

INTRODUCTION TO LAB 2
AND
SOCKET PROGRAMMING

Vengatanathan Krishnamoorthi , Minh-Ha Le

BEFORE WE
START ...



Soft deadline for lab 2: Apr 24



Finish assignment 1 as soon as possible if you have not yet.



Hard deadline for assignments: May 20

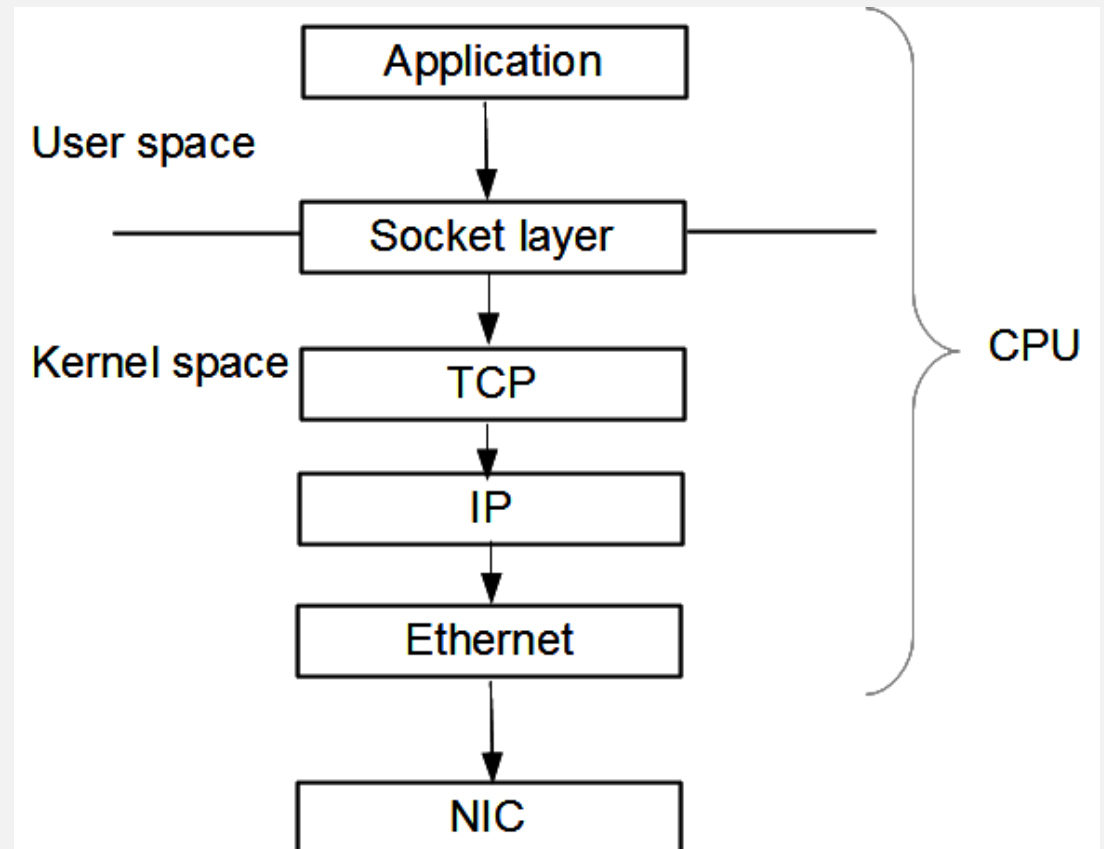


Check with the TA if you plan to use languages other than those prescribed

WHAT WILL WE DO IN LAB 2?

Goals:

- Learn about WWW and HTTP
- Learn TCP/IP socket programming to understand HTTP and WWW better
- Build a simple proxy



WHAT IS WWW?



It is a world-wide system of interconnected servers which distribute a special type of document.



Documents are marked-up to indicate formatting (Hypertexts)

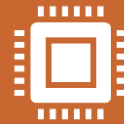


This idea has been extended to embed multimedia and other content within the marked-up page.

WHAT IS HTTP?



HTTP is WWW's application layer protocol.



HyperText Transfer Protocol (HTTP) to transfer HyperText Markup (HTML) pages and embedded objects.



Works on a client-server paradigm.



Needs reliable transport mechanism (TCP).

HTTP



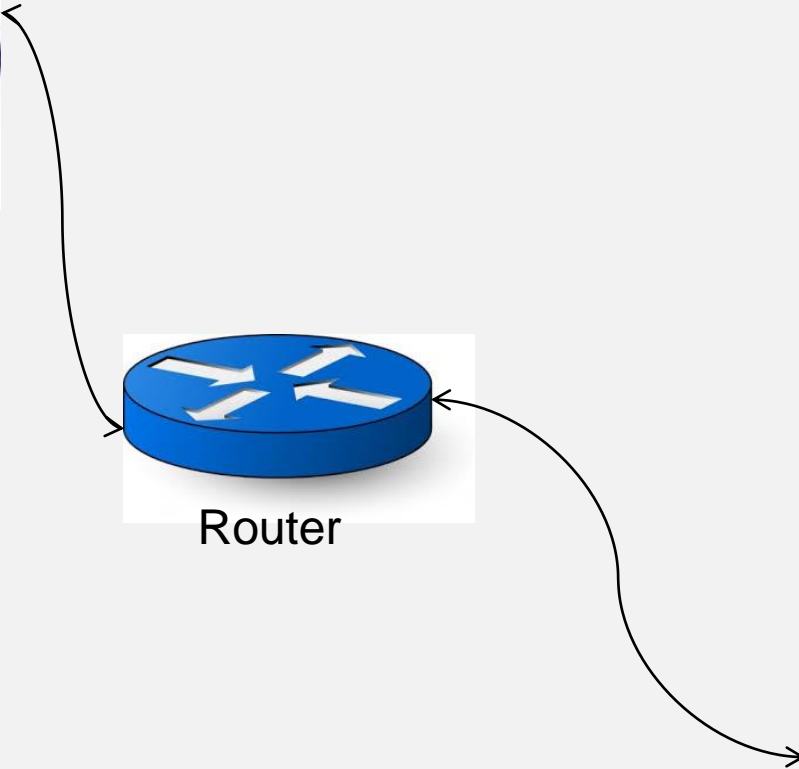
Server



Router



Client



HTTP



Server

Note: HTTP server always runs on port 80



Router



Client

HTTP



Server

Note: HTTP server always runs on port 80

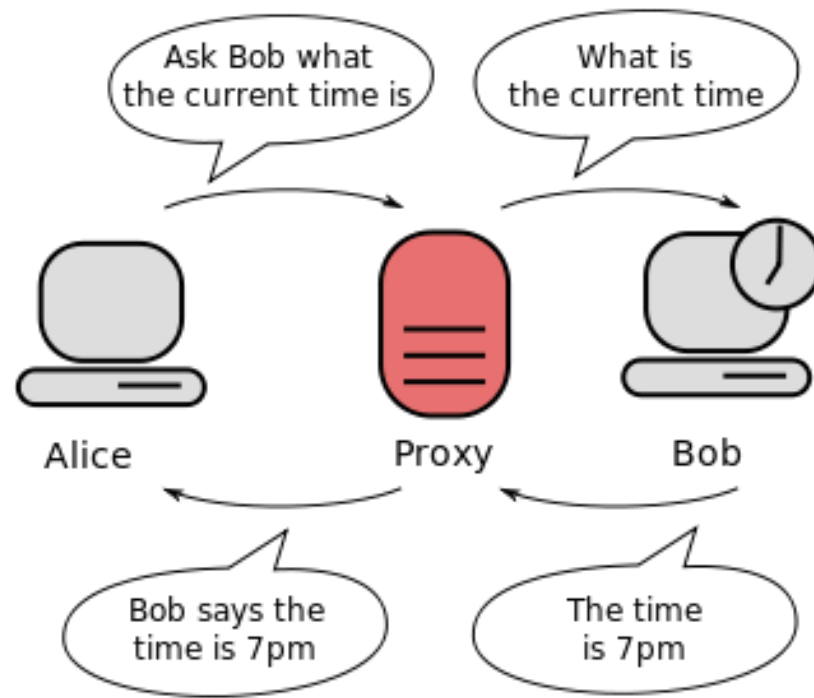


Router

Note: Client can use any unrestricted port
Generally >1024



Client



PROXY

Acts as intermediary between client and server.

BENEFITS OF A PROXY



Hide your internal network information (such as host names and IP addresses).



You can set the proxy to require user authentication.



The proxy provides advanced logging capabilities.



Proxy helps you control which services users can access.



Proxy-caches can be used to save bandwidth.

HTTP WITH PROXY



Server

Note: HTTP server always runs on port 80

Proxy listens on a port (>1024) and talks to server on another (>1024)

Router



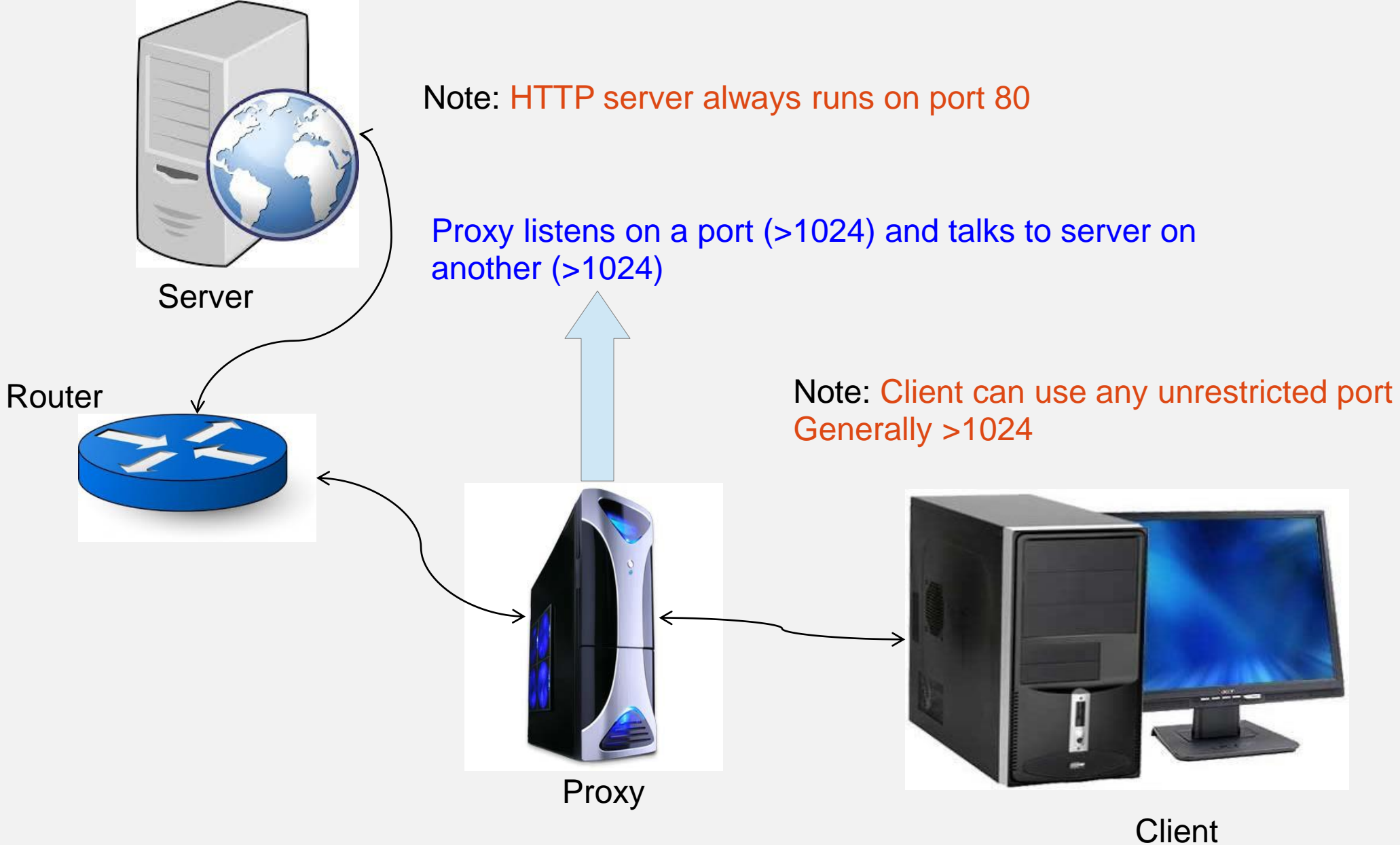
Note: Client can use any unrestricted port
Generally >1024



Proxy



Client



WHAT IS A PORT?

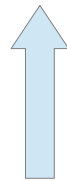
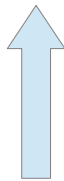
- A port is an application-specific or process-specific software construct serving as a communications endpoint.

WHAT IS A PORT?

- A port is an application-specific or process-specific software construct serving as a communications endpoint.
- The purpose of ports is to uniquely identify different applications or processes running on a single computer and thereby enable them to share a single physical connection to a packet-switched network like the Internet.

PORT CONT

- Port only identifies processes/applications.
- With regard to the Internet, ports are always used together with IP.
- Notation 192.168.1.1:80

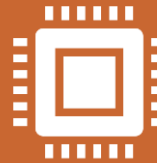


IP address Transport protocol port
UDP/TCP

SOCKET PROGRAMMING



These are software constructs used to create ports and perform operations on them.



It is a way to speak to other programs using standard Unix file descriptors.



We will talk about these types of sockets:

- Datagram socket
- Stream socket
- SSL sockets

DATAGRAM SOCKETS

They are connectionless

Do not guarantee in order delivery

No form of loss recovery

No congestion control

No flow control

Uses different calls (sendto and receivefrom)

DATAGRAM SOCKETS

They are connectionless

Do not guarantee in order delivery

No form of loss recovery

No congestion control

No flow control

Uses different calls (sendto and receivefrom)

Datagram sockets use UDP

STREAM SOCKETS



Connection oriented sockets



In order and guaranteed delivery



Error identification and recovery



Congestion control



Flow control



Stream sockets use TCP protocol



SSL sockets are similar to stream sockets, but include functions to handle encryption

SOCKET PROGRAMMING CALLS

- **getaddrinfo()**
 - Get address information
 - Takes as input
 - Host name
 - Service type (HTTP) or only port number if local
 - Information about IP family(v4 or v6), type of socket. (struct addrinfo)
 - Returns
 - A pointer to a linked list. Lets call this 'result'

SOCKET PROGRAMMING CALLS

- **socket()**
 - Takes as input
 - Address family
 - Socket type
 - Protocol
 - Returns
 - Socket object

SOCKET PROGRAMMING CALLS

- **bind()**
 - Takes as input
 - Address information obtained from `getaddrinfo()`
- What does this do?
 - Associate the socket with a port number

SOCKET PROGRAMING CALLS

- **listen()**
 - Takes as input: None
- Setting socket object on listening mode
- This must run at the server side to listen to incoming connection

SOCKET PROGRAMING CALLS

- **connect()**
 - Takes as input
 - Address information obtained from `getaddrinfo()`
- What does this do?
 - Attempts to setup a connection with the other end

SOCKET PROGRAMING CALLS

- **accept()**
 - Takes as input
 - Socket object
 - Address info
- Reads through the backlog and picks one from the list to connect to it.
- Runs at the server side

SOCKET PROGRAMING CALLS

- **send()**
 - Takes as input
 - Socket object
 - Message
 - Length
 - Returns
 - Number of bytes sent
- Send is always best effort. If it cant send the whole message, the value returned is smaller.

SOCKET PROGRAMING CALLS

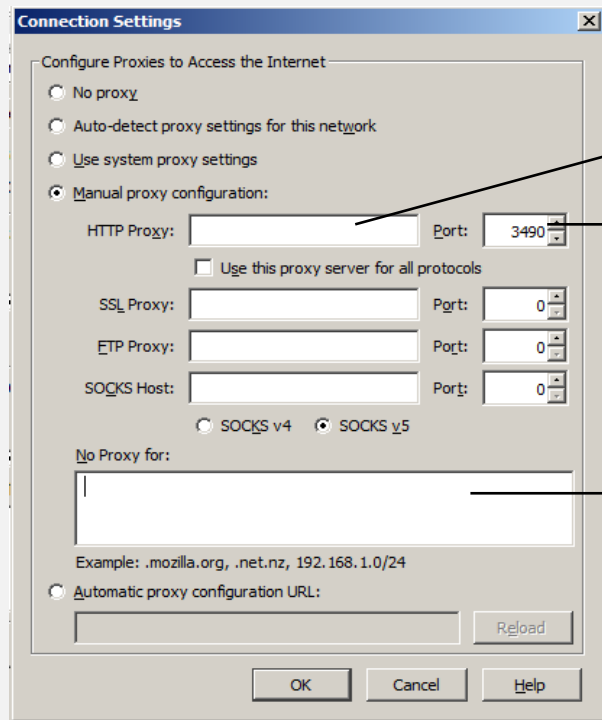
- **recv()**
 - Takes as input
 - Socket object
 - Buffer
 - Max buffer length
 - Returns
 - Number of bytes received
 - Or -1 on error

SOCKET PROGRAMING CALLS

- **close()**
 - Takes as input
 - Socket object
- Closes the stream socket (TCP connection tear down)

BROWSER CONFIGURATION

- Proxy listens on a particular port



→ 127.0.0.1

→ Proxy's port number

→ Make sure it is blank

```
Transmission Control Protocol, Src Port: 50139 (50139), Dst Port: http (80), Seq: 1, Ack: 1, Len: 276
Hypertext Transfer Protocol
GET /vod/final_1.3.f4m HTTP/1.1\r\n
Host: 130.236.182.199\r\n
Connection: keep-alive\r\n
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.103 Safari/537.36\r\n
Accept-Encoding: gzip,deflate, sdch\r\n
Accept-Language: en-US,en;q=0.8,ms;q=0.6\r\n
\r\n
[Full request URI: http://130.236.182.199/vod/final_1.3.f4m]
```

HTTP BASICS

- Recollect lab 1. It contains things that you need in lab 2.
- HTTP request
 - Get
 - Syn, SynAck, Ack

```
ssion Control Protocol, Src Port: http (80), Dst Port: 50139 (50139)
sembled TCP Segments (5595 bytes): #248(1460), #249(1460), #251(1460)
xt Transfer Protocol
1.1 200 OK\r\n
  Sun, 07 Sep 2014 10:06:36 GMT\r\n
  r: Apache/2.2.17 (Unix) DAV/2\r\n
  nt-Length: 5354\r\n
  Modified: Tue, 04 Feb 2014 12:25:40 GMT\r\n
  Alive: timeout=15, max=100\r\n
  ction: Keep-Alive\r\n
  nt-Type: text/xml\r\n
```

HTTP BASICS

- HTTP response
 - OK

HTTP BASICS

HTTP 1.0 vs HTTP 1.1

- Many differences read <http://www8.org/w8-papers/5c-protocols/key/key.html>
- For this assignment
 - Connection: close
 - Handshake-Get-response-OK-Teardown
 - Connection: keep-alive
 - Handshake-Get-response-OK-wait-Get-response

What should you use for the proxy?

HOW TO HANDLE CONNECTIONS



With connection: keep-alive, the connection is kept open. You are responsible to figure out when the response is completed.



With connection: close, the server closes the connection after the response is sent.

HOW TO HANDLE CONNECTIONS



With connection: keep-alive, the connection is kept open. You are responsible to figure out when the response is completed.



With connection: close, the server closes the connection after the response is sent.



How can you enforce connection: close on HTTP 1.1?

ASSIGNMENT 2 DESCRIPTION

- You are to develop a “Fake News” proxy that can modify the content sent from the server before returning to the browser.
- Build a proxy to which a user can connect to
- The proxy connects to the server on the user's behalf (recollect how proxy works)
- Proxy receives the response from the server
- Alters any occurrences of Smiley and Stockholm
- Redirects the (potentially) altered content to the user

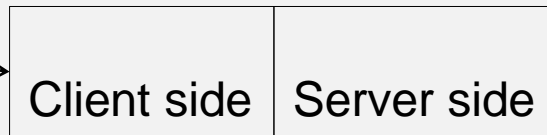
ASSIGNMENT 2 REQUIREMENTS

- • Handles simple HTTP GET interactions between client and server
- Alters any text occurrences of “Smiley” and “Stockholm”
- Replaces any images of Smiley
- Uses at least one TCP socket
- Imposes no limit on the size of the transferred HTTP data
- Uses only *basic* libraries (e.g. socket, threading, time, os, sys modules for Python should be enough)
- Works with all web browser and systems (HTTP only)

GENERAL OVERLAY



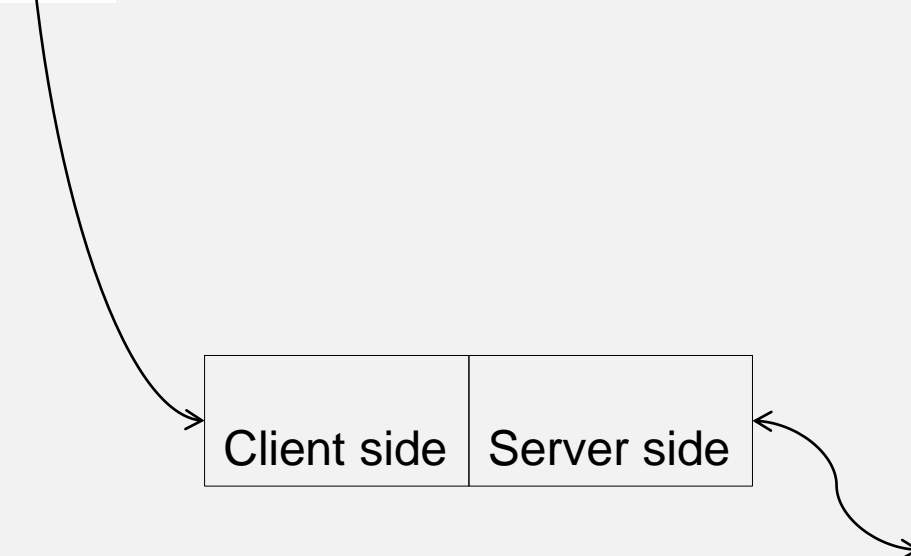
Server



Proxy



Client

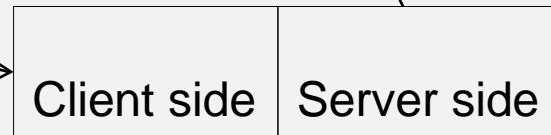


GENERAL OVERLAY



Server

Server side: listens on a port, accepts, receives, forwards to client side



Proxy



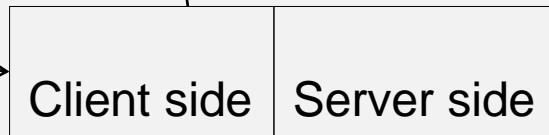
Client

GENERAL OVERLAY



Server

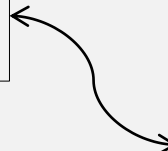
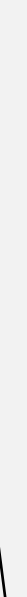
Client side: connects to the server, send request, receive response,
Forwards to server side



Proxy



Client



CONTENT ALTERING

Need to be able to filter based on content.

In which of the two halves of the proxy will you implement content altering ?

How to actually do content altering ?

CONTENT ALTERING



Response from the server comes in segments



Remember TCP segmentation?



Reconstruct the message in a temporary buffer



No dynamic sizing of buffer, chose a value (using debug info) and stick with it



Do not type cast non-text data



Then run filtering only on the text message

TEXT VS OTHER BINARY DATA

What is the requirement for filtering with regard to binary data?

- Only that you have to be smart in handling any data type

What will happen if you attempt to reconstruct an image or video and try to filter it?

Solutions?

```
⊕ Transmission Control Protocol, Src Port: http (80), Dst Port: 50139 (50139), Seq: 4381, Ack: 277, Len: 1215
⊕ [4 Reassembled TCP Segments (5595 bytes): #248(1460), #249(1460), #251(1460), #252(1215)]
⊖ Hypertext Transfer Protocol
  ⊕ HTTP/1.1 200 OK\r\n
    Date: Sun, 07 Sep 2014 10:06:36 GMT\r\n
    Server: Apache/2.2.17 (Unix) DAV/2\r\n
  ⊕ Content-Length: 5354\r\n
    Last-Modified: Tue, 04 Feb 2014 12:25:40 GMT\r\n
    Keep-Alive: timeout=15, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/xml\r\n
    \r\n
```

TEXT VS BINARY DATA

- Content-type header
- Differentiate content type
 - Run/don't run filtering
 - Send data or block the client

HOW TO BLOCK SPECIFIC CONTENT

- You are supposed to return a specific response based on content filtering

HTTP REDIRECT

- If filtering confirms presence of inappropriate words
 - *HTTP/1.1 301 Moved Permanently*
- Else send response

DEBUGGING ADVICE

- Stick to simple web pages initially
- Debug incrementally
- Check and double check request string for formatting and completeness
 - Source of many errors like 'server closed connection unexpectedly'
- If developing on own computers, use wireshark to debug. Can save a lot of time!

DEBUGGING ADVICE

- HTTP vs HTTPS
 - Requirements do not ask for a proxy which works with HTTPS
 - Avoid testing on any site to which you are signed in
 - Restrict yourselves to simple sites and basic test cases

DEBUGGING ADVICE

- Header manipulation
 - First thing to check at a proxy is the URL that it sends out to the server
 - It might require different manipulations based on the site. Be sure that you test for all sites mentioned in the test scenario
 - If you change some fields in the header, the packet length has to be changed or brought back to the original length



THE END

Question?