

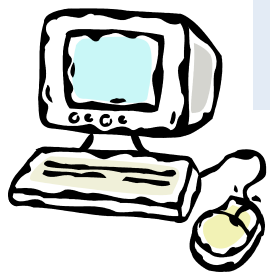
Overview

- The state management problem
 - For now, assume a single web server
- Possible solutions
 - IP address
 - Query string
 - Hidden form fields
 - Cookies
- State management in PHP

The state management problem

- HTTP protocol

- Designed as a stateless, request/response protocol
- No provisions for sessions spanning multiple request/response cycles



```
GET /products.html HTTP/1.1
Host: www.mtech.edu
User-agent: Mozilla/4.0
```



```
HTTP/1.1 200 OK
Date: Thu, 17 Nov 2011 15:54:10 GMT
Server: Apache/2.2.16 (Debian)
Last-Modified: Wed, 14 Sep 2011 17:04:27 GMT
Content-Length: 285
```



```
GET /checkout.html HTTP/1.1
Host: www.mtech.edu
User-agent: Mozilla/4.0
```



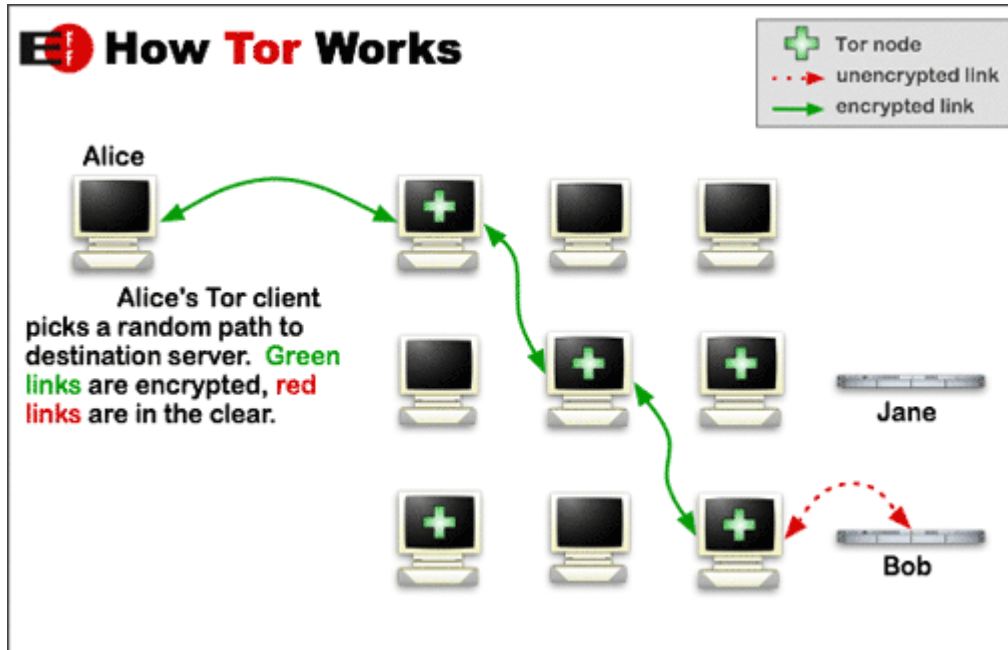
The goal of session-ness

- We want web apps to behave like desktop apps
 - Series of query/responses made to appear as an one ongoing user experience
 - e.g. We don't want to re-login on every page
- Problem 1: How to uniquely ID a session?
 - What exactly do we mean by "session"?
 - Person
 - Person + computer
 - Person + computer + browser
 - Person + computer + browser + browser tab
- Problem 2: Where to store the session data?
 - On the client, on the server, on the page?

Identifying a session

- Option 1: IP address

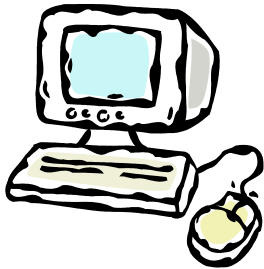
- Web server could use the IP address of the requestor
- Not very reliable, broken by:
 - NAT - Different users on same network look like same user
 - Anonymity services such as Tor



Identifying a session

- Option 2: Put something in the URLs

- First hit to server, just plain URL
- Server adds to every returned link, e.g. ?id=528fa623



```
GET /index.html HTTP/1.1
Host: www.mtech.edu
User-agent: Mozilla/4.0
```



```
HTTP/1.1 200 OK
Date: Thu, 17 Nov 2011 15:54:10 GMT
Server: Apache/2.2.16 (Debian)
Content-Length: 285
```



```
<html><body>
<h1><blink>Welcome to Montana Tech.</blink></h1>
<a href="mission.php?id=528fa623">Mission statement</a>
<a href="press.php?id=528fa623">Press releases</a>
<a href="alumninews.php?id=528fa623">Alumni news</a>
...
```

Identifying a session

- Option 2: Put something in the URLs

- First hit to server, just plain URL

- Server adds to every returned link, e.g. ?id=528fa623

- Advantages:

- Will work, can't be disabled

- Could support independent sessions in same browser

- Disadvantages:

- User can easily see/change/delete the ID

- Bookmark and use much later, no built-in expiration

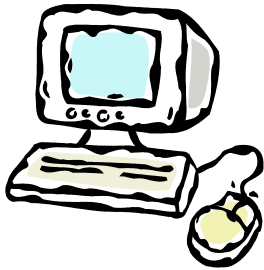
- Only lives as long as the browser window

- Leaks ID to log files on web server, firewalls, other web sites via HTTP REFERER field, to friends via emailing link

Identifying a session

- Option 3: Use hidden form fields

- First hit to server, add ID to hidden field in returned form
- User submits form, hidden ID passed to server



```
GET /index.html HTTP/1.1
Host: www.mtech.edu
User-agent: Mozilla/4.0
```



```
HTTP/1.1 200 OK
Date: Thu, 17 Nov 2011 15:54:10 GMT
Server: Apache/2.2.16 (Debian)
Content-Length: 285
```



```
<html><body>
<form action="submit.php" method="POST">
<input type="text" name="username" value="" />
<input type="hidden" value="528fa623" />
</form>
...
```


Identifying a session

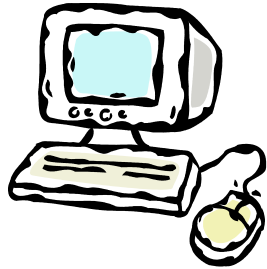
- **Option 3: Use hidden form fields**
 - First hit to server, add ID to hidden field in returned form
 - User submits form, hidden ID passed to server
 - Advantages:
 - Will work, can't be disabled
 - Using POST
 - Not as obvious as embedding in URL string
 - Doesn't appear in bookmarks, log files, etc.
 - Could support independent sessions in same browser
 - Disadvantages:
 - More complicated web pages
 - Everything becomes a form submission

Cookies



- Option 4: Browser Cookies
 - Introduced in Netscape, 1994
 - e-commerce app, needed to implement a shopping cart
 - Name/value pairs originally sent from server
 - Stored in browser (if cookies enabled)
 - User can delete
 - Expire after session
 - Expire after elapsed time
 - Browser makes HTTP request to `www.example.com`
 - Sends any cookies marked for that domain
 - Server can use cookie to ID session
 - They cannot carry malware
 - They can have privacy implications

Anatomy of a cookie



```
GET /index.php HTTP/1.1
Host: www.mtech.edu
User-agent: Mozilla/4.0
```



```
HTTP/1.1 200 OK
Date: Thu, 17 Nov 2011 15:54:10 GMT
Server: Apache/2.2.16 (Debian)
Content-Length: 285
Set-Cookie: sessionID=528fa623; path=/; Expires=Wed, 09
Mar 2014 11:00:00 GMT
```



```
<html><body>
<a href="mission.php">Mission statement</a>
<a href="press.php">Press releases</a>
...
```

```
GET /mission.php HTTP/1.1
Host: www.mtech.edu
User-agent: Mozilla/4.0
Cookie: sessionID=528fa623
```



Cookie attributes

- Domain

- Defaults to domain of page that set cookie
- Browser only sends cookie back to this domain
- All hosts at a domain, leave off first part: .mtech.edu
- Cannot be different from page sent from

```
HTTP/1.1 200 OK
```

```
Date: Thu, 17 Nov 2011 15:54:10 GMT
```

```
Server: Apache/2.2.16 (Debian)
```

```
Content-Length: 285
```

```
Set-Cookie: sessionID=528fa623; path=/; Expires=Wed, 09 Mar 2014 11:00:00 GMT
```

```
<html><body>
```

```
...
```

Cookie attributes

- Path

- Defaults to path of page that set cookie
- Browser only sends cookie back to pages below the path
 - If path is "/products/" only sent to pages in products directory of web site

```
HTTP/1.1 200 OK
```

```
Date: Thu, 17 Nov 2011 15:54:10 GMT
```

```
Server: Apache/2.2.16 (Debian)
```

```
Content-Length: 285
```

```
Set-Cookie: sessionID=528fa623; path=/; Expires=Wed, 09 Mar 2014 11:00:00 GMT
```

```
<html><body>
```

```
...
```

Cookie attributes

- Expires
 - If not set, delete when browser closes
 - Specifies date and time of expiration
 - DOW, DD, MON, YYYY HH:MM:SS GMT
- Max-Age
 - Alternate to Expires, seconds in future

```
HTTP/1.1 200 OK
```

```
Date: Thu, 17 Nov 2011 15:54:10 GMT
```

```
Server: Apache/2.2.16 (Debian)
```

```
Content-Length: 285
```

```
Set-Cookie: sessionID=528fa623; path=/; Expires=Wed, 09 Mar 2014 11:00:00 GMT
```

```
<html><body>
```

```
...
```

Cookie attributes

- **Secure**
 - If present, browser only sends if on secure web page
 - Server probably should only set on secure page!
- **HttpOnly**
 - Use cookies via HTTP protocol only
 - e.g. Can't access via JavaScript
 - Avoid a cross-site scripting attack stealing cookies

```
HTTP/1.1 200 OK
```

```
Date: Thu, 17 Nov 2011 15:54:10 GMT
```

```
Server: Apache/2.2.16 (Debian)
```

```
Content-Length: 285
```

```
Set-Cookie: sessionId=528fa623; path=/; Expires=Wed, 09 Mar 2014 11:00:00 GMT;  
Secure; HttpOnly
```

```
<html><body>
```

Third party cookies

- Cookies are sent only to server setting them
 - Except a site can link external images
 - e.g. banner ads
 - Most browsers default to allowing third-party cookies



Storing session data

- Now that we have a session ID, where to store other session data?
- On the client:
 - In the URL string
 - In hidden form fields
 - In browser cookies
- On the server, using session ID as key:
 - In shared memory
 - In a disk file
 - In a database

Cookie-only state

- Complete state in client-side cookies
 - No server-side state, great for scalability!
 - Limits to the amount of data (browser dependent)
 - Only one session per browser
 - Only works if cookies enabled
 - Users can delete or muck with cookies

```
HTTP/1.1 200 OK
```

```
Date: Thu, 17 Nov 2011 15:54:10 GMT
```

```
Server: Apache/2.2.16 (Debian)
```

```
Content-Length: 285
```

```
Set-Cookie: products=343984,545454,98983;username=bob; path=/; Expires=Wed, 09  
Mar 2014 11:00:00 GMT
```

```
<html><body>
```

```
...
```

PHP sessions

- PHP session support
 - Provides a unique session ID
 - Done via cookies and/or URL fallback
 - Stores name/value pairs according to session ID
 - Defaults to using a temp file in /tmp
 - Works for a single server
 - Multiple servers requires a shared store:
 - Database (e.g. MySQL)
 - Shared memory cache (e.g. memcached)
 - Shared file system

Using PHP sessions

- Starting a PHP session

- Assume for now cookies are enabled
- Every PHP script runs `session_start()` function
 - You must do this before any other output on the page!
- Sets a unique ID the first time
- Returns the same ID every other time
- ID is available using `session_id()` function

```
<?
    session_start();
    echo "<p>Your session ID is " . session_id() . "</p>";
?>
```

Using PHP sessions

- Storing sessions variables
 - Make sure session is started at top of page
 - Use `$_SESSION` superglobal to get/set values

```
<?
    session_start();
    $_SESSION["product1"] = "Snickers";
    $_SESSION["price1"]   = "1.25";
    echo "Product added.";
?>
```

First page that sets two session variables.

```
<?
    session_start();
    echo "Product: " . $_SESSION["product1"] . "<br />";
    echo "Price: "   . $_SESSION["price1"]   . "<br />";
?>
```

Second page that shows what is currently stored in the session variables.

Using PHP sessions

- What if cookies not enabled?
 - If cookie value not available, constant `SID` set
 - Append SID to GET parameters of every link on page
 - `session_start()` will load from GET instead of cookie
 - Session variables work as normal
 - **BUT: turned off in PHP by default to defend against exploits**

```
<a href="order.php<? echo SID; ?>">Order form</a>
```



```
<a href="order.php?PHPSESSID=738feaw23872">Order form</a>
```

Summary

- Forcing state onto stateless HTTP protocol
- Find a way to unique track session
 - Using URLs
 - Using hidden form fields
 - Using cookies
- Store state somewhere
 - Client-side
 - Server-side