Session Management

Dan Boneh

Sessions

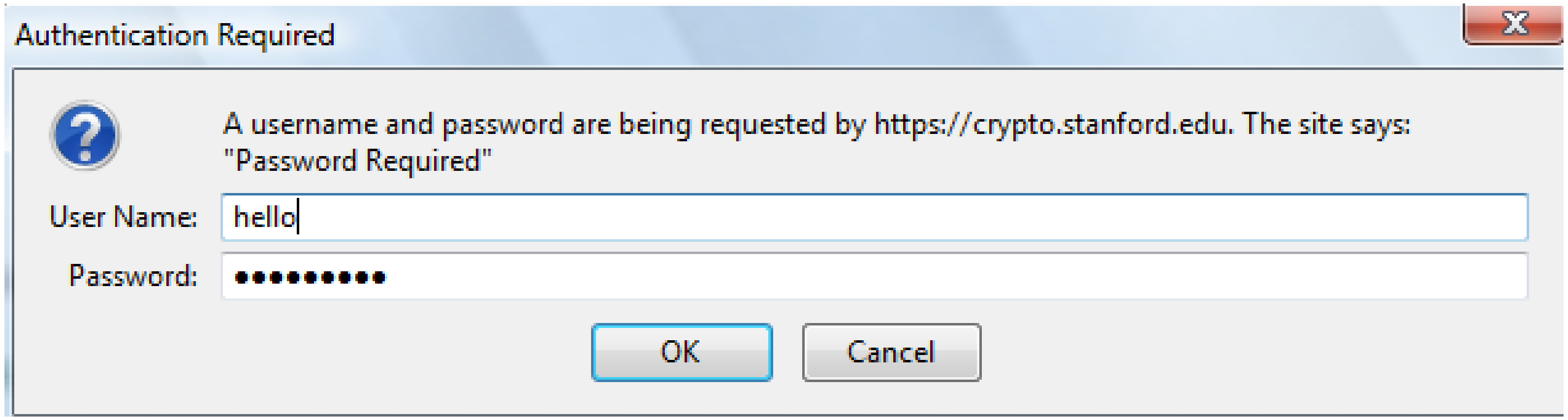
- A sequence of requests and responses from one browser to one (or more) sites
  - Session can be long (Gmail - two weeks)
    or short
  - without session mgmt:
    users would have to constantly re-authenticate
- Session mgmt:
  - Authorize user once;
  - All subsequent requests are tied to user

Pre-history: HTTP auth

HTTP request: GET /index.html

HTTP response contains:

WWW-Authenticate: Basic realm="Password Required“



Authentication Required

A username and password are being requested by https://crypto.stanford.edu. The site says: "Password Required"

User Name: hello

Password: ●●●●●●●●●●

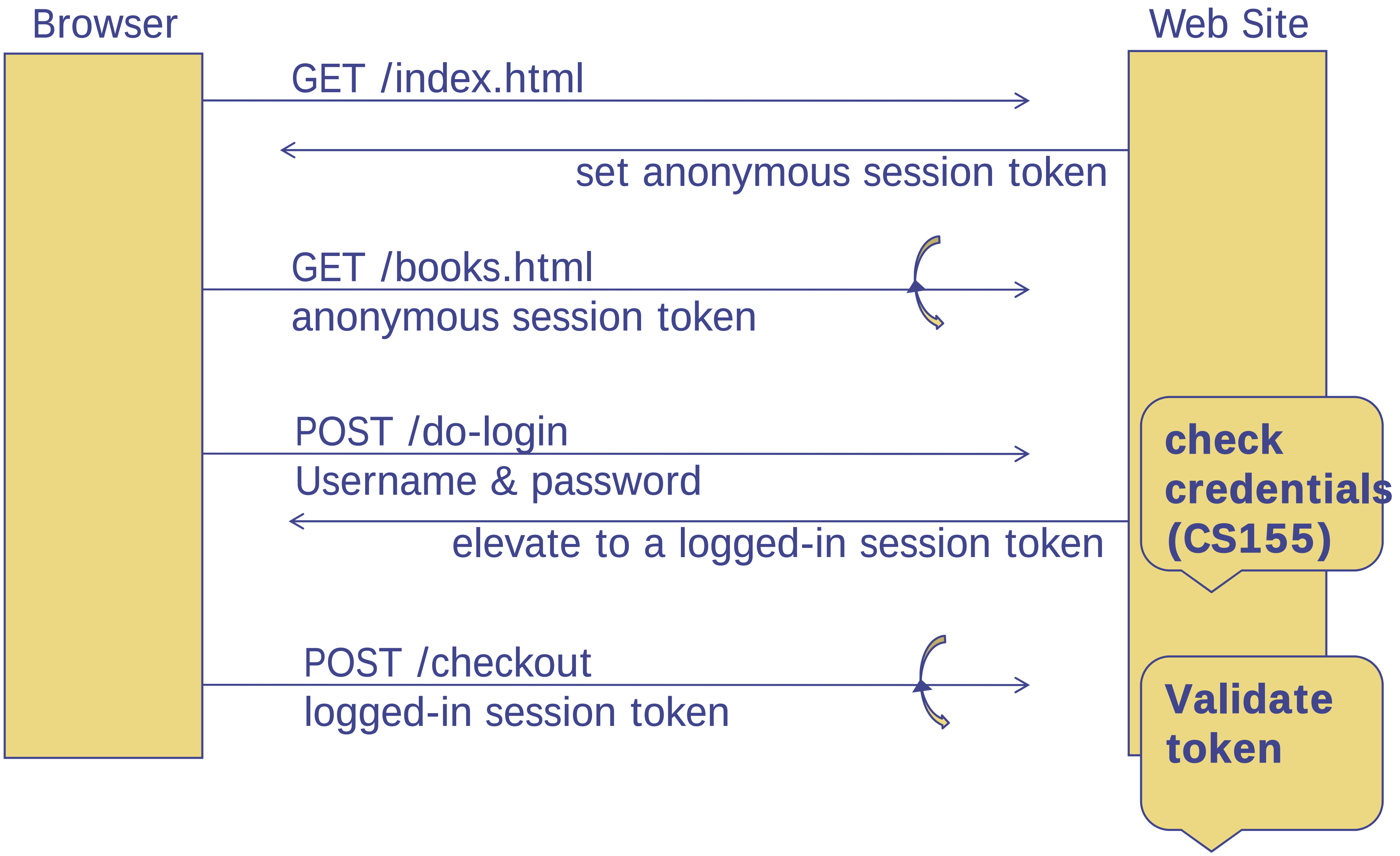
Browsers sends hashed password on all subsequent HTTP requests:

Authorization: Basic ZGFddfibzsdfgkjheczI1NXRleHQ=

HTTP auth problems

- Hardly used in commercial sites
  - User cannot log out other than by closing browser
    - What if user has multiple accounts?
    - What if multiple users on same computer?
  - Site cannot customize password dialog
  - Confusing dialog to users
  - Easily spoofed

Session tokens



Storing session tokens:

Lots of options (but none are perfect)

- Browser cookie:

  Set-Cookie: SessionToken=fduhye63sfdb

- Embedd in all URL links:

  https://site.com/checkout ? SessionToken=kh7y3b

- In a hidden form field:

  <input type="hidden" name="sessionid" value="kh7y3b">

- Window.name DOM property

Storing session tokens: problems

- Browser cookie:

  browser sends cookie with every request,
  even when it should not (CSRF)

- Embed in all URL links:

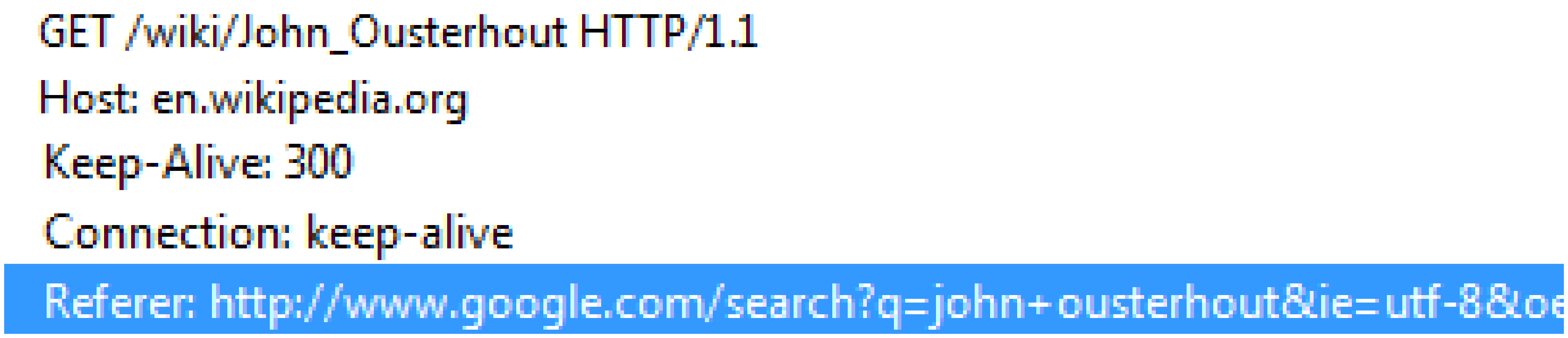
  token leaks via HTTP Referer header

- In a hidden form field: short sessions only

Best answer: a combination of all of the above.

why? next lecture.

The HTTP referer header



```
GET /wiki/John_Ousterhout HTTP/1.1
Host: en.wikipedia.org
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.google.com/search?q=john+ousterhout&ie=utf-8&oe
```

Referer leaks URL session token to 3rd parties

SESSION HIJACKING

Attacker waits for user to login;

then attacker obtains user’s Session Token and “hijacks” session

1. Predictable tokens

- Example: counter (Verizon Wireless)

  ⇒ user logs in, gets counter value, can view sessions of other users

- Example: weak MAC (WSJ)
  - token = **{userid, MAC$_k$(userid) }**
  - Weak MAC exposes **k** from few cookies.

Session tokens must be unpredicatble to attacker:

Use underlying framework.

Rails: token = MD5(current time, random nonce)

2. Cookie theft

- Example 1: login over SSL, but subsequent HTTP
  - What happens as wireless Café ?
  - Other reasons why session token sent in the clear:
    - HTTPS/HTTP mixed content pages at site
    - Man-in-the-middle attacks on SSL

- Example 2: Cross Site Scripting (XSS) exploits

- Amplified by poor logout procedures:
  - Logout must invalidate token on server

Session fixation attacks

- Suppose attacker can set the user's session token:
  - For URL tokens, trick user into clicking on URL
  - For cookie tokens, set using XSS exploits

- Attack: (say, using URL tokens)
  1. Attacker gets anonymous session token for site.com
  2. Sends URL to user with attacker's session token
  3. User clicks on URL and logs into site.com
     - this elevates attacker's token to logged-in token
  4. Attacker uses elevated token to hijack user's session.

Session fixation: lesson

- When elevating user from anonymous to logged-in,

  always issue a new session token

- Once user logs in, token changes to value unknown to attacker.

  ⇒ Attacker’s token is not elevated.

Generating session tokens

Goal: prevent hijacking and avoid fixation

Option 1: minimal client-side state

- SessionToken = [random unpredictable string]

  (no data embedded in token)

  - Server stores all data associated to SessionToken:

    userid, login-status, login-time, etc.

- Can result in server overhead:
  - When multiple web servers at site,
    lots of database lookups to retrieve user state.

Option 2: lots of client-side state

- SessionToken:

  SID = **[userID, exp. time, data]**

  where data = (capabilities, user data, …)

  SessionToken = **Enc-then-MAC (k, SID)**

  (as in CS255)

  k: key known to all web servers in site.

- Server must still maintain some user state:
  - e.g. logout status (should be checked on every request)

- Note that nothing binds SID to client's machine

Binding SessionToken to client's computer; mitigating cookie theft

approach: embed machine specific data in SID

- **Client IP Address**:
  - Will make it harder to use token at another machine
  - But honest client may change IP addr during session
    - client will be logged out for no reason.

- **Client user agent**:
  - A weak defense against theft, but doesn't hurt.

- **SSL session key**:
  - Same problem as IP address (and even worse)

MORE ON CROSS SITE SCRIPTING (XSS)

Recall: reflected XSS

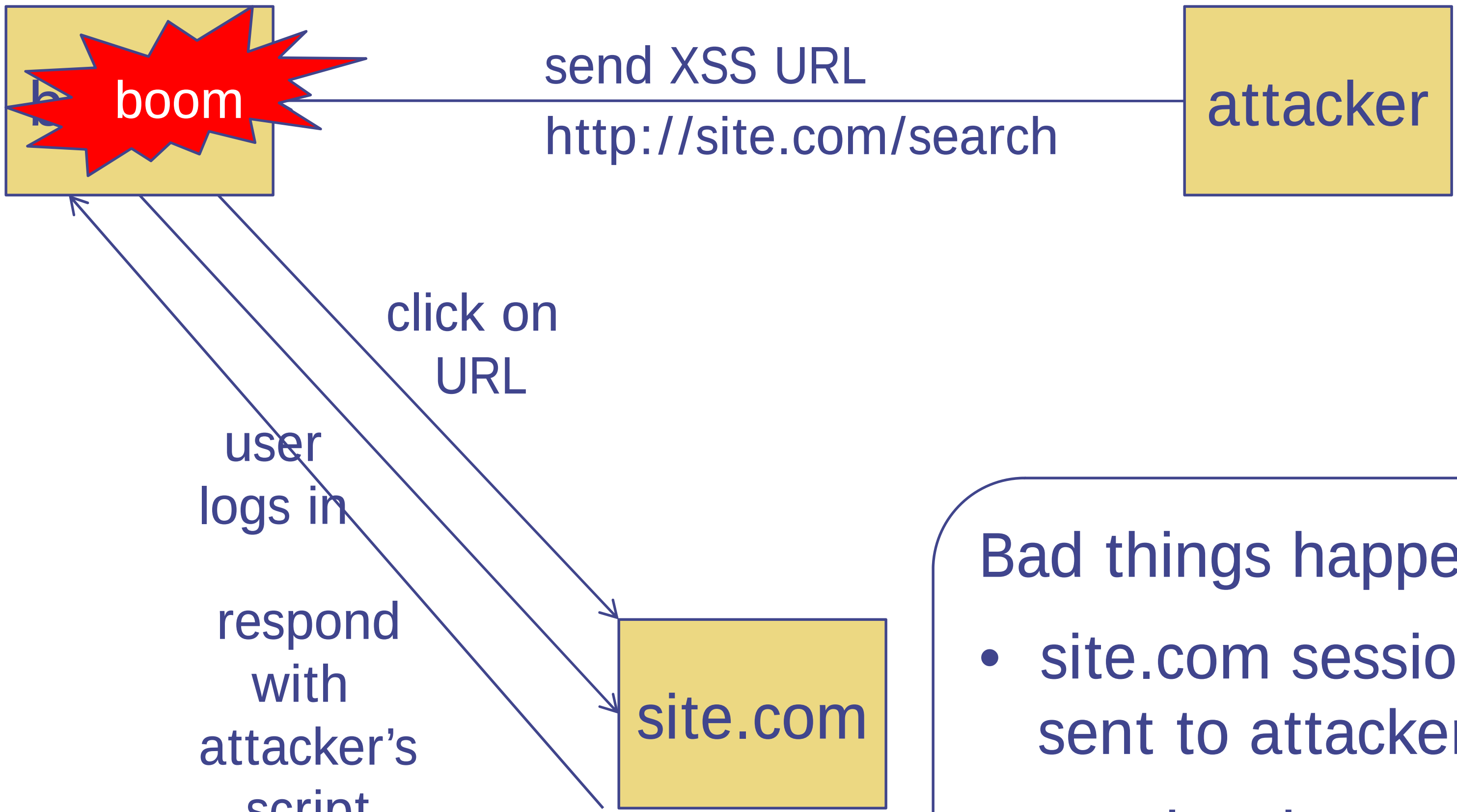
- search field on victim.com:
  - **http://victim.com/search.php ? term = apple**
- Server-side implementation of **search.php**:
  - Echo search term directly into HTML response

(no filtering of user input)

- To exploit, attacker crafts a URL containing a script

```
http://victim.com/search.php ? term =
     <script> do_something_bad  </script>
```

Reflected XSS: the exploit



Bad things happen:

- site.com session token sent to attacker
- rewrite site.com DOM
- rewrite links and persist

Persistent XSS

- XSS script is injected into blog, message board, etc.
  - When user’s view the block, the malicious script runs in their browser
  - ⇒ blogs must filter uploaded content

- The famous MySpace Samy worm: (2005)
  - Bypassed MySpace script filters
  - Script spread from user to user making everyone Samy’s friend

http://namb.la/popular/tech.html

Persistent XSS using images

Suppose pic.jpg on web server contains HTML !

- request for http://site.com/pic.jpg results in:

```
HTTP/1.1  200 OK
...
Content-Type:  image/jpeg

<html>  fooled ya  </html>
```

- IE will render this as HTML (despite Content-Type)

- Consider photo sharing sites that support image uploads
  - What if attacker uploads an “image” that is a script?

Universal XSS

- Adobe PDF viewer “feature” : (version <= 7.9)

  http://site.com/abc.pdf # whatever=javascript: --- code –

  viewer will execute the javascript in origin of current domain!

- Any site that hosts a single PDF is vulnerable to XSS !

  (in fact, PDF files in Windows can also be used)