Web Security: Session management and CSRF

CS 161: Computer Security

Prof. Raluca Ada Popa

February 10, 2016

Credit: some slides are adapted from previous offerings of this course and from CS 241 of Prof. Dan Boneh

Announcements

- Project 1 due Feb 16 11:59pm
- Instructors' office hours
  - David: Wed 4-5pm and Fri 1-2pm in 733 Soda
  - Raluca: Fri 3-5pm in 729 Soda

HTTP is mostly stateless

- Apps do not typically store persistent state in client browsers
  - User should be able to login from any browser
- Web application servers are generally "stateless":
  - Most web server applications maintain no information in memory from request to request
    - Information typically store in databases
  - Each HTTP request is independent; server can't tell if 2 requests came from the same browser or user.
- Statelessness not always convenient for application developers: need to tie together a series of requests from the same user

HTTP cookies

Outrageous Chocolate Chip Cookies

1676 reviews

Made 321 times

Recipe by: Joan

"A great combination of chocolate chips, oatmeal, and peanut butter."

Save | I Made it | Rate it | Share | Print

Ingredients

25 m | 18 servings | 207 cals

- 1/2 cup butter
- 1/2 cup white sugar
- 1/3 cup packed brown sugar
- 1 cup all-purpose flour
- 1 teaspoon baking soda
- 1/4 teaspoon salt
- 1/2 cup rolled oats
- 1 cup semisweet chocolate chips

On Sale

On

What's on sale near you.

TARGET

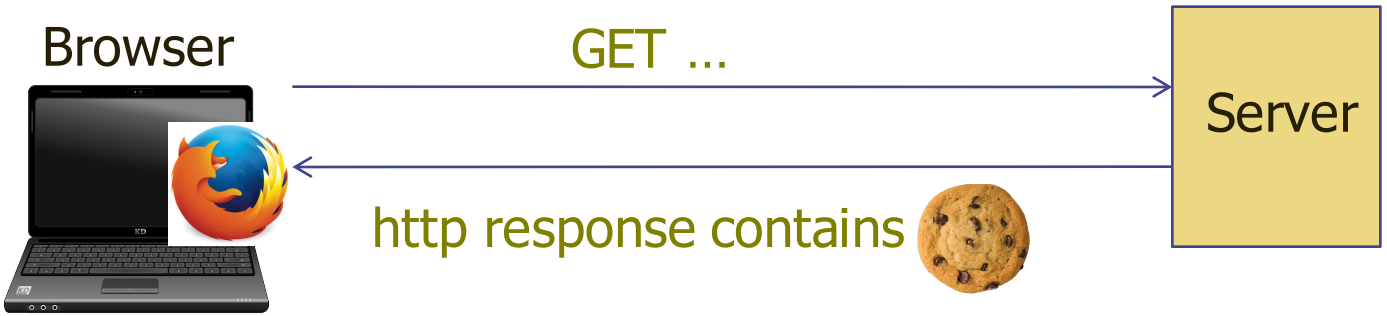
Target
1057 Eastshore Hwy
ALBANY, CA 94710
Sponsored

May we suggest

These nearby stores have ingredients on sale!

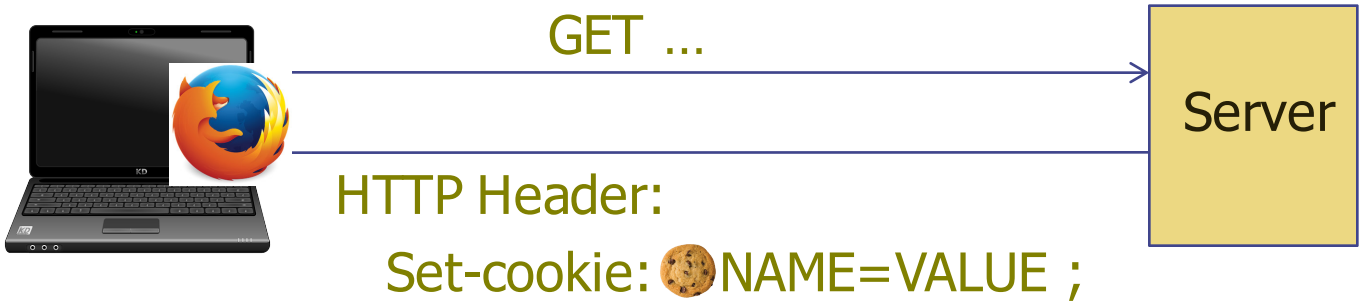
Cookies

- A way of maintaining state



Browser maintains cookie jar

Setting/deleting cookies by server



- The first time a browser connects to a particular web server, it has no cookies for that web server
- When the web server responds, it includes a **Set-Cookie:** header that defines a cookie
- Each cookie is just a name-value pair

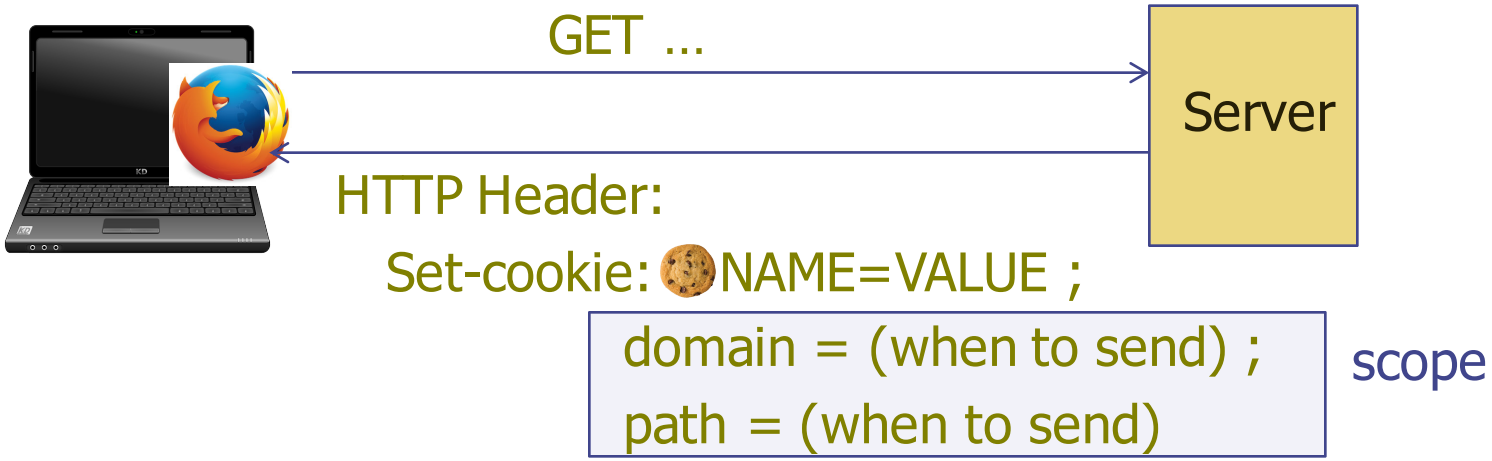
View a cookie

In a web console (firefox, tool->web developer->web console), type

document.cookie

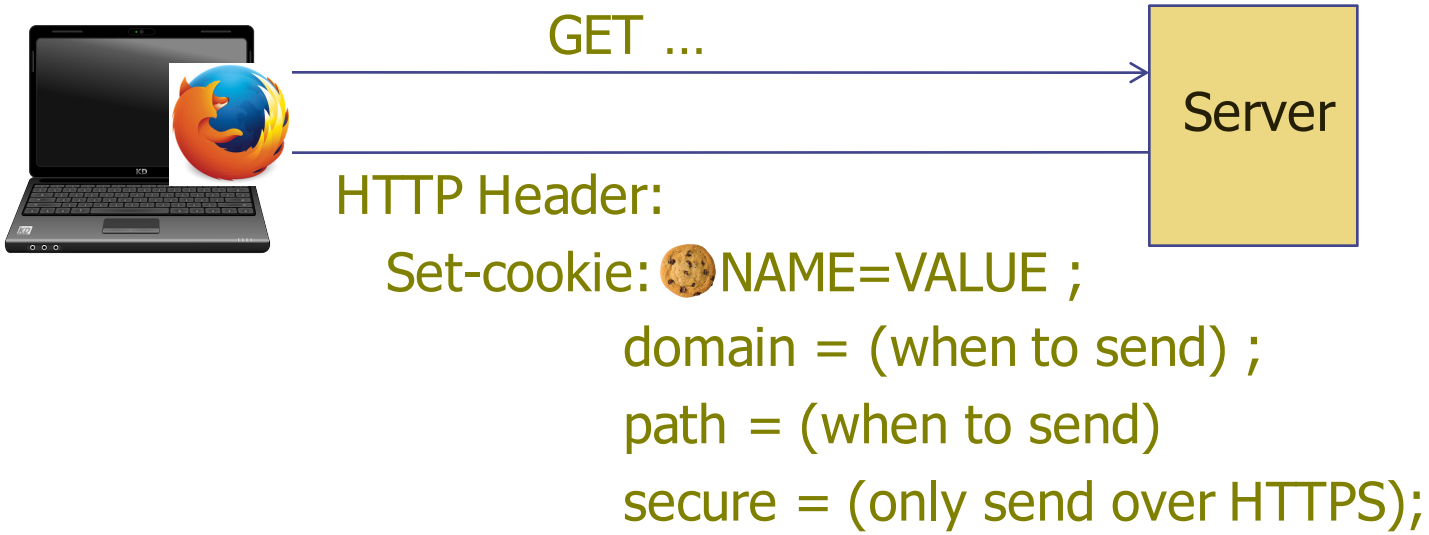
to see the cookie for that site

Cookie scope



- When the browser connects to the same server later, it includes a Cookie: header containing the name and value, which the server can use to connect related requests.
- Domain and path inform the browser about which sites to send this cookie to

Cookie scope



- Secure: sent over https only
  - https provides secure communication (privacy and integrity) – we'll see later in course

Cookie scope

GET …

Server

HTTP Header:

Set-cookie: NAME=VALUE ;

domain = (when to send) ;

path = (when to send)

scope

secure = (only send over SSL);

expires = (when expires) ;

HttpOnly

- Expires is expiration date
  - Delete cookie by setting "expires" to date in past
- HttpOnly: cookie cannot be accessed by Javascript, but only sent by browser

Cookie scope

- Scope of cookie might not be the same as the URL-host name of the web server setting it

Rules on:

1. What scopes a URL-host name is allowed to set
2. When a cookie is sent to a URL

What scope a server may set for a cookie

domain: any domain-suffix of URL-hostname, except TLD

[top-level domains, e.g. '.com']

example: host = "login.site.com"

allowed domains	disallowed domains
login.site.com	**user.site.com**
.site.com	**othersite.com**
	.com

⇒ **login.site.com** can set cookies for all of **.site.com** but not for another site or TLD

Problematic for sites like .berkeley.edu

path: can be set to anything

Examples

Web server at foo.example.com wants to set cookie with domain:

domain	Where it will be sent
(value omitted)	*foo.example.com* (exact)
bar.foo.example.com	
foo.example.com	**.foo.example.com*
baz.example.com	
example.com	
ample.com	
.com	

Credits: The Tangled Web: A Guide to Securing Modern Web Applications, by Michał Zalewski

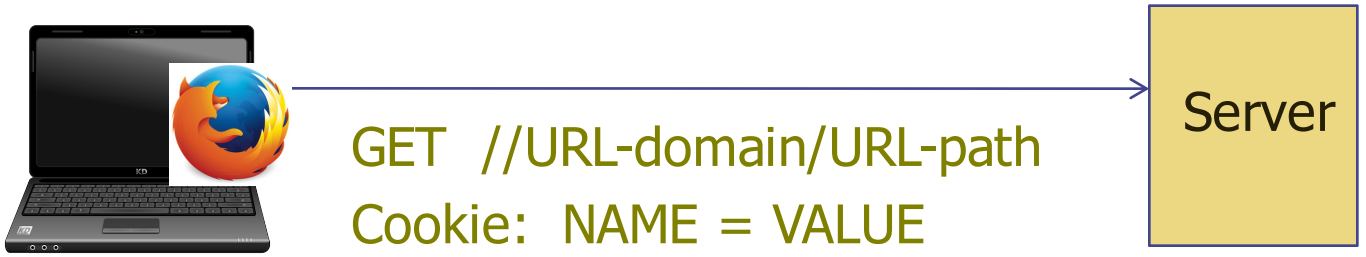
Examples

Web server at foo.example.com wants to set cookie with domain:

domain	Where it will be sent
(value omitted)	*foo.example.com* (exact)
bar.foo.example.com	Cookie not set: domain more specific than origin
foo.example.com	**.foo.example.com*
baz.example.com	Cookie not set: domain mismatch
example.com	**.example.com*
ample.com	Cookie not set: domain mismatch
.com	Cookie not set: domain too broad, security risk

Credits: The Tangled Web: A Guide to Securing Modern Web Applications, by Michał Zalewski

When browser sends cookie

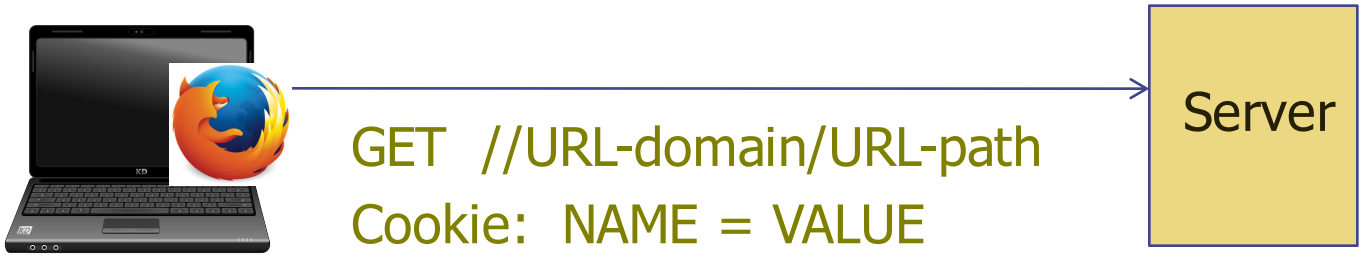


Goal: server only sees cookies in its scope

Browser sends all cookies in URL scope:

- cookie-domain is domain-suffix of URL-domain, and
- cookie-path is prefix of URL-path, and
- [protocol=HTTPS if cookie is "secure"]

When browser sends cookie



A cookie with

domain = example.com, and

path = /some/path/

will be included on a request to

http://foo.example.com/some/path/subdirectory/hello.txt

Examples

cookie 1
name = **userid**
value = u1
domain = **login.site.com**
path = **/**
non-secure

cookie 2
name = **userid**
value = u2
domain = **.site.com**
path = **/**
non-secure

http://checkout.site.com/	cookie: userid=u2
http://login.site.com/	cookie: userid=u1, userid=u2
http://othersite.com/	cookie: none

Examples

cookie 1
name = **userid**
value = u1
domain = **login.site.com**
path = **/**
secure

cookie 2
name = **userid**
value = u2
domain = **.site.com**
path = **/**
non-secure

http://checkout.site.com/ cookie: userid=u2

http://login.site.com/ cookie: userid=u2

https://login.site.com/ **cookie: userid=u1; userid=u2**

(arbitrary order)

Client side read/write: document.cookie

- Setting a cookie in Javascript:

  document.cookie = "name=value; expires=...; "

- Reading a cookie: alert(document.cookie)

  prints string containing all cookies available for document (based on [protocol], domain, path)

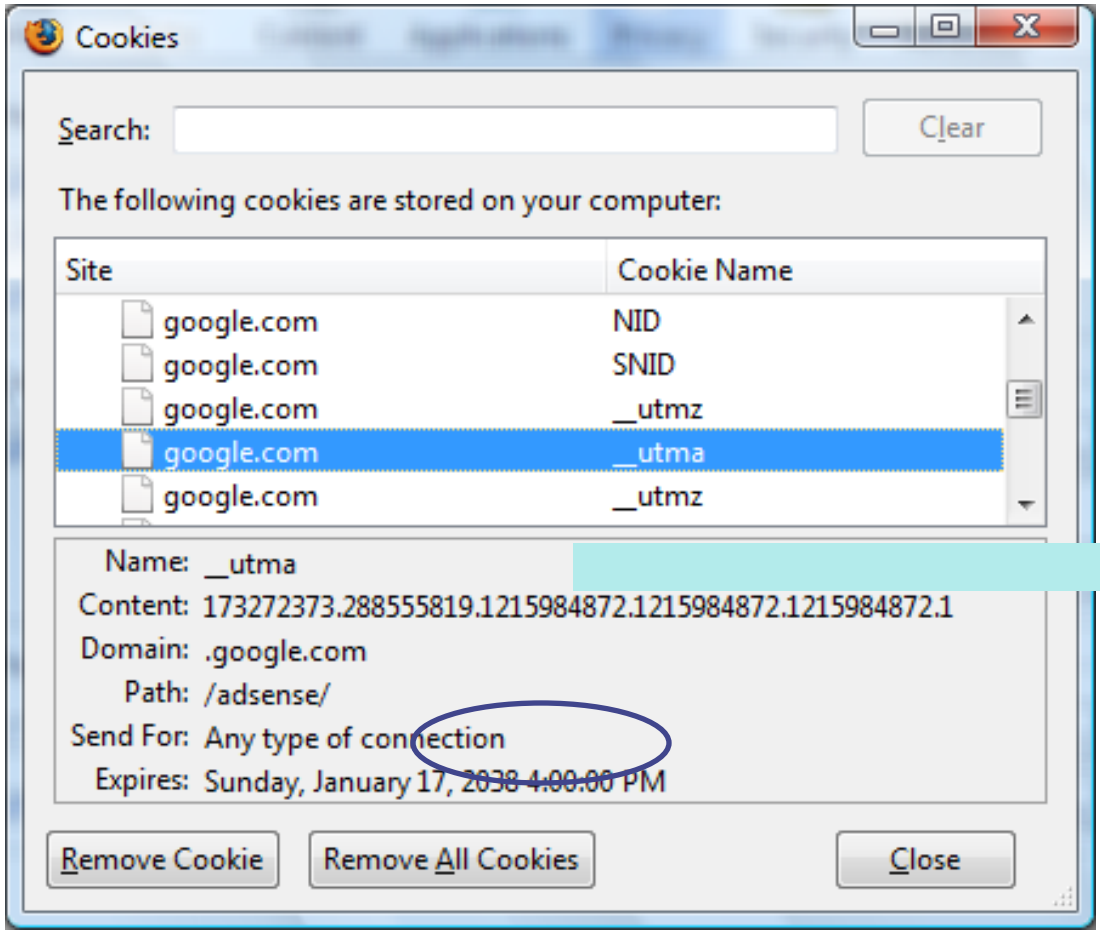
- Deleting a cookie:

  document.cookie = "name=; expires= Thu, 01-Jan-70"

document.cookie often used to customize page in Javascript

Viewing/deleting cookies in Browser UI

Firefox: Tools -> page info -> security -> view cookies



Session management

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

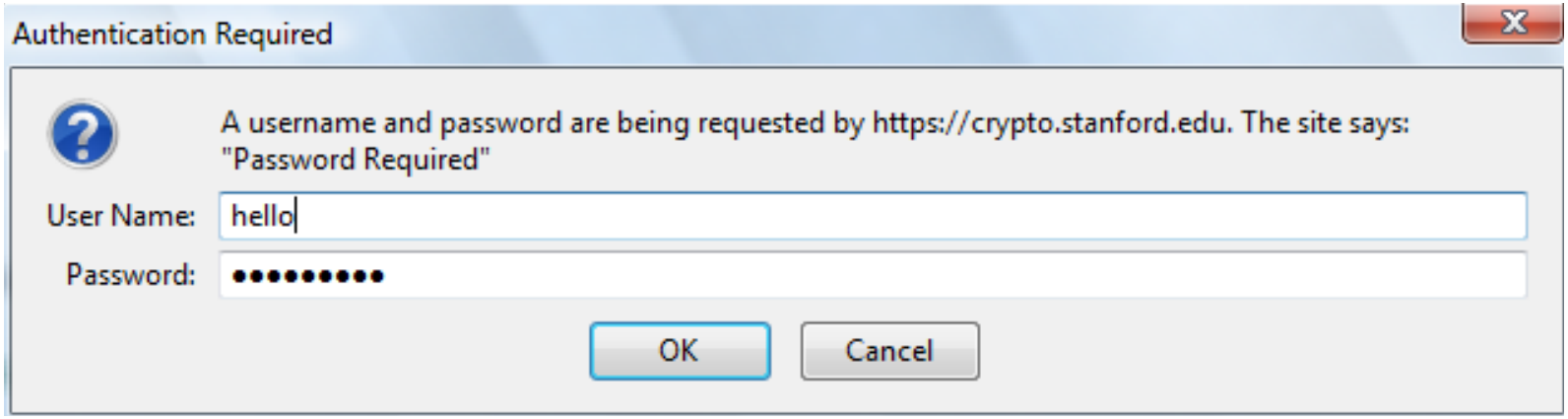
- Session mgmt:
  - Authorize user once;
  - All subsequent requests are tied to user

Pre-history: HTTP auth

HTTP request: GET /index.html

HTTP response contains:

WWW-Authenticate: Basic realm="Password Required"



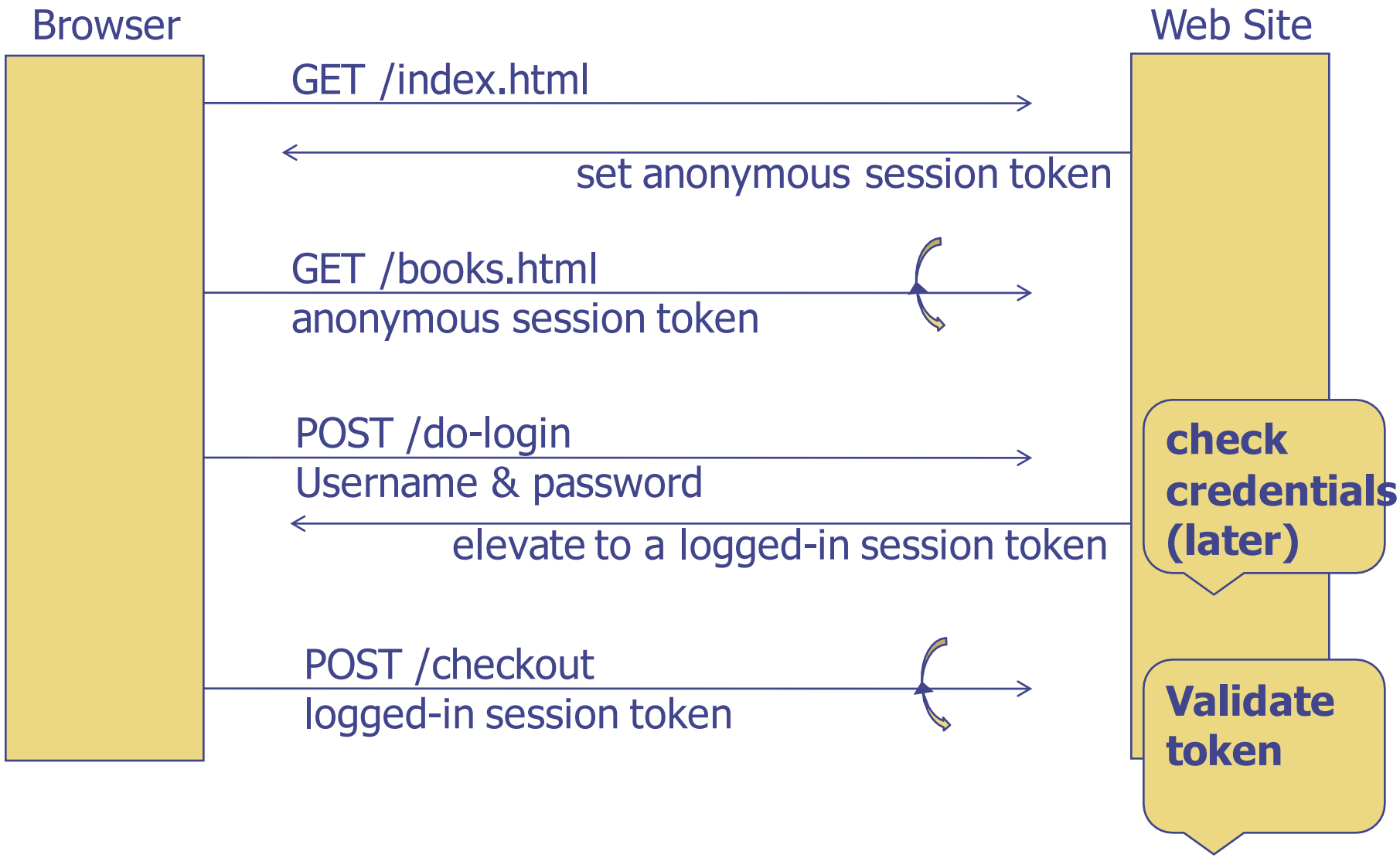
Browsers sends hashed password on all subsequent HTTP requests:

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">

Storing session tokens: problems

- Browser cookie:

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

- Embed in all URL links:

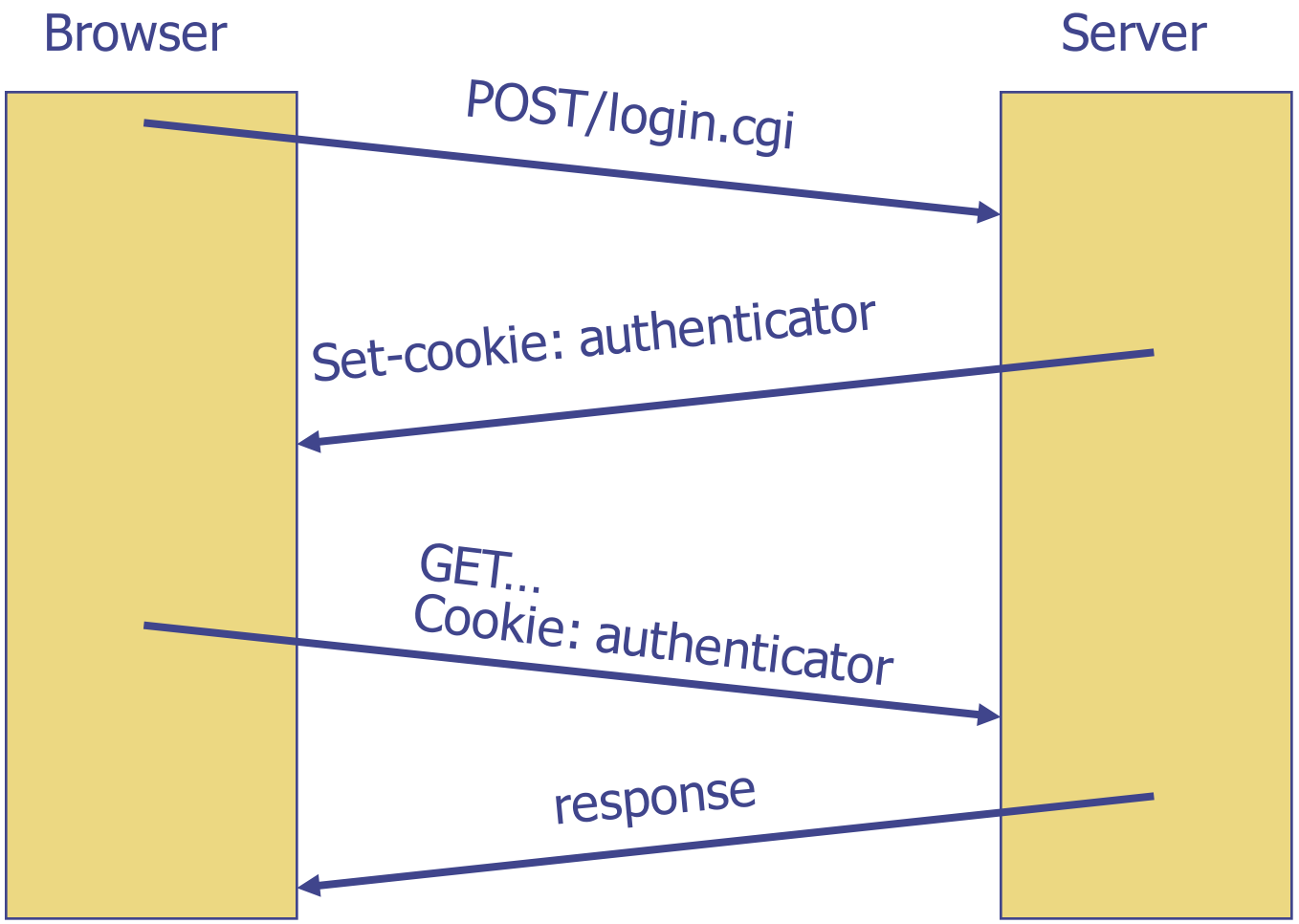
  token leaks via HTTP Referer header

- In a hidden form field: short sessions only

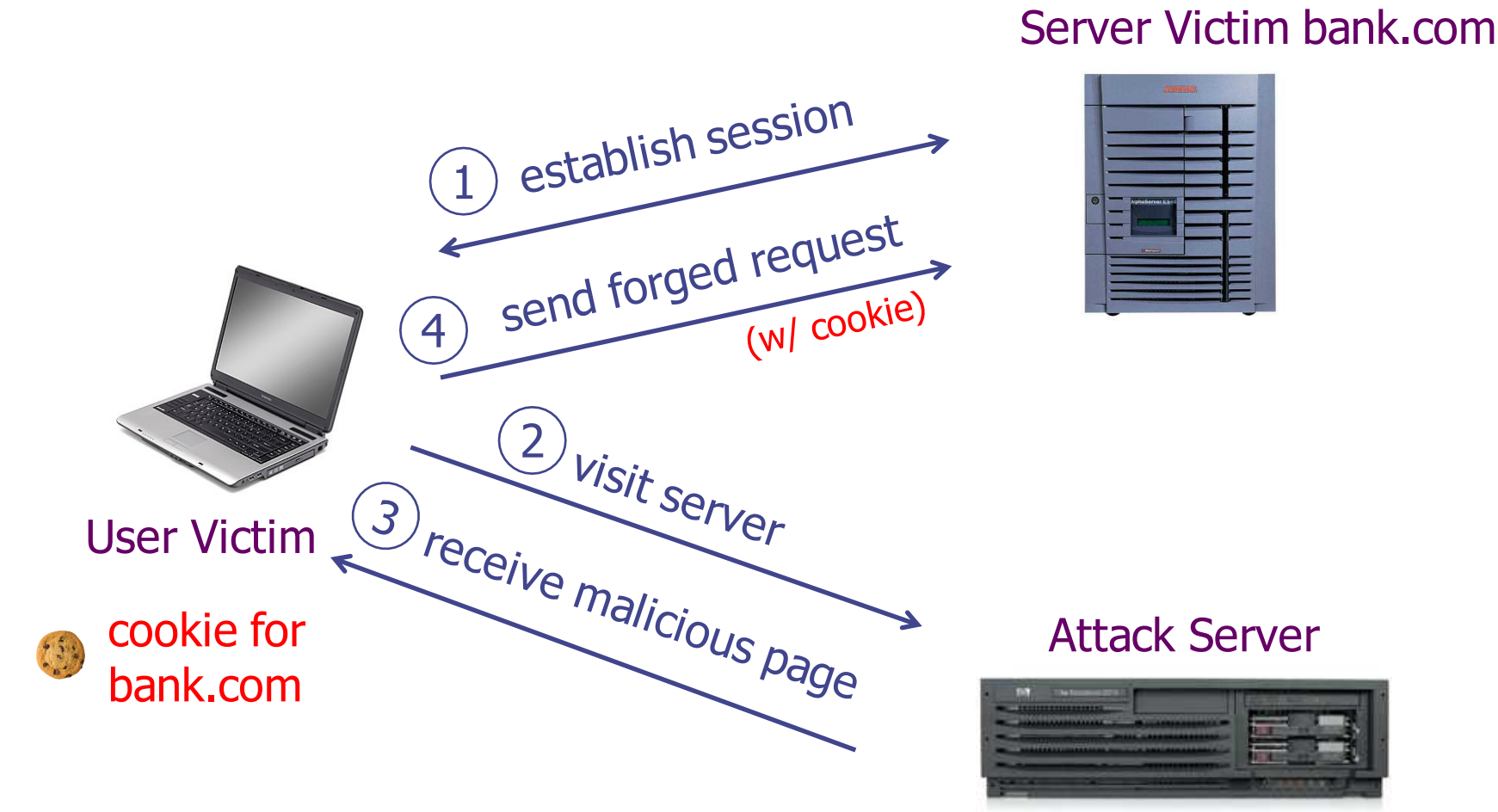
Best answer: a combination of all of the above.

Cross Site Request Forgery

Recall: session using cookies



Basic picture



What can go bad?

URL contains transaction action

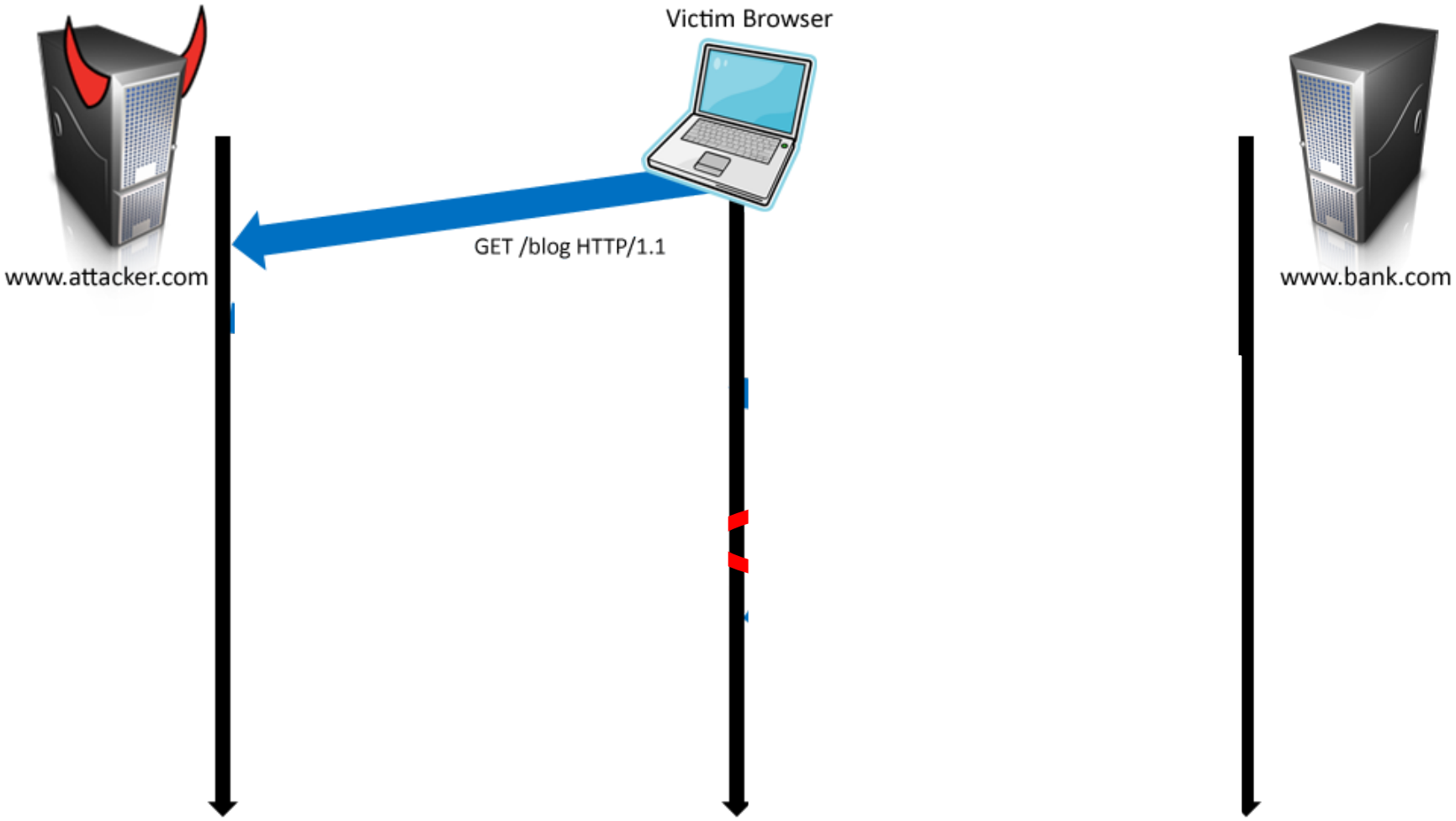
Cross Site Request Forgery (CSRF)

- Example:
  - User logs in to bank.com
    - Session cookie remains in browser state
  - User visits malicious site containing:

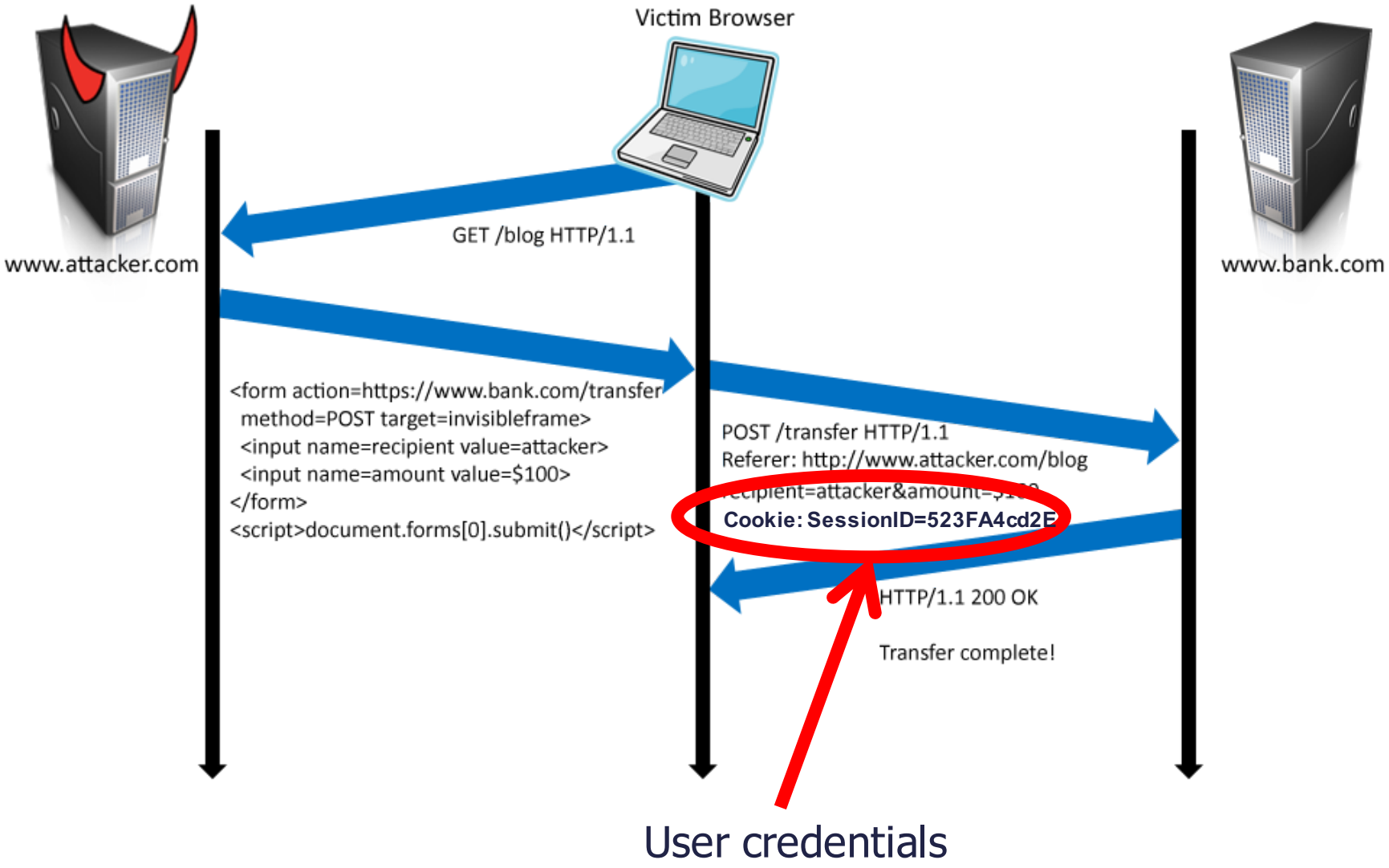
```
<form name=F action=http://bank.com/BillPay.php>
<input name=recipient value=badguy> …
<script> document.F.submit(); </script>
```

  - Browser sends user auth cookie with request
    - Transaction will be fulfilled
- Problem:
  - cookie auth is insufficient when side effects occur

Form post with cookie



Form post with cookie



2008 CSRF attack

An attacker could

- add videos to a user's "Favorites,"
- add himself to a user's "Friend" or "Family" list,
- send arbitrary messages on the user's behalf,
- flagged videos as inappropriate,
- automatically shared a video with a user's contacts, subscribed a user to a "channel" (a set of videos published by one person or group), and
- added videos to a user's "QuickList" (a list of videos a user intends to watch at a later point).

Facebook Hit by Cross-Site Request Forgery Attack

By Sean Michael Kerner / August 20, 2009

Angela Moscaritolo

September 30, 2008

Popular websites fall victim to CSRF exploits

Defenses

CSRF Defenses

- Secret Validation Token


```
<input type=hidden value=23a3af01b>
```

- Referer Validation


```
Referer: http://www.facebook.com/home.php
```

- Others (e.g., custom HTTP Header)


```
X-Requested-By: XMLHttpRequest
```

Secret Token Validation

1. goodsite.com server includes a secret token into the webpage (e.g., in forms as a hidden field)
2. Requests to goodsite.com include the secret
3. goodsite.com server checks that the token embedded in the webpage is the expected one; reject request if not

Can the token be?

- 123456
- Dateofbirth

Validation token must be hard to guess by the attacker

Variants

- Session identifier
- Session-independent token
- Session-dependent token

Session identifier

- The user's session id is used as the secret validation token
- On every request the server validates if the token matches the session id
- Disadvantage is that anyone who reads the contents of the page, which contains the user's session id in the form of CSRF token, can impersonate the user till the session expires

Session independent nonce

- goodsite.com server sets a random nonce in a cookie when the user first visits the site. Other sites don't know this random nonce
- The nonce is included as a hidden form field as well
- Browser sends nonce and cookie to goodsite.come on all form POSTs
- Disadvantage is that an active network attacker can overwrite the session independent nonce with his or her own CSRF token

Session-dependent nonce

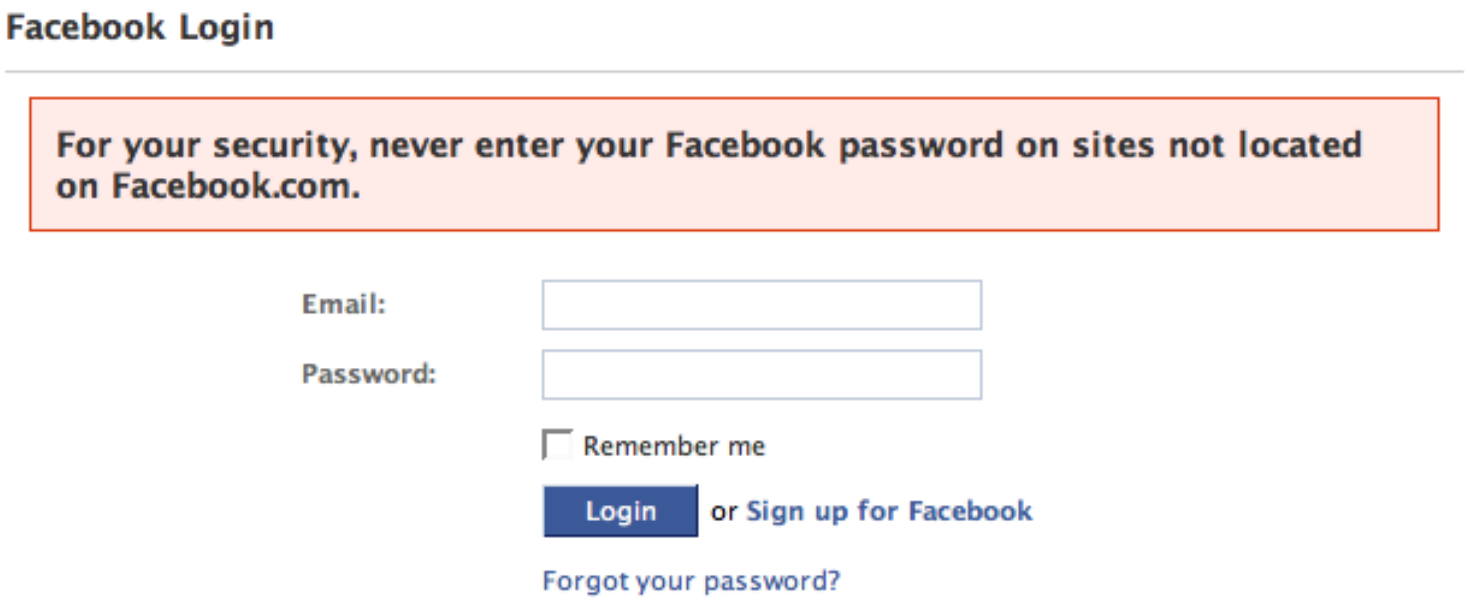
- The server stores state that binds the user's CSRF token to the user's session id
- Embeds CSRF token in every form
- On every request the server validates that the supplied CSRF token is associated with the user's session id
- Disadvantage is that the server needs to maintain a large state table to validate the tokens.

Answer: Token will be cryptographically bound to session id, attacker cannot create token

Other CRSF protection: Referer Validation

- When the browser issues an HTTP request, it includes a referer header that indicates which URL initiated the request
- This information in the Referer header could be used to distinguish between same site request and cross site request

Referer Validation



Referer Validation Defense

- HTTP Referer header
  - Referer: http://www.facebook.com/
  - Referer: http://www.attacker.com/evil.html
  - Referer:
    - Strict policy disallows (secure, less usable)
    - Lenient policy allows (less secure, more usable)

Privacy Issues with Referer header

- The referer contains sensitive information that impinges on the privacy
- The referer header reveals contents of the search query that lead to visit a website.
- Some organizations are concerned that confidential information about their corporate intranet might leak to external websites via Referer header

Referer Privacy Problems

- Referer may leak privacy-sensitive information

  `http://intranet.corp.apple.com/projects/iphone/competitors.html`

- Common sources of blocking:
  - Network stripping by the organization
  - Network stripping by local machine
  - Stripped by browser for HTTPS -> HTTP transitions
  - User preference in browser

Custom HTTP Headers

- Browsers prevent sites from sending custom HTTP headers to another site but allow sites to send custom HTTP headers to themselves.
- Cookie value is not actually required to prevent CSRF attacks, the mere presence of the header is sufficient.
- To use this scheme as a CSRF Defense, a site must issue all state modifying requests using XMLHttpRequest, attach the header and reject all requests that do not accompany the header .

Custom Header Defense

- XMLHttpRequest is for same-origin requests
  - Can use setRequestHeader within origin
- Limitations on data export format
  - No setRequestHeader equivalent
  - XHR2 has a whitelist for cross-site requests
- Issue POST requests via AJAX:

- Doesn't work across domains

```
X-Requested-By: XMLHttpRequest
```

Summary

- Cookies add state to HTTP
  - Cookies are used for session management
  - They are attached by the browser automatically to HTTP requests
- CSRF attacks execute request on benign site because cookie is sent automatically
- Defenses for CSRF:
  - embed unpredicatable token and check it later
  - check referer header