Chapter 15: Socket Security

1. Sockets 101 (ok more like 99)
    a. A socket represents a single connection between two pieces of software – typically on different machines, but can be on the same machine
    b. Bidirectional
    c. Often discussed in terms of a client (the one who initiates contact) / server relationship, but this is not always the case
    d. Each OS has a set of socket libraries
        i. Berkeley socket library on *nix
        ii. Winsock on windows
        iii. There are others…
        iv. Functions like open( ) , close( ) , read( ), and write( ) are used
    e. Three interface types to sockets
        i. Connection based (stream socket -- TCP): two parties establish contact, then exchange data
        ii. Connection-less (datagram based -- UDP): either party sends a packet and does not care about acknowledgment from the other – these connections are 'fast'
        iii. Raw socket -- IP: used for low level protocol development
    f. Addresses and Ports
        i. Connection involves an IP address and a port (http is port 80, smtp is port 25, SSH is 22, DNS is 53, FTP is 20 and 21, etc.) (large numbered ports (> 1000) are for general programming use)
    g. Establishing connection (TCP 3-way handshake)
        i. The active open is performed by sending a SYN to the server.
        ii. In response, the server replies with a SYN-ACK.
        iii. Finally the client sends an ACK (usually called SYN-ACK-ACK) back to the server.
    h. Sending data with TCP allows for the following
        i. Error-free data transfer
        ii. Ordered data transfer
        iii. Retransmission of lost packets
        iv. Discarding duplicate packets
        v. Congestion throttling
    i. Closing connection (TCP 4-way handshake)
        i. FIN (from first party)
        ii. ACK (from second party)
        iii. FIN-ACK (from first party)
        iv. FIN-ACK (from second party
2. TCP and UDP do not mitigate current threats – IPSec helps
3. Server hijacking: when an application allows a local user to intercept and manipulate information meant for a server that the local user did not start herself
    a. Socket is first bound to a port
    b. More than one socket can be bound to the same part (this is the problem)

        i. Socket libraries will send packet to address whose binding is most specific (so a specific IP address will take precedence over a connection that uses INADDR_ANY (which says any IP address))
        ii. So you should not establish a general connection if possible
            1. can specify all IPs on the server (this can be lengthy)
            2. SO_EXCLUSIVEADDRUSE stops this on old windows machines (things before XP SP2)
            3. Use DACL to allow access to current user and administrators only to the socket
4. TCP window attack
    a. Silly window: TCP uses a window size advertisement in ACK packets to help server send data no faster than client can receive
    b. If client buffers are full, window of size 0 can be specified
    c. This causes server to wait to send more data
    d. A malicious client sets window size to very small and causes server to send data slowly (and with high overhead) (for every few bytes of data, there are about 40 bytes worth of TCP/IP stuff in one of these packets)
    e. Defend by checking returns on send calls – if something is unacceptably small (and thus slow) for a period of time, terminate the connection
5. Server interfaces: how to choose and set up
    a. Reduce the number of interfaces exposed to outside world (reduce your surface area of attack)
    b. Perform filtering
    c. Take advantage of router and firewalls (these can be compromised)
    d. Try and make sure any service that will be provided allows for
        i. Specifying which interface is listening
        ii. Specifying which IP address and port it will listen on
        iii. Specifying which clients can connect to the service
6. How to accept connections
    a. UDP: obtain IP and port and then decide whether to process request
    b. TCP: because of handshaking process, attacker can slow system down by making and breaking connections, or by hacking her IP stack so it does not respond to handshaking requests (causing your server to wait needlessly)
        i. Set delay in response to be small for handshaking (thus causing a quicker close if no response)
        ii. Can write code to decide whether to even accept connection in the first place (perhaps use a DNS lookup)
7. Firewall friendly code
    a. Utilize open ports through firewall if at all possible
    b. Sometimes communication only needs to be one way so use UDP
        i. Faster
        ii. Does not allow/need data to come through port from outside
    c. Use one connection only (if possible)
    d. Avoid connections back to client (if possible)
    e. If bidirectional communication is necessary, use a connection-based protocol (they are easier to secure)

    f. Avoid host IP addresses in app layer data (information disclosure that can lead to attacks on those hosts)
8. Spoofing and Host-based and Port-based trust
    a. Spoofing involves: attacker, victim, innocent third party
        i. Attacker makes victim believe a connection is from an innocent system
        ii. Trivial to do this with connection-less protocols (find a good host, tinker with source address of packets, send the packets on their way)
    b. DNS corruption
    c. Remember you can use a shared secret to confirm identity
    d. Ports: make sure port is legit (just because it's a low numbered port (< 1024) does not mean it must be ok)
    e. IPv6
        i. 128 bit size: makes it infeasible to scan internet for all IP addresses (as you could with IPv4)
        ii. Link local, site local, global
            1. gives much finer control over what your app will accept
            2. site local cannot be routed globally
        iii. IPSec is always present with IPv6
            1. packet privacy
            2. packet integrity