CSE 484 / CSE M 584 (Spring 2012)

Web Security

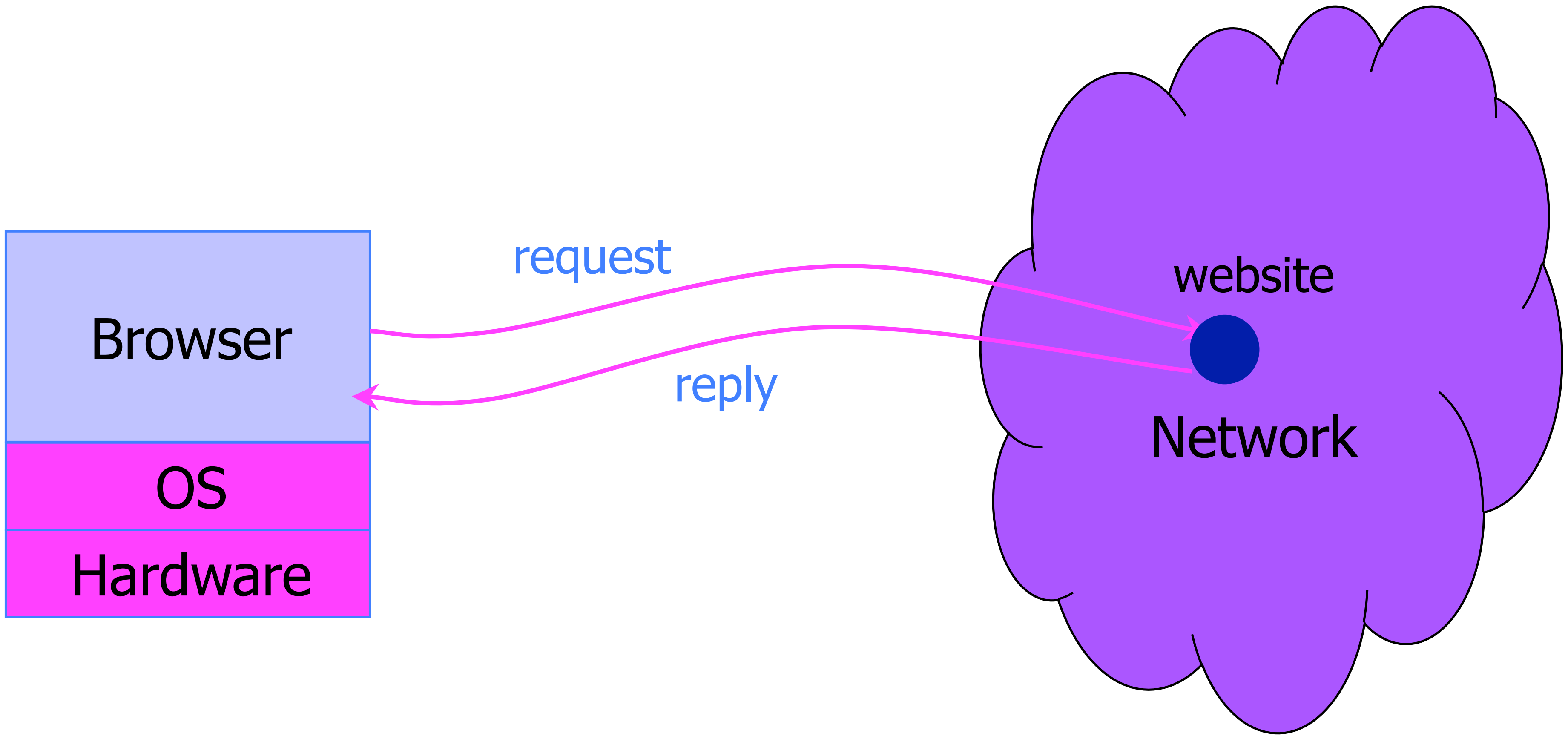
Tadayoshi Kohno

Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Goals for Today

- Web Security
- (Back later to asymmetric cryptography)

Browser and Network



Types of problems

- Web browser problems (client side)
  - Exploit vulnerabilities in browsers
  - Install botnets, keyloggers
  - Exfiltrate data
- Web application code (server side)
  - Exploit vulnerabilities in code running on servers (and coming from servers)
  - Examples: XSS, XSRF, SQL injection, clickjacking, insecure parameters, security misconfigurations
  - Steal user credentials, data from databases, ...

Example Questions

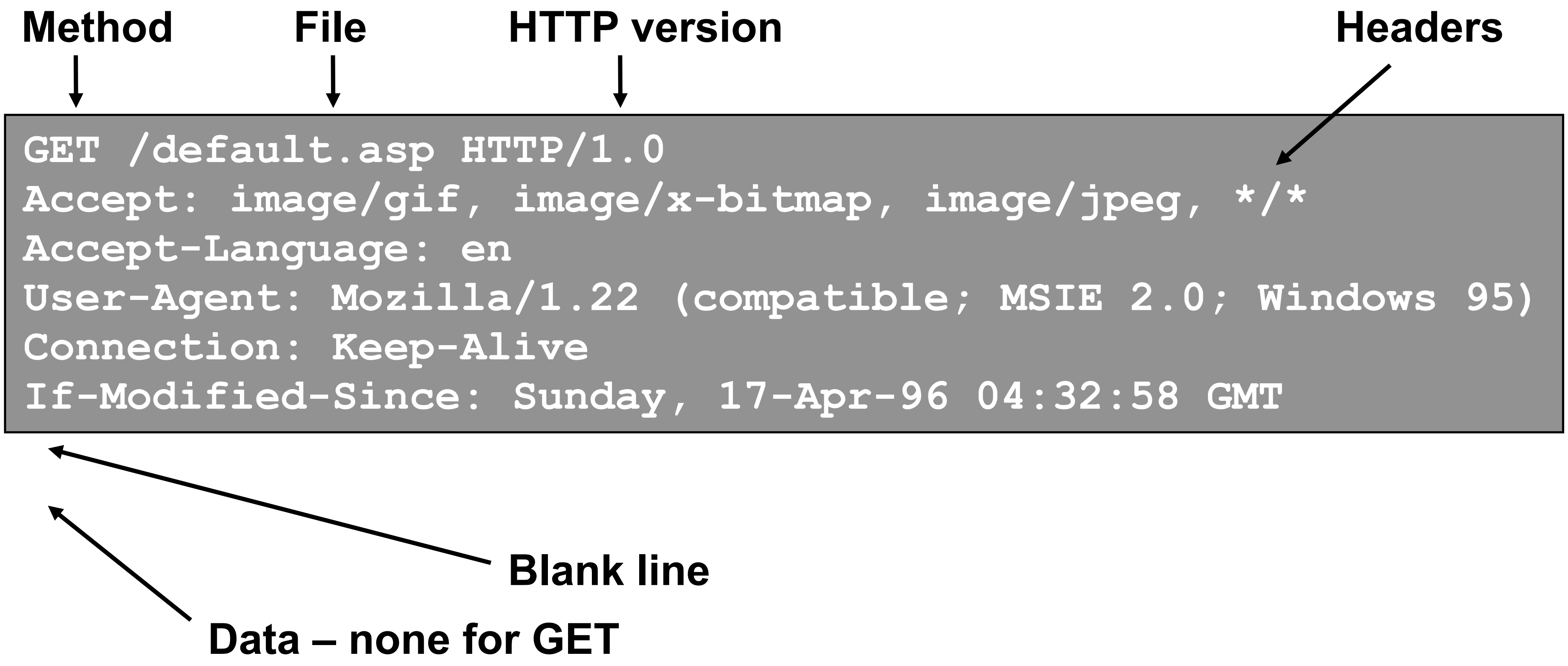
- How do websites know who you are?
- How do you know who the website is?
- Can someone intercept traffic ?
- Related: How can you better control flow of information?

- Our focus: High-level principles (lab focuses on pragmatics)
- Focus on a bit of history: How we got here

HTTP: HyperText Transfer Protocol

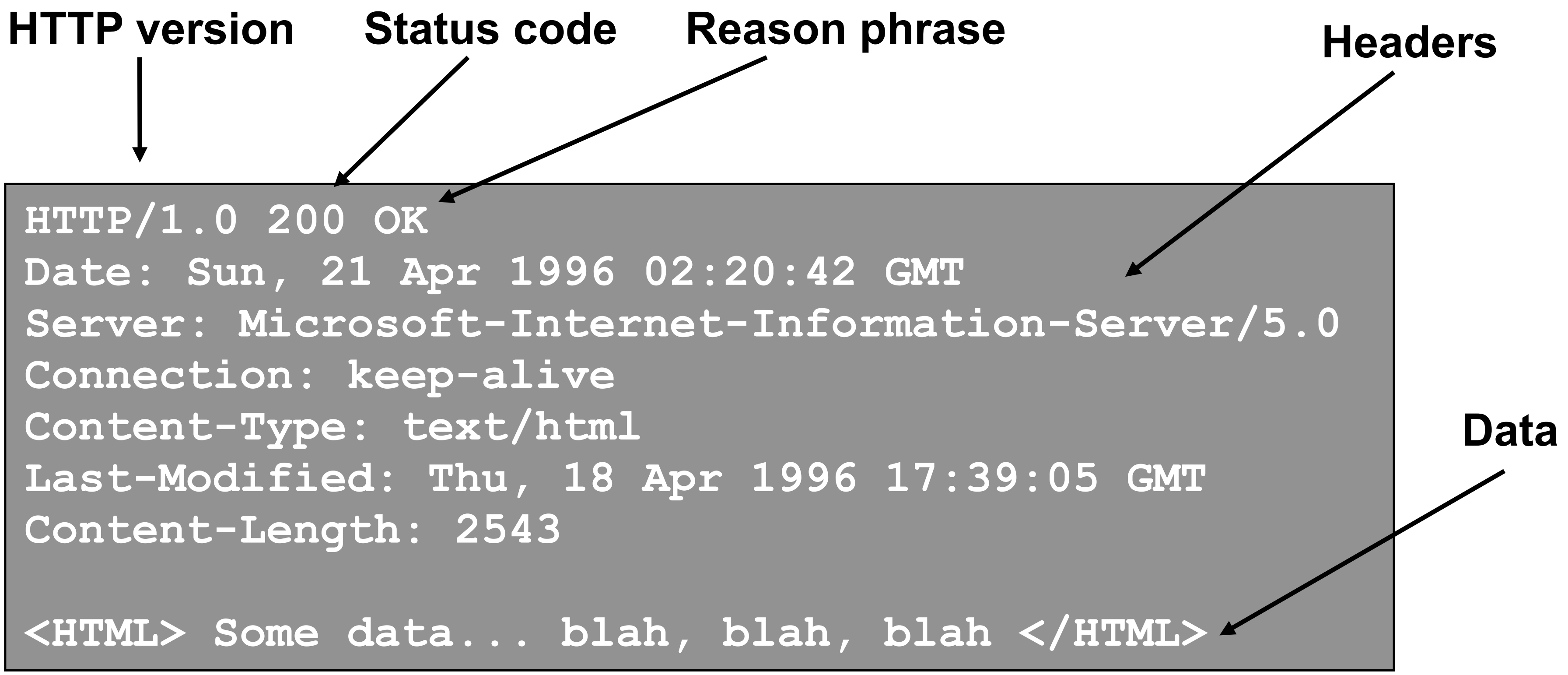
- Used to request and return data
  - Methods: GET, POST, HEAD, …
- Stateless request/response protocol
  - Each request is independent of previous requests
  - Statelessness has a significant impact on design and implementation of applications
- Evolution
  - HTTP 1.0: simple
  - HTTP 1.1: more complex
  - … HTML 5 …

HTTP Request



```
GET /default.asp HTTP/1.0
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Connection: Keep-Alive
If-Modified-Since: Sunday, 17-Apr-96 04:32:58 GMT
```

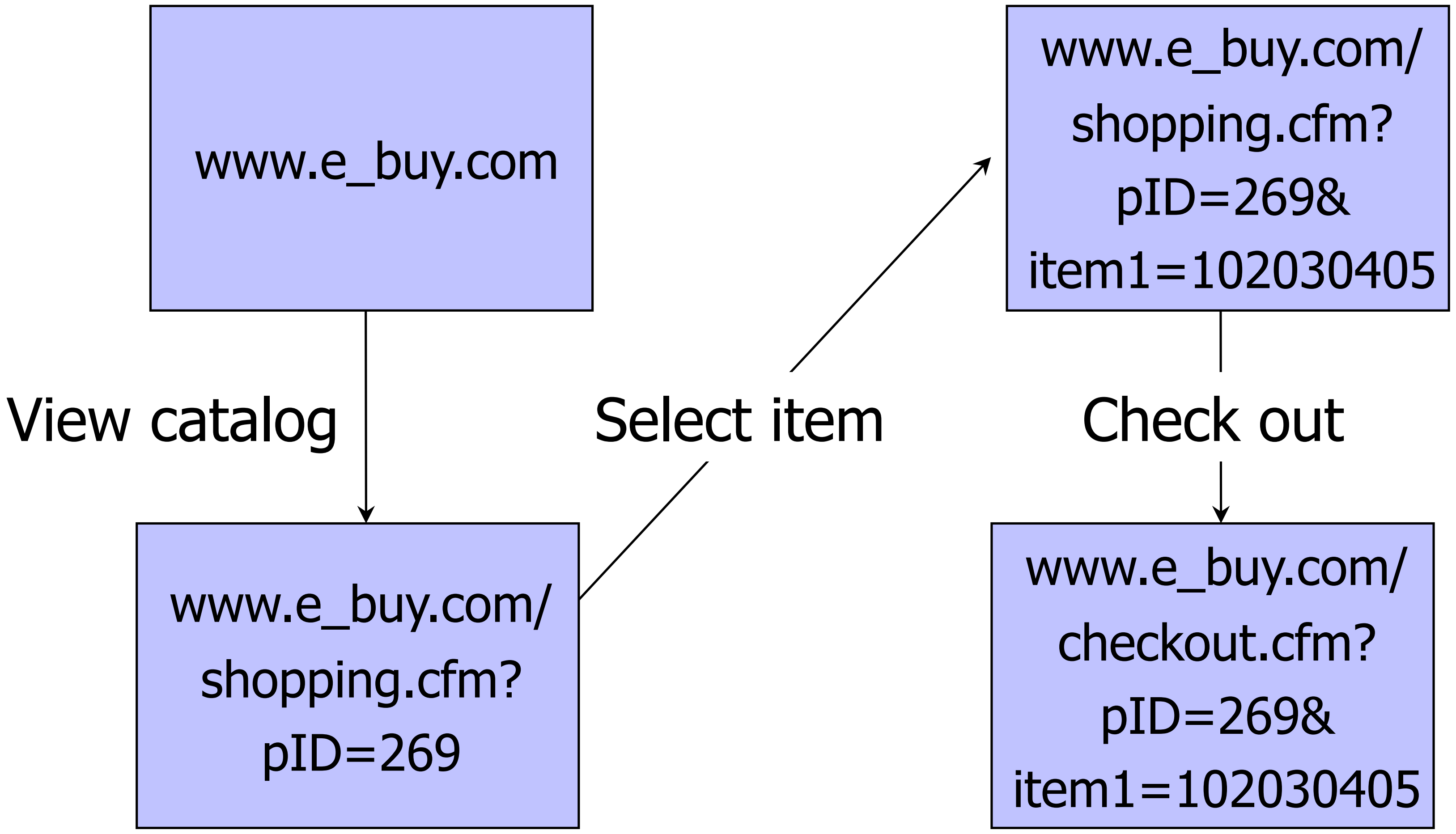
HTTP Response



```
HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Content-Length: 2543

<HTML> Some data... blah, blah, blah </HTML>
```

Primitive Browser Session


www.e_buy.com
View catalog
www.e_buy.com/ shopping.cfm? pID=269
Select item
www.e_buy.com/ shopping.cfm? pID=269& item1=102030405
Check out
www.e_buy.com/ checkout.cfm? pID=269& item1=102030405

Store session information in URL; easily read on network

FatBrain.com circa 1999 [due to Fu et al.]

- User logs into website with his password, authenticator is generated, user is given special URL containing the authenticator

  https://www.fatbrain.com/HelpAccount.asp?t=0&p1=me@me.com&p2=540555758

  - With special URL, user doesn't need to re-authenticate
    - Reasoning: user could not have not known the special URL without authenticating first. That's true, BUT…
- Authenticators are global sequence numbers
  - It's easy to guess sequence number for another user

    https://www.fatbrain.com/HelpAccount.asp?t=0&p1=SomeoneElse&p2=540555752
  - Partial fix: use random authenticators

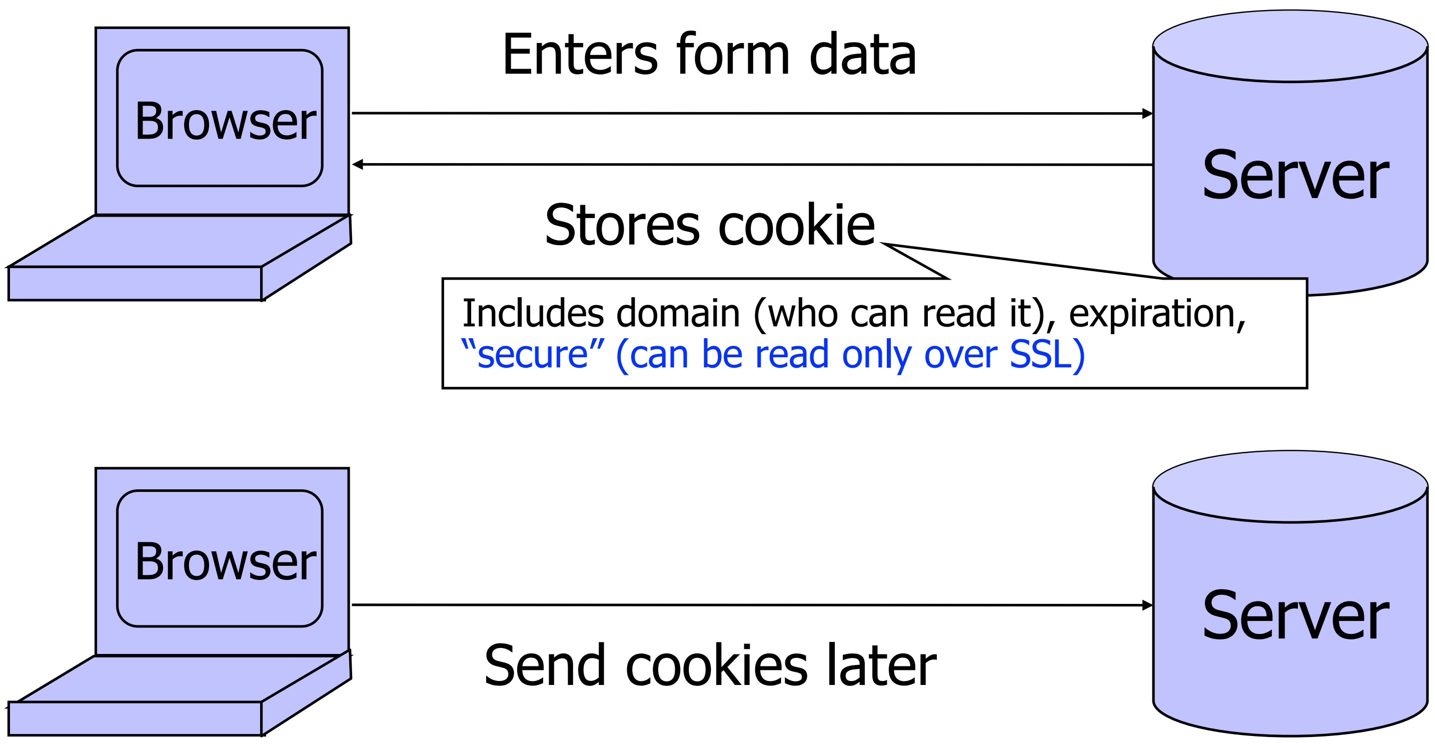
Bad Idea: Encoding State in URL

- Unstable, frequently changing URLs
- Vulnerable to eavesdropping
- There is no guarantee that URL is private

Cookies

Storing Info Across Sessions

- A cookie is a data blob created by an Internet site to store information on your computer


Browser
Enters form data
Server
Stores cookie
Includes domain (who can read it), expiration,
"secure" (can be read only over SSL)
Browser
Send cookies later
Server

HTTP is traditionally a stateless protocol; cookies add state

What Are Cookies Used For?

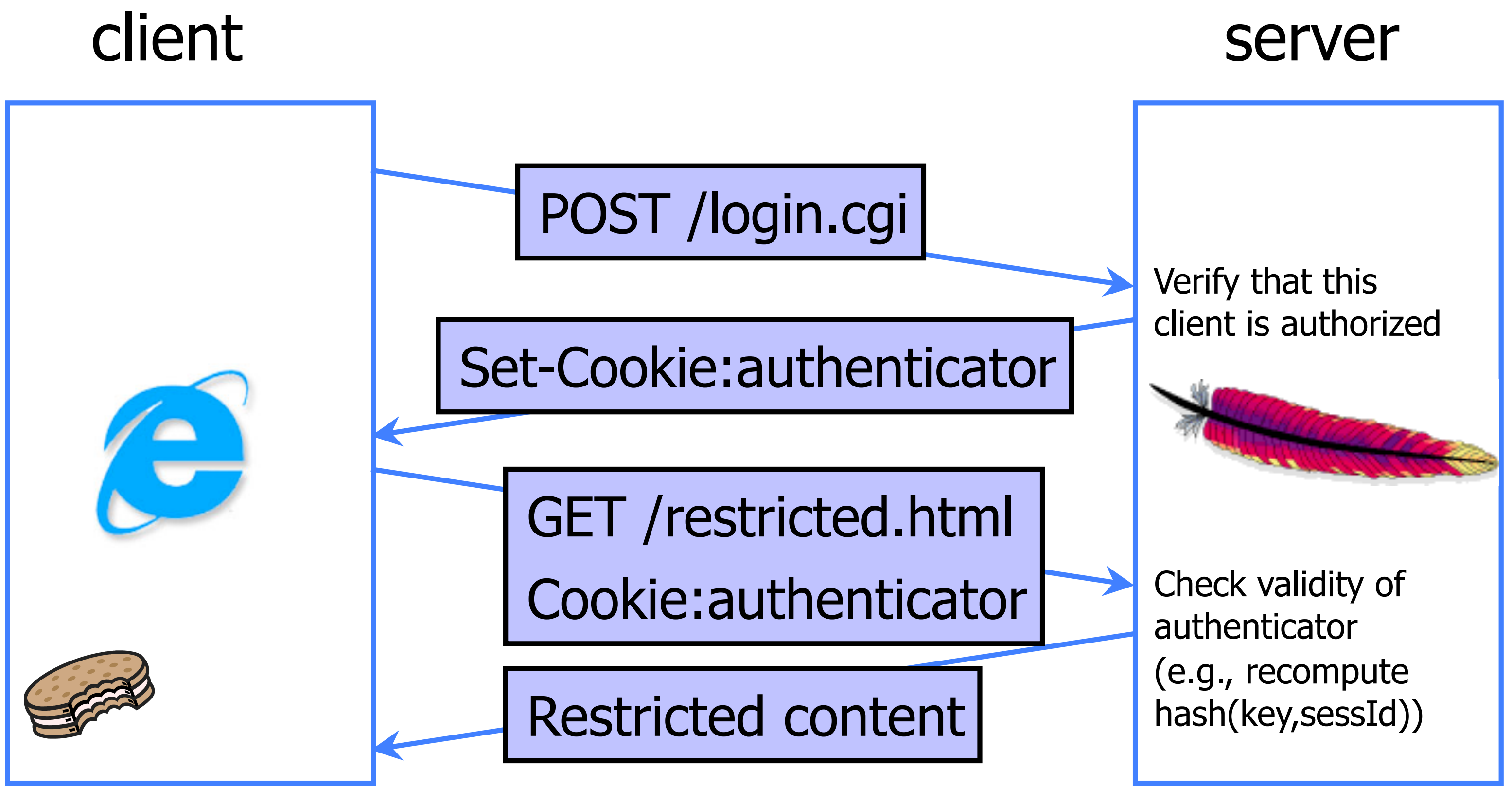
- Authentication
  - Use the fact that the user authenticated correctly in the past to make future authentication quicker
- Personalization
  - Recognize the user from a previous visit
- Tracking
  - Follow the user from site to site; learn his/her browsing behavior, preferences, and so on

Web Authentication via Cookies

- Need authentication system that works over HTTP and does not require servers to store session data

- Servers can use cookies to store state on client
  - When session starts, server computes an authenticator and gives it back to browser in the form of a cookie
    - Authenticator is a value that client cannot forge on his own
    - Example: MAC(server's secret key, session id)
  - With each request, browser presents the cookie
  - Server recomputes and verifies the authenticator
    - Server does not need to remember the authenticator

Typical Session with Cookies



Authenticators must be unforgeable and tamper-proof

(malicious client shouldn't be able to compute his own or modify an existing authenticator)

Cookie Management

- Cookie ownership
  - Once a cookie is saved on your computer, only the website that created the cookie can read it (supposedly)
- Variations
  - Session cookies
    - Stored until you quit your browser
  - Persistent cookies
    - Remain until deleted or expire
  - Third-party cookies
    - Set by sites embedded within other sites (e.g., ads)

Privacy Issues with Cookies

- Cookie may include any information about you known by the website that created it
  - Browsing activity, account information, etc.
- Sites can share this information
  - Advertising networks
- Browser attacks could invade your privacy

  November 8, 2001 (and **many more** since):

  Users of Microsoft's browser and e-mail programs could be vulnerable to having their browser cookies stolen or modified due to a new security bug in Internet Explorer (IE), the company warned today

Storing State in Browser

- Dansie Shopping Cart (2006)
  - "A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order."

```
<FORM METHOD=POST
 ACTION="http://www.dansie.net/cgi-bin/scripts/cart.pl">

  Black Leather purse with leather straps<BR>Pri

  <INPUT TYPE=HIDDEN NAME=name     VALUE="Black leather purse">
  <INPUT TYPE=HIDDEN NAME=price    VALUE="20.00">
  <INPUT TYPE=HIDDEN NAME=sh       VALUE="1">
  <INPUT TYPE=HIDDEN NAME=img      VALUE="purse.jpg">
  <INPUT TYPE=HIDDEN NAME=custom1  VALUE="Black leather purse     with
leather straps">

  <INPUT TYPE=SUBMIT NAME="add" VALUE="Put in Shopping Cart">

</FORM>
```

Change this to 2.00

Shopping Cart Form Tampering

http://xforce.iss.net/xforce/xfdb/4621

- Many Web-based shopping cart applications use hidden fields in HTML forms to hold parameters for items in an online store. These parameters can include the item's name, weight, quantity, product ID, and price. Any application that bases price on a hidden field in an HTML form is vulnerable to price changing by a remote user. A remote user can change the price of a particular item they intend to buy, by changing the value for the hidden HTML tag that specifies the price, to purchase products at any price they choose.
- **Platforms Affected:**
  - 3D3.COM Pty Ltd: ShopFactory 5.8 and earlier   @Retail Corporation: @Retail Any version
  - Adgrafix: Check It Out Any version   Baron Consulting Group: WebSite Tool Any version
  - ComCity Corporation: SalesCart Any version   Crested Butte Software: EasyCart Any version
  - Dansie.net: Dansie Shopping Cart Any version   Intelligent Vending Systems: Intellivend Any version
  - Make-a-Store: Make-a-Store OrderPage Any version   McMurtrey/Whitaker & Associates: Cart32 2.6
  - McMurtrey/Whitaker & Associates: Cart32 3.0   pknutsen@nethut.no: CartMan 1.04
  - Rich Media Technologies: JustAddCommerce 5.0   SmartCart: SmartCart Any version
  - Web Express: Shoptron 1.2

Storing State in Browser Cookies

- Set-cookie: price=299.99
- User edits the cookie… cookie: price=29.99
- What’s the solution?
- Add a MAC to every cookie, computed with the server’s secret key
  - Price=299.99; MAC(ServerKey, 299.99)

Storing State in Browser

- **Dansie Shopping Cart (2006)**
  - "A premium, comprehensive, Perl shopping cart. Increase your web sales by making it easier for your web store customers to order."

MAC(K, "$20")

```
<FORM METHOD=POST
 ACTION="http://www.dansie.net/cgi-bin/scripts/cart.pl">

  Black Leather purse with leather straps<BR>Price: $20.00<BR>

  <INPUT TYPE=HIDDEN NAME=name      VALUE="Black leather purse">
  <INPUT TYPE=HIDDEN NAME=pricemac VALUE=~~"F13A3....B2">~~  A319F...3C
  <INPUT TYPE=HIDDEN NAME=sh        VALUE="1">
  <INPUT TYPE=HIDDEN NAME=img       VALUE="purse.jpg">
  <INPUT TYPE=HIDDEN NAME=custom1   VALUE="Black leather purse     with
leather straps">

  <INPUT TYPE=SUBMIT NAME="add" VALUE="Put in Shopping Cart">

</FORM>
```

MAC(K, "$2")

Better: MAC(K, "$20,Black leather purse, product number 12345, …")

WSJ.com circa 1999 [due to Fu et al.]

- Idea: use user,hash(user||key) as authenticator
  - Key is secret and known only to the server. Without the key, clients can't forge authenticators.
  - || is string concatenation
- Implementation: user,crypt(user||key)
  - crypt() is UNIX hash function for passwords
  - crypt() truncates its input at 8 characters
  - Usernames matching first 8 characters end up with the same authenticator
  - No expiration or revocation
- It gets worse… This scheme can be exploited to extract the server's secret key

Attack

username	crypt(username,key,“00”)	authenticator cookie
AliceBob1	008H8LRfzUXvk	AliceBob1008H8LRfzUXvk
AliceBob2	008H8LRfzUXvk	AliceBob2008H8LRfzUXvk

“Create” an account with a 7-letter user name…

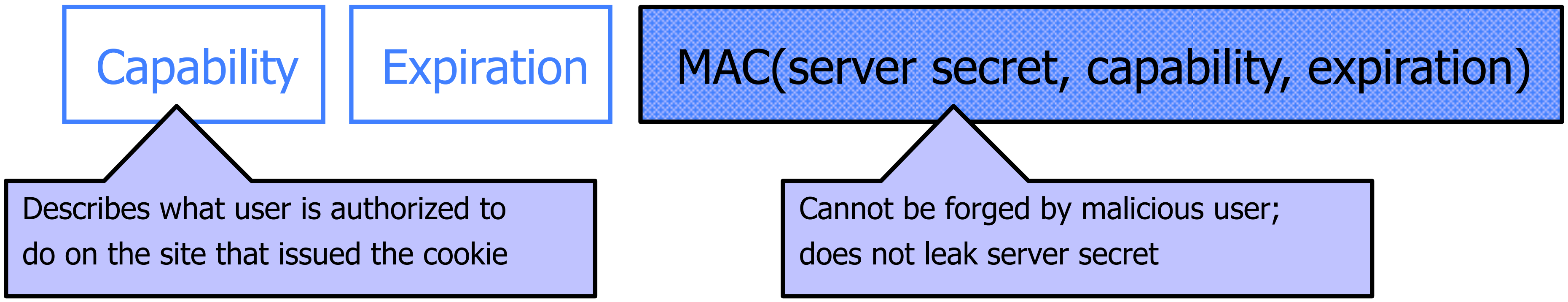
AliceBoA	0073UYEre5rBQ	Try logging in: access refused
AliceBoB	00bkHcfOXBKno	Access refused
AliceBoC	00ofSJV6An1QE	Login successful! 1st key symbol is C

Now a 6-letter user name…

AliceBCA	001mBnBErXRuc	Access refused
AliceBCB	00T3JLLfuspdo	Access refused… and so on

- Only need 128 x 8 queries instead of intended 128^8
- Minutes with a simple Perl script vs. billions of years

Better Cookie Authenticator


Capability
Expiration
MAC(server secret, capability, expiration)
Describes what user is authorized to do on the site that issued the cookie
Cannot be forged by malicious user; does not leak server secret

- Main lesson: be careful rolling your own
  - Homebrewed authentication schemes are easy to get wrong
- There are standard cookie-based schemes

Web Applications

- Online banking, shopping, government, etc.
- Website takes input from user, interacts with back-end databases and third parties, outputs results by generating an HTML page
- Often written from scratch in a mixture of PHP, Java, Perl, Python, C, ASP, ...
- Security is a potential concern.
  - Poorly written scripts
  - Sensitive data stored in world-readable files

General issue: Inadequate Input Validation

- http://victim.com/copy.php?name=username

Supplied by the user!

- copy.php includes

```
system("cp temp.dat $name.dat")
```

- User calls

  http://victim.com/copy.php?name="a; rm *"

- copy.php executes

```
system("cp temp.dat a; rm *.dat");
```