Understanding WEB Server

A. Objectives
1. Install web server in Ubuntu 12.04
2. Learn about basic Python Network Programing
3. Practice with Shell scripting
4. Learn about netstat command
5. Monitor web traffic
6. Learn about HTTP protocol

B. Time of Completion
This laboratory activity will take about 2 hours to complete.

C. Requirements
Read the FTP lab for more information on required commands in this lab.

D. Procedure

Pay attention to how the PC numbers are setup. As you complete each part respond to each question. Submit

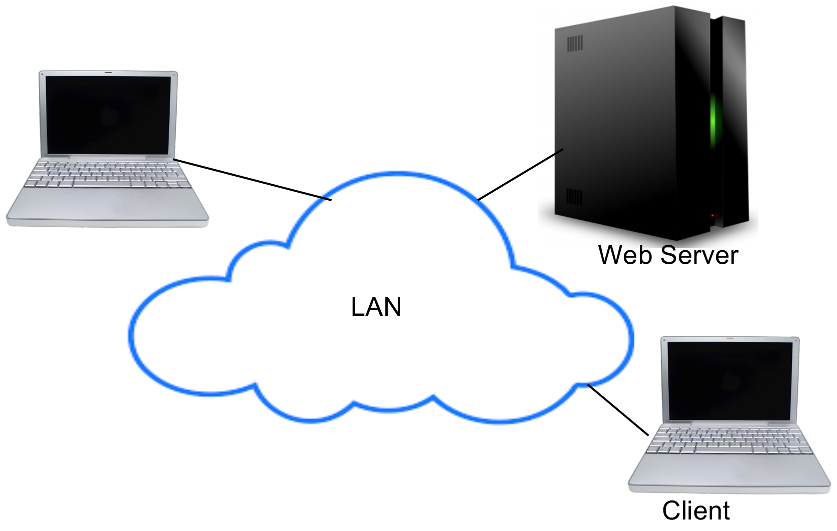


Figure 1 – Example of Web Server Connection to the Clients in the Lab.

PART I

In this section you are expected to setup your own web sever in Ubuntu 12.04.

1. Make sure your computer is connected to the Internet.
2. Make sure you `update` your Ubuntu OS by typing the appropriate command in a terminal. **IMPORTANT**: **DO NOT** `upgrade` the OS and avoid using the update manager in Ubuntu. Note that this may take a few minutes.
3. Install the `appache2` package using the appropriate command in your terminal. Note that this may also take a while. Eventually, you should see a message indicating that the server has started, as shown below.

```
* Starting web server apache2
apache2: Could not reliably determine the server's fully qualified domain name,
using 127.0.1.1 for ServerName
                                                                        [ OK ]
Setting up apache2 (2.2.22-1ubuntu1.7) ...
Processing triggers for libc-bin ...
ldconfig deferred processing now taking place
```

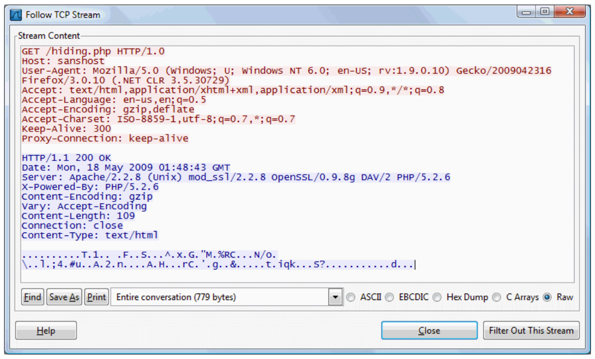
4. Identify the following on your server:

IP address	
localhost address	
mask address	
domain name	
hostname	

5. In your browser type `http://localhost/` What happens? What do you see?
6. Change the localhost to your IP address in the above URL address. What happens?
7. Using `netstat` command identify which ports are open and listening. You can use `netstat -a | more`
8. What happens if you use `-an` option for the above command?
9. Use `clear` command to clear your terminal.
10. Stop the server by typing `sudo /etc/init.d/apache2 stop`
11. Try opening the web page by typing your localhost IP address in the URL. What happens? What is the IP address you typed?
12. Start the web server by typing the appropriate command.
13. Go to `/var/www` directory. What file do you see? Copy it as a backup file. Call it `index_back.html`
14. Using `nano` editor, edit the file (`index.html`) and type your name in there. Save the file. Make sure the file extension does not change.
15. Go back to your browser and make sure you can see the changes in the browser.
16. Ask your partner from a different station to log into your web server. The remote user should see the web page.

Show your results to the Instructor before you proceed!

17. At this point run the Wireshark on the web server.
18. Using a different machine access the server and record the packets in a file. Analyze the captured packets and clearly explain the packet types that are exchanged between the client and the server.
    - What is the widow size?
    - What are the flag types?
    - Using a timing diagram show all the packet types.
    - Can you see web page content? How? Take a snapshot. You should see something like below.



Follow TCP Stream

Stream Content

```
GET /hiding.php HTTP/1.0
Host: sanshost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.0.10) Gecko/2009042316
Firefox/3.0.10 (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive

HTTP/1.1 200 OK
Date: Mon, 18 May 2009 01:48:43 GMT
Server: Apache/2.2.8 (Unix) mod_ssl/2.2.8 OpenSSL/0.9.8g DAV/2 PHP/5.2.6
X-Powered-By: PHP/5.2.6
Content-Encoding: gzip
Vary: Accept-Encoding
Content-Length: 109
Connection: close
Content-Type: text/html

..........T.1.. .F..S...^.x.G."M.%RC...N/o.
\..l.;4.#u..A.2.n....A.H...rC.'.g..&.....t.iqk...S?..........d...
```

Entire conversation (779 bytes)   ASCII   EBCDIC   Hex Dump   C Arrays   [✓ Raw]

19. Stop the server.
20. Using a different machine log into the server. Run the Wireshark software on the **client**.
21. What type of TCP packets (in terms of flag values) the client receives?
22. Restart the server. What command did you use?
23. Ask a random client in the lab to log into your web server. Does it work?

PART II
In this section you analyze the web traffic.

1. Using Wireshark it is possible to monitor the web users. Start the Wireshark first. Then have three different users try to access the web server exactly at the same time. Save the file and analyze the captured file.
   - How does the port addresses change?
   - Take a snapshot and clearly explain what is happening.
2. Change your web page so that you allow users to download a file (`Wireshark-win32.exe`). Here is an example HTML file. Note that all files must be in the same directory.

```
<!DOCTYPE html>
<HTML><HEAD><TITLE>Example File</TITLE></HEAD>
<BODY>
<h1>My Name Is Steve! </h1>
<p>Please feel free to download my <a href="Wireshark-win32.exe">Wireshark</a> file!<BR>
</p>
<P>&nbsp;</P>
</BODY></HTML>
```

- Start `Wireshark`. Download the large file (`Wireshark-win32.exe`) from the web server. Download the file TWO more times (total of three times). Once the download process is completed stop `Wireshark`. Save the captured packets in a file; call it `Captured_WEB_FILE`.
- Answer the following questions:
  a. On Average, how long did it take to download the files?
  b. On average, how many bytes you downloaded?
  c. What does command *get* do?
  d. How can you *upload* a file in the ftp server?

Open your `Captured_WEB_FILE` using *wireshark*. Answer the following questions.

3. Go to Statistics→IO Graphs. Note that the graph can be saved using the **SAVE** button in *jpeg* or any other format. Answer the following questions:
    a. Take a snapshot of the IO Graphs as shown by *wireshark.*
    b. How many **peaks** do you observe in the graph? Why?
    c. In terms of Bytes/Second what is the peak transmission rate for each peak?
    d. Go to Statistics→End Points. Click on IPV4. How many bytes your terminal received during this transition?
    e. Go to Statistics→End Points. Click on IPV4. How many packets your terminal received during this transition?
    f. What is the MAC address of the WEB server?

PART III
General questions:

1. How is HTTP application encapsulated? Use a diagram to show it.
2. Write a Shell script that shows all the open ports on your machine. The result of your script should look something like this (**NO** UDP or TCP6 must be included):

```
tcp4       0      0  127.0.0.1.56083        *.*                    LISTEN
tcp4       0      0  127.0.0.1.56082        *.*                    LISTEN
tcp4       0      0  127.0.0.1.26166        *.*                    LISTEN
```

3. It is possible to write a simple HTTP client to fetch some data from the web server. Use the code below and download the data from your Web server. Note that this code must be run from the client side. Run the code using the following command format on the client machine:
`$>python file_name.py --host=your_address`

Python code to download data from an HTTP server:

```
!/usr/bin/env python
# This program is optimized for Python 2.7.

import argparse
import httplib

REMOTE_SERVER_HOST = 'YOUR_WEB_SERVER'
REMOTE_SERVER_PATH = '/'

class HTTPClient:   #this class fetches the data from the remote server

  def __init__(self, host):
    self.host = host

  def fetch(self, path):    # This is what is actually grabbed.
    http = httplib.HTTP(self.host)

    # Prepare header
    http.putrequest("GET", path)
    http.putheader("User-Agent", __file__)
    http.putheader("Host", self.host)
    http.putheader("Accept", "*/*")
    http.endheaders()

    try:
      errcode, errmsg, headers = http.getreply()

    except Exception, e:
        print "Client failed error code: %s message:%s headers:%s" %(errcode, errmsg, headers)
    else:
      print "Got homepage from %s" %self.host

    file = http.getfile()
    return file.read()

if __name__ == "__main__":
  parser = argparse.ArgumentParser(description='HTTP Client Example')
  parser.add_argument('--host', action="store", dest="host",  default=REMOTE_SERVER_HOST)
  parser.add_argument('--path', action="store", dest="path",  default=REMOTE_SERVER_PATH)
  given_args = parser.parse_args()
  host, path = given_args.host, given_args.path
  client = HTTPClient(host)
  print client.fetch(path)
```