ETHICAL HACKING V2 LAB SERIES

Packet Crafting with Scapy

Material in this Lab Aligns to the Following	
Books/Certifications	**Chapters/Modules/Objectives**
All-In-One CEH Chapters ISBN-13: 978-1260454550	3: Scanning and Enumeration
EC-Council CEH v10 Domain Modules	3: Scanning Networks 4: Enumeration
CompTIA Pentest+ Objectives	2.1: Given a scenario, conduct information gathering using appropriate techniques 4.2: Compare and contrast various use cases of tools
CompTIA All-In-One PenTest+ Chapters ISBN-13: 978-1260135947	7: Network-Based Attacks

Document Version: 2020-08-24

Contents

Introduction

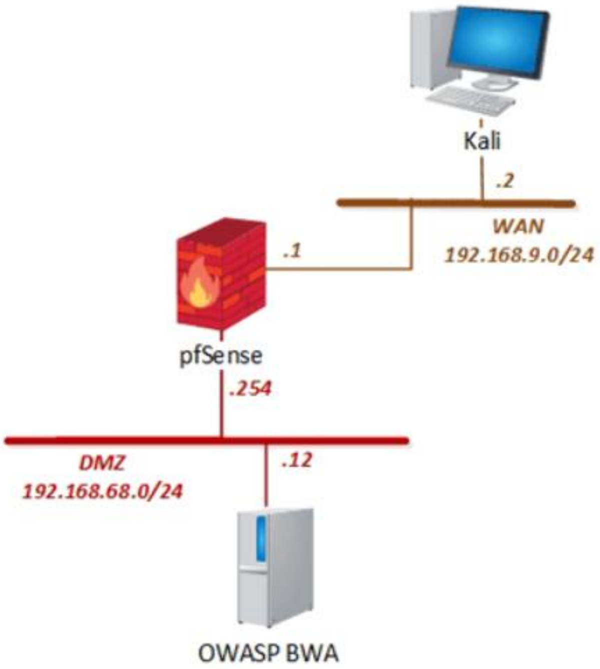
Building a packet field-by-field demonstrates how someone could manipulate the packet traffic entering or leaving a network. This lab shows how to build packets layer-by-layer using *Scapy*, a packet manipulation tool, and then implementing the finished packets to perform various network functions.

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

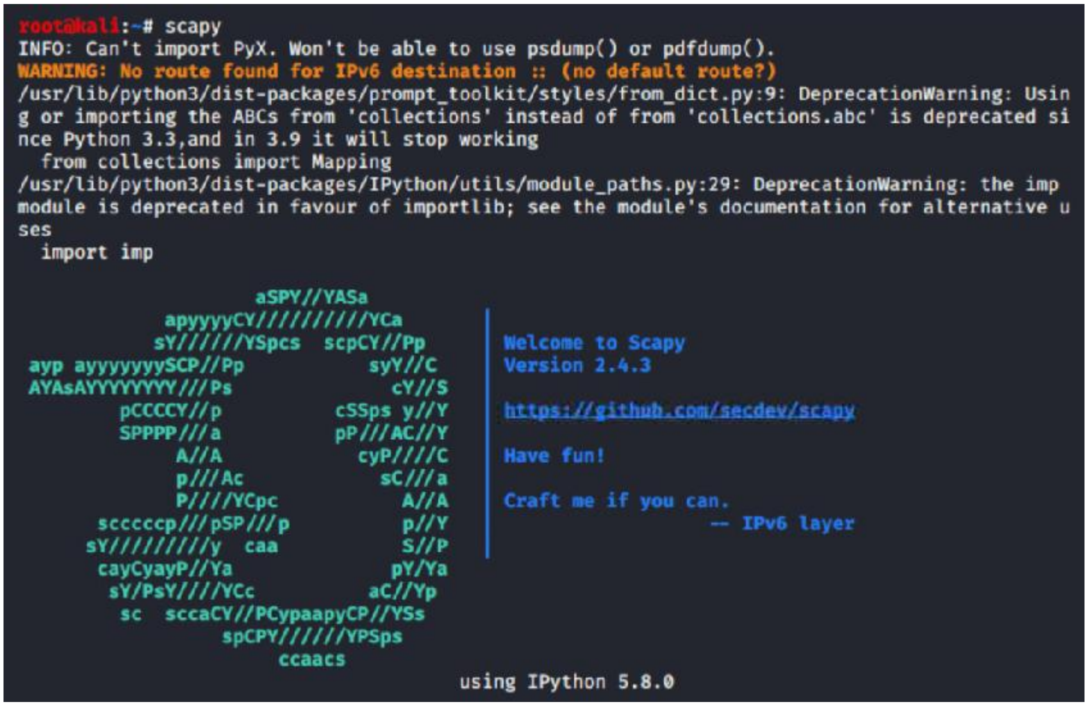
The information in the table below will be needed in order to complete the lab. The task sections below provide details on the use of this information.

Virtual Machine	IP Address	Account (if needed)	Password (if needed)
Kali Linux	192.168.9.2 192.168.0.2	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

1 Creating Packets with Scapy

1. Click on the **Kali** tab.
2. Click within the console window and press **Enter** to display the login prompt.
3. Enter **`root`** as the *username*. Press **Tab**.
4. Enter **`toor`** as the *password*. Click **Log In**.
5. Open a new terminal by clicking on the **Terminal** icon located at the top of the page, if the terminal is not already opened.
6. In the new *Terminal* window, type the command below to initialize the *Scapy* application. Press **Enter**.

```
scapy
```

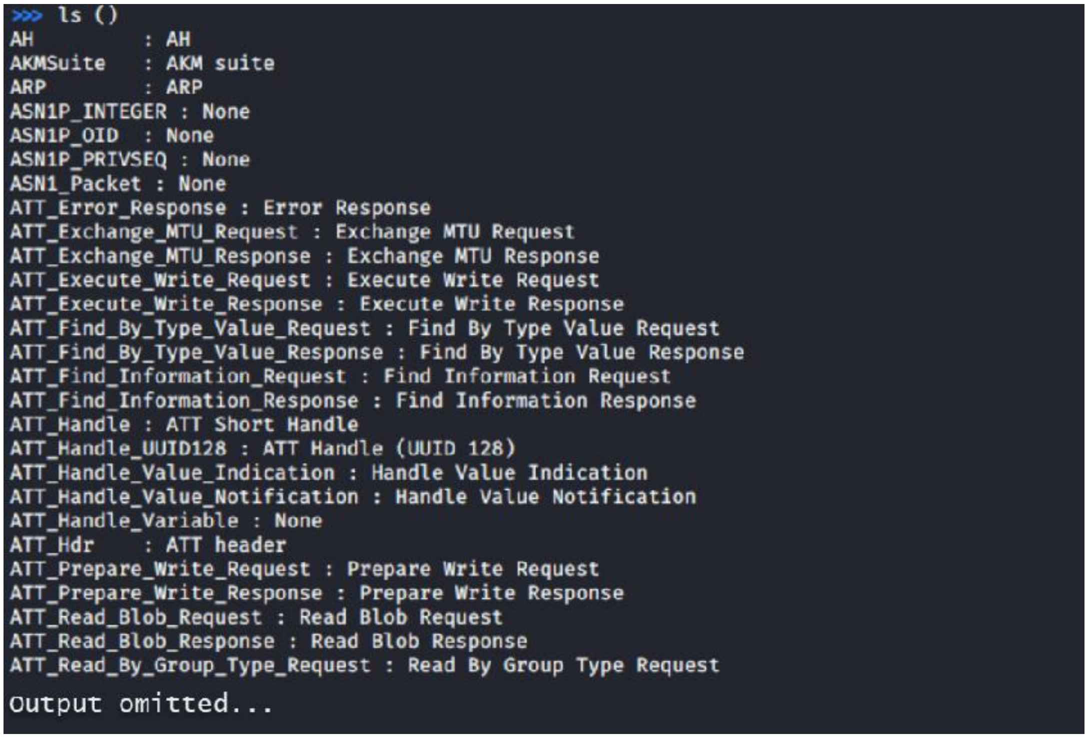


```
root@kali:~# scapy
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)
/usr/lib/python3/dist-packages/prompt_toolkit/styles/from_dict.py:9: DeprecationWarning: Usin
g or importing the ABCs from 'collections' instead of from 'collections.abc' is deprecated si
nce Python 3.3,and in 3.9 it will stop working
  from collections import Mapping
/usr/lib/python3/dist-packages/IPython/utils/module_paths.py:29: DeprecationWarning: the imp
module is deprecated in favour of importlib; see the module's documentation for alternative u
ses
  import imp

                     aSPY//YASa
             apyyyyCY//////////YCa       |
            sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C    | Version 2.4.3
 AYAsAYYYYYYYY///Ps              cY//S   |
         pCCCCY//p          cSSps y//Y   | https://github.com/secdev/scapy
         SPPPP///a          pP///AC//Y   |
              A//A            cyP////C   | Have fun!
              p///Ac            sC///a   |
              P////YCpc           A//A   | Craft me if you can.
       sccccp///pSP///p           p//Y   |               -- IPv6 layer
      sY/////////y  caa           S//P   |
       cayCyayP//Ya              pY/Ya
        sY/PsY////YCc          aC//Yp
         sc  sccaCY//PCypaapyCP//YSs
                  spCPY//////YPSps
                       ccaacs
                                       using IPython 5.8.0
```

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

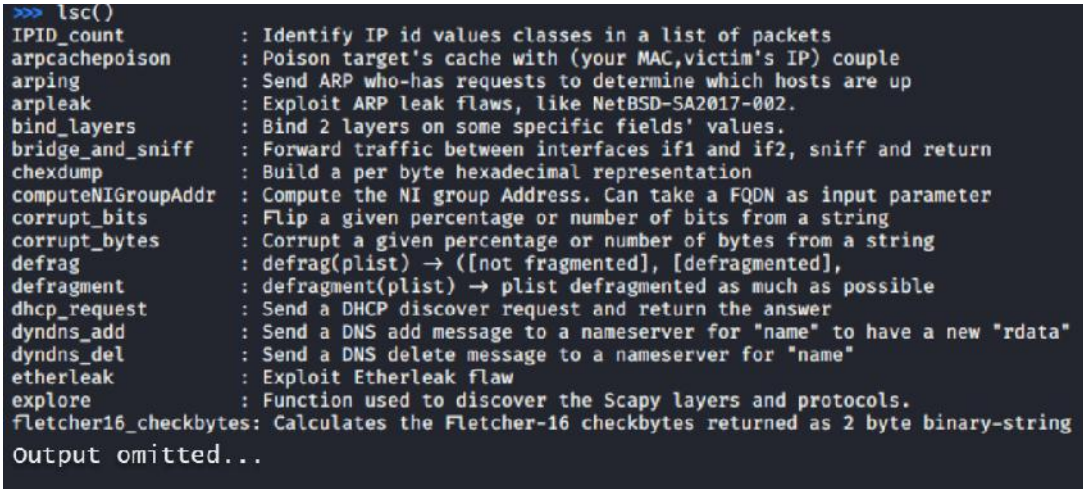
```
ls()
```



```
>>> ls ()
AH         : AH
AKMSuite   : AKM suite
ARP        : ARP
ASN1P_INTEGER : None
ASN1P_OID  : None
ASN1P_PRIVSEQ : None
ASN1_Packet : None
ATT_Error_Response : Error Response
ATT_Exchange_MTU_Request : Exchange MTU Request
ATT_Exchange_MTU_Response : Exchange MTU Response
ATT_Execute_Write_Request : Execute Write Request
ATT_Execute_Write_Response : Execute Write Response
ATT_Find_By_Type_Value_Request : Find By Type Value Request
ATT_Find_By_Type_Value_Response : Find By Type Value Response
ATT_Find_Information_Request : Find Information Request
ATT_Find_Information_Response : Find Information Response
ATT_Handle : ATT Short Handle
ATT_Handle_UUID128 : ATT Handle (UUID 128)
ATT_Handle_Value_Indication : Handle Value Indication
ATT_Handle_Value_Notification : Handle Value Notification
ATT_Handle_Variable : None
ATT_Hdr    : ATT header
ATT_Prepare_Write_Request : Prepare Write Request
ATT_Prepare_Write_Response : Prepare Write Response
ATT_Read_Blob_Request : Read Blob Request
ATT_Read_Blob_Response : Read Blob Response
ATT_Read_By_Group_Type_Request : Read By Group Type Request
Output omitted...
```

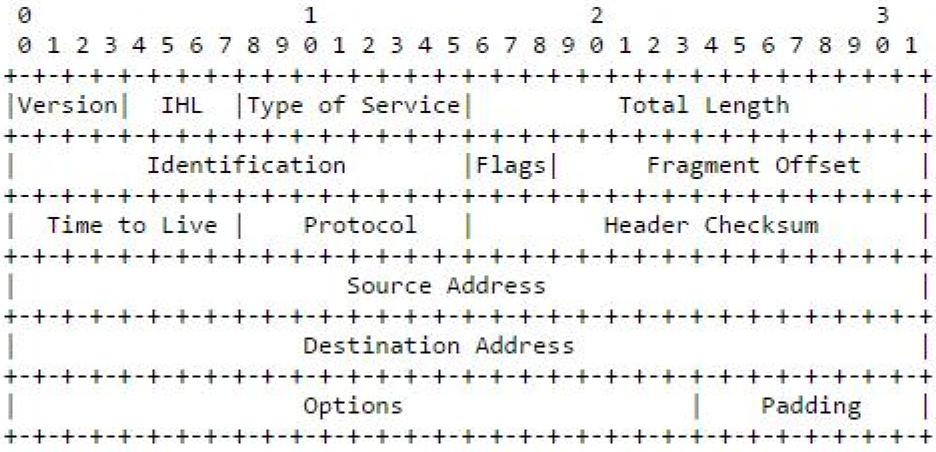
8. Enter the command below to list the available commands.

```
lsc()
```



```
>>> lsc()
IPID_count          : Identify IP id values classes in a list of packets
arpcachepoison      : Poison target's cache with (your MAC,victim's IP) couple
arping              : Send ARP who-has requests to determine which hosts are up
arpleak             : Exploit ARP leak flaws, like NetBSD-SA2017-002.
bind_layers         : Bind 2 layers on some specific fields' values.
bridge_and_sniff    : Forward traffic between interfaces if1 and if2, sniff and return
chexdump            : Build a per byte hexadecimal representation
computeNIGroupAddr  : Compute the NI group Address. Can take a FQDN as input parameter
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 string
defrag              : defrag(plist) -> ([not fragmented], [defragmented],
defragment          : defragment(plist) -> plist defragmented as much as possible
dhcp_request        : Send a DHCP discover request and return the answer
dyndns_add          : Send a DNS add message to a nameserver for "name" to have a new "rdata"
dyndns_del          : Send a DNS delete message to a nameserver for "name"
etherleak           : Exploit Etherleak flaw
explore             : Function used to discover the Scapy layers and protocols.
fletcher16_checkbytes: Calculates the Fletcher-16 checkbytes returned as 2 byte binary-string
Output omitted...
```

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.

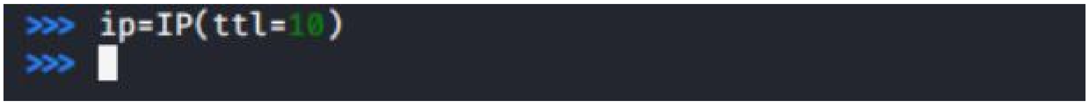


```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         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 to set the *Time to Live* for the IP packet.

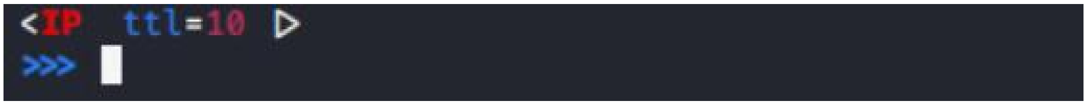
```
ip=IP(ttl=10)
```



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

11. Check to ensure the previous modification took effect by entering the command below.

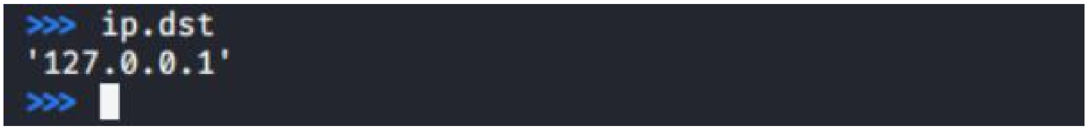
```
ip
```



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

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

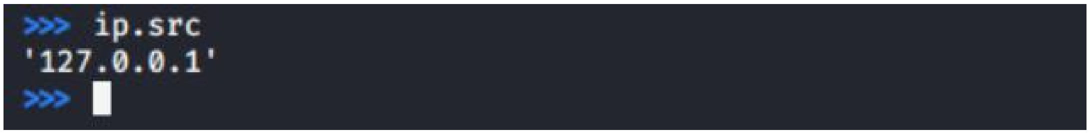
```
ip.dst
```



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

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

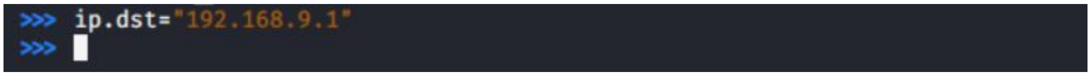
```
ip.src
```



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

14. Change the IP destination.

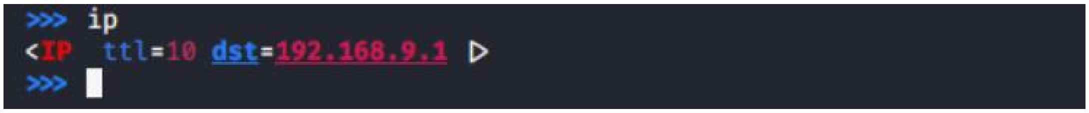
```
ip.dst=”192.168.9.1”
```



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

15. Verify the modifications.

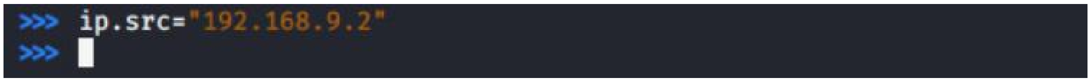
```
ip
```



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

16. Change the IP source address.

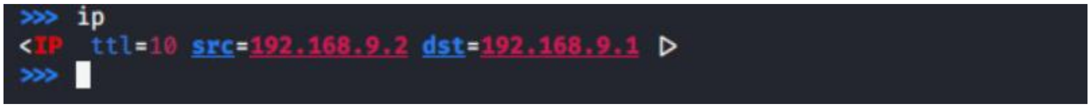
```
ip.src=”192.168.9.2”
```



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

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

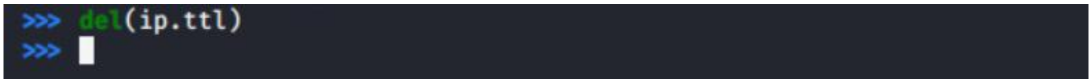
```
ip
```



```
>>> ip
<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.

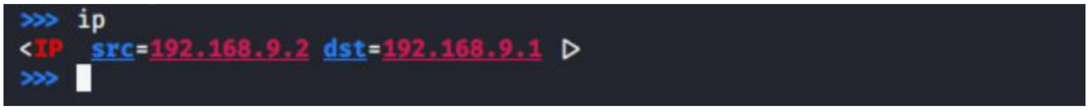
```
del(ip.ttl)
```



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

19. Verify the removal.

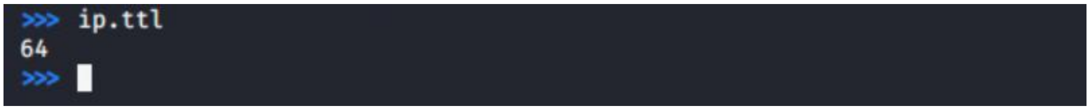
```
ip
```



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

20. Verify the *TTL* is the *RFC* value of **64**.

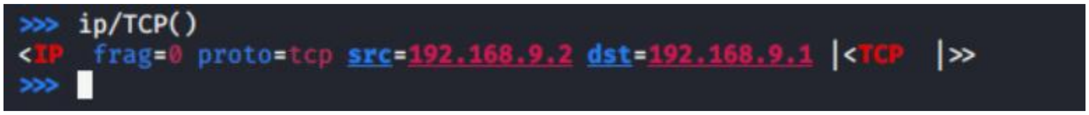
```
ip.ttl
```



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

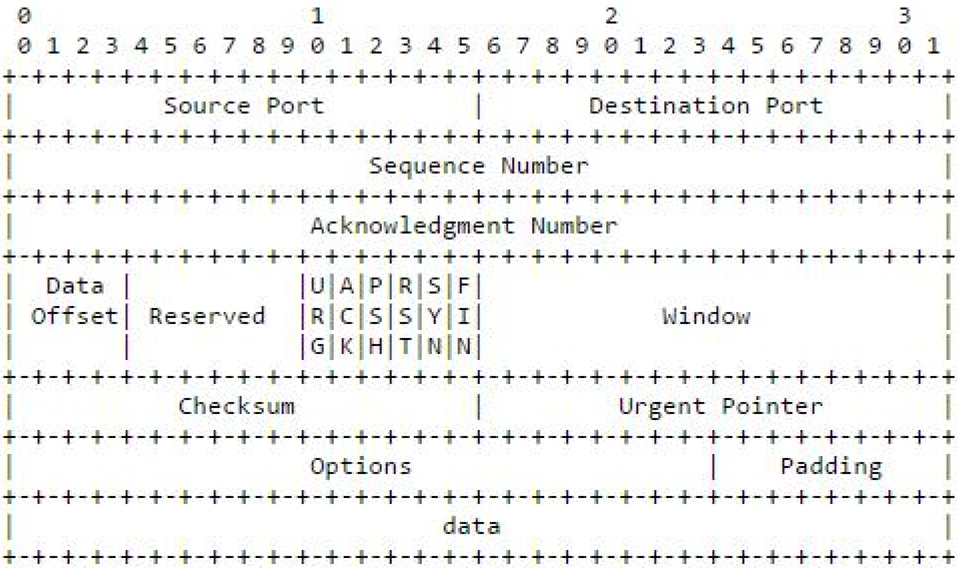
21. Add additional protocol layers by adding *TCP* on top of *IP*.

```
ip/TCP()
```



```
>>> ip/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*.

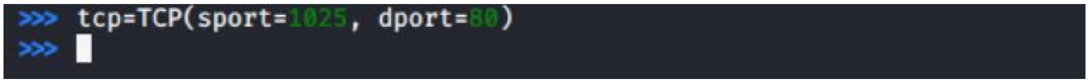


```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        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.

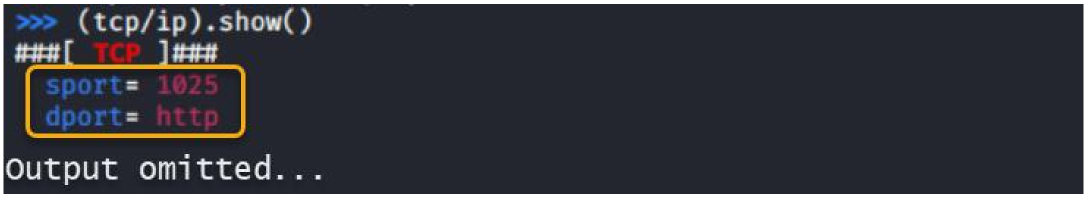
```
tcp=TCP(sport=1025, dport=80)
```



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

24. Show the TCP stack.

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

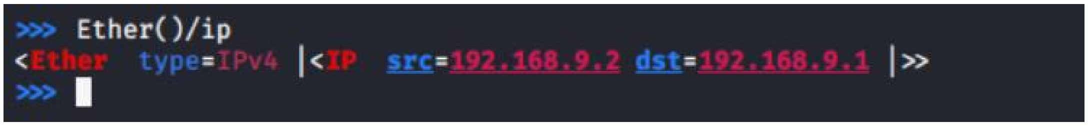


```
>>> (tcp/ip).show()
###[ TCP ]###
  sport= 1025
  dport= http
Output omitted...
```

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=IPv4 |<IP  src=192.168.9.2 dst=192.168.9.1 |>>
>>> █
```

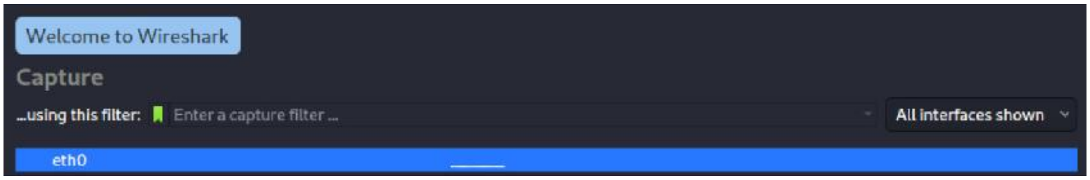
26. Leave the *Terminal* open for the next task.

2 Sending Crafted Packets

1. Launch a new **Terminal** by clicking the **File** dropdown menu option from the already existing *Terminal* window and select **New Window**.
2. In the new *Terminal* window, type the command below, followed by pressing the **Enter** key.

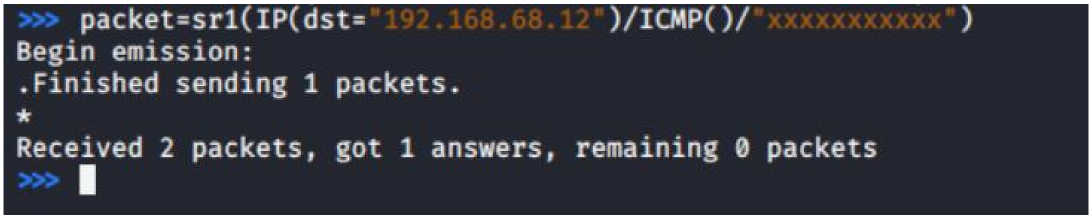
```
wireshark
```

3. If prompted with a warning message about running *Wireshark* as the root user, click **OK**.
4. Within the *Wireshark* window, select the **eth0** interface from the *Capture* panel and press **CTRL+E** to start capturing.



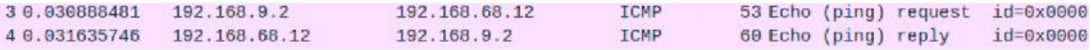
5. Navigate back to the **Terminal** window with the *Scapy* prompt.
6. Generate a single *ICMP* packet to be sent to the *OWASP* machine. Enter the command below.

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



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

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

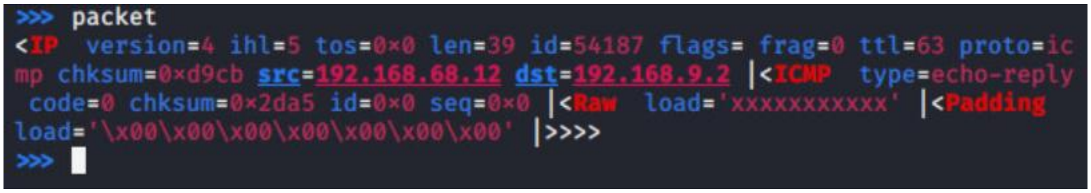


```
3 0.030888481   192.168.9.2      192.168.68.12     ICMP     53 Echo (ping) request   id=0x0000
4 0.031635746   192.168.68.12    192.168.9.2       ICMP     60 Echo (ping) reply     id=0x0000
```

9. Navigate back to the **Terminal** with the *Scapy* prompt.

10. Enter the command below to show the contents of the packet, which was created.

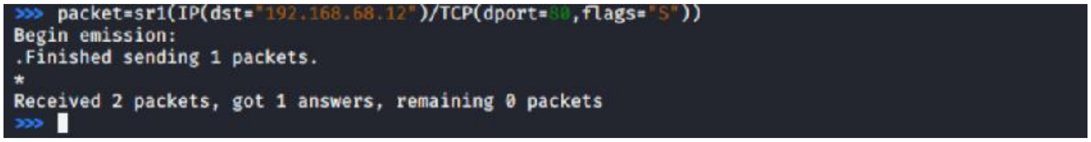
```
packet
```



```
>>> packet
<IP  version=4 ihl=5 tos=0x0 len=39 id=54187 flags= frag=0 ttl=63 proto=ic
mp chksum=0xd9cb src=192.168.68.12 dst=192.168.9.2 |<ICMP  type=echo-reply
 code=0 chksum=0x2da5 id=0x0 seq=0x0 |<Raw  load='xxxxxxxxxxx' |<Padding
load='\x00\x00\x00\x00\x00\x00\x00' |>>>>
>>> 
```

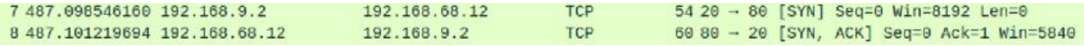
11. Enter the command below in an attempt to initiate a simple SYN scan on a single port.

```
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 sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
>>> 
```

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



```
7 487.098546160 192.168.9.2         192.168.68.12        TCP        54 20 → 80 [SYN] Seq=0 Win=8192 Len=0
8 487.101219694 192.168.68.12       192.168.9.2          TCP        60 80 → 20 [SYN, ACK] Seq=0 Ack=1 Win=5840
```

14. You may now end your reservation.