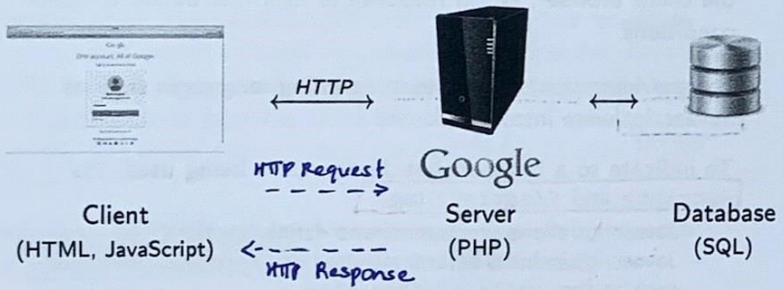
Web applications



URLs

A web browser identifies a website with a uniform resource locator (URL).

arguments

`Protocol://host:port/path?arg1=val1&arg2=val2#statement`

This naming scheme allows referring to content on distant computers in a simple and consistent manner:

- `Protocol:` protocol to access the resource (`http`, `https`, `ftp`, ...)
- `host:` domain or IP address of the server storing the resource
- `FilePath:` path to the resource on the host
- Resources can be static (`file.html`) or dynamic (`do.php`)
- URLs for dynamic content usually include arguments to pass to the process (`arg1`, `arg2`)

HTTP requests

GET request

Path of the resource that is being requested
Version of protocol

```
GET / HTTP/1.1 → REQUEST LINE
Host: www.inf.ed.ac.uk
User-Agent: Mozilla/5.0
            (X11; Ubuntu; Linux x86_64; rv:29.0)
            Gecko/20100101 Firefox/29.0
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
Referer: https://www.google.com
Connection: keep-alive
```

- After establishing a TCP connection to the web server, the browser sends HTTP requests to that server
- HTTP requests begin with a request line (`GET` or `POST` command)
- An HTTP request consist of the headers section, and the message body

Server processes request and prepares a response

HTTP responses

status code

```
HTTP/1.1 200 OK
Server: Apache
Cache-control: private
Set-Cookie: JSESSIONID=B7E2479EC28064DF84DF4E3DBEE9C7DF;
            Path=/
Content-Type: text/html;charset=UTF-8
Date: Wed, 18 Mar 2015 22:36:30 GMT
Connection: keep-alive
Set-Cookie: NSC_xxx.fe.bd.vl-xd=ffffffffc3a035...423660; path=/
Content-Encoding: gzip
Content-Length: 4162

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
    Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
    xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xml:lang="en" lang="en">
<head>
<title> Informatics home | School of Informatics </title>
...
```

Main body of the web page encoded in HTML

Hypertext Markup Language (HTML)

- The main body of a web page is encoded using HTML.
- HTML provides a structural description of a document using special tags.
- HTML includes a mechanism called forms to allow users to provide input to a website in the form of variables represented by name-value pairs.
- Forms can submit data either using the `GET` (name-value pairs encoded in the URL) or the `POST` method (name-value pairs encoded in the message body).

i.e. A login form

```
<form method="POST" action="login.php">
   FirstUsername: <input type="text" name="username">
   Password: <input type="password" name="password">
   <input type="submit" value="submit">
</form>
```

Upon clicking the submit button, the browser will issue a POST request w/ the 2 arguments to the login script that lives somewhere in the server

Javascript - example

```
<html>
  <body>
    <p id="p1">Hello World!</p>

    <script>
      document.getElementById("p1").innerHTML =
                                          "New text!";
    </script>
  </body>
</html>
```

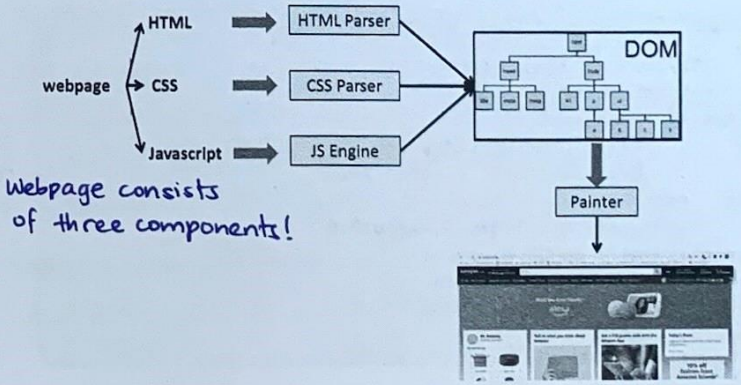
JAVASCRIPT

Go and change the content of paragraph p1 into "New text!"

Dynamic content

- Pages with dynamic content can change after their delivery to the client browser, eg. in response to user interaction or other conditions.
- For providing dynamic content, scripting languages such as Javascript were introduced.
- To indicate to a browser that Javascript is being used, the `<script>` and `</script>` tags:
  - Javascript allows programmers to define functions
  - Javascript includes several standard programming constructs such as `for`, `while`, `if/then/else`, ...
  - Javascript also handles events, eg. user clicks on a link, user hover mouse pointer over a portion of the page

Webpage rendering



Webpage consists of three components!

- The Document Object Model (DOM) is a means for representing and accessing the content of a page. (Tree data struct.)
- Scripts can alter/manipulate the content of a page by accessing/updating the DOM of the page.

→ JS Engine is constantly running to listen for user's actions! (Source of many web security issues!)

How is state managed in HTTP sessions

HTTP is stateless: when a client sends a request, the server sends back a response but the server does not hold any information on previous requests

The problem: in most web applications a client has to access various pages before completing a specific task and the client state should be kept along all those pages. How does the server know if two requests come from the same browser?
Example: the server doesn't require a user to log at each HTTP request

The idea: insert some token into the page when it is requested and get that token passed back with the next request

Two main approaches to maintain a session between a web client and a web server

1) use hidden fields
2) use cookies

1. Hidden fields (1)

The principle

Include an HTML form with a hidden field containing a session ID in all the HTML pages sent to the client. This hidden field will be returned back to the server in the request.

Example: the web server can send a hidden HTML form field along with a unique session ID as follows:

e.g. (not visible to user)

```
<input type="hidden" name="sessionid" value="12345">
```

When the form is submitted, the specified name and value are automatically included in the GET or POST data.

Hidden fields (2)

→ An attacker gaining access to the GET/POST variables being submitted by a user could hijack their session and assume their identity. → Use HTTPS!

(−) **Disadvantage of this approach**

- it requires careful and tedious programming effort, as all the pages have to be dynamically generated to include this hidden field
- session ends as soon as the browser is closed

(+) **Advantage of this approach**

All browser supports HTML forms

2. Cookies (1)

- A cookie is a small piece of information that a server sends to a browser and stored inside the browser. A cookie has a name and a value, and other attribute such as domain and path, expiration date, version number, and comments
- The browser automatically includes the cookie in all its subsequent requests to the originating host of the cookie
- Cookies are only sent back by the browser to their originating host and not any other hosts. Domain and path specify which server (and path) to return the cookie
- A server can set the cookie's value to uniquely identify a client. Hence, cookies are commonly used for session and user management

APPLICATIONS

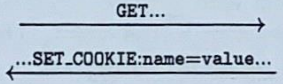
- Cookies can be used to hold personalised information, or to help in on-line sales/service (e.g. shopping cart)...

(−) **Main limitation**

Users may disable cookies in their browser

Cookies (2)

Cookies are set on the client's system when the server uses the `Set-Cookie` field in the HTTP header of its response:



Google

A cookie has several attributes:

```
Set-Cookie:  name=value[; expires=date]
        [; domain=dom][; path=p][; Secure][; HttpOnly]
  expires : (whentobedeleted)
  domain : (whentosend)   } scope
  path : (whentosend)     }
  Secure : (onlyoverSSL)
  HttpOnly : (onlyoverHTTP)
```

→ Default is when client closes browser

It is recommended tht. users erase cookies on a regular basis!

Security goals

Web applications should provide the same security guarantees as those required for standalone applications

1. visiting `evil.com` should not infect my computer with malware, or read and write files
   Defenses: Javascript sandboxed, avoid bugs in browser code, privilege separation, *etc*

2. visiting `evil.com` should not compromise my sessions with `gmail.com`
   Defenses: same-origin policy – each website is isolated from all other websites

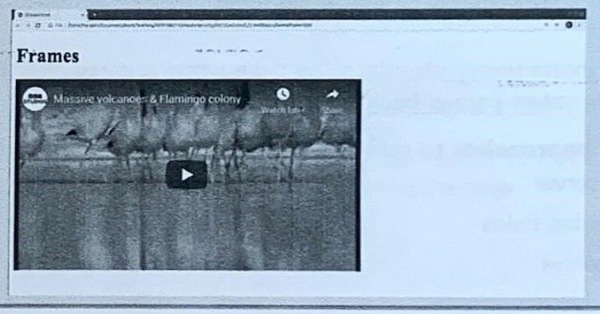
3. sensitive data stored on `gmail.com` should be protected

To hide the exec. of code, an attacker can use an invisible iframe

Frames

- Embed a "inner" webpage within an "outer" webpage
- `<iframe src="URL"></iframe>`
- The outer webpage specifies the size and position of the inner webpage within the outer webpage

```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/owsfdh4gxyc"
frameborder="0" allowfullscreen></iframe>
```



Threat model

2 TYPES OF ATTACKER

1. **Web attacker**
   - controls `evil.com`
   - has valid SSL/TLS certificates for `evil.com`
   - victim user visits `evil.com`

2. **Network attacker**
   - controls the whole network: can intercept, craft, send messages

A Web attacker is weaker than a Network attacker