LEBANESE AMERICAN UNIVERSITY

**Electrical and Computer Engineering Dept**

COE 431

Computer Networks

Spring 2013

# Project II

## Due date: Tuesday June 4, 2013

## Objective

A port scanning tool can be used to probe machines for open ports. Port scanners are usually employed by network administrators to check to see how secure the machines in the network are. They are also of interest to malicious users since they equip them with the capability of finding machines to compromise. This is done simply by looking for software versions with known vulnerabilities. **Nmap** is an example of a well-known open source port scanner and it can be downloaded from: http://insecure.org/namp.

In this project, your task (if you choose to accept it ☺) is to develop a Java-based TCP port scanner that the network administrators can use to verify that machines on their network are running only expected services. This will help you gain a deeper understanding of the interplay between firewalls, transport protocols and operating systems.

## Scanner details

The program that you will be implementing is expected to identify and recognize the following five port states:

1. **Open**: this means that the application is actively accepting TCP connections on this port. Determining whether or not a given port number is open is one of the primary goals of your program.
2. **Closed**: this state implies that the port is accessible to the port scanner but there is no application using that port.
3. **Filtered**: Filtering devices, such as firewalls, can prevent a port scanner from inferring if a port is open. In such cases, your program should infer if a port is filtered.

For example, to explore a specific TCP port, say port 6789, on a target host, **nmap** will send a TCP SYN segment with destination port 6789 to that host. There are three possible outcomes:

1. *The source host receives a TCP SYNACK segment from the target host*. Since this means that an application is running with TCP port 6789 on the target host, namp return “open”.
2. *The source host receives a TCP RST segment from the target host*. This means that the SYN segment reached the target host, but the target host is not running an application with TCP port 6789. But the attacker at least knows that the segment destined for the host at port 6789 are not blocked by any firewall on the path between the source and target hosts.
3. *The source receives nothing*. This likely means that the SYN segment was blocked by an intervening firewall and never reached the target host.

## Project specification

The basic idea behind a port scanner is simple: given the IP address of a machine and a list of interesting TCP ports to scan, the scanner will connect on each port using TCP sockets, then determine whether the port is in one of the above-listed states before moving on to the next port to scan.

Your scanner should support the following options:

- -help (display invocation options). When your port scanner is invoked on the command line argument with this option, it should display the various options available to the user.
- -ports (ports to scan). Your port scanner should scan all well-known ports [1-1024] by default. However, if this option is specified, it will scan the ports specified by the user on the command line instead. The latter could be individual ports separated by commas, or even a range.
- -IP (IP address of the host whose ports should be scanned).

After each invocation, your port scanner should output a succinct summary of the list of TCP ports on the user-supplied IP address along with their associated states. Additionally, for each open port, it will include the name of the service that is likely running. To find the services associated with port numbers 1 through 1024, visit: http://www.iana.org/assignments/port-numbers.

As the machines you are expected to scan are not under your control, your program should be designed to handle smartly bad responses from the remote machine. No-response from the remote machine should not cause your program to crash. Similarly, your program should timeout in a reasonable period of time (generally a handful of seconds), no matter how the remote machine responds.

## Resources and restrictions

Begin by familiarizing yourself with the **nmap** software tool. A simple starting point is to scan your machine. For this project, you are required to use the Java language. You should consider using an external Java library to complete your project especially that Java does not support opening sockets through socket system calls.

**You must not copy any code from the internet and you must make sure to be explicit about what resources you use, including web tutorials and any other web resources.**

## What to turn in?

The project is due at the beginning of class on the due date. You have to turn in the following material in both hard and soft copies.

| **Criteria** | **Percentage** |
|---|---|
| **Documentation** of your solution including explanations and illustrations in one or two pages along with short write-up of questions and/or problems that you encountered while doing this assignment. | 2 pts (10%) |
| **Source code** that contains an appropriate amount of comments. Well-organized and correct code receives 16 pts, messy yet working code receives 12 pts, code with bugs receives 4 pts, and incomplete code receives 1 pt. | 16 pts (80 %) |
| **Execution output** such as a snapshot of the contents of standard output. A correct output receives 2 pts, the one with minor errors receives 1 pts, and an incomplete output receives 0 pts. | 2 pts (10%) |
| **Total** | 20pts (100%) |