

Servlet

Recall

Problems presenting continuity
e.g. zip code, e.g. auth

HTTP Stateless

Appearance of continuity between many pages
From server's point of view, the HTTP requests all come in independently

Solution 1 -- Hidden field

(Breadcrumbs)

(e.g. HW3)
Note the information in a hidden field
Stored on the client side
Sent back to the server on a later request
This works ok

Solution 2 -- Session

Store information about the series of pages -- the session -- on the server side.
i.e. a hash table on the server side
Maybe just stored in the memory of the server
Maybe written to a "session" database, so even if the server crashes, or there are
multiple servers, the session-id still works

1. Session ID

Arbitrary identifier for session -- "123xyz"

2. Server

Create an empty session when first see the client
Create an id
Server side -- store the session under its id
Pass the id to the client (e.g. hidden field id=154xyz)

3. Client

Contact the server
Get back HTML etc. with the id in it (e.g. hidden field)
Send the info back to the server on each request

4. Server again

On later client request

See the id=123xyz binding from the client

Look up the appropriate session object

Look up or edit state in the session

e.g. Cart

Store the "cart" collection of items in the server side session

The client surfs through various product pages

When they click "buy", the server adds that item to the session cart collection

The only state the client maintains is id=123xyz, so even if they use the back button and then start forward again, the server still pulls up the right cart info from the one, current state in the server side session

e.g. Auth

Client logs in to the server with user= xxxx and password=xxx bindings

If the password is correct, server notes auth=true, and auth-time=current-time in the session

Server generates a random session id, and sends that back to the client

Any client request that comes in with the right id, we conclude is part of this authorized session

This works, since we only send the random id back to the client that had the right password. That's the only party that has the id

The id is a temporary shared secret -- known only to the client and server

HTTP is susceptible to eavesdropping -- could use secure HTTP to avoid that problem

Auth notes

Do not store auth=true on client side

Store id on client side

Server has the data, the client just points to it

Server can note the passage of time, and after, say, an hour delete the server session, or just set auth=false so that the client will be asked for their password again

Storing State on Client side

The server sends something to the client

On a later request, the client sends it back to the server -- in this way the server can keep track of its many dialogs

#1 - Hidden Fields

We've seen this in detail

#2 - URL re-writing

Modify HTML

Modify every URL on the generated HTML to include the session id.

At the end of the url put .../cgi-bin/foo.pl/session-id=34567

In a way, this is similar to the hidden field, but it does not require forms. We alter the HTML we send the client.

Brute-Force

This is a bit clumsy, but it works.

Problems if they come to the site from some other url, such as on another site.

Our site -> follow link to site B -> follow link back to our site -- hidden fields and url-rewrites don't make it back

#3 Cookies

A netscape technology, now widely adopted

Privacy concerns -- way overblown

Set-Cookie: Server Response

Set-cookie: id=123xyz

The server adds this to the **HTTP response** header

The client notes the cookie, and the server it came from

Cookie features

Binding just a name=value pair, stick to a-z0-9 chars

Lifetime = session: not written to file system, just works while the browser is running. Used for session id's.

Lifetime = persistent: written to the users prefs, so that later runs of the browser will remember it.

Send Cookie: Client Request

When sending a request back to the server, include a cookie binding id=123 on the **HTTP response**

"Here's the cookie binding you sent to me earlier. I'm sending it back to you as a reminder."

The server can look at the cookie binding in the header if it chooses to.

Cookie Applications

Session tracking -- use cookies to store the session id.

- Works with the back button (at least as well as hidden fields did)

- Works, even if they surf through another site.

- Works without messing with the HTML

Preferences -- use persistent cookies to recognize the user when they first hit the site, and so get some things right for them.

- e.g. slashdot remembers your username and reading prefs.

- e.g. amazon knows it's you when you go to their site (but they still know to ask for the password if you try to do something)

Privacy vs. Cookies

Site tracking -- amazon

Cross-site-tracking

- GIF with cookie trick

- e.g. doubleclick.com

- Attempt to correlate surfing on multiple sites

Harmless: they may be able to do better market research (which movies go with which books).

Harmful: more marketing directed at you -- this is a legitimate fear, but it does not have much to do with cookies. We'll have a separate lecture on the economics of marketing in this way.