Bonus Assessment #2  
 Frame Manipulation and Attacks using Scapy

|  |  |
| --- | --- |
| Name: |  |
| Section: |  |
| Faculty: | Preet Dsouza |
| Due Date: |  |
| Submission Mode: | **Due in class in Week 14.**  **Students must be ready with their work BEFORE their lab. All troubleshooting is the students’ responsibility!  Lab time will only be used for evaluation! No submission needed** |

**Lab Requirements:**

Kali VM running Wireshark.  
Compatible USB Wi-Fi Adapter

Read this document in full before you begin

# Introduction

Scapy is a tool based in Python that allows the creation and manipulation of packets by stacking protocol layers. Scapy provides support for a wide range of protocols which are defined as Python classes. Objects can be created for each protocol header and stacked using forward slashes to create a new frame.

Scapy provides support for the IEEE802.11 frame header and can be used to craft, sniff and/or inject IEEE802.11 frames. Since Scapy is implemented in Python3, we can effortlessly write Python scripts to perform some wireless attacks.

# Part 1 – Using Scapy (pre-installed in Kali)

1. Install python packages to be used for graphical representation of packets:   
   **sudo apt install python3-pyx python3-matplotlib**
2. Invoke the Scapy command line using command: **sudo scapy**

The Scapy command-line looks as below:

A screen shot of a computer screen

Description automatically generated

1. List available modules using function: **ls()**

You will see familiar protocols like IP, IPv6, Ether (Ethernet), ICMP, TCP, UDP, DNS, Dot11 (802.11).

1. Let’s explore the fields in the IP protocol using command: **ls(IP())**   
     
   A screen shot of a computer

   Description automatically generated

First coloumn 🡪 Protocol fields

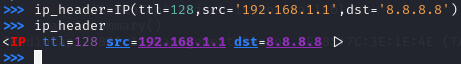
Second column 🡪 Field description

Third column 🡪 Field values  
Fourth column 🡪 Default field values

1. The protocols are modules (classes) and can be invoked using objects. For example:

**ip\_header = IP(ttl=128, src=192.168.1.1, dst=8.8.8.8)**

**ip\_header**

****

**A computer screen with white text

Description automatically generated**

1. Once an object of a protocol class has been created, you can modify its attributes (fields) just like you would for any Python object:

A screenshot of a computer

Description automatically generated

This part carries no points.

# Part 2: Crafting an ICMP Echo Request

1. Let’s prepare each protocol headers. The order does not matter. After the headers are ready, we will stack them.

**icmp=ICMP(type=8,code=0)/("Knock Knock!!!")** # Object containing ICMP Header + Data

**ip=IP(src='192.168.2.73',dst='192.168.2.1')** # Object containing IP Header

**eth=Ether()** # Object containing Ethernet Header

**frame=eth/ip/icmp** # Stack protocol objects

Note: The stacking here is akin to encapsulation.

A screenshot of a computer

Description automatically generated  
  
Alternatively, you can create the frame on a single line as shown below:



1. You can view the entire frame using command:

A screenshot of a computer screen

Description automatically generated

1. Now send the frame onto the network:

**sendp(frame,iface='eth0')**

A black background with white text

Description automatically generated

1. You can use the srp() function to send the frame and check for a reply.

A screen shot of a computer

Description automatically generated

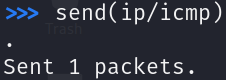
1. You instead assign the return value from the srp() function to a variable as shown below. This variable will store a tuple.

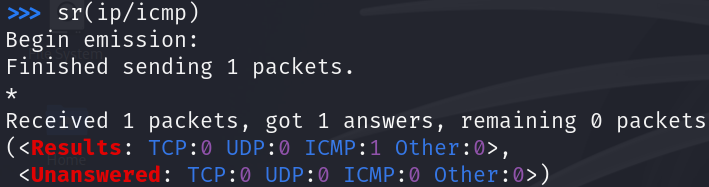


1. Below is the Wireshark capture on the local Ethernet interface showing the ICMP echo request sent along with the ICMP echo reply received.

A screen shot of a computer

Description automatically generated

1. Alternatively, you could create the packet i.e. IP header + ICMP header + Data and send it along. Scapy will insert the Ethernet header on its own.   
   To send packets, we use the **send()** function:  
     
   
2. We can send and receive a reply, using function **sr()** as follows:

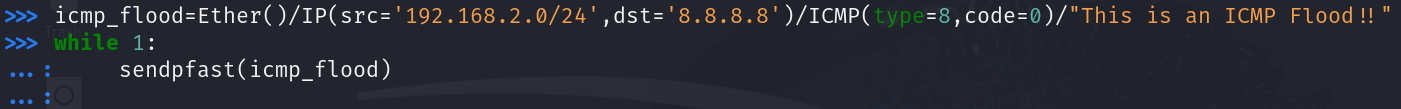


This part carries no points.

# Part 3: ICMP/Ping Flood

1. We can flood a destination with multiple ICMP echo requests thereby causing a ‘flood’. When done at high volume, this can result in a DoS on the destination because the destination will attempt to process each ICMP echo request and respond with an ICMP echo reply.

Below, we specify the source IP as a network range resulting in scapy sending an ICMP echo request with each source IP address. We use the sendpfast() function to send frames rapidly and we put this function in an infinite loop to ensure it indefinitely sends the ICMP echo requests.



1. The Wireshark capture can be seen below. Note that the destination 8.8.8.8 does reply to ICMP echo requests but the only the reply meant for the VM’s IP address will be received.

A screenshot of a computer

Description automatically generated

This part carries no points.

# Part 4: Sending ARP Request frame

1. Create an ARP request frame as follows:

**arp=ARP(psrc='192.168.2.73',pdst='192.168.2.1',op=1)** # Object ARP header

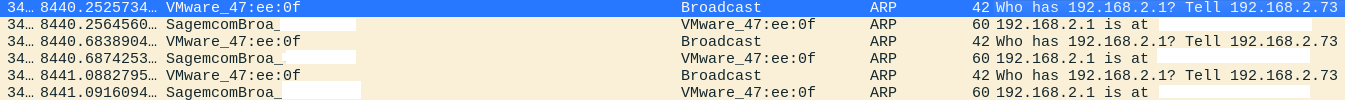
**frame\_arp=Ether()/arp** #Stacking/Encapsulating ARP header within Ethernet header

**sendp(frame\_arp,iface='eth0')** #Sending frames onto Ethernet network

A screen shot of a computer code

Description automatically generated

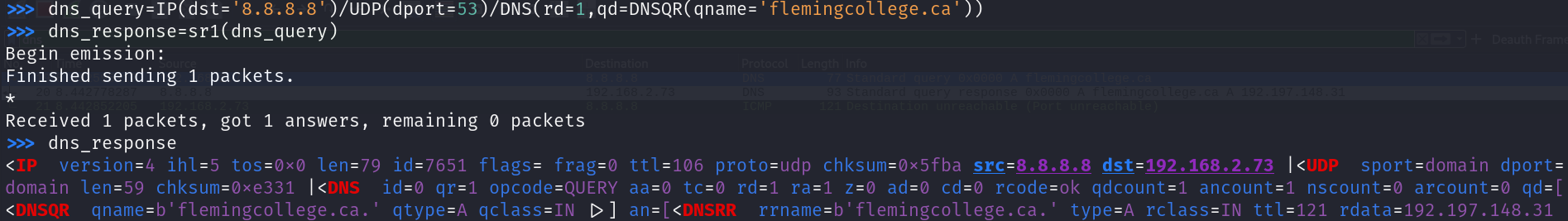
1. The frames can be seen in Wireshark as shown below. Notice the ARP replies being received:



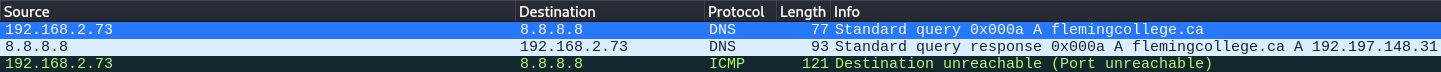
This part carries no points.

# Part 5: Sending a DNS Query

1. Craft a DNS query for *flemingcollege.ca* to DNS server 8.8.8.8  
   Enable recursive lookup.



1. Below is the Wireshark capture showing the DNS Query and Response:



**Note:** The ICMP Destination unreachable (Port unreachable) was sent because when Scapy sent the DNS Query, with source port = 35000, it did not actually ask the OS to allocate the port and so when the DNS Query Response arrived with destination port = 35000, the OS responded back with an ICMP Destination unreachable (Port unreachable) because this port is not open on the OS.

This part carries no points.

**Part 6: Send a TCP SYN to a web server**

1. You will recall that TCP is a connection-oriented protocol which means a TCP connection has to be set up before segments can be exchanged. This connection is set up using the 3-way handshake as follows:

Client sends SYN to Server  
Server replies with SYN,ACK to Client  
Client replies with ACK to Server  
  
The TCP connection has now been established.

1. We will send a TCP SYN and will receive a TCP SYN,ACK from the server. However, we will stop short of the sending a TCP ACK.  
   We can craft a TCP SYN to send to a web server as follows:

A black screen with colorful text

Description automatically generated with medium confidence

1. The Wireshark capture below shows the TCP SYN that was crafted and sent and the TCP SYN,ACK that was received from flemingcollege.ca :



Note: When scapy sent the TCP SYN with source port = 3000, it did not notify the OS to open that port. As a result, when the TCP SYN,ACK was received from the server with destination port = 3000, the OS replied back with a TCP RST which means to reset the connection.

1. Now create a TCP SYN flood yourself.

This part carries no points.

# Part 7 – Sniffing and injecting Beacon Frames

1. Scapy allows us to sniff any IEEE802.11 frame, make changes to it and inject it back onto the medium. We can also craft our own frames from scratch.
2. In this part we will sniff a Beacon frame, change its SSID parameter and then inject the modified Beacon back into the air.
3. Ensure your wireless interface is in Monitor mode.  
   **sudo airmon-ng check kill  
   sudo airmon-ng start wlan0**
4. Run scapy in a terminal: **sudo scapy**
5. Execute the code below line by line within the scapy CLI.

|  |
| --- |
| # capture a single beacon frame  **beacon=sniff(iface='wlan0mon', filter=’type mgt subtype beacon’,count=1)**    #Here, the beacon object is a list containing a single frame  **beacon[0].show()** # Displays the content of the frame present in the beacon list  #We can make changes to this beacon and then inject it back onto the medium  **beacon[0][Dot11Elt].info='SSID-Preet-Dsouza'**  # Set the SSID parameter as SSID-Full-Name  **beacon[0][Dot11Elt].len=len('SSID-Preet-Dsouza')** # Set SSID length  **s=conf.L2socket(iface='wlan0mon')** # Create a Layer 2 socket for frame injection  **while 1:**  **s.send(beacon[0])** # continuously inject the modified beacons over the Layer 2 port  # Check your Wireshark live capture to see the injected Beacons |

The modified Beacon must be visible in both Wireshark and using a Wi-Fi Analyzer. Must be signed off by Faculty in class. Successful completion will earn you 3 points.

# Part 8: Authentication Flood against AP

1. To overwhelm an AP, Authentication frames can be crafted with different source addresses (fake clients) and injected toward the AP. The AP will attempt to reply to each fake client but because of the overwhelming number of fake clients, the AP will be slow to respond to legitimate clients leading to failed connection attempts.
2. Use the script Part8-Auth-Flood.py provided alongside this document in D2L to perform an Authentication flood against your AP (hotspot). Be sure to read the comments in the script indicating where to insert your AP’s MAC address and SSID.
3. Your wireless interface must be in Monitor mode.   
   **sudo airmon-ng check kill  
   sudo airmon-ng start wlan0**
4. Set wireless interface to the same channel as your AP.   
   **sudo iwconfig wlan0mon** *channel\_number*  
     
   Execute the script as follows: **sudo python3 Part8-Auth-Flood.py**

Clients attempting to connect to the AP will be unsuccessful. Must be signed off by Faculty in class. Successful completion will earn you 3 points.

# Part 9: Deauth script to block clients from connecting to WPA/WPA2/WPA3 networks

1. Spoofed Deauthentication frames can be used to kick clients off a WPA/WPA2 protected network. However, these frames cannot be used to kick a client off a WPA3 protected network since WPA3 requires the mandatory use of the IEEE802.11w amendment that provides for Protected Management frames. Specifically, the Deauthentication, Dissociate and Action frames are protected. However, this protection only takes effect AFTER the 4-way handshake has been completed. Before the 4-way handshake, the clients will still respond to spoofed Deauthentication frames.
2. The script below will sniff for an Association Request frame from a client that’s attempting to associate with an AP. When found, the script will inject spoofed Deauthentication frames. Specifically, it will inject 100 broadcast Deathentication frames and 100 unicast Deauthentication frames. The latter is because some clients will not respond to the broadcast Deauthentication frames. This effectively deauthenticates the client sending it back to State 1 where the client is unauthenticated and unassociated with the AP (recall the IEEE802.11 State Machine).
3. The script is provided in D2L alongside this document. It must be run from the Kali terminal and NOT from the scapy CLI. The script is run as follows:

**sudo python3 Part9-Deauth.py** *<channel> <bssid>*

So for example, if your AP is on channel 11 and has MAC address 01:02:03:04:05:06 the command will be:

**sudo python3 Part9-Deauth.py 11 01:02:03:04:05:06**

Clients attempting to connect to the AP will be deauthenticated before the 4-way handshake can take place. Must be signed off by Faculty in class. Successful completion will earn you 4 points.

# Grading Rationale:

1. Graded out of 10 points.
2. Not performing a step correctly or an incomplete step including not answering a question correctly results no points earned.
3. Deduction for poorly organized work including not using the lab document, poor/incorrect arrangement of answers, missing labels for each Part and Step. Depending upon severity, deduction of 1-10 points.
4. Incorrect/incomplete/incompetent work can result in zero if fundamental requirements are not met.