Lesson Outline

Time	Topic	Description
5 min	Announcements	Reminders
15 min	Network Basics	Background and Definitions
15 min	Legality of Packet Sniffing	Ethics Discussion
20 min	Practical Component	Assignment Overview
20 min	Find Users	Identify Unique IP addresses
60	Identify Users	Identify owners of IP addresses

Lesson Objectives

- Introduce and motivate the need for security on wireless networks.
- Define and discuss packet sniffing.
- Use Python concepts to analyze packets captured over a network.

Before Class

- Optional: set up video playlist.
- Make sure that students have Scapy installed on their machines, or access to a machine with Scapy installed on it.
- Post sample code and PCAP file: sample.py, cyber_lab.pcap

Network Basics (15 min)

Today, we will be discussing packet sniffing on unsecured wireless networks. Ask the class how many people have heard of WireShark. *Ask them what Wireshark is. Someone should say something along the lines that it is a network traffic analyzer and/or packet sniffer/analyzer. Next, ask the students what a packet sniffer is and what packets are.*

- A **packet sniffer** is a piece of software that is capable of capturing and perhaps even analyzing the packets that get passed within a network.
- A **packet** is a small amount of information that gets sent across the network. A user on the network can send or receive packets. These packets contain information such as the websites the user visits, and sometimes even usernames and passwords.

To understand these terms in greater detail, let's quickly discuss networks. A **network** is simply a collection of computers or other devices that are connected together some way, usually via a router. A **router** is a simple device that is capable of transferring information between the devices that are connected to it. These devices could be different computers, printers, or even other routers who can then pass the information on.

Each computer on the network is assigned an **IP address**. This address is subject to change. If the user connects to the internet via a different router or at a different time, they may have a different IP address. Contrast this to the **MAC address**, which is unique to a particular device. If you bring a router down, you affect all the devices that are connected to it, and any other routers it is connected to. If a router is offline, the devices connected to it are also offline, as they cannot obtain IP addresses. Thus, the internet at any particular time can be defined as the set of visible routers.

Packets and Packet Sniffers: Let's talk about packets. If you want to send something over the internet (say sending a picture to your grandma), it gets broken down into tiny pieces called packets. Packets are a lot like the packages you send in the mail. The packet header contains the information on how the packet should be handled, similar to the labels you see on a a regular package. The source and destination IP address is like the return and address labels you place on packages you mail. The port and protocol indicate how the packet should be routed, similar to the priority mail stickers you see. The header is designed to assist the router to get the package to you, similar to how a package's labels are designed to help the post office get your package to you.

The next (and most important component) of a packet is the **payload**. The payload refers to the actual contents of the packet. This could be an e-mail message, passwords, websites, etc. Packets traditionally travelled between routers via wires. These wires could be telephone wires, television cable wires, or large fibre-optic cables running deep underwater. That's one of the reasons why many of the common internet providers were originally telephone and cable providers.

Since everyone has a unique IP address and a unique path to the network, information transfer is usually assumed to be "safe". This changed with wireless networks. In a wireless network, all the computers connected to the wireless router sends their information through the air (similar to Mike TV and Wonka Vision in Willie Wonka and the Chocolate Factory). The wireless router, sends the information back to all the computers via the air, in a communication mechanism called broadcast. In broadcasting, every computer gets a perfect copy of all the packets sent over the network. However, since every computer knows its unique IP address, it discards the packets that are not addressed to it, just like you're supposed to discard any mail you receive that's not addressed to you. At least, that's how it's supposed to work.

A malicious individual can place a special device on a network called a **packet sniffer**. Instead of discarding the packets that it gets, it stores all of them in a **packet capture**. You have no knowledge that a packet sniffer has a copy of your packets, because, of course, you get them too. It's one of the consequences of being able to easily create and distribute perfect copies of digital content.

Ethics of Packet Sniffing (15 min)

Ask the students if they think packet sniffing is illegal. Have a discussion with them about this, getting their thoughts on the matter. Unauthorized wiretapping (putting a listening device on a wire) is illegal under US law, but remember there are no wires here. In fact, in order to enable the US to spy on its enemies using radio communication, the wiretapping law was written in such a manner that says "it shall not be unlawful under this chapter or chapter 121 of this title for any person to intercept or access an electronic communication made through an electronic communications system that is configured so that such electronic communication is readily accessible to the general public (Section 2511(2))" And later, "readily accessible to the general public means, with respect to a radio communication that such communication is not scrambled or encrypted, transmitted using modulation techniques." Thus, wiretapping secured wireless networks is illegal; but what about the unsecured "free" WiFi hotspots? In other words, are unsecured wireless networks (such as those found in coffee shops, hotels, airports, etc.) configured in a manner that is "readily accessible to the general public"? And if so, is it legal? Spend a few minutes

and think about this.

So, what conclusions the students come to? Is it illegal? Remember, this question is different from "SHOULD it be illegal". Do people using unsecured wireless networks have some "expectation of privacy"? The law is currently unclear, and certainly groups advocate amending the Federal Wiretap Act to include a clause that protects wireless networks. The problem with the digital world is that our technology is progressing at a speed that far outpaces the laws written in the physical world.

In Real Life: Let's look at a practical example that is currently cycling the judicial system. *Ask the students how many of them have heard about Google Street Cars.* Google Street Cars go up and down residential and commercial streets, taking pictures. There are a host of privacy complaints against Google because of the Street Cars, but that's a separate issue, and we won't discuss that much here. What is pertinent to our discussion is that most of those vehicles also had packet sniffers. When a Google Street Car came in range of a wireless network, it would automatically connect to it, collect the packets for a brief amount of time, before it went out of range and connected to the next, and so on. All in all, Google collected about 600 GB worth of data from more than 30 countries, including usernames, passwords, personal e-mails.

Google said they were doing this "unintentionally"; they didn't meant to collect the data, and didn't plan to do anything with it. Also, it's not like their technology was breaking in to secured wireless networks and stealing the data; no, this was data that was available on publicly accessible wireless networks. So, how did they break Federal Wiretapping Law? The courts disagreed, and Google was slapped with a $7 million fine. At the time, the courts ruled that unsecured wifi networks were not configured to be readily available to the public.

The following year saw another court case. Innovatio IP Ventures, a non practicing entity that buys up patents for the sole process of making money on litigation (more commonly known as a patent troll) wanted to sue between 8,000-12,000 businesses for violating 17 of its patents. To gather evidence that these companies were violating their patents, the patent troll wanted to run a packet sniffer on the networks, promising only to keep the headers, but delete the payloads. A federal judge in Illinois said ok. To quote: "Many WiFi networks provided by commercial establishments .. are unencrypted, and open to such interference from anyone with the right equipment. In light of the ease of "sniffing" WiFi networks, the course concludes that the communications sent on an unencrypted WiFI network are readily available to the general public"

Google appealed. The law was on their side now, right? But, in 2013, it lost the appeal. The US Court of Appeals for the Ninth Circuit ruled that Google's data collection did not qualify for the exemption under the Federal WireTap Act; "Even if its commonplace for members of the general public to connect to a neighbor's unencrypted Wi-FI network, members of the general public do not typically mistakenly intercept, store, and decode data transmitted by other devices on the network". Google is now taking their case to the Supreme Court. So the jury is still out on this one. Regardless, no student should go to coffee shops and run a packet sniffer! The exercises presented here are for educational purposes only.

Why not outlaw packet sniffers? Here is another question for discussion: why not just outlaw packet sniffers altogether? Isn't packet sniffing obviously bad? Not exactly. Packet sniffers are really useful and can be use for legitimate purposes. For example, a company can use a packet sniffers on their own networks to monitor network traffic, and ensure their networks are being used for the correct and intended purposes, and to detect network intrusion attempts. They are also useful for debugging networks, and verifying the effectiveness of access control systems, such as spam filters and firewalls. Wireshark, a very popular application that has been around the late 1990s, is a packet sniffer (or a packet analyzer) that is used by many security professionals and has won many industrial awards. In fact, Wireshark can

be thought of a fancier version of an even older tool, `tcpdump`, which was developed in the late 1980s by Lawrence Berkley National Labs, and prints out the contents of packets.

So here is the thing to remember: technology itself is amoral. While the use of a technology can raise ethical and legal concerns, the technology itself should have no morality attached to it. Knives are useful for chopping food, performing complex surgeries, opening letters, and various other things. All those particular knives could be used to seriously maim or kill someone. But it's infeasible to think that knives should be outlawed.

Practical Component (20 min)

The scenario we will be studying today is similar to the ones we have discussed so far in this lesson. Suppose you are at a coffee shop with your trusty Linux machine. There are four other individuals at the coffee shop. You start `tcpdump` and capture all the packets being transmitted during the 15 minutes that you are there. These packets are stored in `cyber_lab.pcap`. Your assignment: analyze the PCAP file to find out as much as you can about the four individuals! This includes: websites visited, pictures they looked at, e-mails, passwords, names, occupations, everything you can find. Don't worry – these are four artificially created characters. We created the packet capture on our own sandbox networks, so no laws were broken. The four characters were created for educational purposes. Any similarity to real characters is purely coincidental.

Analyzing the Starter Code: Have the students download the starter code file and PCAP file from the webpage. We are using a Python package called Scapy, a powerful packet manipulation module implemented for use with the Python language. For the purposes of these exercises, we aren't going to use Scapy for any other purpose other than reading a packet capture, which is stored in the provided PCAP file.

Let's take a look at the starter code:

```
1  from scapy.all import *
2  import sys
3
4  def parsePCAP(pkts):
5    for pkt in pkts:
6      print "Source IP: " + pkt[IP].src
7      print "Destination IP: " + pkt[IP].dst
8      print "Source port: " + str(pkt[TCP].sport)
9      print "Destinations port: " + str(pkt[TCP].dport)
10     print "Packet Payload: " + str(pkt[TCP].payload)
11
12 if __name__ == "__main__":
13   if len(sys.argv) < 2:
14     print "usage: python lab3.py [pcap]"
15     sys.exit()
16   pcap= rdpcap(sys.argv[1])
17   pcap = [pkt for pkt in pcap if TCP in pkt]
18   parsePCAP(pcap)
```

The first line imports a number of functions and constant variables into our python session from Scapy. This allows us to call the `rdpcap()` function directly, instead of typing `scapy.all.rdpcap()`.

Other reserved keywords here are `IP` and `TCP`. Stress to the students that they absolutely should NOT create their own variables called `IP` and/or `TCP`. If they do, they will encounter a lot of problems, given the order with which Python references namespaces.

The second line references the `sys` module. We call sys so we can use the `sys.argv` object, which will hold command line arguments. The first element in `sys.argv` (`argv[0]`) is always the name of the python program. This allows us to run our program as follows:

```
1 python sample.py cyber_lab.pcap > output
```

where `argv[1]` will contain `cyber_lab.pcap`. When the code runs, the first thing it calls is the `rdpcap()` function on line 16. The `rdpcap()` function parses the PCAP file and returns a list of Scapy packets. We do a bit of preprocessing on line 17 to include only TCP packets. This particular expression is an example of a list comprehension. If you haven't encountered list comprehensions before, don't worry – for now, it's just a line you should leave alone that is needed as a pre-processing step.

Let's look at the `parsePCAP()` function. The `parsePCAP()` function takes a list of Scapy packets as its single parameter. For each packet in the list of packets, we print out its information. Lines 6-9 are the header information of our packet: the source IP, destination IP, source port, and destination port respectively. Line 10 is the actual packet payload. Notice that while the source and destination IP addresses are by default strings, this is not true for the source and destination port, and the packet payload. The source and destination port are both of type `int`, and the packet payload is a special Scapy type, which is organized like a dictionary. Each Scapy packet has an IP and a TCP layer. The IP layer has two components: `src` and `dst` which are used to reference the source and destination IPs respectively. The TCP layer has three components: the source port (`sport`). the destination port (`dport`), and the packet payload (`payload`). We use the `str()` function to cast each of these non-string types to a string, to enable the proper execution of the concatenation operations on lines 8-10.

Ports: Recall the source and destination IP address refer to the IP address that the packet is coming from and the IP address that the packet is traveling to. This is akin to the return address and mailing address on a package you send through the postal service. The concept of a port is a little less obvious. When you browse webpages, send and receive email, and do other things on the web, you are actually connecting to a **server**. A server is a special application on a machine that is designed to respond to network requests. Sometimes, a single machine is dedicated to a particular type of network requests, such as web server or a mail server. However, it is also possible for a single machine to run multiple server applications. To distinguish between various requests, machines use ports to route and handle different categories of network requests. So if you visit `www.google.com`, you are making a request to a google web server, which connects to port 80. If that server also happens to run webmail services, your are connecting to ports 25 and 143. Students can see a full list of ports and services under `\etc\services`, but the three ports (80,25,143) are all that they need to pay attention to for the scope of this lab.

Running the code: To run the program enter the following command on the command line:

```
1 python sample.py cyber_lab.pcap > output
```

Prior to processing, the PCAP is a binary file. Note that it is very important to redirect the output to a file. If you let the output print to a terminal, you will get a lot of errors, as the terminal attempts to read and render some of the binary fonts. While the `rdpcap()` function removes a lot of the binary

components, there are still binary portions of the parsed PCAP file. This can include gzip'ed images, which are compressed prior to sending over the web. Similarly, do not open the `output` file in an editor! You will run into the same issues as the editor desperately tries to install fonts. I recommend previewing the file using a tool such as `less`.

Currently, the packet capture is simply a long list of packets. While we can view each packet's info individually, they are currently all jumbled together! Let's make it easier to target a single person's packets. To that, we need to know the IP addresses of the other people who were with us in the coffee shop.

Exercise 1

Identify the IP addresses of the other people in the coffee shop. (Easier version of the exercise: Your source IP is 10.3.0.18). You know that all the other machines on the coffee shop's local area network have the same subnet as you; in other words, they share the first three bytes in their IP address as you. Create a new function called `findOtherIPs()` that finds the other IPs of the people in the coffee shop. Need a hint? Remember that the other IPs have the same subnet as you. In other words their IPs start with `10.3.0.` Your function should return the collection of all the IPs of the people in the coffee shop. Note: to make this exercise harder, don't give them the starter IP address. Instead, point out the fact that the set of IP addresses that send and receive the most packets (in other words, the IPs that produce the most chatter) are likely the IPs of the machines your fellow coffee shop patrons are using. Suggest that they use a dictionary to solve this version of the exercise.

Solution: Below are some sample solutions to the simpler version of this exercise:

```
def findOtherIPs(pkts):
  unique = []
  for pkt in pkts:
    if "10.3.0" in pkt[IP].src:
      if pkt[IP].src not in unique:
        unique.append(pkt[IP].src)
  return unique

#one line solution (for advanced students only)
def findOtherIPsv2(pkts):
  return list(set([pkt[IP].src for pkt in pkts if "10.3.0" in pkt[IP].src]))
```

Exercise 2

Now, create a new function called `parsePCAP_mail()` that takes an IP address (type `str`) along with the list of Scapy packets. Your function should write to a unique file all the packets originating from a given IP address and accessing a server's destination mail ports (143 and 25). So, if the IP address is 10.3.0.18 you should write all the mail packets associated with 10.3.0.18 to `10.3.0.18mail.txt`. Ignore packets if their payload field is empty.

Once you are done with this function, write a wrapper function that generates separate files for all the IP addresses in the coffee shop.

Solution: The solution below reflects the exercise in which students are asked to create a mail file for each IP.

```
def writeFiles(pkts, ipList):
  for myIP in ipList:
    parsePCAP_mail(pkts, myIP)

def parsePCAP_mail(pkts, myIP):
  #opens file for writing
  out = open(myIP+"packets.txt", "w")
  for pkt in pkts:
    if pkt[IP].src == myIP or pkt[IP].dst == myIP:
      if pkt[TCP].dport == 25 or pkt[TCP].dport == 143:
        out.write("Packet Payload: " + str(pkt[TCP].payload)+"\n")
  out.close()
```

At this point, the students should be able to read several e-mails and get a lot of information about people's identities. Information that will appear will include logon information for AOL accounts. You should encourage the students to try and log on the AOL accounts (they will be able to!). Please tell the students NOT to change the passwords on the AOL accounts. If they do, it will break this aspect of the exercise for everyone else who tries to run this exercise!

Exercise 3 (v.1 and v.2)

If students are checking the right ports (25 and 143), you probably see a lot of good stuff already. However, the port 80 info is pretty hard to decipher. There are two keywords that we should pay attention to when looking at port 80:

GET: These types of requests include information that is retrieved from a particular web server.

POST: These types of requests include the information sent to a particular web server.

Below is a particular packet from the packet capture file:

```
GET /s/a/hpc4.png HTTP/1.1
Host: www.bing.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.26) ... Firefox
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Keep-Alive: 115
Connection: keep-alive
Referer: http://www.bing.com/
```

This particular packet is a GET packet, denoted by the keyword GET. This is immediately followed by the resource being requested and then HTTP, signaling that is being passed via the hypertext transfer protocol. A carriage return (not a new line!) follows, and we have the name of the host. After host name, there is other information about the request, such as what format and language to accept the information in. The referer indicates the URL from which the request originated.

Parsing Referer links from port 80: Have the students write a function called `parseReferer()` that gets all the unique referer links in an individual's packets. Be sure to ignore empty packets! A solution to this exercise is shown below:

```
def parseReferer(pkts, myIP):
  refers = []
  for pkt in pkts:
    payload = str(pkt[TCP].payload)
    if pkt[IP].src == myIP and pkt[TCP].dport== 80 and payload!= "":
      sections=payload.split('\r')
      for section in sections:
        if "Referer" in section:
          link = section.strip().split()[1]
          if link not in refers:
            refers.append(link)
          break
  return refers
```

A closer look at GET: NOTE: This is the original, older version of this exercise! I recommend that you use `parseReferer()`! If we really want to, we can actually view ALL the information the user requested by combining the host info with the information being retrieved. So, for the above packet sample, it's necessary to combine `www.bing.com` and `/s/a/hpc4.png` to get `www.bing.com/s/a/hpc4.png`. In this case, this image is a part of a bing search background. Have the students go to www.bing.com to see that this is indeed the case.

As a bonus exercise, have students implement a function called `parseGET()` that accomplishes this. For advanced students, have them exclude java script and css files (which will be designated in the extension).

Solution: Here is an implementation of the `parseGET()` function (excludes js and CSS files):

```
def parseGET(pkts, myIP):
  print "GET info for IP address {0}:".format(myIP)
  for pkt in pkts:
    payload = str(pkt[TCP].payload)
    if pkt[IP].src == myIP and pkt[TCP].dport== 80 and payload!= "":
      sections=payload.split('\r')
      if "GET" in sections[0] and len(sections)>1:
        info = sections[0].split()[1]
        if "Host" in sections[1]:
          host = sections[1].split(': ')[1]
          url = host+info
          ext = url[-4:]
          if "css" not in ext and "js" not in ext:
            print host+info
```

Hope you have fun packet sniffing! Have students turn in a text file containing a summary (links, e-mails etc.) that they've been able to discover about each individual. What do these people do? What are they doing on the internet?

Reflection

So how do we protect ourselves against packet sniffing? The answer is not to simply use password protected wireless networks. Someone who is intent on stealing your data will likely break the password. Once they are in the network, they can then run the packet sniffer and your information will be lost. So what do we do? If you noticed, there was one individual out of the four (10.3.0.18) that were unable to get any information on. This isn't because it's simply us; our info would have also shown up in the packet capture. Instead, it's because we were **encrypting** our packets before sending them over the web! Encryption algorithms are based on really hard computer science problems (NP-Hard) that will take an incredibly long time to brute force. Consequently, this is considered the only truly "safe" way to transmit packets. Banks, online stores, and other entities that deal with financial transactions on a regular basis encrypt the packets being sent to them. That is why you always connect to an `https` address with them.

One of the good things that come out of Google's packet sniffing debacle is that all Google searches are now encrypted by default. You can also install third party apps on Firefox such as *HTTPS everywhere* that will open a secure, encrypted session with SSL with whatever website you visit, if an SSL channel is available. *Close with a discussion with students what will need to change in order for organizations and governments to adopt packet encryption as mainstream.*