COURSE TECHNOLOGY
CENGAGE Learning

JavaScript, Sixth Edition

Chapter 9
Managing State and Information Security

Objectives

When you complete this chapter, you will be able to:

- Save state information with query strings, hidden form fields, and cookies
- Describe JavaScript security issues and employ coding practices designed to address them

Understanding State Information

- State information
  - Information about individual visits to a Web site
- HTTP original design: stateless
  - No persistent data about site visits stored
- Reasons for maintaining state information
  - Customize individual web pages
  - Store information within a multipart form
  - Provide shopping carts

Saving State Information with Query Strings

- Query string
  - Set of name-value pairs
    - Appended to a target URL
  - Consists of a single text string
    - Contains one or more pieces of information
- Passes information from one web page to another

Saving State Information with Query Strings (cont'd.)

- Passing data with a query string
  - Add a question mark (?) immediately after a URL
    - Followed by the query string (in name-value pairs) for the information to preserve
  - Ampersands (&)
    - Separates individual name-value pairs within the query string
  - Example:

```
<a href="http://www.example.com/↵
    addItem.html?isbn=9780394800165&quantity=2">Order Book</a>
```


Saving State Information with Query Strings (cont'd.)

- Passed query string assigned to target web page `Location` object `search` property
  - `search` property of the `Location` object contains a URL's query or search parameters
- From preceding example:
  - Query string
    - "?isbn=9780394800165&quantity=2"
  - Available as the value of the `search` property of the `Location` object
    - After addItem.html document opens

Saving State Information with Query Strings (cont'd.)

- Remove the question mark
  - Using the `substring()` method combined with the `length` property
  - Example:

```
// Assign the query string to the queryData variable
var queryData = location.search;
// Remove the opening question mark from the string
queryData = queryData.substring(1, queryData.length);
```


Saving State Information with Query Strings (cont'd.)

- Convert individual pieces of information into array elements
  - Using the `split()` method
  - Example:
    - Convert the information in the `queryData` variable into an array named `queryArray[]`

```
// split queryData into an array
var queryArray = queryData.split("&");
```


Saving State Information with Hidden Form Fields

- Hidden form field
  - Special type of form element
  - Not displayed by web browser
  - Allows information to be hidden from users
  - Created with the `input` element
  - Temporarily stores data sent to a server along with the rest of a form
  - No need for user to see hidden data
- Syntax

```
<input type="hidden">
```


Saving State Information with Hidden Form Fields (cont'd.)

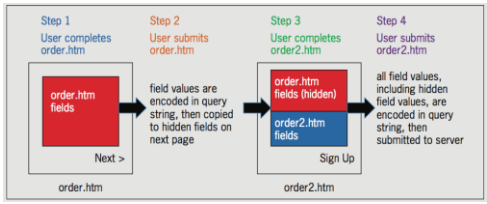

Figure 9-7 Submitting data from multiple forms using hidden fields

Saving State Information with Cookies

- Query strings and hidden form fields temporarily maintain state information
- Cookies
  - Small pieces of information about a user
  - Stored by a web server in text files
  - Stored on the user's computer
- When Web client visits a web server
  - Saved cookies sent from client to the server
- Temporary cookies
  - Available only for current browser session

Storing State Information

- Query strings and hidden form fields temporarily maintain state information
- Two methods of storing state information on user's computer
  - cookies
  - Web Storage

Storing State Information with Cookies

- Cookies
  - Small pieces of information
  - Stored in text files
- Temporary cookies
  - Remain available only for current browser session
- Persistent cookies
  - Remain available beyond current browser session
    - Stored in a text file on a client computer

Storing State Information with Cookies (cont'd.)

- 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 kilobytes
- Not enforced by all browsers

Creating and Modifying Cookies

- Use the `cookie` property of the `Document` object
  - Creates cookies in name-value pairs
  - Syntax
    document.cookie = *name* + "=" + *value*;
  - Cookie property created with a required `name` attribute and four optional attributes:
    - `expires`, `path`, `domain`, `secure`

Creating and Modifying Cookies (cont'd.)

ATTRIBUTE	DESCRIPTION
`domain=domain`	For a web server with multiple subdomains, specifies whether the cookie is available only to the subdomain from which it originated (default)
`expires=date`	When a cookie will be deleted from the system, where `date` is a date in UTC format; setting `expires` to a date value in the past and deletes a cookie immediately
`name=name`	The name of the cookie (required)
`path=path`	A path on the web server in which documents must be located to have access to the cookie; by default, a cookie is available to all web pages in the same directory
`secure=boolean`	Whether the cookie can be transmitted only over a secure protocol, such as HTTPS (`true`), or may be transmitted over a nonsecure protocol (`false`, the default)

Table 9-1 Cookie attributes

Creating and Modifying Cookies (cont'd.)

- `name` attribute
  - Only required parameter
  - Specifies cookie's name-value pair
  - Cookies created with only the `name` attribute:
    - Temporary cookies
  - `cookie` property adds another entry to a list of cookies

Creating and Modifying Cookies (cont'd.)

- Example: build a list of cookies

```
document.cookie = "username=mw101";
document.cookie = "member=true";
document.cookie = "audio=false";
```

- Resulting value of `cookie` property:

username=mw101; member=true; audio=false

Creating and Modifying Cookies (cont'd.)

- Cookies cannot include semicolons or special characters
  - Can use special characters in cookies if using encoding
- Encoding involves converting special characters in a text string
  - To corresponding hexadecimal ASCII value preceded by a percent sign

Creating and Modifying Cookies (cont'd.)

- `encodeURIComponent()` function
  - Used for encoding the individual parts of a URI
  - Converts special characters in the individual parts of a URI to corresponding hexadecimal ASCII value
  - Syntax: `encodeURIComponent(text)`
- `decodeURIComponent()` function
  - Counterpart of `encodeURIComponent()` function
  - Syntax: `decodeURIComponent(text)`

Creating and Modifying Cookies (cont'd.)

- Testing Applications That Use Cookies
  - Browsers open local documents without HTTP
  - Some browsers don't support setting cookies when HTTP protocol not used
  - Can focus testing on browsers that support cookies on files opened without HTTP
  - More robust option
    - run local web server
    - open files using HTTP protocol
    - some code editors include built-in testing server

Creating and Modifying Cookies (cont'd.)

- `expires` attribute
  - Determines how long a cookie can remain on a client system before being deleted
  - Cookies created without this attribute
    - Available current browser session only
  - Syntax: `expires=date`
    - name-value pair and the `expires=date` pair separated by a semicolon
  - Do not encode `expires` attribute

Creating and Modifying Cookies (cont'd.)

- Can manually type a string in UTC format or:
  - Can create string with the `Date` object
- Use the `toUTCString()` method to convert the `Date` object to a string
- Example:

```
var expiresDate = new Date();
var username = document.getElementById("username").value;
expiresDate.setFullYear(expiresDate.getFullYear() + 1);
document.cookie = "username=" +↵
  encodeURIComponent(username) + "; expires=" +↵
  expiresDate.toUTCString();
```


Creating and Modifying Cookies (cont'd.)

- Configuring availability of cookies to other web pages on the server
  - use `path` attribute
    - Default: cookie available to all web pages in the same directory
    - To make cookie available to all directories on a server
      - Use a slash

Creating and Modifying Cookies (cont'd.)

– Example:

```
var username = document.getElementById("username").value;
document.cookie = "username=" +↵
  encodeURIComponent(username + "; path=/advertising");
```

– Example:

```
document.cookie = "username=" +↵
  encodeURIComponent(username + "; path=/");
```

– Cookies from other programs stored in the same directory
  - Can cause JavaScript program to run erratically

Creating and Modifying Cookies (cont'd.)

- Sharing cookies across a domain
  – use `domain` attribute
    - Example:

```
var username = document.getElementById("username").value;
  document.cookie = "username=" +↵
  encodeURIComponent(username +↵
  "; domain=.example.com");
```

    - Cannot share cookies outside of a domain

Creating and Modifying Cookies (cont'd.)

- Securing cookie transmissions
  – `secure` attribute
    - Indicates cookie can only be transmitted across a secure Internet connection
      – Using HTTPS or another security protocol
    - Example:

```
var username = document.getElementById("username").value;
  document.cookie = "username=" +↵
  encodeURIComponent(username + "; secure=true");
```


Reading Cookies with JavaScript

- Parsing a cookie
  – Decode it using `decodeURIComponent()` function
  – Use the `String` object methods to extract individual name-value pairs
  – Similar to parsing query strings
    - Do not need to remove the question mark at the beginning of the string
    - Individual cookies separated by a semicolon and a space instead of ampersands

Reading Cookies with JavaScript (cont'd.)

– Example:
  - Create three encoded cookies
  - Read them from the `cookie` property, decode them
  - Use the `split()` method to copy each name-value pair into `cookieArray[]` array elements
  - Determine which cookie holds needed value
    – `for` loop to cycle through array elements
    – `if` statement to check name portion of each name-value pair

Reading Cookies with JavaScript (cont'd.)

```
document.cookie = "username=" +↵
  encodeURIComponent(username);
document.cookie = "member=" +↵
  encodeURIComponent(member);
document.cookie = "audio=" +↵
  encodeURIComponent(audio);
var cookieString = decodeURIComponent(document.cookie);
var cookieArray = cookieString.split("; ");
```

Reading Cookies with JavaScript (cont'd.)

```
var currentUsername;
var unBox = document.getElementById("username");
for (var i = 0; i < 3; i++) {
  currentUsername = cookieArray[i];
  if
  (currentUsername.substring(0,currentUsername.indexOf("="))
  === "username") {
    unBox.value =
      currentUsername.substring(currentUsername.
      indexOf("=") + 1,currentUsername.length);
    break;
  }
}
```


Deleting Cookies with JavaScript

- Not intuitive
  - Must set cookie expiration to a date in the past
- Example:
  - Delete `username` cookie by setting its `expires` attribute to one week ago

```
var expiresDate = new Date();
var username = document.getElementById("username").value;
expiresDate.setDate(expiresDate.getDate() - 7);
document.cookie = "username=" +
  encodeURIComponent(username) + "; expires=" +
  expiresDate.toUTCString();
```


Deleting Cookies from Your Browser

- Persistent cookies can interfere with testing
- Good idea to delete cookies periodically

BROWSER	KEYBOARD SHORTCUT	MENU STEPS
Internet Explorer 9+	Press **Ctrl** + **Shift** + **Del**, then in the window that opens, make sure the **Cookies and website data** box is checked, and then click **Delete**.	Click the **Tools** icon, if necessary point to **Safety** on the menu, click **Delete browsing history** in the submenu, then in the window that opens, make sure the **Cookies and website data** box is checked, and then click **Delete**.
Firefox	Press **Ctrl** + **Shift** + **Del** (Windows) or **command** + **shift** + **delete** (Mac), then in the window that opens, select the desired time range in the "Time range to clear" list, if necessary click the Details arrow, make sure the **Cookies** box is checked, and then click **Clear Now** (Windows) or **OK** (Mac).	Click the **Open menu** button, click **History**, click **Clear Recent History**, then in the window that opens, select the desired time range in the "Time range to clear" list, if necessary click the Details arrow, make sure the **Cookies** box is checked, and then click **Clear Now** (Windows) or **OK** (Mac).
Chrome	Press **Ctrl** + **H** (Windows) or **command** + **Y** (Mac), click **Clear browsing data**, then in the window that opens, select the desired time range in the "Obliterate the following items from" list, make sure the **Cookies and other site and plug-in data** box is checked, and then click **Clear browsing data**.	Click the **Customize and control Google Chrome** button, click **History**, click **Clear browsing data**, then in the window that opens, select the desired time range in the "Obliterate the following items from" list, make sure the **Cookies and other site and plug-in data** box is checked, and then click **Clear browsing data**.

Table 9-2 Steps to delete cookies in IE, Firefox, and Chrome

Storing State Information with the Web Storage API

- Common uses of cookies never originally planned
- Web Storage
  - Draft specification of W3C
  - Not supported by some older browsers
  - Cookies are still the standard
  - Web Storage offers additional features
  - Applications targeted at specific browsers can use Web Storage

Storing State Information with the Web Storage API (cont'd.)

- Two Web Storage properties of `Window` object
  - `localStorage`
    - Remains until you run code to delete it
    - Similar to persistent cookies
  - `sessionStorage`
    - Removed automatically when user closes browser tab or window
    - Like temporary cookies

Storing State Information with the Web Storage API (cont'd.)

- Storing/reading Web Storage easier than cookies
  - Example:

```
var firstName = document.getElementById("fname").value;
localStorage.fName = firstName;
```

- To use a method
  - Preface it with `localStorage` or `sessionStorage`

Storing State Information with the Web Storage API (cont'd.)

METHOD	DESCRIPTION
`clear()`	Removes all data
`getItem(name)`	Retrieves the value of the item with the name *name*
`removeItem(name)`	Removes the item with the name *name* from storage
`setItem(name, value)`	Stores an item with the name *name* and the value *value*

Table 9-3 Methods of the Web Storage API

Understanding Security Issues

- Security threats
  - Viruses, worms, data theft by hackers
- Consider web server security and secure coding issues
- Web server security technologies
  - Firewalls
  - Secure Socket Layer (SSL)
- JavaScript programs downloaded and execute locally

Secure Coding with JavaScript

- Secure coding or defensive coding
  - Writing code to minimize any intentional or accidental security issues
- All code: 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
- Code injection attack
  - Program or user enters JavaScript code that changes function of web page
    - Validating forms helps prevent this
    - Escaping characters also important

JavaScript Security Concerns

- Another security concern
  - Privacy of individual client information in the web browser window
- Important JavaScript security feature
  - Lack of certain types of functionality
    - File manipulation
    - Create a network connection
    - Cannot run system commands or execute programs on a client

The Same Origin Policy

- Restricts how JavaScript code access between windows/tabs/frames
- To view and modify elements in other windows and frames
  - They must have the same protocol and exist on the same web server
- Same origin policy applies to:
  - The domain name
  - The server on which a document located

The Same Origin Policy (cont'd.)

- Policy prevents:
  - Malicious scripts from modifying content of other windows and frames
  - Theft of private browser information and information displayed on secure web pages
- Policy protects integrity of web page design

The Same Origin Policy (cont'd.)

- `domain` property of the `Document` object
  - Changes 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

Using Third-Party Scripts

- Sometimes you want to run scripts from other domains
  - Known as third-party scripts
  - Examples:
    - Widgets that run from provider's server
    - Using Content Delivery Network (CDN)
- To enable third-party script
  - Include `script` element with `src` value pointing to third-party content
  - Exception to same origin policy

Summary

- State information
  - Information about individual visits to a web site
- HTTP originally designed to be stateless
- Most common tools for maintaining state information:
  - Query strings, hidden form fields, cookies, Web Storage
- Query string
  - Set of name-value pairs appended to a target URL
- Can hide information from users in a hidden form field

Summary (cont'd.)

- Cookies
  - Small pieces of information about a user
  - Stored by a web server
  - Can be temporary or persistent
- Cookie property created with a required `name` attribute
- Can use special characters in cookies if using encoding
- `encodeURIComponent()` function encodes the individual parts of a URI

Summary (cont'd.)

- When reading a cookie or other text string encoded
  - Must first decode it with the `decodeURIComponent()` function
- Cookies: one continuous string that must be parsed
- Web Storage similar to cookies
  - Offers more features, but not as widely supported
- "Secure coding" or "defensive coding"
  - Writing code to minimize any intentional or accidental security issues