Project 2: Network Project

This project is due on **05Oct2020** at **6 p.m. (1800) CT** and late submissions will be handled as discussed in the syllabus. Submissions will recieve a zero (0) if submitted after 08Oct2020 at 6 p.m. Please plan accordingly for your schedule and turn in your project early if appropriate.

The code and other answers you submit must be entirely your own work. You are encouraged to consult with other students about the concepts of the project and the meaning of the questions via Piazza but you may not look at any part of someone else's solution or collaborate on solutions. You may consult published references, provided that you appropriately cite them (e.g. with program comments), as you would in an academic paper.

Problem 1: Automated Anomaly Detection

In Homework 1, you manually explored a network trace to extract information from the protocols communicating in it. In Project 2, you will programmatically analyze a network trace file to detect suspicious behavior. Specifically, you will identifying hosts which are port scanning across the pcap's vantage point.
As we discussed in-class, port scanning is a technique used to find hosts that have services listening on one or more target ports. It can be used offensively to locate vulnerable systems in preparation for an attack, or defensively for research or network administration. In one kind of port scan technique, known as a "SYN scan", the scanner sends TCP SYN packets and watches for hosts that respond with SYN+ACK packets thereby indicating that they are willing to actively communicate. Since most hosts are not prepared to receive connections on any given port, a much smaller number of hosts will respond with SYN+ACK packets than originally received SYN packets. By observing this effect in a packet trace, you can identify source addresses that may be attempting a port scan.
Your task is to develop a program that analyzes a pcap file in order to detect possible SYN scans. You may do so in any reasonable language that you choose to but are recommended to use Python3[1]. It is *highly* recommended that you use a common and well-known library to parse the pcap file as well as the packets themselves:

- Python3 — dpkt library + tutorial
- Golang — gopacket library + tutorial

Your program will take the path to a pcap file to be analyzed as a command-line argument and should output the set of IP addresses (one per line) that sent more than 3 times as many SYN packets as the number of SYN+ACK packets they received. Your program should silently ignore packets that are malformed or that are not using Ethernet, IP, and TCP.

[1]Python2 is not a "reasonable" language.

What to Submit

You should submit a tarball named `problem_1.tar`[2] containing A) your source code and B) a `Makefile` supporting two commands:

1. `make build` — This should build your application from the included source code. You may assume that `dpkt, gopacket, and libpcap` are installed locally. For Python3, this command may be a no-op.
2. `make run` *filename*`.pcap` — This should cause your locally compiled program to read the input file, determine which (if any) hosts are scanning, and print the IP address of those that are to the screen each on their own line.

You are *highly* recommended to create your own pcap file consisting of a small number of TCP connections for you to develop your program on[3]. You may also use the moderately-sized pcap file available via AU-Box to validate your program's operation. When run, it should output (in no particular order):

```
128.3.23.2
128.3.23.5
128.3.23.117
128.3.23.150
128.3.23.158
128.3.164.248
```

Problem 2: Web Attacks

As we talked about in class, Internet websites are far from a safe and secure technology. In addition to tracking, phishing, and server-launched attacks against the client, a common web vulnerability is injection of content. To highlight not only the dangers of poorly designed website but also to explore with how they can be exploited, you will build two attacks against a "new and improved" version of the National Bank of COMP-5370. After your demonstration of length-extension vulnerabilities in Project 1, the security team has experienced significant turnover and the website was rebuilt from scratch. Instead of attacking the API which locks and unlocks safes, you will turn your attention to the API which shows users their account balances.

Among other changes to the website overall, the URL tokens are now generated using a secure HMAC function and an unknown, but assumed to be long and random, secret is used as the HMAC key[4]. Much like the safes API, the URLs contain information such as the customer account to display, the person that authorized that account to be displayed, and a URL-specific HMAC token to prove the authenticity of that URL. Working examples are provided in the `example-urls.txt` file on Canvas but take the form of:

`https://problem-2.project-2.comp5370.org/view?approver=admin&customer=Bob`
`&token=9487c368b7f0b5f5c58eb6ccf1dd6c2d67f71a73a075409729fe4d899462ff65`

[2]Create a tarball with the bash command `tar -cf problem_1.tar Makefile` *src_file*_1 *src_file*_2 ....

[3]Remember that you can export specific streams from a larger capture using the techniques from Homework 2.

[4]In other words, you would be wasting your time if you try to exploit a length-extension vulnerability.

You have permission to use the API on the problem-2.project-2.comp5370.org server until the 0% deadline to experiment with your attacks. You may use the API as much as you like but **no automated interaction with the server is allowed**. Copy-pasting the URL into a browser is perfectly acceptable and a highly recommended way to check whether it is exploiting the vulnerability as expected. For both problems, you may assume that a user does not examine the URL prior to clicking on it and that they would not understand it even if they did.

Problem 2.1: Content Injection

For this portion, you should explore ways to inject arbitrary text into the website in order to change what is displayed to the viewer. Specifically, you should create a URL that effectively hides the existing charges to the account[5] and replaces them with:

Date	Vendor	Amount
14Sept202	Dunkin Donuts	($5.74)
14Sept202	Domino's Pizza	($17.93)
15Sept202	Carls Jr.	($8.37)
16Sept202	Wendy's	($9.53)
17Sept202	Whole Foods	($32.94)

Problem 2.2: XSS

For this portion, you should expand your technique from 2.1 into a full-on Cross-site scripting attack (XSS). This attack should display an alert box to the viewer that informs them that the website is insecure with the text "I highly recommend you find another bank because your bank's website is insecure."

What to Submit

For Problems 2.1 and 2.2, you should submit an ascii-only text file named `problem_2.txt` that contains your attack links in the form of:

2.1 --- *full URL*

2.2 --- *full URL*

[5]It is not expected to hide the existing charges entirely but they should be hidden in a way that a reasonable user would not be aware of them.