

Overview

- Web enabled applications are everywhere and are playing an increasingly important role in the marketing strategies of many companies
- For some companies, a web interface to their applications is even central to their existence e.g Amazon, Rabobank. . .
- This trend is set to continue and leads us on to our next topic in secure coding: web security
- For us, web security involves simplifying the increasingly complex web applications that lie behind web interfaces in order to understand the threats they face and to ensure we write code resistant to those threats



Overview

- At the highest level, web security requires:
 1. Securing the web server and the data on it
 2. Securing the data between the server and browser
 3. Securing the end user's computer
- Our concern here is 1 but before getting into it, in this lecture we start off with a brief look at the how the web works and the structure of the simplest of web applications
- Material is largely drawn from the following sources:
 - *Web Security, Privacy and Commerce* by Garfinkel
 - *Innocent Code* by Huseby



Internet Architecture

- The Internet dates back to research conducted into packet switching by the U.S. Department of Defence's Advanced Research Projects Agency (DARPA) in the 1960's
- Before packet switching, a dedicated communication line between two machines was required for reliable communication
- Thanks to packet switching, even in the absence of a direct communication line, message delivery is ensured as:
 1. Each packet contains a source and destination address
 2. Intervening networks will forward packets to their intended destination based on dynamically updated routing tables



Internet Architecture

- The Internet can therefore be viewed as a set of publicly available, interconnected computer networks that transmit messages by packet switching
- Machines are addressed using the Internet Protocol (IP), messages are packaged and transmitted according to the Transmission Control Protocol (TCP) and web applications communicate using the HyperText Transfer Protocol (HTTP)
- Machines in DCU's School of Computing communicate with the Internet through a proxy server
- Between the proxy server and the Internet is a gateway running firewall software



World Wide Web

- “Invented” in 1990 by Tim Berners-Lee at CERN
- The Internet was in place but there existed no common means of accessing the different types of information it made available: Tim gave us the common means
- Three key enablers:
 1. Uniform resource locators (URL) were specified and provided a single addressing scheme for all information
`protocol://machine/object`
 2. HyperText Markup Language (HTML) was specified and provided a standard way of structuring textual information
 3. HTTP provided a standard means of communicating requests and responses



Domain Name Service

- Internet domain names must be converted into IP addresses before communication can be established
- DNS servers perform the translation
- Originally all names and corresponding IP addresses were stored in a single file
- Nowadays, instead of a single centralised listing a distributed database is used
- Organisations are themselves responsible for their part of the global namespace



Domain Name Service

- When a host name must be resolved to an IP address a DNS request packet is sent by your browser to a local name server
- If the name server knows the answer it responds otherwise it forwards the request to another name server that will find out
- Should it need to go there your local name server knows the addresses of the thirteen root name servers:
`A-M.ROOT-SERVERS.NET`



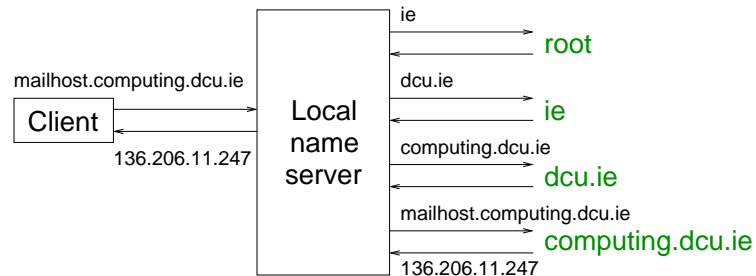
Domain Name Service

- For example, you visit `www.computing.dcu.ie`:
 1. If the local name server does not have the corresponding IP address cached it asks a root name server for the address of the `.ie` domain name server
 2. It then asks the `.ie` name server for the address of the `.dcu` name server
 3. It finally asks the `.dcu` name server for the address of the `computing web server`
- We can use utilities like `dig` and `nslookup` to query name servers interactively e.g. we can ask a domain server for Virgin's IP address like this:

```
$ nslookup www.virgin.co.uk
```



Domain Name Service



Using The Web

- Once the IP address of a remote machine is known a TCP/IP connection can be established
- The connecting machine sends a HTTP request containing the host name and location of the object requested
- The web server sends a reply containing the page contents and closes the connection
- If the document contains hypertext that references embedded contents the browser will need to send multiple requests to download and display these items
- HTTP is thus a client/server protocol, the client being usually a browser but it need not be: it can be any piece of software capable of speaking HTTP



Internet Service Providers

- At the other end of your Internet connection sits equipment that routes your packets to other machines on the network: ISPs provide that equipment
- Mostly owned by or affiliated with telephone and cable TV companies
- An ISP might operate the following: domain name servers, web servers, communications lines, data centres where the computers reside, network operations centres for status monitoring etc.
- Packets must travel between ISPs and so they often group together to put multiple routers belonging to each of the organisations involved at a common location, so-called network access points



Internet Service Providers

- ISPs pay to use each other's lines for packet delivery through peering and transit arrangements
- The route from DCU to UL can be traced as follows:

```
$ traceroute www.ul.ie
traceroute to www.ul.ie
 1  router-11.computing.dcu.ie
 2  136.206.13.254
 3  cisdcu.dcu.ie
 4  kuiper-gige2-1.dcu.access.hea.net
 5  hyperion-gige4-2.cwt.core.hea.net
 6  rhea-gige3-1.lim.core.hea.net
 7  tempus-gige0-3.lim.access.hea.net
 8  ul.lim.client.hea.net
 9  ulfort.ext.ul.ie
10  new-web-bup.ul.ie
```



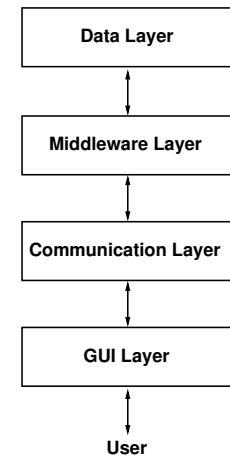
Web Application Framework

- With our basic understanding of how the web works in place we can examine the design behind the simplest of typical web applications
- A web application comprises four technology layers:
 1. GUI
 2. Communication
 3. Middleware
 4. Data
- We look briefly at the functioning of each layer and some technologies found in each
- Confronted with a new technology, knowing where it fits into the web application framework will at least give us a working handle on it



Web Application Framework

- Given the multitude of web development technologies available, we cannot study them all
- We restrict ourselves for now to understanding where a technology fits into the overall web application framework and its role within that layer



The GUI Layer

- Includes technologies for displaying and requesting information through a browser
- For us, it's simply HTML forms with GET or POST submit methods
- Optional plugins can add functionality without cluttering a browser's design (e.g. Shockwave, Flash, XUL etc.)
- A JVM (may be considered a plugin) allows a browser to run applets
- Client-side scripting with JavaScript allows simple control over browser behaviour through an API



The Communication Layer

- Documents requested with GET or POST HTTP methods
- GET supports the sending of optional search parameters with attribute:value pairs e.g.
`http://www.d.com/cgi-bin/lookup?name=Sonia`
- The user sees these pairs in the URL and they're stored in browser history, local cache, proxy cache, proxy logs and may leak through `Referer` HTTP header field
- GET is restricted to sending a maximum 1024 bytes
- GET submissions may be silently reissued by clicking "back" in a browser: unfortunate if paying a bill



The Communication Layer

- POST allows for the sending of unlimited information in the body of the HTML request itself
- Again, submitted by way of attribute:value pairs
- POST requests cannot be reissued without the browser asking first for permission to do so
- Rule: use POST requests when operations have side effects
- The method of submission is specified in the HTML form definition



The Middleware Layer

- The middleware layer accepts incoming requests, processes them using the resources provided by the web server and returns formatted results
- CGI (Common Gateway Interface) provides the most basic gateway to web server resources where Perl/PHP are often the programming languages of choice because of their strong text formatting capabilities
- Other technologies include SSI, ASP and COM/Active-X, Java and Visual Basic



The Data Layer

- The middleware layer typically operates on data, retrieving it per the request and formatting it appropriately before returning it to your browser
- That data may be stored on the server as raw text, simple HTML or in XML format
- Often data will be stored in a database e.g. MySQL, Oracle, MS Access, Postgres
- Talking to various databases is made easier by using a database independent layer such as Perl's DBI module



HTTP Requests

- HTTP is a line oriented protocol with the browser sending requests like this:

```
GET / HTTP/1.0
Host: www.computing.dcu.ie
Accept: text/html, text/plain, image/*
Accept-Language: en
User-Agent: Mozilla/5.0
```
- The first line is a request for a document, the root document, and the browser states it expects the server to speak HTTP 1.0
- Following this request line is a series of request headers that add more control to the conversation



HTTP Requests

- The `Accept` header tells the server what media formats the client supports
- The server may send different responses based on the `User-Agent` so that's sent along too
- The above is an example of a GET request, what does a POST look like?



HTTP Requests

- Here's one...

```
POST /lookup.cgi HTTP/1.0
Host: www.computing.dcu.ie
User-Agent: Mozilla/5.0
Referer: http://www.dcu.ie/search.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 17
```

```
name=Sonia&age=34
```

- URL encoding means troublesome characters like `&` can be submitted to the server and will not be interpreted as parameter separators by the HTTP protocol



HTTP Responses

- The server responds with something like this:

```
HTTP/1.1 200 OK
Date: Mon, 12 November 2007 15:00 GMT
Server: Apache/1.4.21 (Unix) PHP/4.2.3
Last-Modified: Sun, 11 November 2007, 21:03 GMT
Content-Length: 565
Connection: close
Content-Type: text/html
```

```
<html>
...
</html>
```



HTTP Responses

- The first line, the status line, tells the client what version of HTTP this server is capable of (not the version it is currently speaking)
- Next comes a 3 digit status code and human readable reason phrase (e.g. 404 Not Found)
- Next the server identifies itself along with version number and platform
- The `Content-Type` header says the response is in HTML



HTTP Headers

Can a web server trust them?

- Definitely not!
- Although your browser does not allow you to play with HTTP headers, it is simple to write a program that will allow you to do so and to submit handcrafted, bogus HTTP header values
- If you couldn't be bothered writing such a program then download a web proxy such as WebScarab
- WebScarab sits between your browser and the Internet and will allow you to manipulate all incoming and outgoing traffic (among many other things)