Dr. Feng Cheng (`feng.cheng@hpi.uni-potsdam.de`)
Prof. Dr. Lijuan Duan (`ljduan@bjut.edu.cn`)
October 15th, 2020

Exercise #1
Internet Security – Weaknesses and Targets
Prof. Dr. Ch. Meinel
Winter Semester 2020/21

Topics: TCP/IP and Reconnaissance,
Linux Shell Commands and Usage
Maximum Points: 100

IP and Routing

To understand the lecture, the concepts of addressing, encapsulation and fragmentation using the Internet Protocol (IP) are necessary. Besides those, another important objective of the Internet Protocol is *routing*, meaning the delivery of IP packets over dynamically computed paths. Make yourself familiar with the concepts of *Distance Vector*-based routing protocols (i.e. RIP, Router Information Protocol) and *Link-State* routing (as in OSPF, Open-Shortest-Path-First).

1. Explain the idea of both routing algorithms. (4 points)
2. Point out the main differences. (2 points)
3. Why was Distance Vector routing superseded by protocols basing on Link-State (three possible reasons)? (4 points)

TCP, UDP

The Transmission Control Protocol (TCP) is supposed to be a *reliable* transport protocol, which implements a handshake protocol for connection initialization and a retransmission protocol for the repeated transmission of lost packets.

Theory

1. Explain the details of the TCP handshake protocol using an example. (4 points)
2. How does the adaptive retransmission work within TCP? What would be an optimal retransmission timeout? How is this timeout chosen in real TCP implementations? (4 points)
3. What are the main differences between TCP and UDP? (2 points)

Practice

About *traceroute* (5 points)

1. How does the program *traceroute* work?
2. How reliable are results generated by *traceroute*? Describe two examples, where *traceroute* might return faulty results.
3. Name another tool, which generates similar results as *traceroute* but works in another way.

Reconnaissance - Port Scanning

This task covers network programming and reconnaissance! The goal is to write a simple TCP port scanning tool. - (30 points)

You should use Python RAW sockets for the implementation of the TCP-SYN and TCP-FIN scan techniques. We know, there are sophisticated packages for Python that allow convenient network packet creation, but the use of those (e.g., scapy) is ***not*** intended in this assignment.

GOAL: Develop a simple TCP port scanner in Python.
Choose two of the three mentioned scan techniques and implement these in your scanning application.

- Support TCP-Connect scans using simple sockets and standard network programming
- Support TCP-SYN scans using packet creation of SYN packets and capturing of response packets
- Support TCP-FIN scans using packet creation of FIN packets and capturing of response packets

REQUIREMENTS:

- Take the target, the scanning method, and the ports as command line input.
- Print open port numbers, one number per line.
- Make sure you are scanning the full range of possible ports!
- Example: python scanner.py IP SCAN_METHOD PORTS
  - IP is the IPv4 address of the target
  - SCAN_METHOD is one of the specified scanning techniques (SYN, FIN or Connect)
  - PORTS is a comma separated list of ports(i.e. 1,80,8080,22)
    or a range(i.e. 20-100)
    or a combination of it(i.e. 21,22,100-200,8080)

BONUS If you are interested in this topic, we suggest you to go further and try to implement an operating system detection. This could be done with a look at TCP window sizes, sequence numbers, acknowledge numbers, etc. These characteristics might give hints about the operating system, which finally allow an educated guess. Furthermore, you could extend your scanner to support IPv6 targets.

HINTS Here is a short list with necessary steps as your guidance:

1. Investigate the capabilities of the packet capturing/creation library!
2. What is special about the TCP-SYN/ TCP-FIN scans?
3. Investigate the difference between TCP-SYN, TCP-FIN, and TCP-Connect scans! Design different algorithms to perform such scans!

NOTES: You will lose points if you don't follow the Hand-In requirements!

1. Make sure the source code is of high quality, is *documented well* in a readme.txt file!
2. Make sure your code is *UNIQUE*. We do not accept the programs copying from any other sources. You should design the scanner individually.
3. Place a note about your development environment (operating system + version, interpreter + version) in the leading comments of your main source file.
4. Compress all source files and the readme file into a zip file.
5. Name your submission as `lastname1_lastname2_portscan.zip` (for a two-students team) or `firstname_lastname_portscan.zip`.

Linux Shell Commands and Usage
Linux and Unix systems offer a multitude of tools and mechanisms on the shell. To gather some experience in shell usage on Linux, we designed the following tasks. All shell commands have to be solved by using ONLY the following commands: `head`, `tail`, `cat`, `awk`, `sed`, `grep`, `sort`, `tr`, `time`, `fold`, `cgrep`, `curl`, `cut`, `wc`, and `uniq`. Please use the `bash` shell which should be present on most Linux distributions.

Theory

1. Describe how log files are created and for which purposes they can be used (10 points)
2. How many types of log files the Aparche HTTP server can generate? Explain the structure and usages of these log files respectively (5 points)

Practice

A log file has been prepared for your analysis. Please download (from the link `sample.log`[1]) and perform the following analysis:

[1]`http://172.21.7.85/media/tmp/sample.zip`

1. Write a shell command using a combination of the allowed programs specified above to create a list of unique IP-Addresses which accessed our server. Don't use any scripting! Your command should return the top 10 IP-Addresses (2 points)

2. Write a small application in Python to determine all unique IP-Addresses. Your script should print all IP-Addresses directly to the standard out seperated by a newline (no additional text to standard out). Compare the execution time of your script to the execution time of the shell command of previous task. Name your script logfile −unique−ip.py (8 points)

3. Extend your application to calculate the amount of data which was transferred to each IP. Specify the 5 IP-Addresses with the highest traffic (5 points)
   Note: Name your script logfile −ip−transfer−size.py. The output format should be a list of IP and data amount separated by a colon. Example output:

   ```
   ...
   79.204.95.120: 1737851
   79.205.105.61: 893343
   ...
   ```

4. Calculate how many "GET" requests were made by each IP-Address using shell commands. Specify the top 5 IP-Addresses with the highest amount of requests (4 points)
   Note: Your shell command should return the top 4 IP-Addresses only.

5. Write a shell command to find all requests which do not have a status code of 200 in the response. Specify the overall amount of those requests (2 points)
   Note: Your shell command should only return the number of requests, which you identified.

6. Write a shell command to find the time with the highest amount of requests (2 points)
   Note: The time should contain hours and minutes only. Your shell command should only return the top 5 hour-minute combinations and the corresponding number of requests.

7. Write a shell command to identify the top domains that referred to our server most often. Therefore, extract the referrer from the logs and select the domain from this field. Calculate the times each domain referred to our server (4 points)
   Note: You can assume that the domain is enclosed in the first slashes of the referrer URL (e.g. https://example.com/some/path ! example.com). Finally, return the top 15 domains and the corresponding numbers each domain referred to our server.

8. Determine the browser distribution of the requests, by calculating the percentage for each browser family (e.g., Firefox, Chrome, Internet Explorer, and Android) with a small Python script (4 points)
   Note: You script should print the top 7 browser family names and the corresponding percentages. You should use one of the following Python libraries: `user-agents` or `ua-parser`. Name your script logfile −request−browser.py. Please write one browser family per line. Example output:

```
...
Opera:  3.446965276967444%
...
```

HINTS

1. Regular Expressions (regex) might be useful for this task. For further understanding you can have a look at the log-format of apache2.
2. The shell commands should be written in **a single line** using the concept of **pipes**.

Important Notes:

- You are allowed to organize a team with max. 2 students for finishing the exercises. The team building should be fixed for this assignment as well as the following two assignments.
- All solutions and Email communication with tutors should be written **in English**. Your teamate should be on cc (if you are working in a two-students team).
- The solutions to all the questions (except the coding tasks) should be written into a pdf file named as `lastname1_lastname2_solution_01.pdf` (for a two-students team) or `first_lastname_solution_01.pdf`.
- All submitted files (including the python scripts, the zip file, and the pdf file) should be named correctly following the corresponding instructions. Submissions with any other names will be directly rejected!

Hand-in per Email: November 15th, 2020, 11:59 p.m. (Beijing Time)