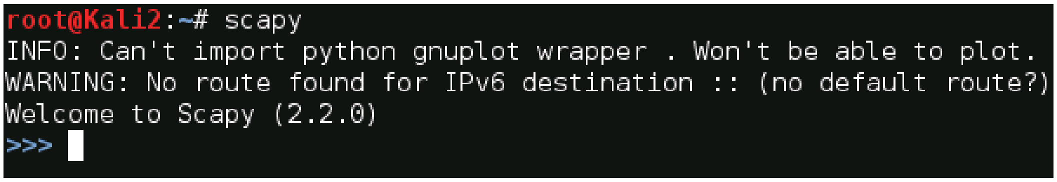
1 Creating Packets with Scapy

1. Click on the **Kali** graphic on the *topology page*.
2. Click anywhere within the *Kali* console window and press **Enter** to display the login prompt.
3. Enter `root` as the *username*. Click **Next**.
4. Enter `toor` as the *password*. Click **Sign In**.
5. Open the *Terminal* by clicking on the **Terminal** icon located on the left panel.



6. In the new *Terminal* window, type the command below to initialize the Scapy application. Press **Enter**.

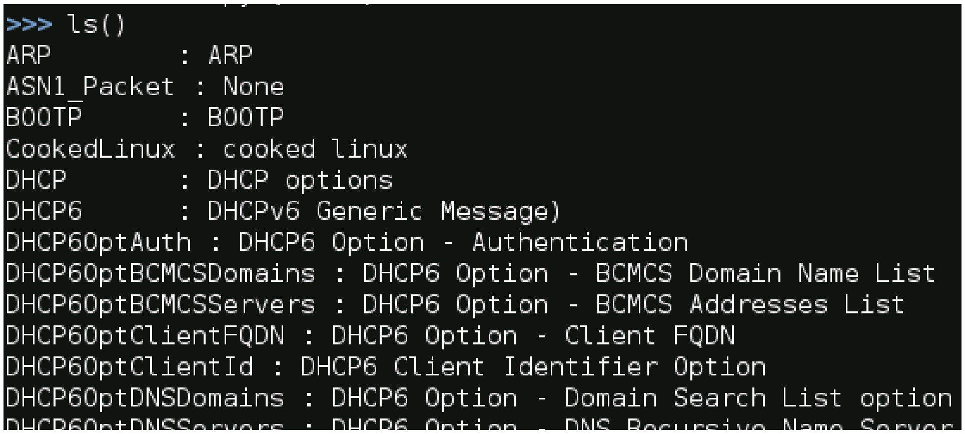
```
scapy
```



```
root@Kali2:~# scapy
INFO: Can't import python gnuplot wrapper . Won't be able to plot.
WARNING: No route found for IPv6 destination :: (no default route?)
Welcome to Scapy (2.2.0)
>>> 
```

7. List out all of the protocols and layers available for packet manipulation by typing the command below followed by pressing the **Enter** key.

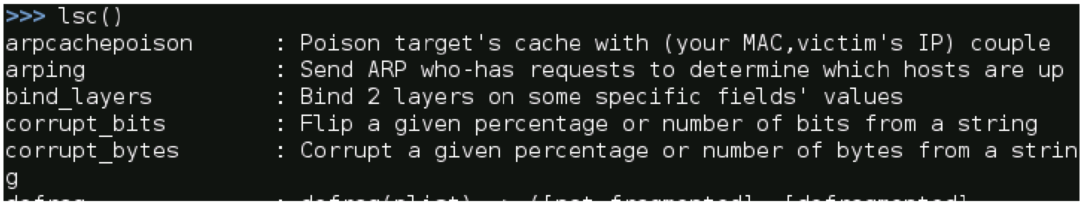
```
ls()
```



```
>>> ls()
ARP          : ARP
ASN1_Packet : None
BOOTP        : BOOTP
CookedLinux : cooked linux
DHCP         : DHCP options
DHCP6        : DHCPv6 Generic Message)
DHCP6OptAuth : DHCP6 Option - Authentication
DHCP6OptBCMCSDomains : DHCP6 Option - BCMCS Domain Name List
DHCP6OptBCMCSServers : DHCP6 Option - BCMCS Addresses List
DHCP6OptClientFQDN : DHCP6 Option - Client FQDN
DHCP6OptClientId : DHCP6 Client Identifier Option
DHCP6OptDNSDomains : DHCP6 Option - Domain Search List option
DHCP6OptDNSServers : DHCP6 Option - DNS Recursive Name Server
```

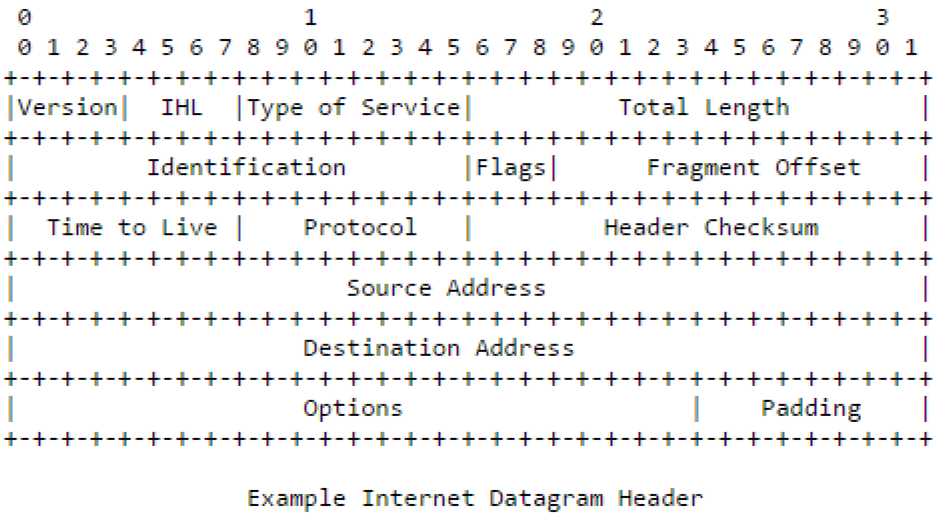
8. Enter the command below to list the available commands.

```
lsc()
```



```
>>> lsc()
arpcachepoison      : Poison target's cache with (your MAC,victim's IP) couple
arping              : Send ARP who-has requests to determine which hosts are up
bind_layers         : Bind 2 layers on some specific fields' values
corrupt_bits        : Flip a given percentage or number of bits from a string
corrupt_bytes       : Corrupt a given percentage or number of bytes from a strin
g
defrag              : defrag(plist) -> ([not fragmented], [defragmented]
```

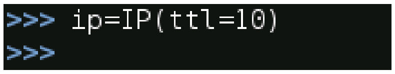
9. To build a simple IP packet, use the RFC 791 to define the IP protocol. The diagram below lists the fields in an IP packet header.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                  Example Internet Datagram Header
```

10. Enter the command below within the Scapy prompt.

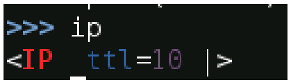
```
ip=IP(ttl=10)
```



```
>>> ip=IP(ttl=10)
>>>
```

11. Enter the command below.

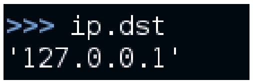
```
ip
```



```
>>> ip
<IP  ttl=10 |>
```

12. Identify the current IP destination by entering the command below.

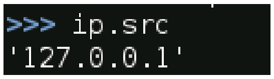
```
ip.dst
```



```
>>> ip.dst
'127.0.0.1'
```

13. Identify the current IP source by entering the command below.

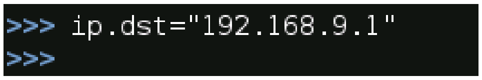
```
ip.src
```



```
>>> ip.src
'127.0.0.1'
```

14. Change the IP destination.

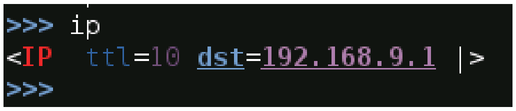
```
ip.dst=”192.168.9.1”
```



```
>>> ip.dst="192.168.9.1"
>>>
```

15. Verify the modifications.

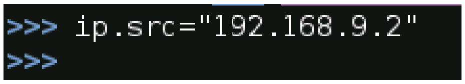
```
ip
```



```
>>> ip
<IP  ttl=10 dst=192.168.9.1 |>
>>>
```

16. Change the IP source address.

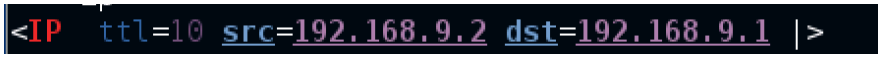
```
ip.src=”192.168.9.2”
```



```
>>> ip.src="192.168.9.2"
>>>
```

17. Verify the modifications including the source, destination and TTL values.

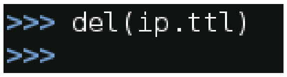
```
ip
```



```
<IP  ttl=10 src=192.168.9.2 dst=192.168.9.1 |>
```

18. With the *TTL*, *source address*, and *destination address* populated, remove the *TTL* and set it to the default *TTL* specified in the *RFC*. Enter the command below.

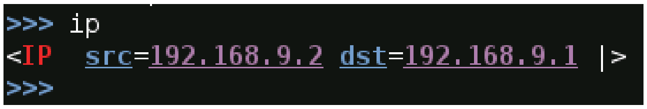
```
del(ip.ttl)
```



```
>>> del(ip.ttl)
>>>
```

19. Verify the removal.

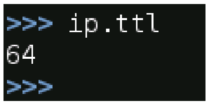
```
ip
```



```
>>> ip
<IP  src=192.168.9.2 dst=192.168.9.1 |>
>>>
```

20. Modify the *TTL* to the *RFC* value of **64**.

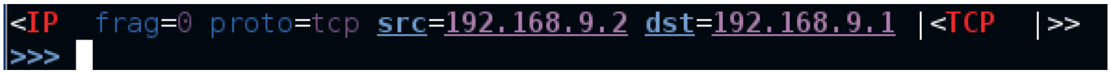
```
ip.ttl
```



```
>>> ip.ttl
64
>>>
```

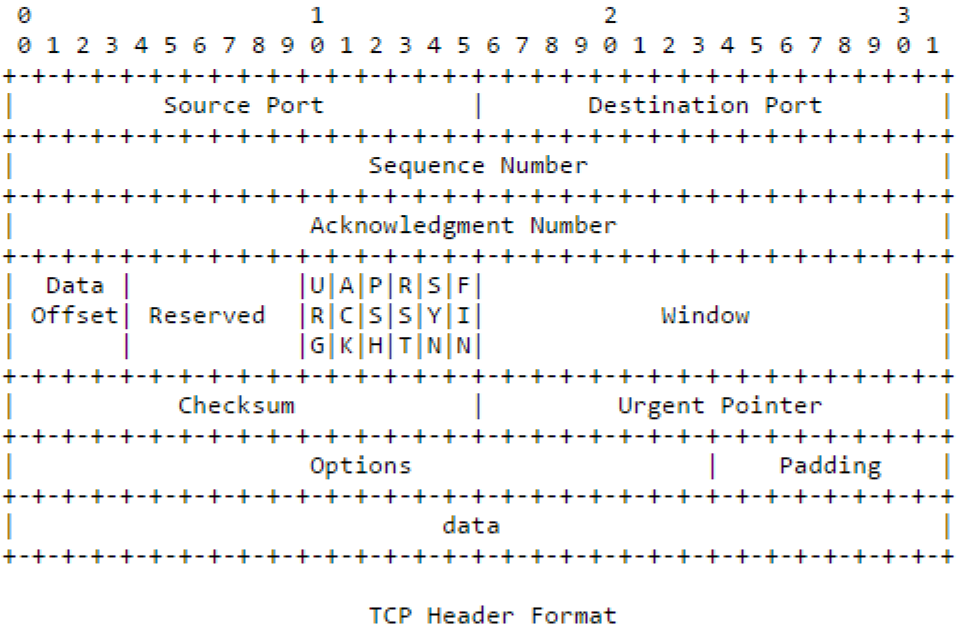
21. Add additional protocol layers by adding *TCP* on top of *IP*.

```
ip/TCP()
```



```
<IP  frag=0 proto=tcp src=192.168.9.2 dst=192.168.9.1 |<TCP  |>>
>>>
```

22. Analyze the *TCP* header from the *RFC 793*.



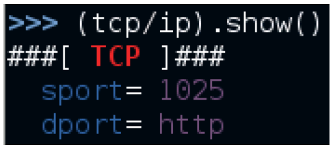
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                         TCP Header Format
```

23. Add some information to the TCP protocol fields.

```
tcp=TCP(sport=1025, dport=80)
```

24. Show the TCP stack.

```
(tcp/ip).show()
```

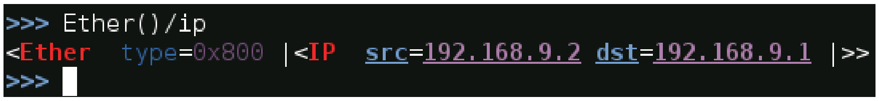


```
>>> (tcp/ip).show()
###[ TCP ]###
  sport= 1025
  dport= http
```

> Notice the packet should now have a *TCP* header with a configured *source port* of *1025* and a *destination port* of *80* stacked on the *IP* protocol.

25. Add an Ethernet layer.

```
Ether()/ip
```

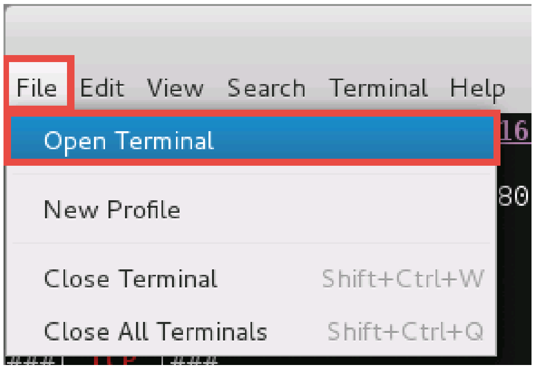


```
>>> Ether()/ip
<Ether  type=0x800 |<IP  src=192.168.9.2 dst=192.168.9.1 |>>
>>> 
```

26. Leave the *Terminal* open for the next task.

2 Sending Crafted Packets

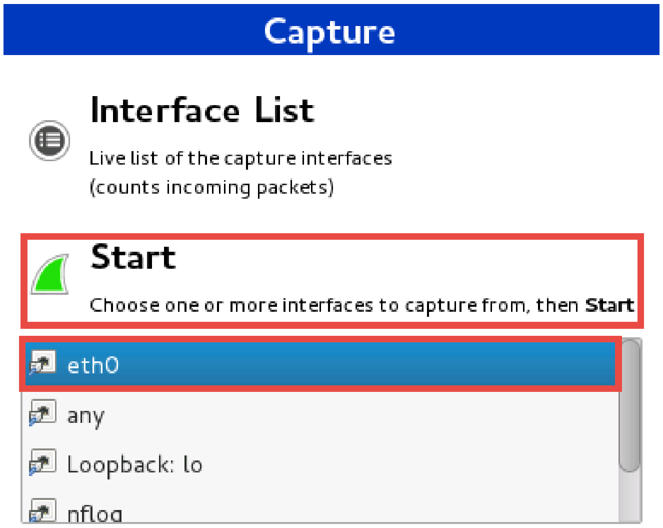
1. Launch a new **Terminal** by clicking the **File** drop-down menu option from the already existing *Terminal* window and select **Open Terminal**.



2. In the new *Terminal* window, type the command below followed by pressing the **Enter** key.

```
wireshark
```

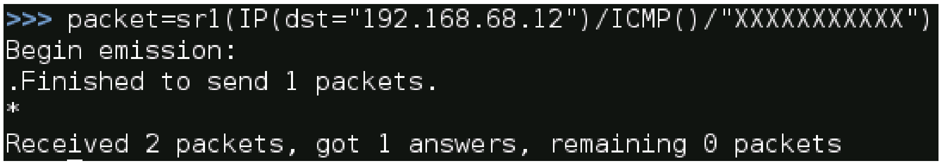
3. Click **OK** if prompted with an error message to continue.
4. If prompted with a warning message about running *Wireshark* as the root user, click **OK**.
5. Within the *Wireshark* window, select the **eth0** interface from the *Capture* panel and click **Start**.


Capture
Interface List
Live list of the capture interfaces
(counts incoming packets)
Start
Choose one or more interfaces to capture from, then Start
eth0
any
Loopback: lo
nflog

6. Navigate back to the **Terminal** window with the *Scapy* prompt.

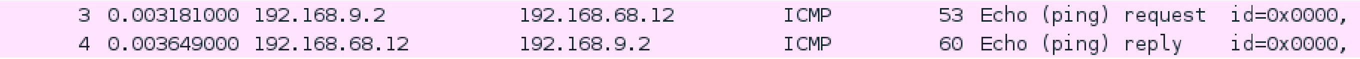
7. Generate a single *ICMP* packet to be sent to the *OWASP* machine. Enter the command below.

```
packet=sr1(IP(dst=”192.168.68.12”)/ICMP()/”XXXXXXXXXXX”)
```



```
>>> packet=sr1(IP(dst="192.168.68.12")/ICMP()/"XXXXXXXXXXX")
Begin emission:
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
```

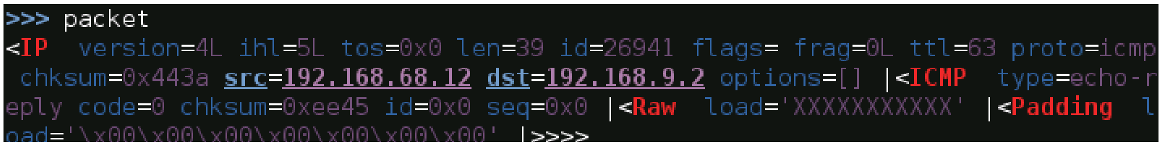
8. Navigate back to the **Wireshark** window.
9. Notice the *Scapy* crafted packet has successfully sent an *ICMP* request to the *OWASP* VM with a response.



```
   3 0.003181000 192.168.9.2        192.168.68.12      ICMP       53 Echo (ping) request  id=0x0000,
   4 0.003649000 192.168.68.12      192.168.9.2        ICMP       60 Echo (ping) reply    id=0x0000,
```

10. Navigate back to the **Terminal** with the *Scapy* prompt.
11. Enter the command below to show the contents of the packet which was created.

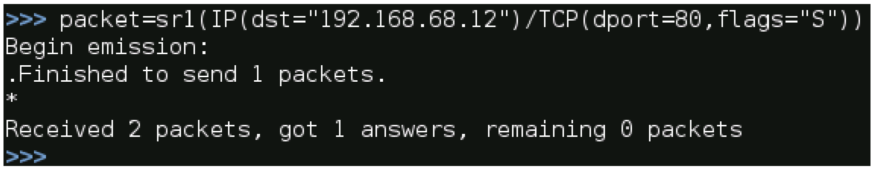
```
packet
```



```
>>> packet
<IP  version=4L ihl=5L tos=0x0 len=39 id=26941 flags= frag=0L ttl=63 proto=icmp
 chksum=0x443a src=192.168.68.12 dst=192.168.9.2 options=[] |<ICMP  type=echo-r
eply code=0 chksum=0xee45 id=0x0 seq=0x0 |<Raw  load='XXXXXXXXXXX' |<Padding  l
oad='\x00\x00\x00\x00\x00\x00\x00' |>>>>
```

12. Enter the command below in an attempt to initiate a simple SYN scan on a single port.

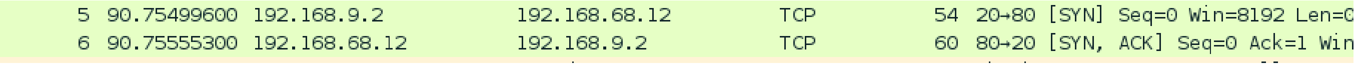
```
packet=sr1(IP(dst=”192.168.68.12”)/TCP(dport=80,flags=”S”))
```



```
>>> packet=sr1(IP(dst="192.168.68.12")/TCP(dport=80,flags="S"))
Begin emission:
.Finished to send 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
>>>
```

13. Navigate back to the **Wireshark** window.

14. Analyze the given **Wireshark** output and notice a *SYN* packet was sent with a *SYN, ACK* packet being received indicating that port *80* is open.



```
   5 90.75499600 192.168.9.2        192.168.68.12      TCP        54 20→80 [SYN] Seq=0 Win=8192 Len=0
   6 90.75555300 192.168.68.12      192.168.9.2        TCP        60 80→20 [SYN, ACK] Seq=0 Ack=1 Win
```

15. Close the **Kali** PC viewer.