Introduction to Network Security – Part 3

acarlstein.com/

Posted by Alejandro G. Carlstein Ramos Mejia on October 27, 2010 October 28, 2010 About Programming / Network Security

NOTIFICATION: These examples are provided for educational purposes. The use of this code and/or information is under your own responsibility and risk. The information and/or code is given 'as is'. I do not take responsibilities of how they are used.

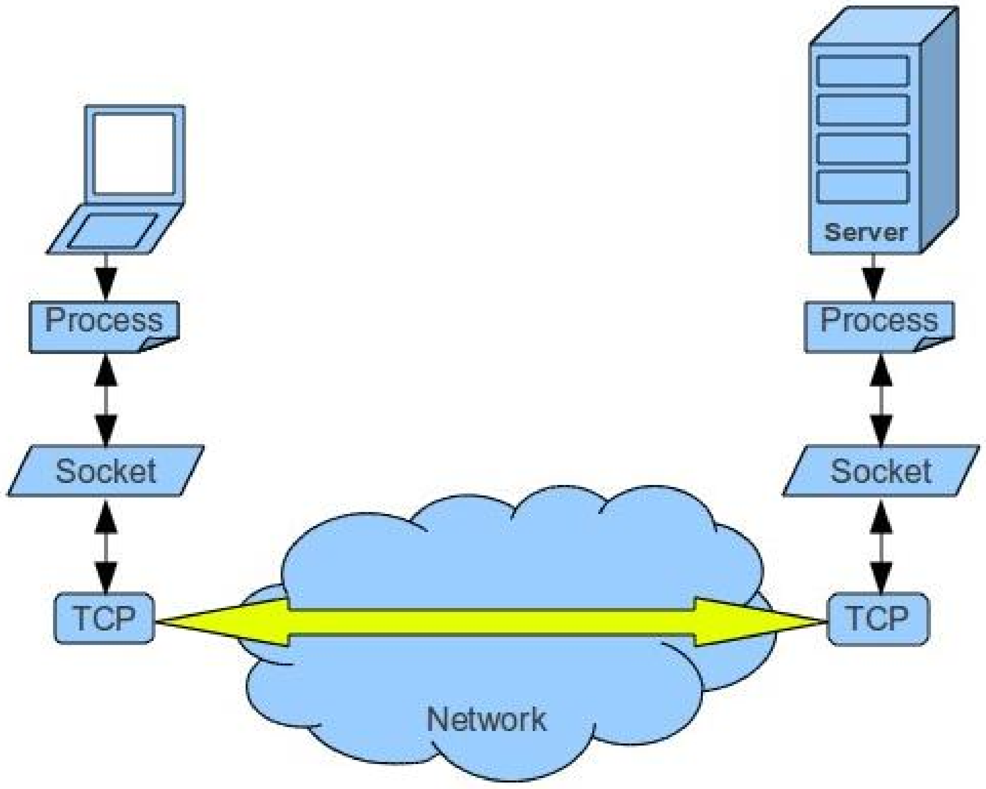
Socket Programming

In networks applications, communications are almost always done using a client-server model.

A client will perform request for a service to the server, and receive a service from the server (which should be usually on). The only thing that the client needs to know is the address of the server and which port to use. Once the connection between the client and the server is establish, then the client and the server can send and receive information back and forwards. Eg. FTP client to a FTP server

The client usually should communicate with only one server at a time; however, there are exceptions such as the web browser. In today web browsers, a web browsers could connect to different servers in order to download all the elements of a page faster. A page could indicate that the images are from one server, while the flash animation is coming from a different server.

In order to a client and a server to communicate they must use sockets in order to send and receive messages from each other. A socket is used by a sending process to send a message to the transport infrastructure which deliver the message to a receiving process which also is using a socket to receive the message.



The first step for a process that is receiving a message is to have an identifier. This identifier must include the IP address and a port number. The port number must be associated with the process. The port number is a 16-bit number used to identify the application process. Notice that an IP address do not identify a process but the port does, so you can have more than one process running at the host which means different ports can be used at the same time with the same IP address.

IP Address

Right now, October 27 of 2010, there are two Internet Protocols (IP) available: IPv4 and IPv6.

1. IPv4 (IP version 4) is a 32-bit binary number represented by using 4 decimal values, separated by periods, in dotted decimal notation. Each decimal value is an octed (8-bits) which range between 0 to 255. Example: 192.168.1.101 would be 11000000.10101000.00000001.01100101.
    1. 192 → 11000000
    2. 168 → 10101000
    3. 1   → 00000001
    4. 101 → 01100101
2. IPv6 (IP version 6) is designed to succeed IPv4 (IP version 4). Instead of a 32-bit binary number, IPv6 use 128-bits. This means that the use of Notation Address Translation (NAT) is not needed as in IPv4. Network security such as IPSec (http://en.wikipedia.org/wiki/IPsec) which allow to authenticate and encrypt IP

packages have being incorporated.

Ports

As explained previously a port is a 16-bit identifier number which identify the type of application process, this means that there are a total number of 65535 ports number available. For more information you can go to the Internet Assigned Numbers Authority (IANA) <http://www.iana.org> . This institution is in charge of the global coordination of the DNS Root, IP addressing, and other Internet protocol resources.

1. Ports from 0 to 1023 are called well-known ports.
    1. Examples of reserved ports: HTTP (Port 80), FTP (Port 21), Telnet (Port 23), STMP (Port 25), etc.
2. Ports from 1024 through 49151 are called reserved ports.
3. Ports from 49152 to 65535 are called Dynamic ports, Private ports, or Ephemeral ports.
4. For more information you can go to <http://www.iana.org/assignments/port-numbers>

TCP Socket on Client Side.

1. Create a socket to create an endpoint for communication.
    1. (In Linux) ANSI C:

       ```
       int socket(int domain, int type, int protocol);
       ```

    2. For more information: <http://linux.die.net/man/7/sockets>
2. Specify the server's IP address and port to connect
3. Establish a connection initiate a connection on a socket with the server.
    1. (In Linux) ANSI C:

       ```
       int connect(int sockfd, const struct sockaddr *addr,  socklen_t addrlen);
       ```

    2. For more information: <http://linux.die.net/man/2/connect>
4. Send data to the server and receive data from the server
5. Close the connection

Here is an example of a client/server program using ANSI C in Linux: <http://www.acarlstein.com/?p=891>

TCP Socket on Server Side

1. Create a socket to create an endpoint for communication.
    1. (In Linux) ANSI C:

       ```
       int socket(int domain, int type, int protocol);
       ```

    2. For more information: <http://linux.die.net/man/7/sockets>
2. Bind the socket with an address to specify the server's IP address and port.
    1. (In Linux) ANSI C:

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

   2. For more Information: <http://linux.die.net/man/2/bind>
3. Listen form incoming connections.
   1. (In Linux) ANSI C:

```
int listen(int sockfd, int backlog);
```

   2. For more information: <http://linux.die.net/man/2/listen>
4. Accept the connection with the client.
   1. (In Linux) ANSI C:

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

   2. For more information: <http://linux.die.net/man/2/accept>
5. Send data to the server and receive data from the server
6. Close the connection

Here is an example of a client/server program using ANSI C in Linux: <http://www.acarlstein.com/?p=891>