Lecture 7
HTTP and Session Management

ENGG5105/CSCI5470 Computer and Network Security

Spring 2014

Patrick P. C. Lee

Acknowledgments

- Some slides are based on Stanford course CS155 Computer and Network Security, Lectures 8 - 11
  http://crypto.stanford.edu/cs155

Topics on Web Security

- 1st week
  - Session management
  - HTTPS
- 2nd week
  - Cross-site attacks, SQL injection
- Goal:
  - To understand existing attacks and defenses of web applications

Top 10 Web Attacks

- A1: Injection
- A2: Cross-Site Scripting (XSS)
- A3: Broken Authentication and Session Management
- A4: Insecure Direct Object References
- A5: Cross-Site Request Forgery (CSRF)
- A6: Security Misconfiguration
- A7: Insecure Cryptographic Storage
- A8: Failure to Restrict URL Access
- A9: Insufficient Transport Layer Protection
- A10: Unvalidated Redirects and Forwards

Reference: http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Case Study

- **FBController** allows people to take complete control of third-party Facebook accounts
- It analyzes Facebook communication, and uses cookie data, to hijack accounts
- Information:
  - http://news.cnet.com/8301-1009_3-10234720-83.html

Lecture Roadmap

- HTTP Basics
- Cookies
- Security issues of cookies
- HTTPS

Web and HTTP

- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file,…
- Web page consists of base HTML-file which includes several referenced objects
- Each object is addressable by a URL
  - URL = Uniform Resource Locator

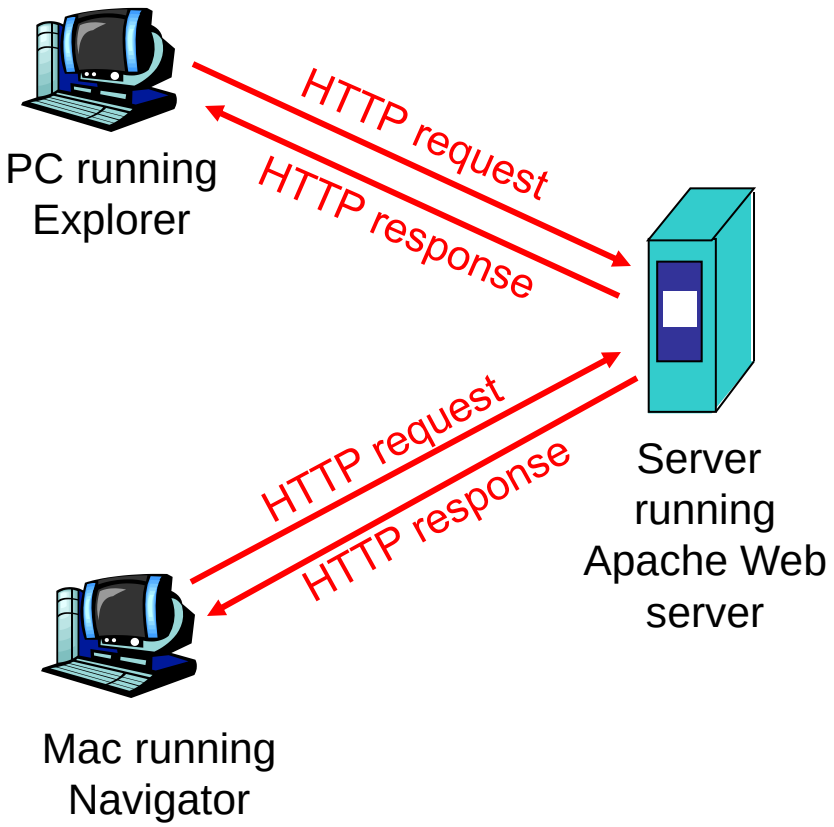
URLs

- URLs are global identifiers of network-retrievable documents
- Example:
  - http://www.cse.cuhk.edu.hk:81/class?name=csci5470#homework
    - Protocol: http
    - Hostname: www.cse.cuhk.edu.hk
    - Port: 81
    - Path: class
    - Query: name=csci5470
    - Fragment: homework

HTTP Overview

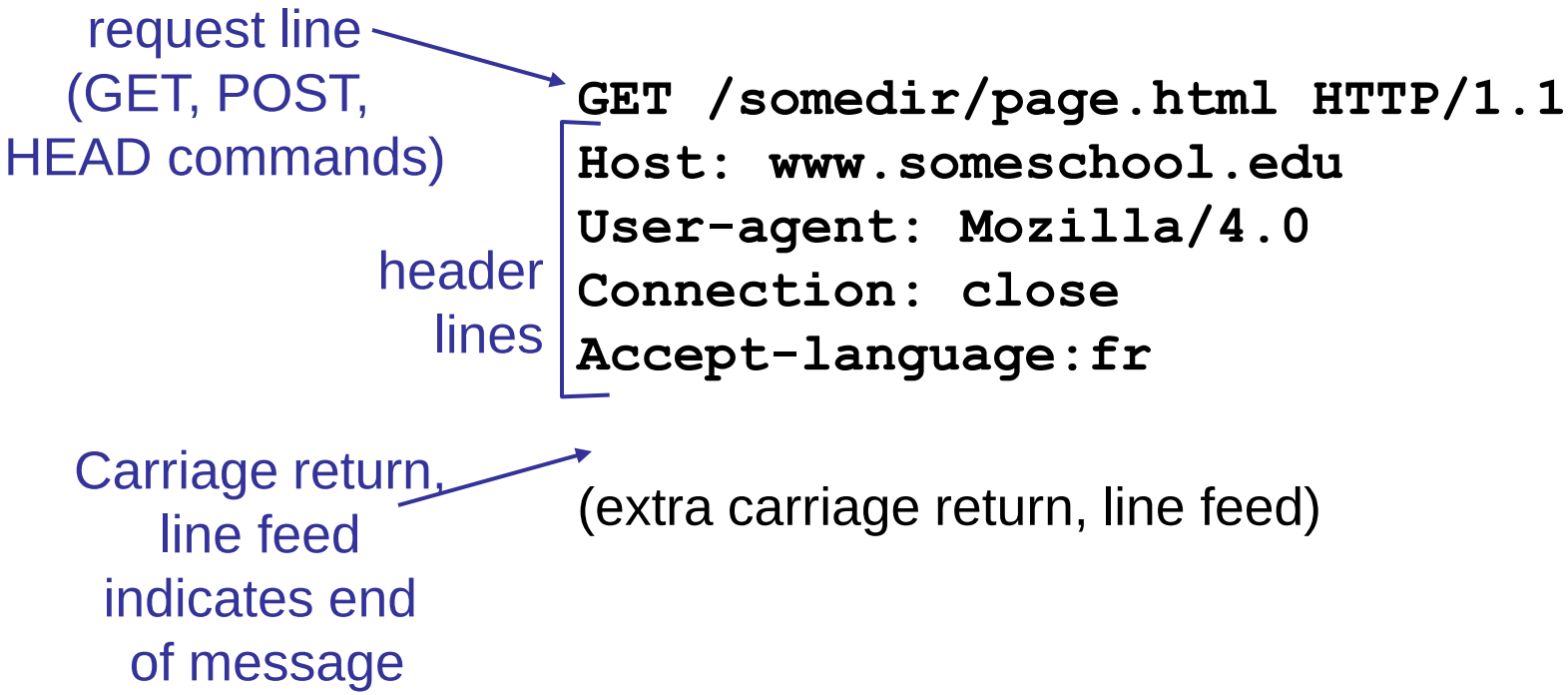
HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests

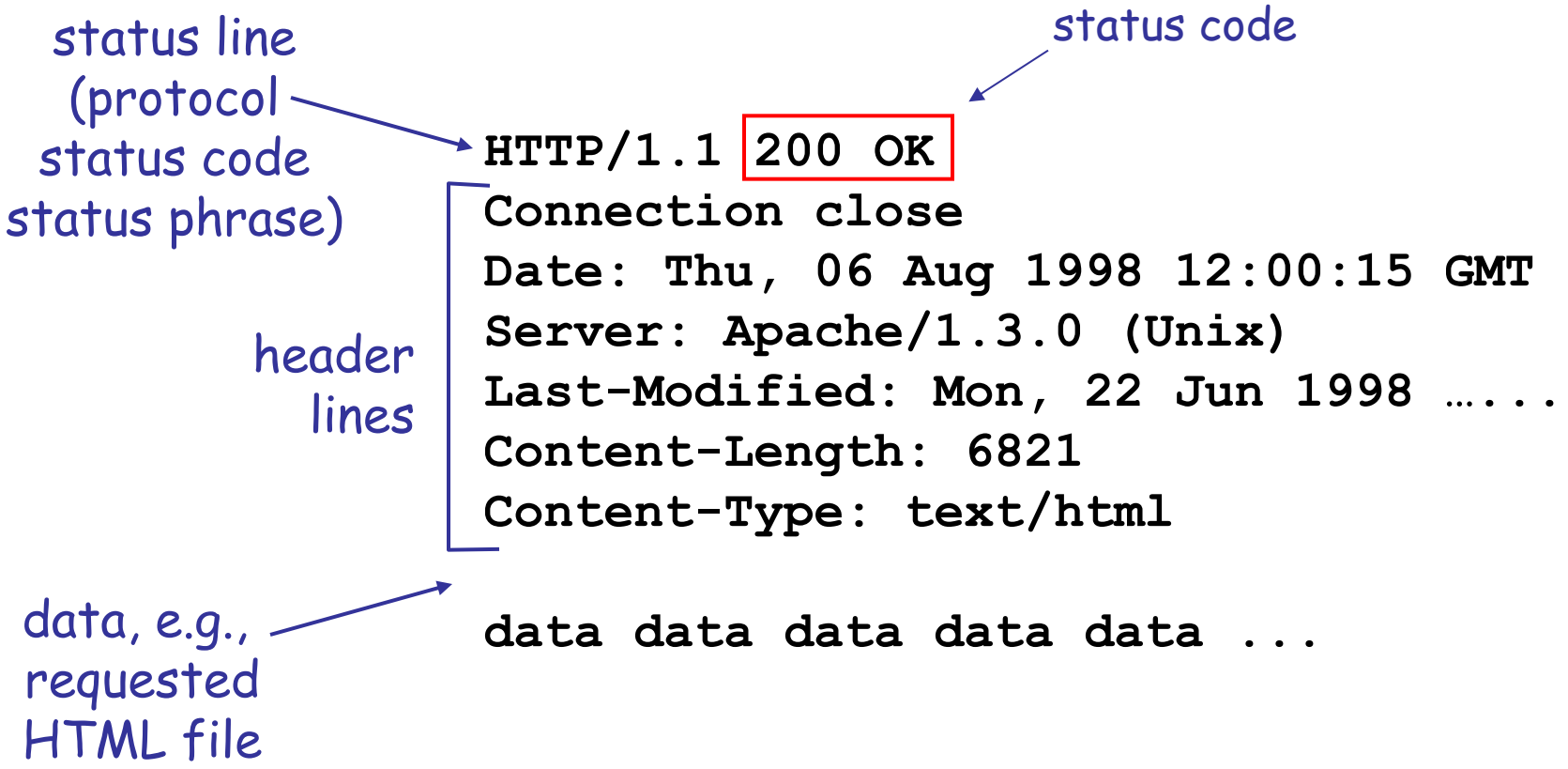


HTTP Request message

- two types of HTTP messages: *request*, *response*
- HTTP request message:
  - ASCII (human-readable format)



HTTP Response Message



Document Object Model (DOM)

- DOM is an object-based representation of HTML document layout in a hierarchical structure with various read/write properties and callable methods
- Examples:
  - Properties : document.alinkColor, document.URL, document.forms[]
  - Methods: document.write(document.referrer)

Same Origin Policy (SOP)

- SOP defines how documents and scripting languages to access DOM properties and methods across domains
  - Most important security concept within modern browsers
- SOP prevents a document or script loaded from one origin from getting or setting properties of a document from another origin
- **Origin = protocol://host:port**

Details discussed in lec8.pdf

Reference: https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript

Same Origin Policy (SOP)

Example:

URL	Outcome	Reason
http://store.company.com/dir2/other.html	Success	
http://store.company.com/dir/inner/another.html	Success	
https://store.company.com/secure.html	Failure	Different protocol
http://store.company.com:81/dir/etc.html	Failure	Different port
http://news.company.com/dir/other.html	Failure	Different host

Making HTTP Stateful

- HTTP is **stateless**
  - HTTP message exchanges do not store any states about HTTP requests. Server simply returns HTTP reply based on the HTTP request.
- **Session management**:
  - Allows web-based systems to create sessions so that users don't have to re-authenticate every time they wish to perform an action
  - In short, to maintain state
  - Three approaches: GET, POST, cookies

GET and POST

- Use GET and POST to maintain state of a particular user
- GET: encodes state info inside query string
  - http://somesite.com/admin.php?sessionID=12345678
    - admin.php determines if sessionID is valid. If so, authenticates the user
- POST: passes data to page through forms
  - e.g., includes a hidden field called sessionID that identifies a unique session.

Cookies

- **Cookies** are tokens of HTTP state
  - Tokens are in *name=value* format
- Persistent cookies: reside in browser's local disk storage (called cookie jar)
  - There's an expiration date that specifies how long the cookies are stored
- Session cookies: exist in memory and expire when browser closed

User-server interaction: cookies

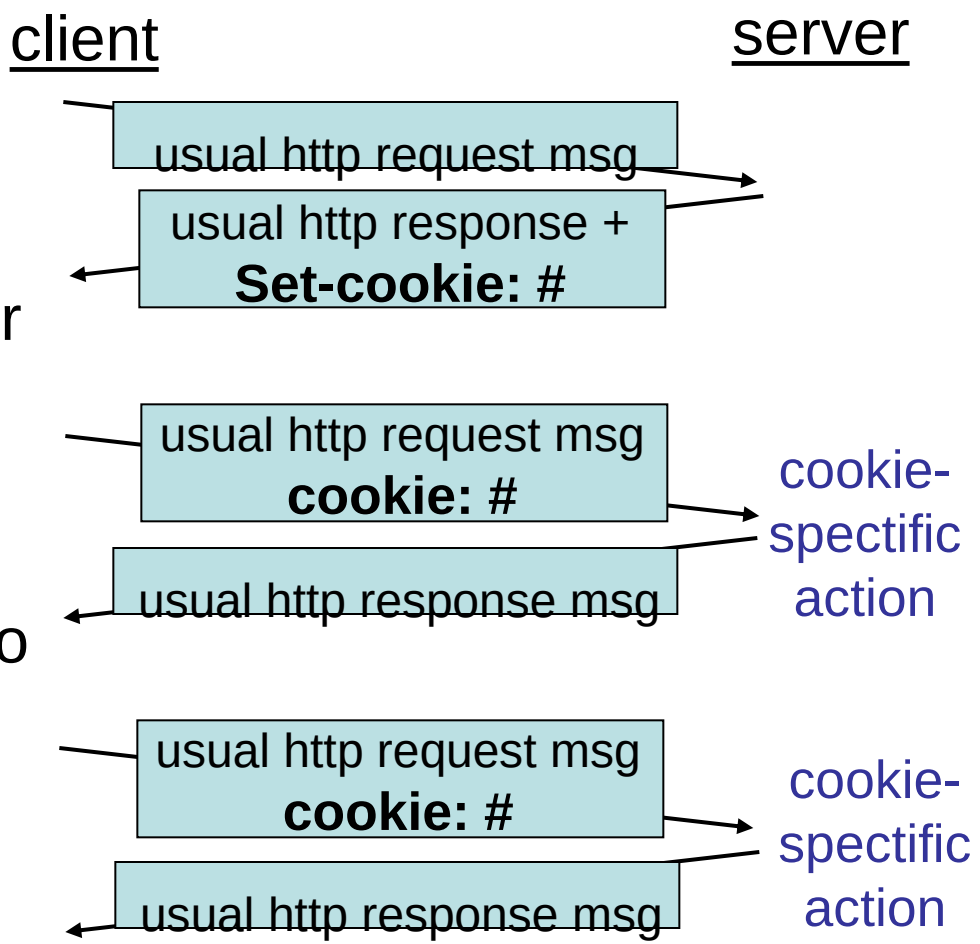
- server sends “cookie” to client in response msg

  `Set-cookie: sid=1678453`

- client presents cookie in later requests

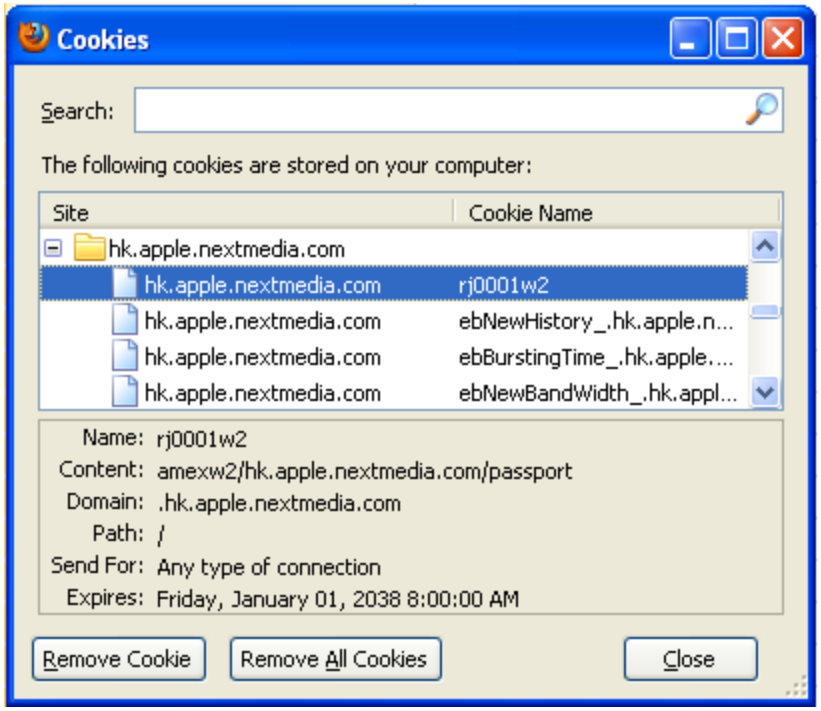
  `cookie: sid=1678453`

- server matches presented-cookie with server-stored info
  - authentication
  - remembering user preferences, previous choices



Find out your Cookies

- Find cookies inside browsers:
  - In Firefox, select Tools → Options → Privacy → Show Cookies



Find out your Cookies

- Find cookies in transmissions
  - Install Firefox plugin LiveHttpHeaders
    - http://livehttpheaders.mozdev.org/
  - Display all HTTP headers in HTTP request and response messages (including cookies)

Find out your Cookies

- HTTP request (captured from LiveHTTPHeaders):

GET / HTTP/1.1
Host: www.google.com.hk
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.10) Gecko/20100914 Firefox/3.6.10 (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Cookie: PREF=ID=0a27b5c6ff8e2e8d:U=f84ebc85b045306d:LD=en:NR=10:TM=1225854781:LM=1285852043:DV=sdw-RZjl_scD:IG=1:SG=0:S=d2JnwJbTtjmpIzQ2; ID=39=O4fR5STWLWxn21qdmUtvrzPYRegO8_GAG1Jg-pgVu3we8icnFt9SX9zPvf5NSR0ZEisoA9j1CCQyeMWHASttQ7o4GW8alJfnbfTFRWjvTLelXM_296pWoZ2BDQeC41Qk; SID=DQAAAJ0AAAD8WklD_wPyI6L8qNdZncbyt7-wxhPTDJ3eeThQuyHXJuTh-KoCAzuVemRh7RXJ0lpWlhH2dJuuLA-J8x2SkbG1Ycp-3REmCG8unrob5AdREiSUOvNazKR0_yX_rWpHlvBLhvhZQkGm_5Z4kp06UD9xHgxpGFz6JoHsktF9PtK0TTXO-cOoSKM5bkepSGPrmELSANYO_Si6LADIBHPo2CZO; HSID=AYwRtsQmndXrE7Uy2

Find out your Cookies

- HTTP response (captured from LiveHTTPHeaders):

```
HTTP/1.1 200 OK
Date: Thu, 30 Sep 2010 13:07:47 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Set-Cookie: PREF=ID=0a27b5c6ff8e2e8d:U=f84ebc85b045306d:LD=en:NR=10:TM=1225854781:LM=1285852067:DV=sdw-
RZjl_scD:IG=1:SG=0:S=jbTDREIcPCJvl5xO; expires=Sat, 29-Sep-2012 13:07:47 GMT; path=/; domain=.google.com.hk
Content-Encoding: gzip
Server: gws
Content-Length: 4878
X-XSS-Protection: 1; mode=block
```

Lecture Roadmap

- HTTP Basics
- Cookies
- Security issues of cookies
- HTTPS

Cookie Scope

- Cookie scope
  - limits cookie access to specified sites
  - enforced by web browsers
  - specified by domain and path
- Example – a server replies a HTTP response and specifies the cookies:
  - Set-Cookie: sid=123; **path=/; domain=.facebook.com**; expires=Mon, 24-Sep-2012 14:04:50 GMT

Cookie Scope

- domain
  - domain=.facebook.com means that only facebook.com and sub-domains (e.g., abc.facebook.com) would have permission to access and modify the cookies
  - That is, a client can retrieve and use the cookies only when s/he visits facebook.com (or sub-domains)
- path
  - specifies a directory under which the cookie is active
    - e.g., path=/cgi-bin
  - path=/ means cookies are valid in entire domain

Cookie Scope

- Cookies are identified by (name, domain, path)
- Two distinct cookies (same name, but different domains):
  - userid=test, domain=login.site.com, path=/
  - userid=test123, domain=.site.com, path=/
- Both cookies are stored inside browser.
- Both cookies are in scope login.site.com
- Many (if not all) browsers treat .site.com and site.com as the same

Setting Cookies by Server

- Server can set cookies on client in http-response
  - **`Set-Cookie: abc=123; domain=.login.com`**
- What are the domain and path that can be set by the server?
  - Any domain-suffix of URL hostname, except top-level domain (e.g., .com, .hk)
  - path can be set to anything
- Example: host = “login.site.com”
  - Allowed domains: login.site.com, .site.com
  - Disallowed domains: user.site.com, other.com, .com

Cookie Attributes by Server

- Server can set cookie attributes
  - e.g., domain, path, expires
  - secure: tells browser to send cookies only over a secure (e.g., HTTPS) connection
  - httponly: tells browser not to let Javascript read cookie values
  - Example:
    - `Set-Cookie: abc=11223344; expires=Mon, 25-Oct-2010 14:04:50 GMT; path=/; domain=.facebook.com; httponly`
- secure and httponly attributes provide some security protection for cookies (but not perfect)

Cookie Attributes: Default

- If the attributes are not set:
  - domain: the host name of the page setting the cookie
  - path: path (or current directory) of the URL that sent the Set-Cookie header
  - expires: session cookie
- Example: Cookies sent by http://www.google.com/accounts/a.html
  - Default domain = www.google.com, default path = /accounts/, default expires: end of session

Reading Cookies by Server

- Server can retrieve cookies from client in http-request
  - **`GET /abc/a.html`**
    `Host: login.site.com`
    `Cookie: sid=1234567`
- Browser sends **all** cookies within scope
  - cookie domain: domain-suffix of URL domain
  - cookie path: prefix of URL path
- Idea: server only seeks cookies in its scope

Accessing Cookies by Client

- In Javascript, use document.cookie to access cookies
- Setting cookies:
  - document.cookie = “name=value; expires=..”
- Reading cookies:
  - alert(document.cookie); // print cookie string
- Deleting cookies:
  - document.cookie = “name=; expires=Thu, 01-Jan-70”;

javascript:alert(document.cookie)

Lecture Roadmap

- HTTP Basics
- Cookies
- Security issues of cookies
- HTTPS

Security Issues of Cookies

- HTTP messages are sent in plain, so are cookies.
  - Confidentiality: Attackers can read session state
  - Integrity: Attackers can modify session state
  - Replay: Attackers replay cookies (e.g., authentication credentials)
- Can HTTPS solve the above problems?
  - Will come back later.

Cookie Guessing Attacks

- Cookies may contain authentication credentials
  - sessionid=10017
- If session ID has small entropy, an attacker can guess a session ID (e.g., sessionid=10032) and generate a new authentication session
- In general, attacker can guess the session ID via brute force

Cookie Discovery Attacks

- Since HTTP is sent in plain, an attacker can discover cookie values by sniffing network traffic
- Cross-site scripting (XSS)
  - Steals cookies by inserting scripts inside HTML pages, and send cookies to attacker
    - Will be discussed next lecture
- DNS poisoning
  - Poison DNS cache to map domain name to attacker's IP address
  - trick browser to send cookies to attacker's IP address
    - Cookie scope is defined by domain name, not by IP address

Cookie Setting Attacks

- Attacker can set a user's session cookies to a value the attacker can control.
- **Session fixation** attack:
  - the attacker fixes the user's session ID before the user even logs into the target server, thereby eliminating the need to obtain the user's session ID afterwards.

How Session Fixation Works?

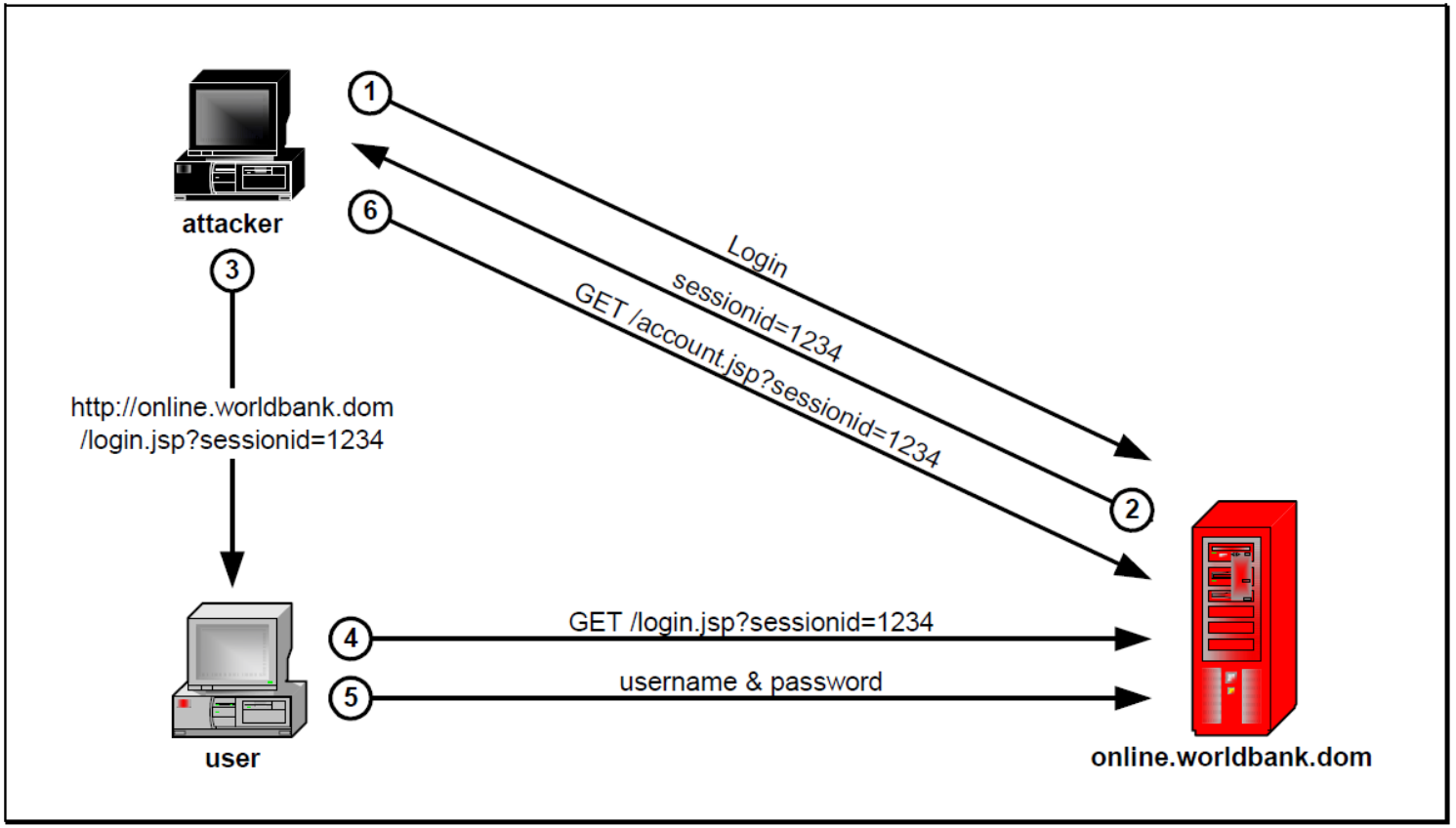


Figure 1: Simple session fixation in an URL-based web banking system

Source: Mitja Kolsek, "Session Fixation Vulnerability in Web-based Application", 2002

How Session Fixation Works?

- Attacker – who in this case is also a legitimate user of the system – logs in to the server (**1**) and is issued a session ID 1234 (**2**).
- Attacker sends a hyperlink http://online.worldbank.dom/login.jsp?sessionid=1234 to the user, trying to lure him into clicking on it (**3**).
- The user clicks on the link, which opens the server's login page in his browser (**4**).

How Session Fixation Works?

- Note that upon receipt of the request for login.jsp?sessionid=1234, the web application has established that a session for 1234 already exists for this user and a new one need not be created.
- Finally, the user provides his credentials to the login script (**5**) and the server grants him access to his bank account. However, at this point, knowing the session ID, the attacker can also access the user's account via account.jsp?sessionid=1234 (**6**)

Exploitation Example 1

- Alice logs in at login.site.com
  - login.site.com sets sessionid=alice with domain=.site.com
- Alice visits evil.site.com
  - evil.site.com sets sessionid=evil with domain=.site.com
- Alice visits csci5470.site.com to submit homework
  - csci5470.site.com thinks it's talking to evil
  - csci5470 cannot tell that sessionid is overwritten

Exploitation Example 2

- Attacker can inject into HTTP response that is sent to Alice:
  - Set-Cookie: sessionid=evil; secure
- Later, if Alice uses HTTPS, she uses sessionid=evil for its sessionid
- Even “secure” cookies can be compromised

Exploitation Example 3: Interaction with DOM SOP

- Recall that SOP for DOM:
  - Origin A can access origin's B DOM if they match on (protocol, domain, port)
- Cookie scope:
  - Based on (domain, path)
  - http://domain/path can only access cookies within scope
- SOP for DOM make things complicated…

Exploitation Example 3: Interaction with DOM SOP

- By cookie scope:
  - x.com/A cannot see cookies of x.com/B
- DOM SOP:
  - x.com/A has access to DOM of x.com/B
  - <!-- in x.com/A -->
    <iframe src="x.com/B"></iframe>
    alert(frames[0].document.cookie)
- Path separation is done for efficiency not security:
  - Only cookies under the path is sent

Exploitation Example 4

- User can change and delete cookie values stored inside browser
- Example:
  - Set-cookie: cart-total = 150
  - Cookie: cart-total = 15
- In short, cookies have no integrity checks

Cryptographic Checksums

- Integrity of cookies
- Server → Browser
  - Server generates **tag** T = F(k, value), where k is the secret key of server
  - Set-cookie: NAME=value+T
- Browser → Server
  - Cookie: NAME=value+T
  - Server verifies if T = F(k, value)
- Drawback:
  - Increase transmission overhead, as every name=value needs a tag

Lecture Roadmap

- HTTP Basics
- Cookies
- Security issues of cookies
- HTTPS

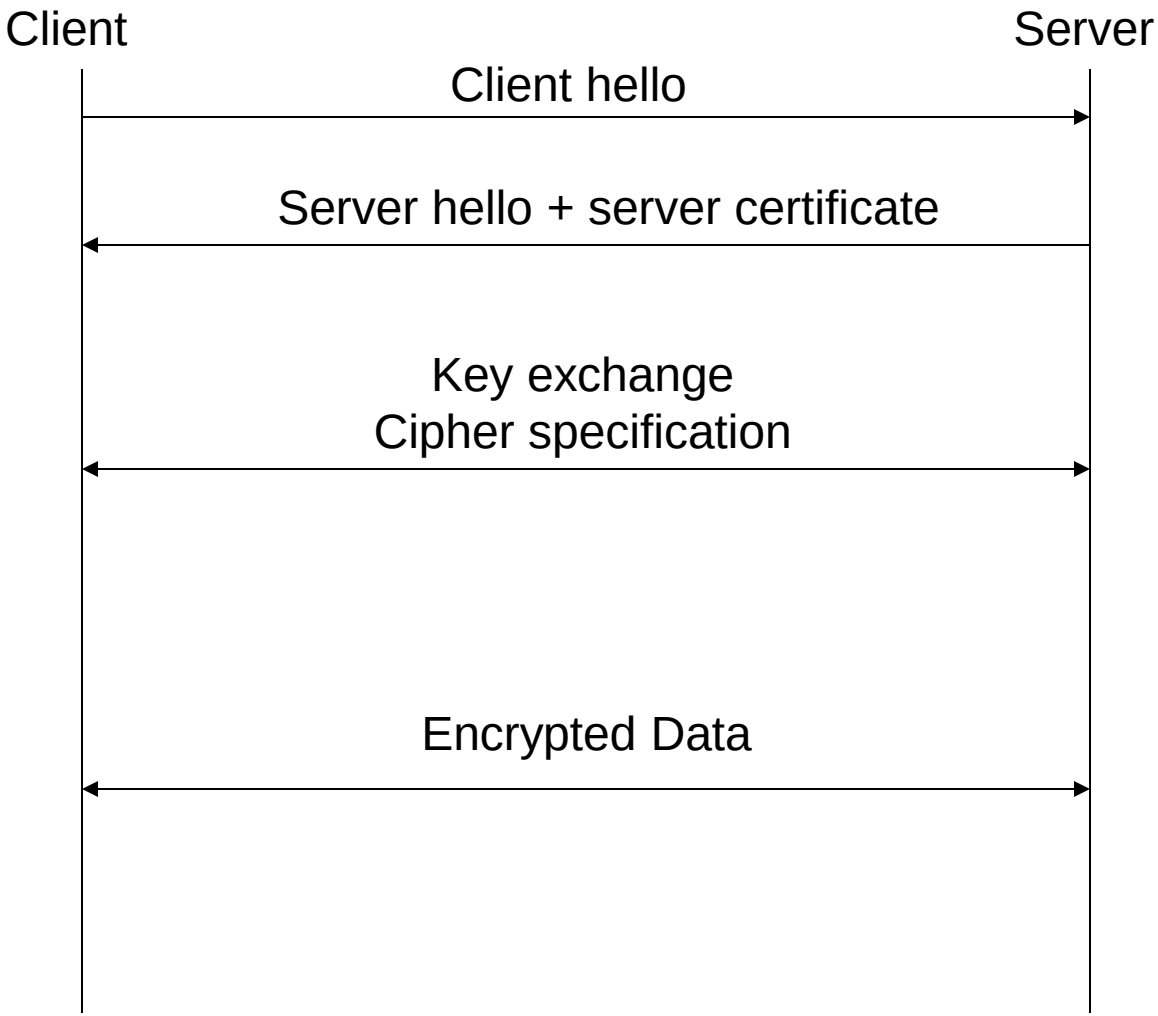
HTTPS

- HTTPS (HTTP with TLS) encrypts everything, including cookies, HTTP header, and HTTP message content
- Features of HTTPS:
  - Server authentication
  - Integrity protection
  - Confidentiality protection
  - Optional client authentication
- Use https:// instead of http://
  - Use port 443 instead of port 80

SSL/TLS

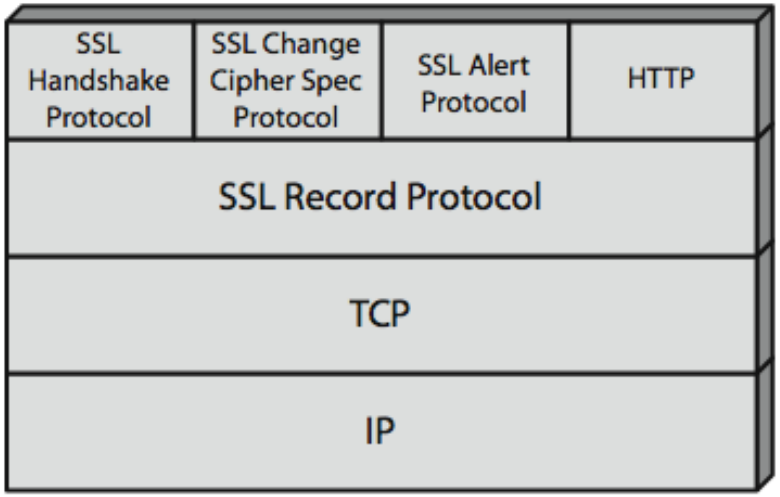
- HTTPS is built atop SSL
- SSL (Secure Socket Layer) provides a transport layer security service
  - Now is version 3
- SSL subsequently became Internet standard known as TLS (Transport Layer Security)
- uses TCP to provide a reliable end-to-end service
- SSL is composed of two layers (shown later)

SSL/TLS


Client
Server
Client hello
Server hello + server certificate
Key exchange
Cipher specification
Encrypted Data

SSL Architecture

- SSL Record Protocol
  - Protect application data (encrypt and authenticate)
- SSL management
  - SSL handshake
  - SSL change cipher spec
  - SSL alert



Is HTTPS Perfect?

- HTTPS provides a baseline level of security, but it's not unbeatable
- HTTPS has a high performance overhead, and it also breaks web caching
- Websites usually use a mix of HTTP and HTTPS
  - open door for attacks
- Use "secure" attribute in cookies to tell browser to "only send the cookies over the network using HTTPS, not HTTP"

HTTPS Has Been Hacked!

- HTTPS is found to be vulnerable to chosen-plaintext attack
  - Attack applies to SSL 3.0 and TLS 1.0
- Theoretical attack was documented 10 years ago, but made real recently

Red alert: HTTPS has been hacked, http://www.infoworld.com/print/174025, Sep 2011
T. Duong, J. Rizzo, “Here Come the ⊕Ninjas”, 2011

HTTPS Has Been Hacked!

- In SSL, plaintext is split into records (blocks)
- Exploit the CBC mode operation with chained initialization vectors (IVs), i.e., IV is simply the last encryption of the previous ciphertext
  - e.g., $C_j = E(C_{j-1} + P_j)$
  - C_{j-1} is the IV being used here

HTTPS Has Been Hacked!

- Suppose the attacker suspects some previous plaintext block $P_i = x$
- Choose P_j ($j > i$) s.t. $P_j = C_{j-1} + C_{i-1} + x$. Then
  - $C_j = E(P_j + C_{j-1})$
    $= E(P_i + C_{i-1})$
    $= C_i$
  - If $C_j = C_i$, then the attacker confirms $P_i = x$
- Defense: randomize IVs for each block

Conclusions

- HTTP Session Management include state information into stateless HTTP
- Cookies are the main component of state information, and need to be protected properly
- Next week:
  - More on session hijacking attacks

References

- Required Readings:
  - Chris Palmer, Secure Session Management for Web Application (and Presentation), 2008
  - Luke Murphey, “Secure Session Management: Preventing Security Voids in Web Application”, 2005
- Optional:
  - Mitja Kolsek, “Session Fixation Vulnerability in Web-based Application”, 2002
  - Browser Security Handbook, part 2 http://code.google.com/p/browsersec/wiki/Part2
  - Stallings, Ch. 16: “Web Security”