Cookies
Objectives

• Learn about creating and accessing cookies, and how this is used to store state information
• Learn about security issues
State Information

• Information about individual visits to a web site

• HTTP was originally designed to be stateless
  – Browsers stored no persistent data about site visits
Why keep state information?

• Customize individual Web pages
• Temporarily store information for a user
• Allow a user to create bookmarks
• Provide shopping carts
• Store user IDs and passwords
• Use counters
State Information with Cookies

• Cookies
  – Small pieces of information about a user that are stored by a server in text files, on the user’s computer

• Each time the client visits a web server, saved cookies are sent from the client to the server

• Temporary cookies
  – Remain available only for current browser session
Persistent Cookies

• Remain available beyond current browser session

• Limitations on the use of cookies
  – Server or domain can store a maximum of 20 cookies
  – Total cookies per browser cannot exceed 300
  – Largest cookie size is 4 KB
Creating Cookies

• Use the `cookie` property of the `Document` object to create cookies in `name=value` pairs

• Optional attributes: `expires`, `path`, `domain`, `secure`

```javascript
document.cookie = "user=eg; expires= " etc.
```
Creating Cookies

• Every time `document.cookie` is assigned a name-value pair, then:
  – if the name already exists (within the same domain), the value is replaced
  – if the name does not exist, a new cookie is created

• This property behaves differently from normal JavaScript properties
name attribute

• Required
• Specifies cookie’s name=value pair
• If cookie only has a name, then temporary cookie
• Cannot include ; or special characters
• Use encoding for special characters
  – replaces certain characters with their hexadecimal value. For example, %40 is the @ character.
Encoding/Decoding

• Convert special characters (e.g. space, punctuation) to hex value
  `encodeURIComponent()`

• Encodes special characters in the individual parts of a URI to their corresponding value

• Likewise
  `decodeURIComponent()`
Creating Cookies...

- You should manually encode and decode cookies

```javascript
document.cookie = "name=" + encodeURIComponent(document.forms[0].name.value);
document.cookie = "address=" + encodeURIComponent(document.forms[0].address.value);
```
expires attribute

• Determines how long a cookie can remain on a client system before it is deleted
• Without this attribute cookies are only available for current browser session
• Be sure not to encode this attribute
expires attribute...

• You can manually type a string in UTC format
• You can create the string with the `Date` object and use `toUTCString()` method to convert it to a string
Example

```javascript
submitForm() {

  var expiresDate = new Date();
  expiresDate.setFullYear(expiresDate.getFullYear() + 1); //add 1 to year
  document.cookie = "registered=" + encodeURIComponent(true) + "; expires=" + expiresDate.toUTCString();
}
```
**path attribute**

- By default, cookie is available to all web pages on the server
- Defining a path restricts to that directory, and all its sub-directories
- To make a cookie available to all directories on a server, use a slash
  
  ```
  ...;
  path=/
  ```
path attribute...

- Cookies from other programs on the web page that are stored in the same directory can cause your JavaScript cookie program to run erratically
- Keep each set of cookies in their own directory
domain and secure attributes

• domain used to share cookies across servers in the same domain
• cookies cannot be shared outside a domain
• secure indicates that a cookie can only be transmitted across a secure internet connection using HTTPS or similar security protocol
Example

document.cookie = "name=eg; expires=" + expiresDate.toUTCString() + "; path=/; domain=.soton.ac.uk; secure=true";
Reading Cookies

- When reading `document.cookie` returns a single `String` object containing all name-value pairs separated using ";;" followed by a space
- Use the methods of the `String` object to extract individual `name=value` pairs
Parsing Strings

• **Use `split()`** to convert the individual pieces of information into array elements
• **Use `trim()`** to remove white space
• **Use `split()` again** to split the name-value pairs
• **Decode the value of each element using `decodeURIComponent()` function**
Example:

```javascript
var cookieData = document.cookie;
//reads and prints contents of the cookies
var cookieArray = cookieData.split(";");
for (var i=0; i<cookieArray.length; ++i){
   var nameValuePair=cookieArray[i].trim().split("=");
   document.write("<p>name:"+nameValuePair[0]+" value:"+decodeURIComponent(nameValuePair[1])+"</p>");
}
```
Security in the News

  
  "Microsoft issued a security advisory on Saturday warning users of a vulnerability in its **Internet Explorer** web browser that could allow malicious 'remote code execution.'"

• **Heartbleed bug**

  "Heartbleed is a **security bug** in the **open-source OpenSSL cryptography** library, which is widely used to implement the Internet's **Transport Layer Security** (TLS) protocol." (Wikipedia)
Secure Coding with JavaScript

• Security threats
  – Viruses, worms, and data theft by hackers

• Consider both Web server security issues and secure coding issues

• Web server security technologies
  – Firewalls
  – Secure Socket Layer (SSL)

• JavaScript programs are downloaded and executed locally
Secure Coding with JavaScript (continued)

• **Secure coding** or **defensive coding**
  – Writing code to minimize any intentional or accidental security issues

• All code is insecure unless proven otherwise

• No magic formula for writing secure code
JavaScript Security Concerns

• Security areas of most concern
  – Protection of a Web page and JavaScript program against malicious tampering
  – Privacy of individual client information
  – Protection of the local file system of the client or Web site from theft or tampering

• Another security concern
  – Privacy of individual client information in the Web browser window

• An important JavaScript security feature
  – Its lack of certain types of functionality
JavaScript Security Concerns (continued)

• Missing functionalities
  – File manipulation
  – Create a network connection
  – Cannot run system commands or execute programs on a client
The Same Origin Policy

• Same origin policy
  – Restricts how JavaScript code in one window or frame accesses a Web page
    • In another window or frame on a client computer
• To view and modify the elements in other windows and frames
  – They must have the same protocol and exist on the same Web server
• Same origin policy applies not only to the domain name
  – But also to server on which a document is located
The Same Origin Policy (continued)

• Policy prevents malicious scripts from modifying the content of other windows and frames
  – And prevents the theft of private browser information and information displayed on secure Web pages
• Policy also protects the integrity of the design of your Web page
• Example
  – Create a frame set in which one frame uses JavaScript code to try to change the status bar text of another frame
The Same Origin Policy (continued)

• **domain property** of the Document object
  – Changes the origin of a document to its root domain name
  – Allows documents from different origins in the same domain to access each other’s elements and properties

• Changes to domain are limited. For example:
  – changing from `ecs.soton.ac.uk` to `soton.ac.uk` is allowed
  – but changing to `ecs.soton.ac` or `ecs.soton.com` is not
Summary

• Information about individual visits to a Web site is called state information
• HTTP was originally designed to be stateless
• Most common tool for maintaining state information is cookies
• Cookies can be temporary or persistent
• Cookies are one continuous string that must be parsed
Summary...

• You can use special characters in your cookies if you use encoding

• The `encodeURIComponent()` built-in function encodes the individual parts of a URI

• When you read a cookie, you must first decode it with the `decodeURIComponent()` function
Summary...

• “Secure coding,” or “defensive coding,” refers to writing of code to minimize any intentional or accidental security issues
Summary...

- For JavaScript to access a web page in another window or frame, the pages must have the same server and protocol.
- The `domain` property of the `document` object changes the origin of a document to its root domain name:
  ```javascript
document.domain = "ecs.soton.ac.uk";
```
Query Strings

• Set of *name=value* pairs appended to a target URL

• Consists of a single text string containing one or more pieces of information

• You can use a query string to pass information from one web page to another
Examples

<a href = "http://www.ecs.soton.ac.uk/myPage.html?firstName=Quintin&lastName=Gee&group=lsl">home page</a>

<a href = "http://www.google.co.uk/search?hl=en&q=martyn+talbot&btnG=Google+Search &meta=cr%3DcountryUK|countryGB" ></a>

Note: %3D represents = in hexadecimal and %20 represents space
Passing Data

• Follow the URL with ? and the query string for the information you want to preserve
• Separate name=value pairs using &
• The passed query string is then assigned to the search property of the target web page location object
Parsing Query Strings

- Use `substring()` and `length` to remove the question mark
- Use `split()` to convert the individual pieces of information into array elements
- Example: Write your own parsing script that extracts and displays the data in the query string
var formData = location.search;

formData = formData.substring(1, formData.length); //remove ?

var formArray = formData.split("&");
for (var i=0; i < formArray.length; ++i) {
  document.write(formArray[i] + "<br /> ");
}

Parsing Data...

**Figure 9-5**
Data submitted to Register.html

Your product registration has been received.

You entered the following data:

- name=Don Gosselin
- address=123 Main Street
- city=Anywhere
- state=MA
- zip=01500
- email=don@gosselin.com
- telephone=(707) 555-1212
- location=home_office
- comments=Great product!
- registered=true
- platform=linux
- quality=true
- price=true
- serial=12345-6
- date=January%2027,%202008
- where=internet

Done
Encoding/Decoding

• URLs only allows certain type of ASCII characters
• Therefore, whenever a query must be *encoded*, which replaces certain characters with their hexadecimal value. For example, %40 is the @ character.
• This is done automatically when submitting a form using the *get* method. In addition, since spaces are not allowed, typically the + sign is used (instead %20)
• Therefore, to parse a query, the values must first be *decoded* – you will learn how to do this in the lab