CSCD 330 – Computer Networks

Lab 6, Scapy introduction

Overview:

Today you'll learn about the python library `scapy` and use it to build packets to complete a three-way handshake and send a valid GET request! Similar to lab 4, handling HTTPS is not expected.

Instructions:

Complete the above program following the steps in the next section. Call this file `lab6.py` and include your name at the top of the file in a comment – `#author: YOUR NAME`. No output is expected on `STDOUT` for this lab. Instead, use `tcpdump` to record network traffic while your program is running. You must submit this network recording as `test.pcap`. You must also create a README explaining how to run your program and answering the questions below. Furthermore, you must create a test script in Bash. Call this file test.sh.

Install `scapy` with: `sudo apt-get install python3-scapy`

You may only use the following imports:

```python
from urllib.parse import urlparse
from random import randint
from scapy.all import *
from socket import gethostbyname
from sys import argv
```

Steps:

Step 0: Setup

`Scapy` allows you to create and send custom network packets by operating outside of the kernel's networking stack. As a result, the kernel cannot track the state of any `scapy` formed connections. When the kernel receives a reply from an untracked connection, the kernel will automatically send a reset packet to terminate the connection.

Use `iptables` to drop all these outgoing reset packets:

```
sudo iptables -I OUTPUT -p tcp --tcp-flags ALL RST -j DROP
```

After you are done, restore the default behavior with:

```
sudo iptables -D OUTPUT -p tcp --tcp-flags ALL RST -j DROP
```

Your test script should run both of these commands automatically. Also note that ***you must run your program as sudo.***

Step 1: Three-way handshake

Initiate and complete the 3-way handshake with the target website, given as a URL as command line input. Your TCP initial sequence number **must** be the number of characters in your first name.

Step 2: GET request

Send a valid GET request for the page/path specified by the input URL. Make sure to take a `tcpdump` (the `tcpdump` can be done in terminal and does not need to be scripted) of your code running and requesting the page. The server should reply with the first packet of the HTTP response.

Step 3: Close the connection (15 points of extra credit)

This step is not required. Acknowledge the packets from the server and cleanly close the connection. If you do complete this step, mention that you did the extra credit in your README.

Questions:

1. In lab 4 you sent a GET request using the sockets library. What did the sockets library do for you?
2. Before you submit this lab, you should check that your pcap contains the correct traffic. What program should you use to analyze your pcap? In your pcap, did the server send you the complete HTML for the website or just a portion of the HTML? (Does the response end with a `</html>` tag?)
3. Is your program guaranteed to receive a complete HTML response from the website? Why or why not.
4. Can you merge the final ACK of the three-way handshake with the GET request? That is, can you merge the two packets into one? If yes, explain how such an option might be beneficial.
5. Can `scapy` be used to send other types of packets? If yes, give an example.

Turn in:

Submit a tarball with the following:

- Your source code (in Python) called `lab6.py`

- Your test script (in Bash) called test.sh – test at least 3 inputs.
- Your README answering the above questions and explaining how to run your program.
- A network recording of your program running called `test.pcap`

In case you have forgotten: `tar -czvf lab6_YOURNAME.tar.gz *.py *.sh *.pcap README`

Example output:

python3 lab6.py http://httpforever.com/

```
# No output is expected.
# Use tcpdump to verify correctness.
```

python3 lab6.py http://www.cs.sjsu.edu/~pearce/modules/lectures/web/html/HTTP.htm

```
# No output is expected.
# Use tcpdump to verify correctness.
```