Lecture 23: Port and Vulnerability Scanning, Packet Sniffing, Intrusion Detection, and Penetration Testing

Lecture Notes on "Computer and Network Security"

by Avi Kak (kak@purdue.edu)

April 17, 2017
12:20am

Goals:

- Port scanners
- The nmap port scanner
- Vulnerability scanners
- The Nessus vulnerability scanner
- Packet sniffers
- Intrusion detection
- The Metasploit Framework
- The Netcat utility

CONTENTS

23.1: PORT SCANNING

- See Section 21.1 of Lecture 21 for the mapping between the ports and many of the standard and non-standard services. As mentioned there, each service provided by a computer monitors a specific port for incoming connection requests. There are 65,535 different possible ports on a machine.

- The main goal of port scanning is to find out which ports are **open**, which are **closed**, and which are **filtered**.

- Looking at your machine from the outside, a given port on your machine is **open** if you are running a server program on the machine and the port is assigned to the server. If you are not running any server programs, then, from the outside, no ports on your machine are open. This would ordinarily be the case with a brand new laptop that is not meant to provide any services to the rest of the world. But, even with a laptop that was "clean" originally, should you happen to click accidently on an email attachment consisting of malware, you could inadvertently end up installing a server program in your machine.

- When we say a port is **filtered**, what we mean is that the packets passing through that port are subject to the filtering rules of a firewall.

- If a port on a remote host is **open** for incoming connection requests and you send it a SYN packet, the remote host will respond back with a SYN+ACK packet (see Lecture 16 for a discussion of this).

- If a port on a remote host is **closed** and your computer sends it a SYN packet, the remote host will respond back with a RST packet (see Lecture 16 for a discussion of this).

- Let's say a port on a remote host is **filtered** with something like an `iptables` based packet filter (see Lecture 18) and your scanner sends it a SYN packet or an ICMP ping packet, you may not get back anything at all.

- A frequent goal of port scanning is to find out if a remote host is providing a service that is vulnerable to buffer overflow attack (see Lecture 21 for this attack).

- Port scanning may involve all of the 65,535 ports or only the ports that are well-known to provide services vulnerable to different security-related exploits.

23.1.1: Port Scanning with Calls to `connect()`

- The simplest type of a scan is made with a call to `connect()`. The manpage for this system call on Unix/Linux systems has the following prototype for this function:

```
#include <sys/socket.h>

int connect(int socketfd, const struct sockaddr *address, socklen_t address_len);
```

  where the parameter **`socketfd`** is the file descriptor associated with the internet socket constructed by the client (with a call to three-argument `socket()`), the pointer parameter **`address`** that points to a **`sockaddr`** structure that contains the IP address of the remote server, and the parameter **`address_len`** that specifies the length of the structure pointed to by the second argument.

- A call to `connect()` if successful completes a three-way handshake (that was described in Lecture 16) for a TCP connection with a server. The header file `sys/socket.h` includes a number of definitions of structs needed for socket programming in C.

- When `connect()` is successful, it returns the integer 0, otherwise it returns -1.