Web Security: Session management

CS 161: Computer Security

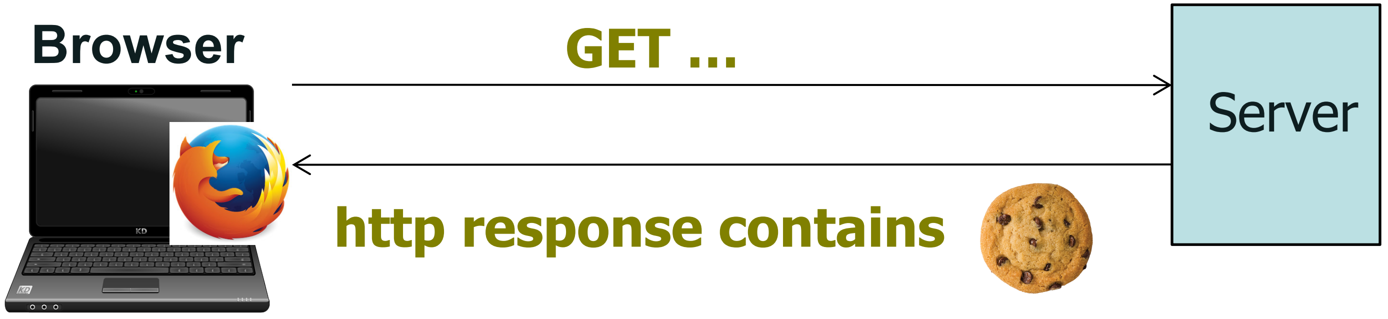
Prof. Raluca Ada Popa

April 10, 2020

Some content adapted from materials by David Wagner or Dan Boneh

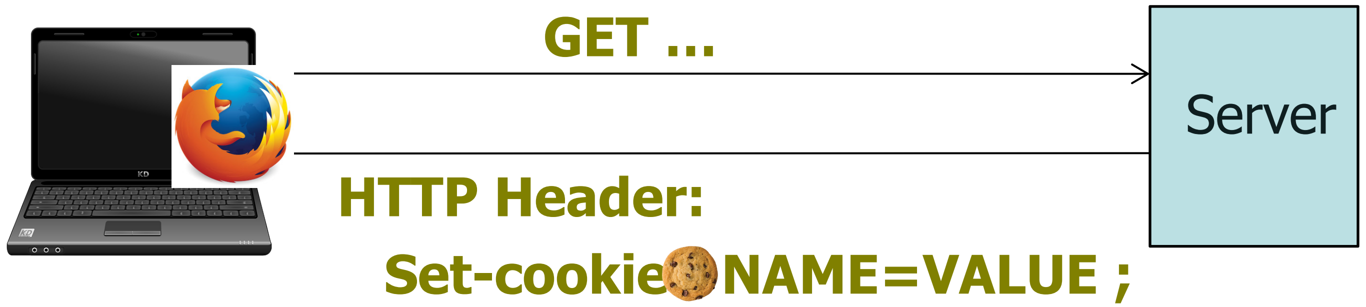
Cookies

A way of maintaining state in the browser



Browser maintains cookie jar with all cookies it receives

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 (with some extra metadata)

View a cookie

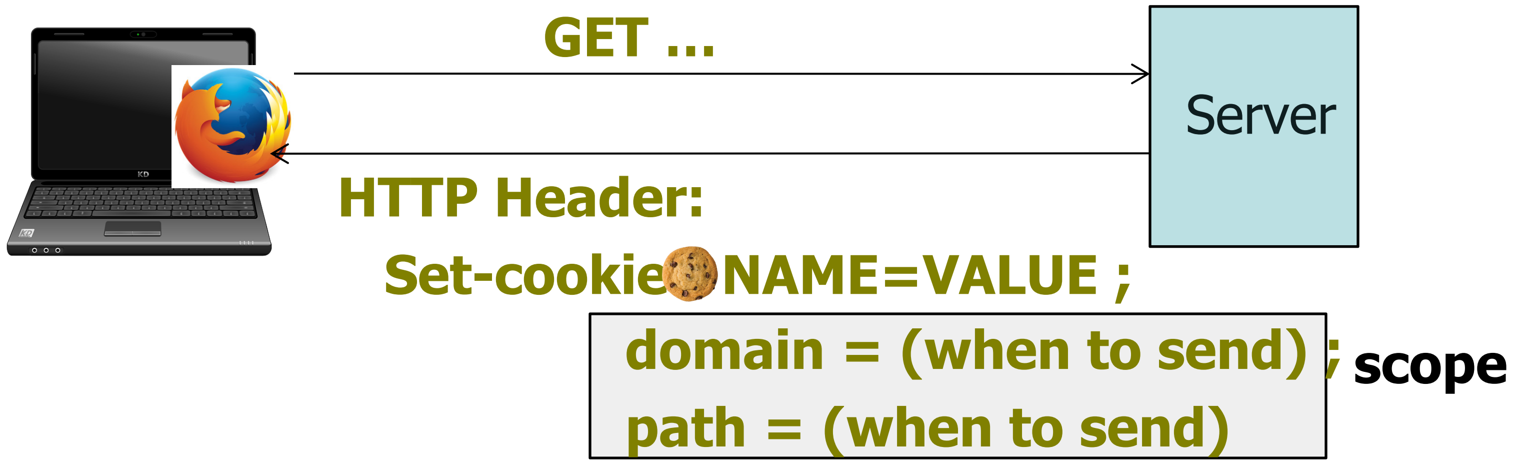
In a web console (firefox, tool->web developer->web console), type

document.cookie

to see the cookie for that site

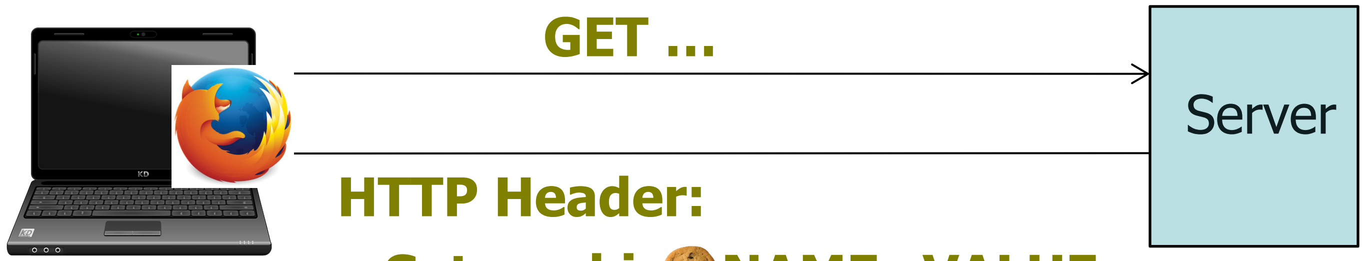
Each name=value is one cookie.
document.cookie lists all cookies in scope for document

Cookie scope



- When the browser connects to the same server later, it automatically attaches the cookies in scope: 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

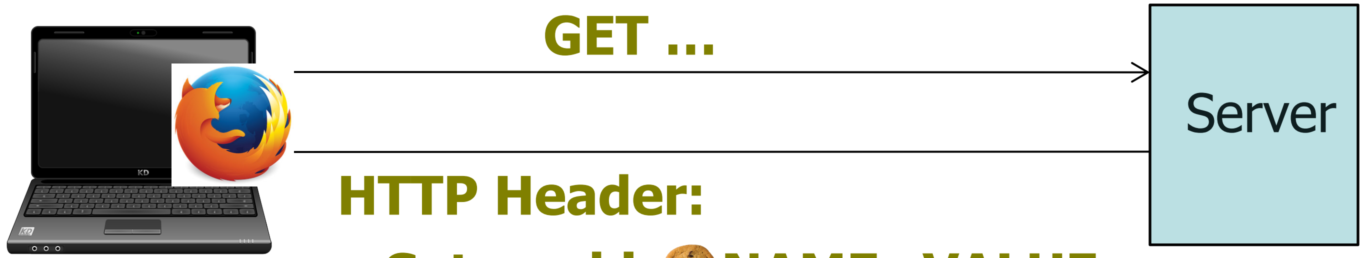
Cookie scope



HTTP Header:
Set-cookie NAME=VALUE ;
domain = (when to send) ;
path = (when to send)
secure = (only send over HTTPS);

- Secure: sent over https only
  - https provides secure communication using TLS (privacy and integrity)

Cookie scope



Set-cookie: NAME=VALUE ;
domain = (when to send) ; scope
path = (when to send)
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 (defense in depth, but does not prevent XSS)

Cookie policy

The cookie policy has two parts:

1. What scopes a URL-host name web server is allowed to set on a cookie
2. When the browser sends a cookie to a URL

Cookie scope

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

What scope a server may set for a cookie

The browser checks if the web server may set the cookie, and if not, it will not accept the cookie.

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

example: host = "login.site.com"

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

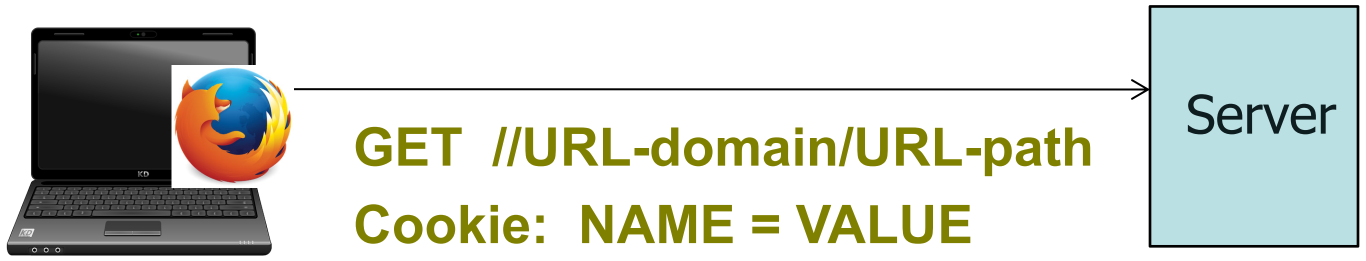
path: can be set to anything

Examples

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

domain	**Whether it will be set, and if so, where it will be sent to**
(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

When browser sends cookie

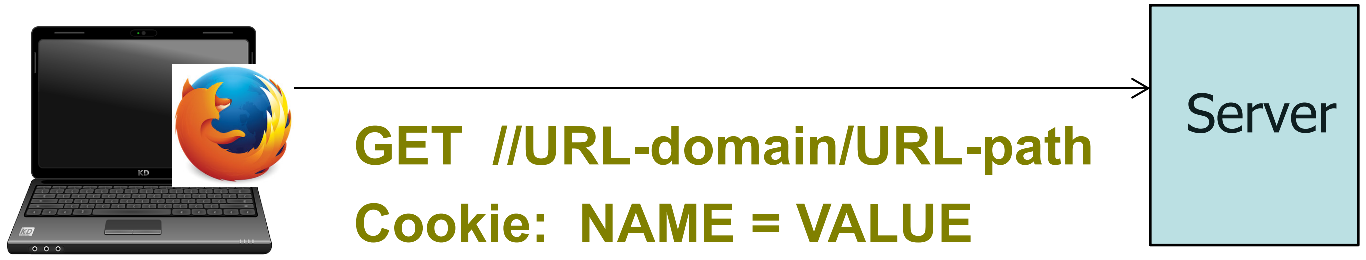


Goal: server only sees cookies in its scope

Browser sends all cookies <u>in URL scope</u>:

- 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: Which cookie will be sent?

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

<u>cookie 2</u>
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

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

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

Examples

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

domain	**Whether it will be set, and if so, where it will be sent to**	
(value omitted)	*foo.example.com* (exact)	**.foo.example.com*
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

Examples

<u>cookie 1</u>
name = userid
value = u1
domain = login.site.com
path = /
secure

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

http://checkout.site.com/	**cookie: userid=u2**
http://login.site.com/	**cookie: userid=u2**
http**s**://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)

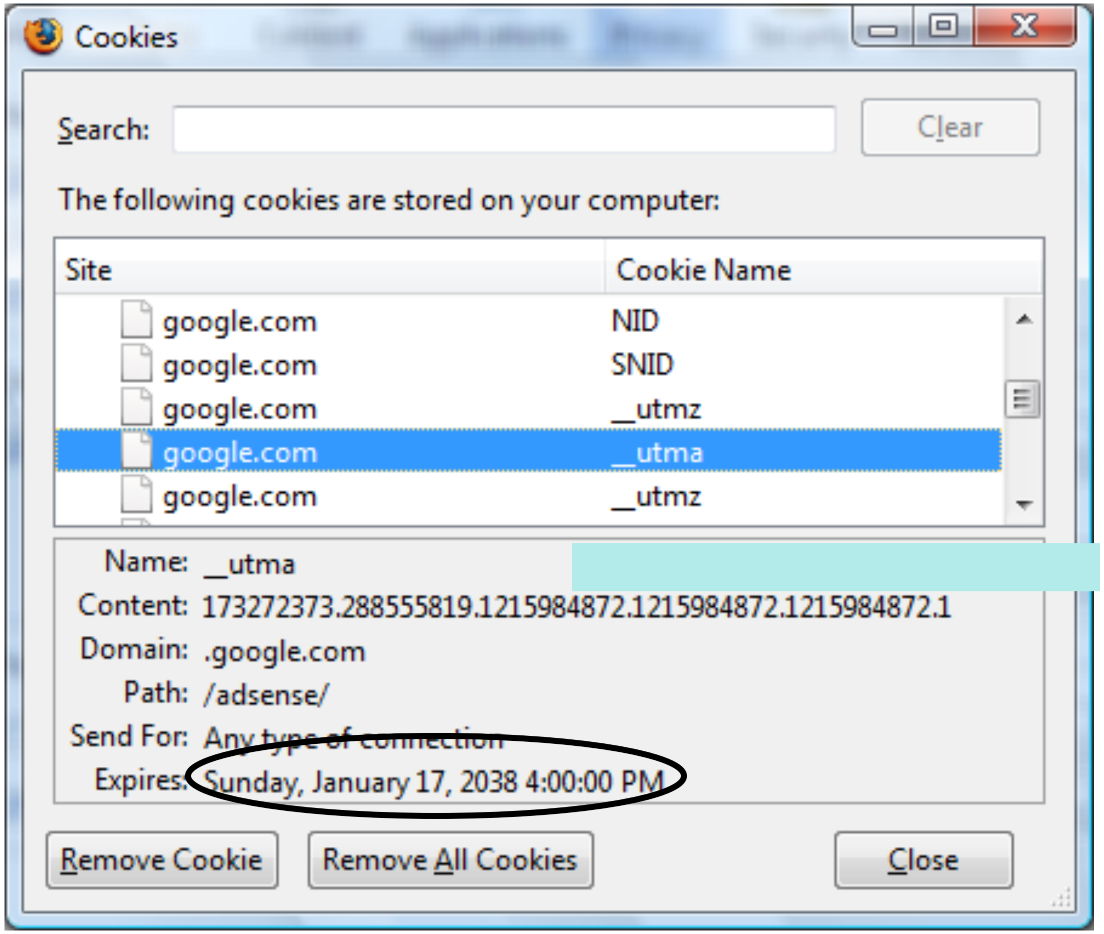
- Deleting a cookie:

  document.cookie = “name=; expires= Thu, 01-Jan-00”

document.cookie often used to customize page in Javascript

Viewing/deleting cookies in Browser UI

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



The following cookies are stored on your computer:

Site	Cookie Name
google.com	NID
google.com	SNID
google.com	__utmz
google.com	__utma
google.com	__utmz

Name:	__utma
Content:	173272373.288555819.1215984872.1215984872.1215984872.1
Domain:	.google.com
Path:	/adsense/
Send For:	Any type of connection
Expires:	Sunday, January 17, 2038 4:00:00 PM

Cookie policy versus same-origin policy

Cookie policy versus same-origin policy

- Consider Javascript on a page loaded from a URL U
- If a cookie is in scope for a URL U, it can be accessed by Javascript loaded on the page with URL U,

  unless the cookie has the httpOnly flag set.

So there isn’t exact domain match as in same-origin policy, but cookie policy instead.

Examples

<u>cookie 1</u>	**<u>cookie 2</u>**
name = userid	**name = userid**
value = u1	**value = u2**
domain = login.site.com	**domain = .site.com**
path = /	**path = /**
non-secure	**non-secure**

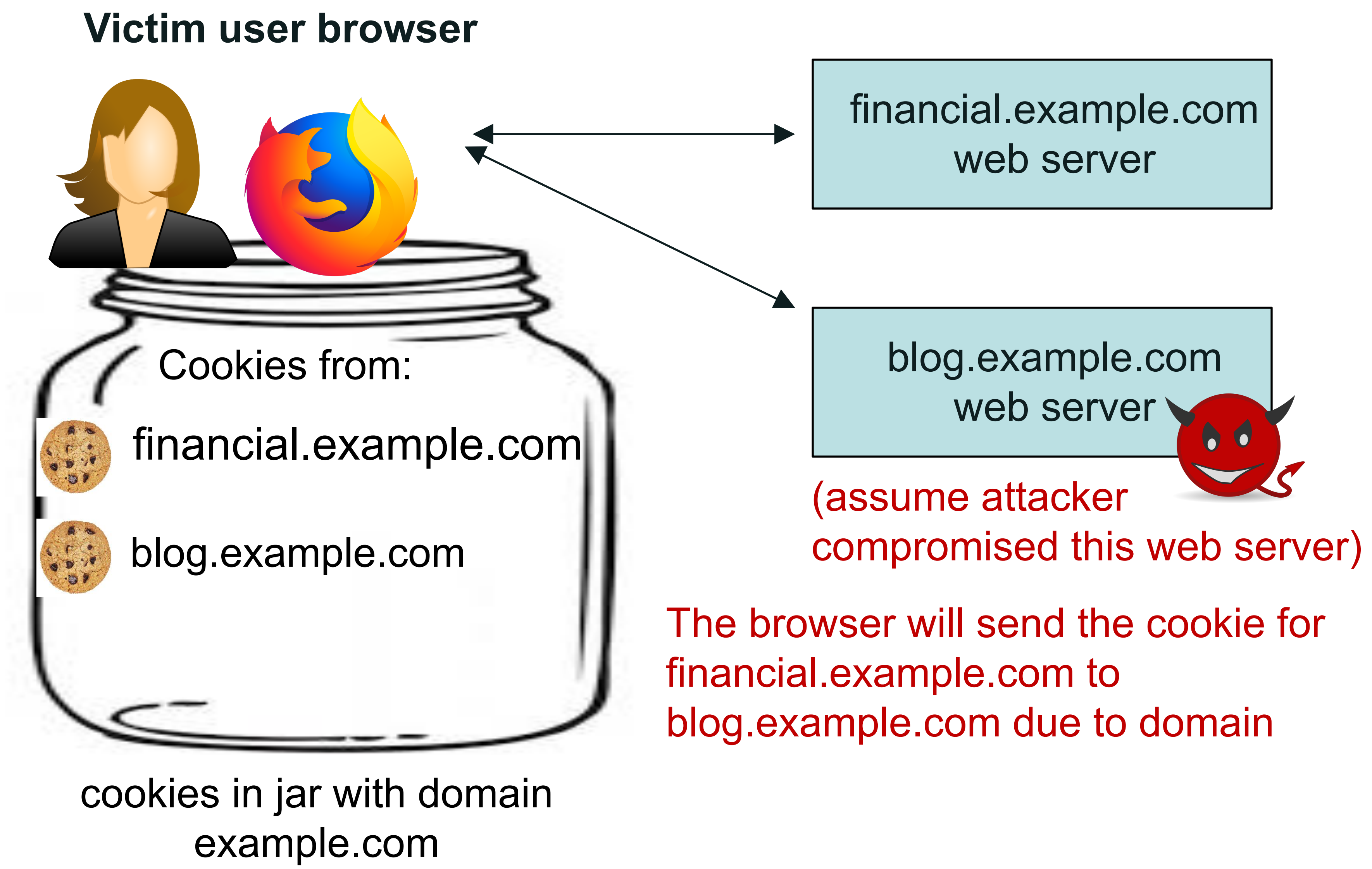
http://checkout.site.com/	**cookie: userid=u2**
http://login.site.com/	**cookie: userid=u1, userid=u2**
http://othersite.com/	**cookie: none**

JS on each of these URLs can access the corresponding cookies even if the domains are not the same

Indirectly bypassing same-origin policy using cookie policy

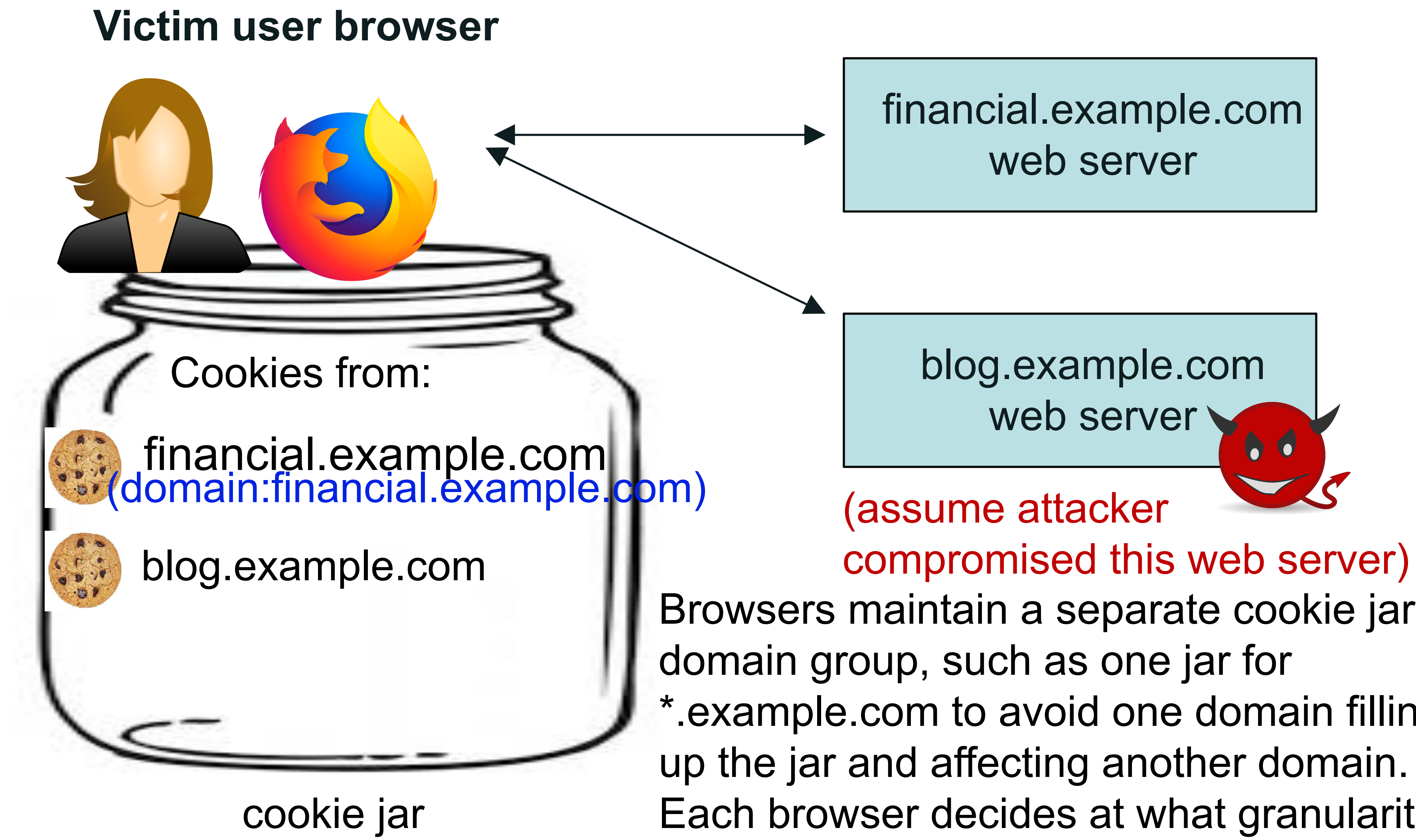
- Since the cookie policy and the same-origin policy are different, there are corner cases when one can use cookie policy to bypass same-origin policy
- Ideas how?

Example


Victim user browser
financial.example.com web server
blog.example.com web server
Cookies from:
financial.example.com
blog.example.com
(assume attacker compromised this web server)
cookies in jar with domain example.com

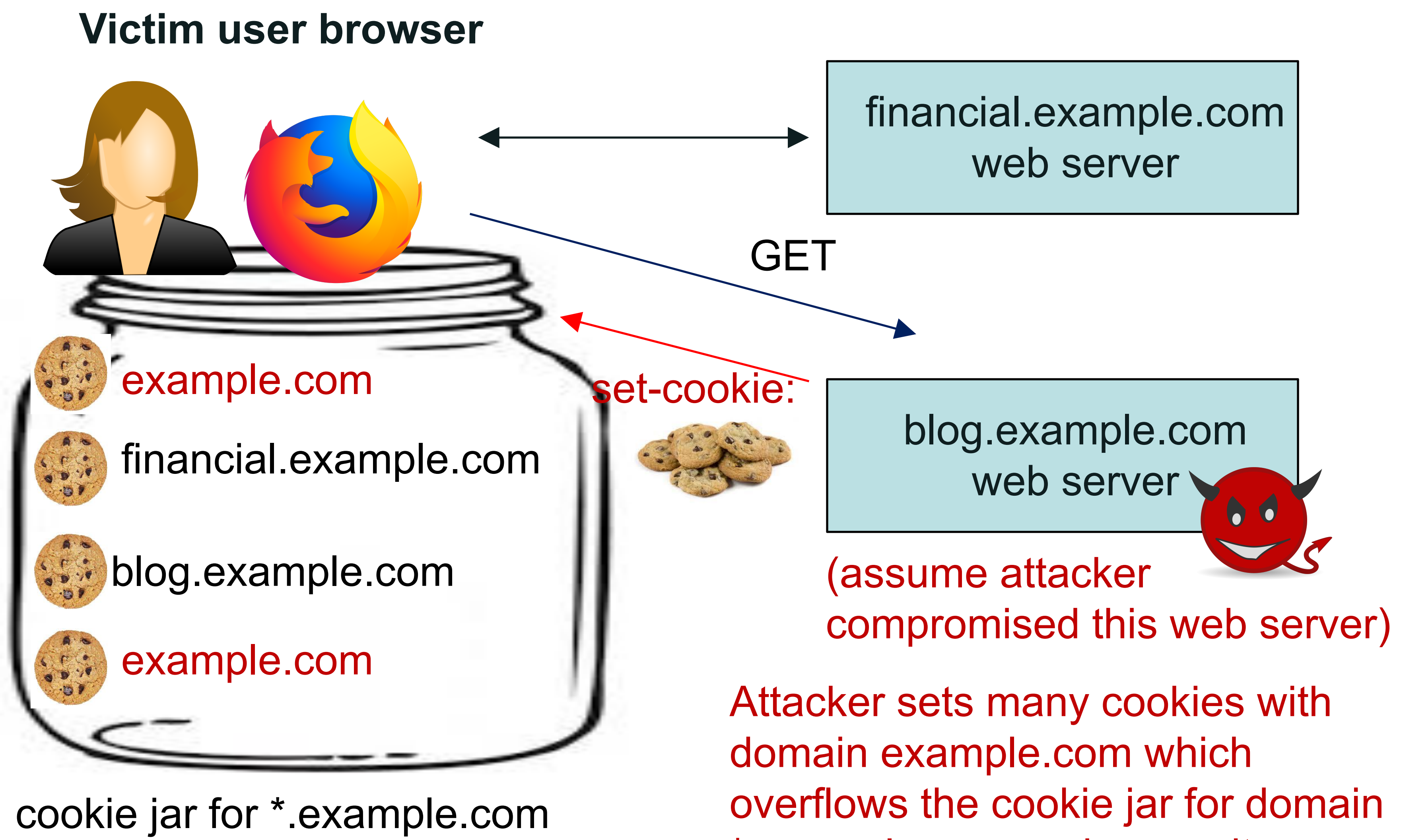
The browser will send the cookie for financial.example.com to blog.example.com due to domain

Example


Victim user browser
financial.example.com web server
blog.example.com web server
Cookies from:
financial.example.com
(domain:financial.example.com)
blog.example.com
cookie jar
(assume attacker compromised this web server)

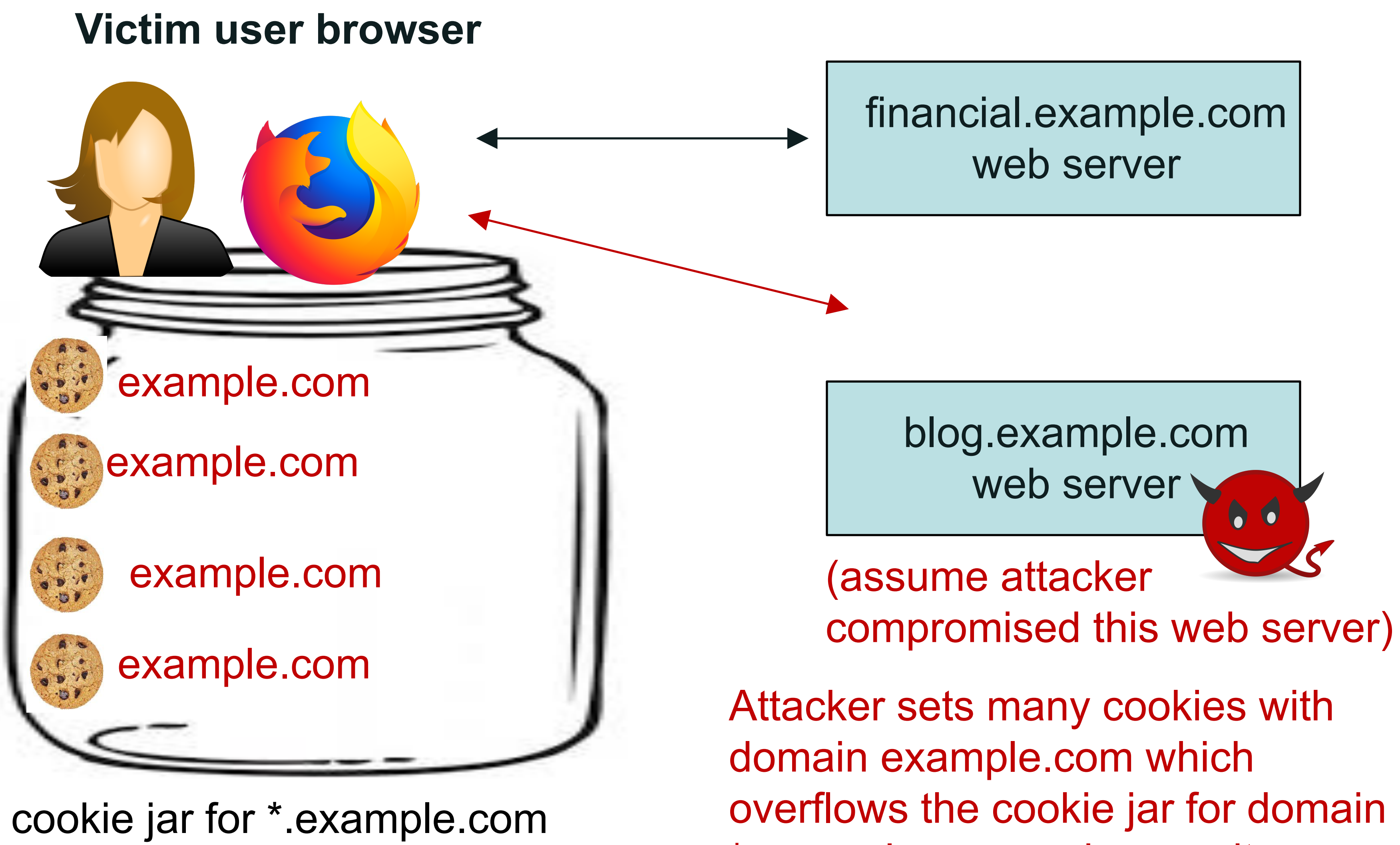
Browsers maintain a separate cookie jar per domain group, such as one jar for *.example.com to avoid one domain filling up the jar and affecting another domain. Each browser decides at what granularity to hold group domains.

Example


Victim user browser
financial.example.com web server
GET
set-cookie:
blog.example.com web server
(assume attacker compromised this web server)
example.com
financial.example.com
blog.example.com
example.com
cookie jar for *.example.com

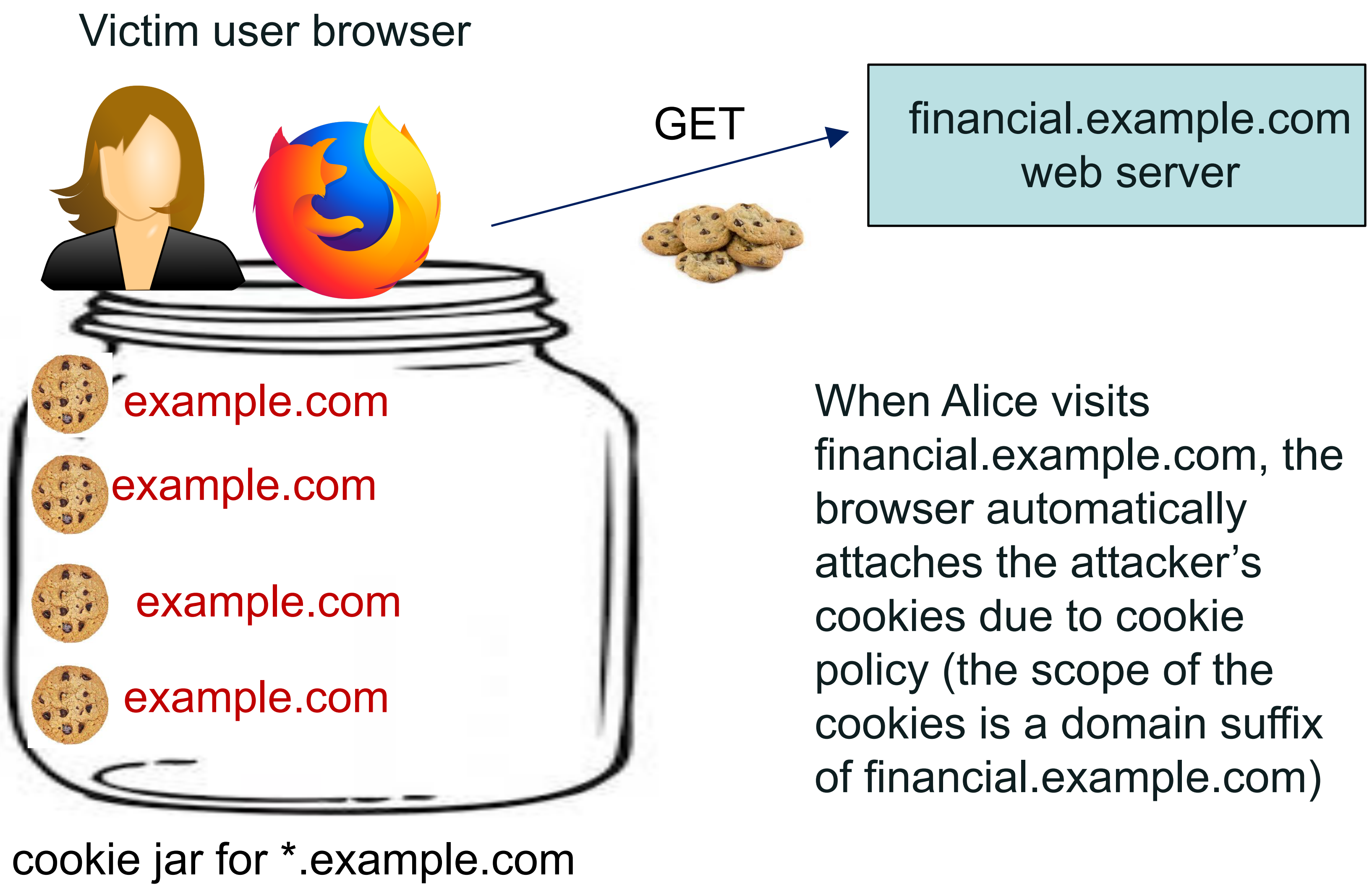
Attacker sets many cookies with domain example.com which overflows the cookie jar for domain *.example.com and overwrites cookies from financial.example.com

Example


Victim user browser
financial.example.com web server
blog.example.com web server
example.com
example.com
example.com
example.com
(assume attacker compromised this web server)
cookie jar for *.example.com

Attacker sets many cookies with domain example.com which overflows the cookie jar for domain *.example.com and overwrites cookies from financial.example.com

Example



When Alice visits financial.example.com, the browser automatically attaches the attacker's cookies due to cookie policy (the scope of the cookies is a domain suffix of financial.example.com)

Why is this a problem?

Indirectly bypassing same-origin policy using cookie policy

- Victim thus can login into attackers account at financial.example.com
- This is a problem because the victim might think its their account and might provide sensitive information
- This also bypassed same-origin policy (indirectly) because blog.example.com influenced financial.example.com

RFC6265

- For further details on cookies, checkout the standard RFC6265 “HTTP State Management Mechanism”

https://tools.ietf.org/html/rfc6265

- Browsers are expected to implement this reference, and any differences are browser specific

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 (banks)
  - without session mgmt:

    users would have to constantly re-authenticate
- Session management:
  - Authorize user once;
  - All subsequent requests are tied to user for a period

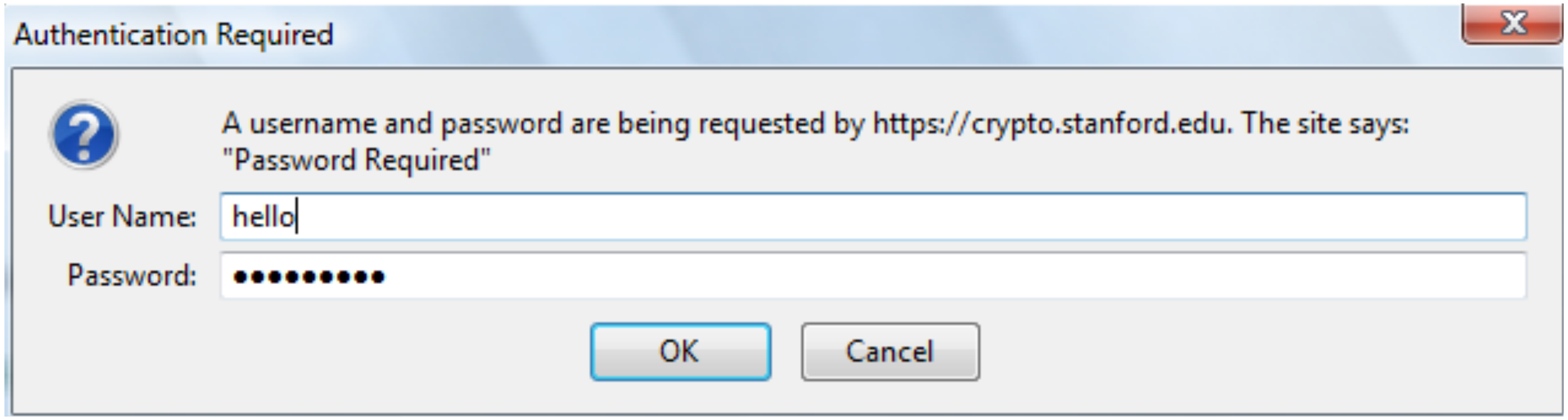
Pre-history: HTTP auth

One username and password for a group of users

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 Token Analogy

Analogy

- Show your ticket and ID
- Receive a wristband
- When you want to re-enter later, show your wristband

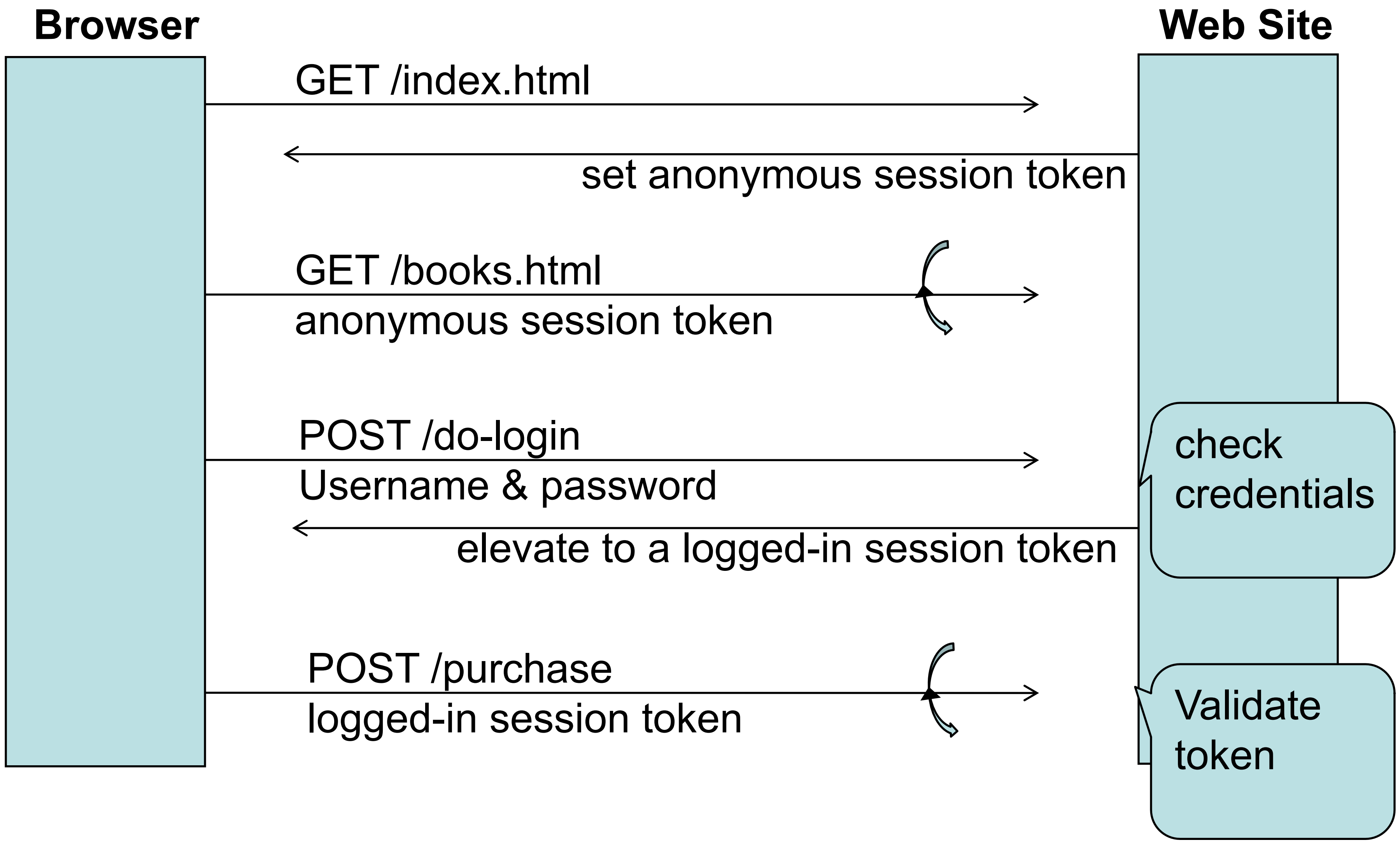
Actual Web

- Send your password
- Receive a session token
- When you want to make another request, send your session token

Session token

- A temporary identifier for a user, usually random or cryptographic so that an attacker cannot guess it
- If an attacker gets a session token, it could access the user’s account for the duration of that token

Session tokens



Storing session tokens:
Lots of options (but none are perfect)

- Browser cookie:

  Set-Cookie: SessionToken=fduhye63sfdb

- Embed 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
  - users might share URLs

- In a hidden form field: short sessions only

Better answer: a combination (1) and (3) above (e.g., browser cookie with CSRF protection using form secret tokens)

Cross Site Request Forgery

HTML Forms

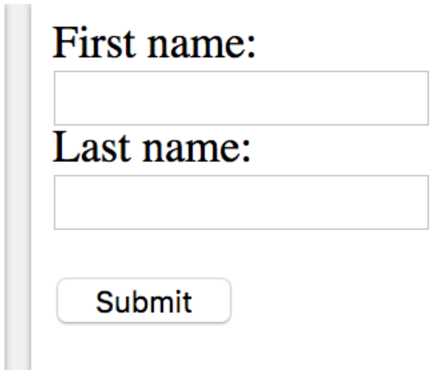
- Allow a user to provide some data which gets sent with an HTTP POST request to a server

```
<form action="bank.com/action.php">

First name:  <input type="text" name="firstname">

Last name:<input type="text" name="lastname">

<input type="submit" value="Submit"></form>
```



When filling in Alice and Smith, and clicking submit, the browser issues

```
HTTP POST request
bank.com/action.php?firstname=Alice&lastname=Smith
```

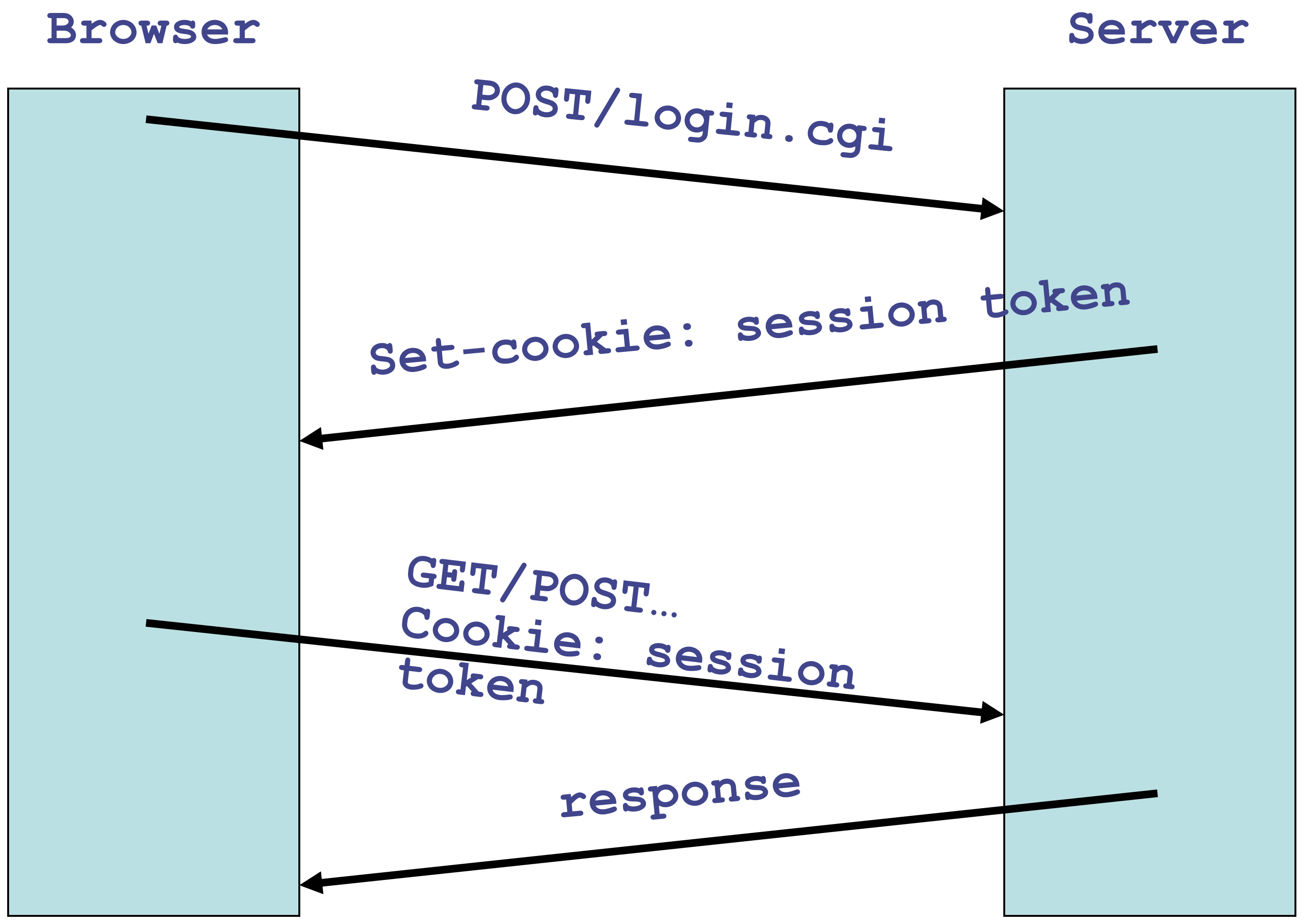
As always, the browser attaches relevant cookies

Consider the cookie stores the session token

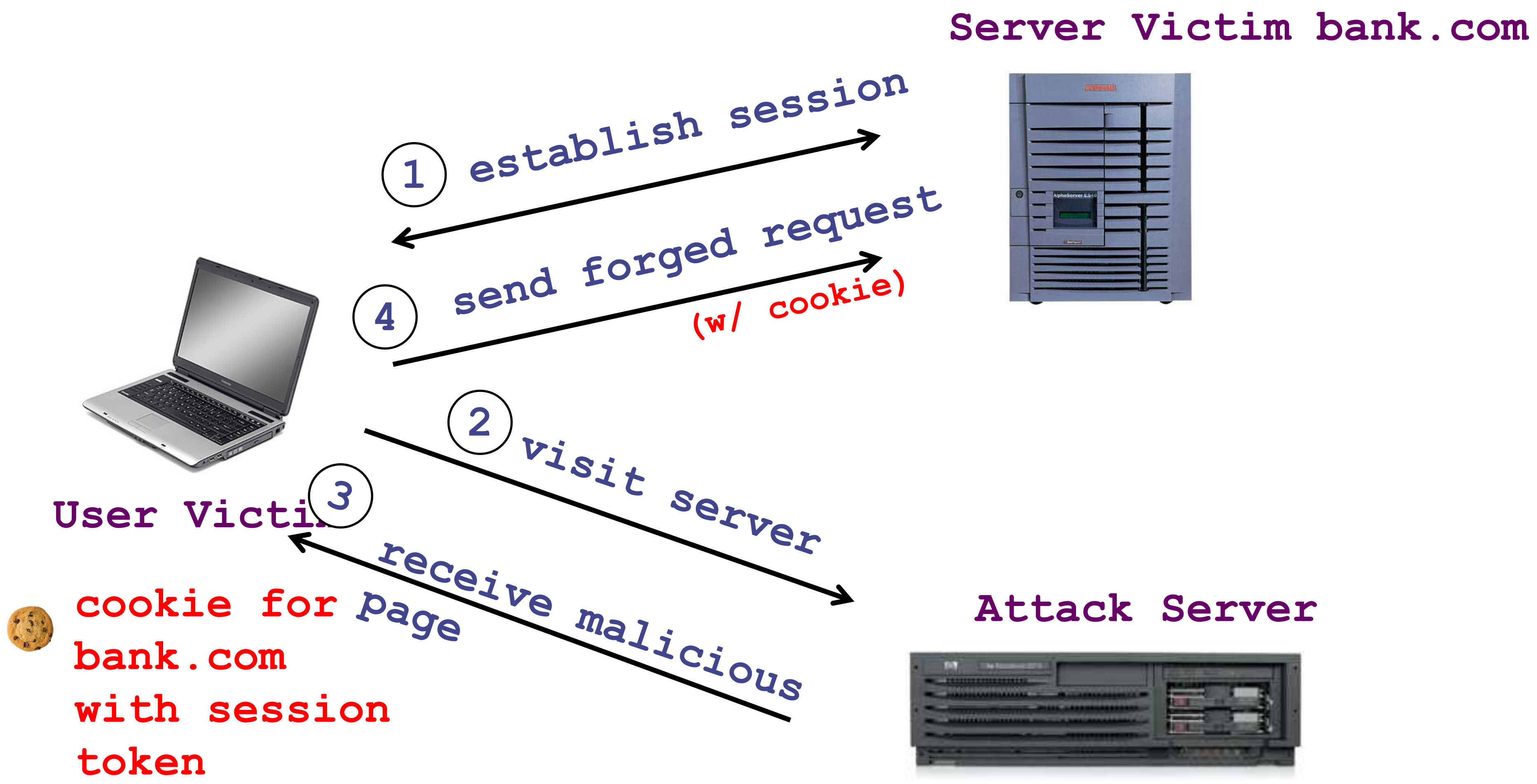
- Server assigns a random session token to each user after they logged in, places it in the cookie
- The server keeps a table of

  [username -> session token], so when it sees the session token it knows which user
- When the user logs out, the server clears the session token

Session using cookies



CSRF Attack Basic Picture



What can go bad? URL contains transaction action

Cross Site Request Forgery (CSRF)

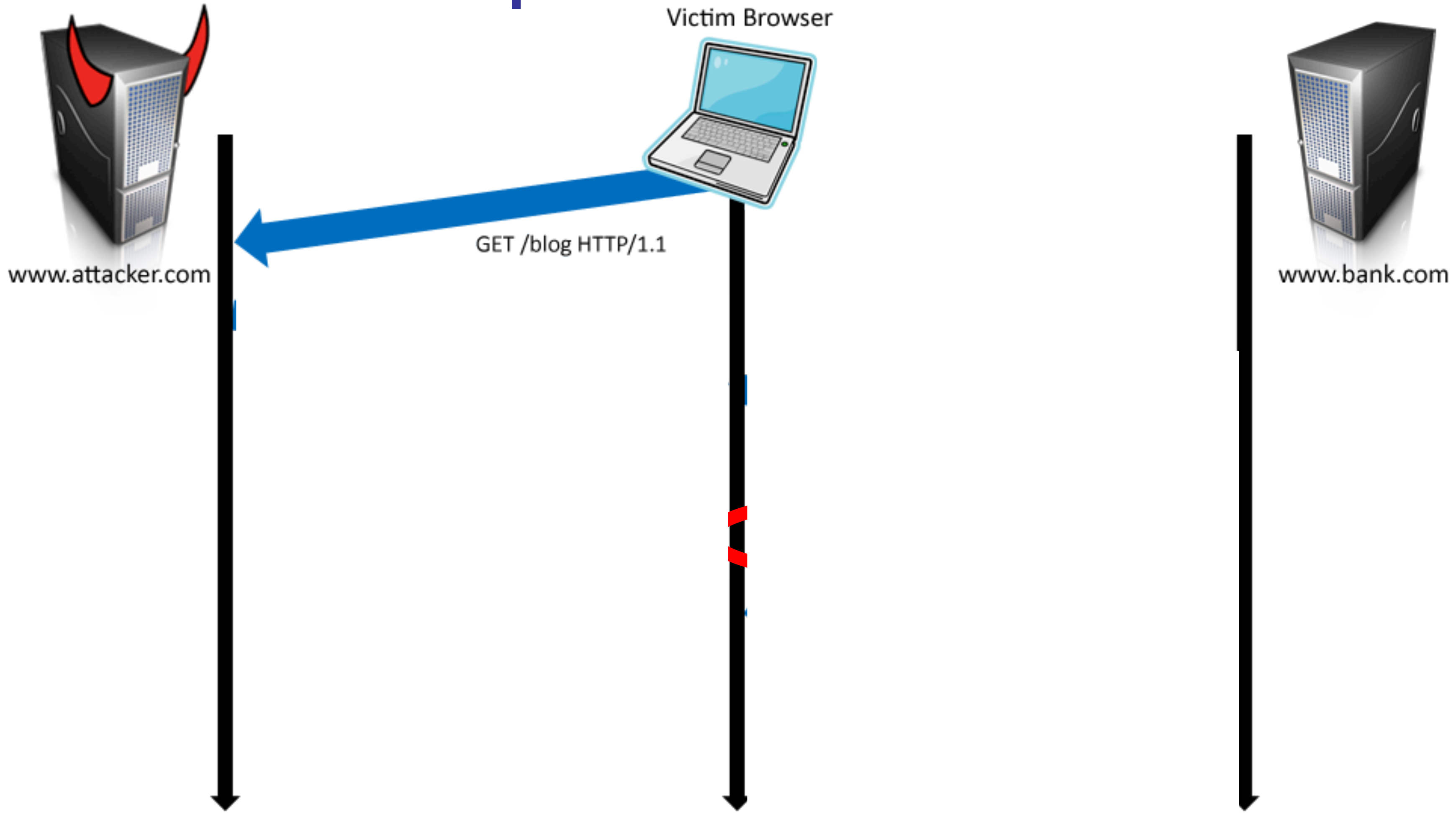
- User logs in to bank.com
  - Session cookie remains in browser state
- User visits malicious site containing:

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

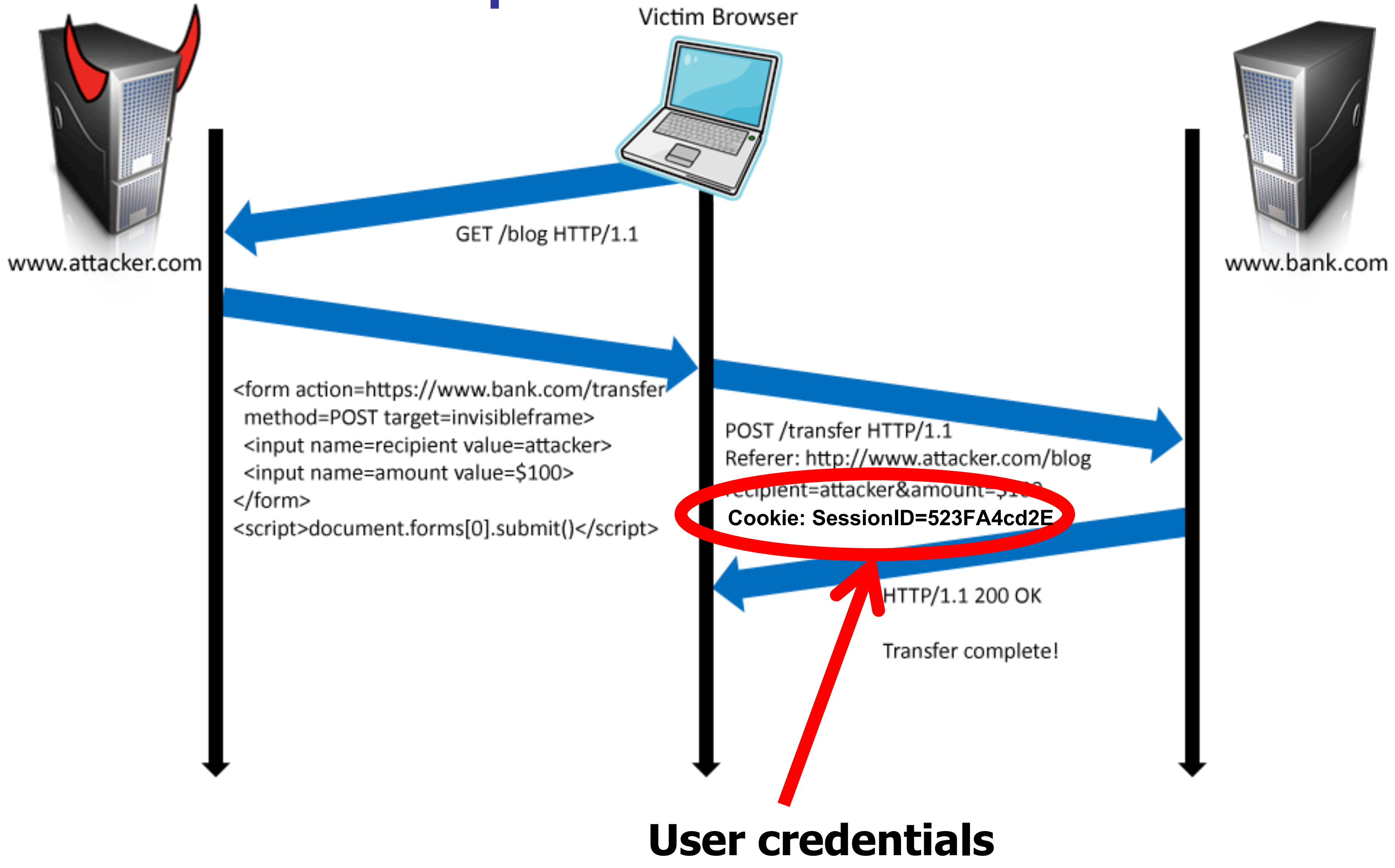
- Browser sends user auth cookie with request
  - Transaction will be fulfilled

- <u>Problem</u>:
  - cookie auth is insufficient when side effects occur

Form post with cookie



Form post with cookie



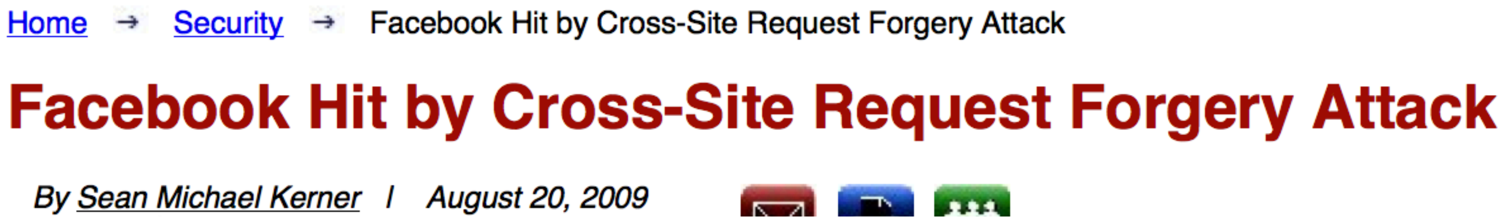
```
<form action=https://www.bank.com/transfer
  method=POST target=invisibleframe>
  <input name=recipient value=attacker>
  <input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100
Cookie: SessionID=523FA4cd2E
```

YouTube 2008 CSRF attack

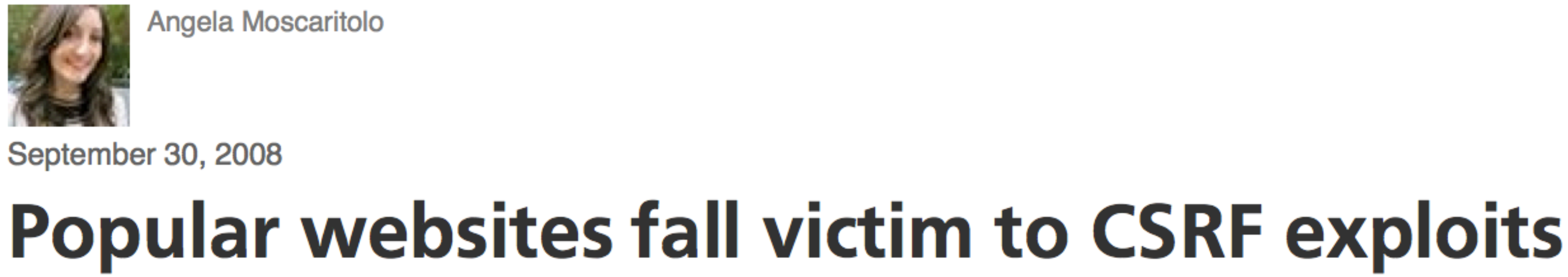
An attacker could

- 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).


Home → Security → Facebook Hit by Cross-Site Request Forgery Attack

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

CSRF Defenses

- CSRF token

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

- Referer Validation

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

- Others (e.g., custom HTTP Header) we won’t go into

CSRF token

1. goodsite.com server wants to protect itself from CSRF attacks, so it 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?

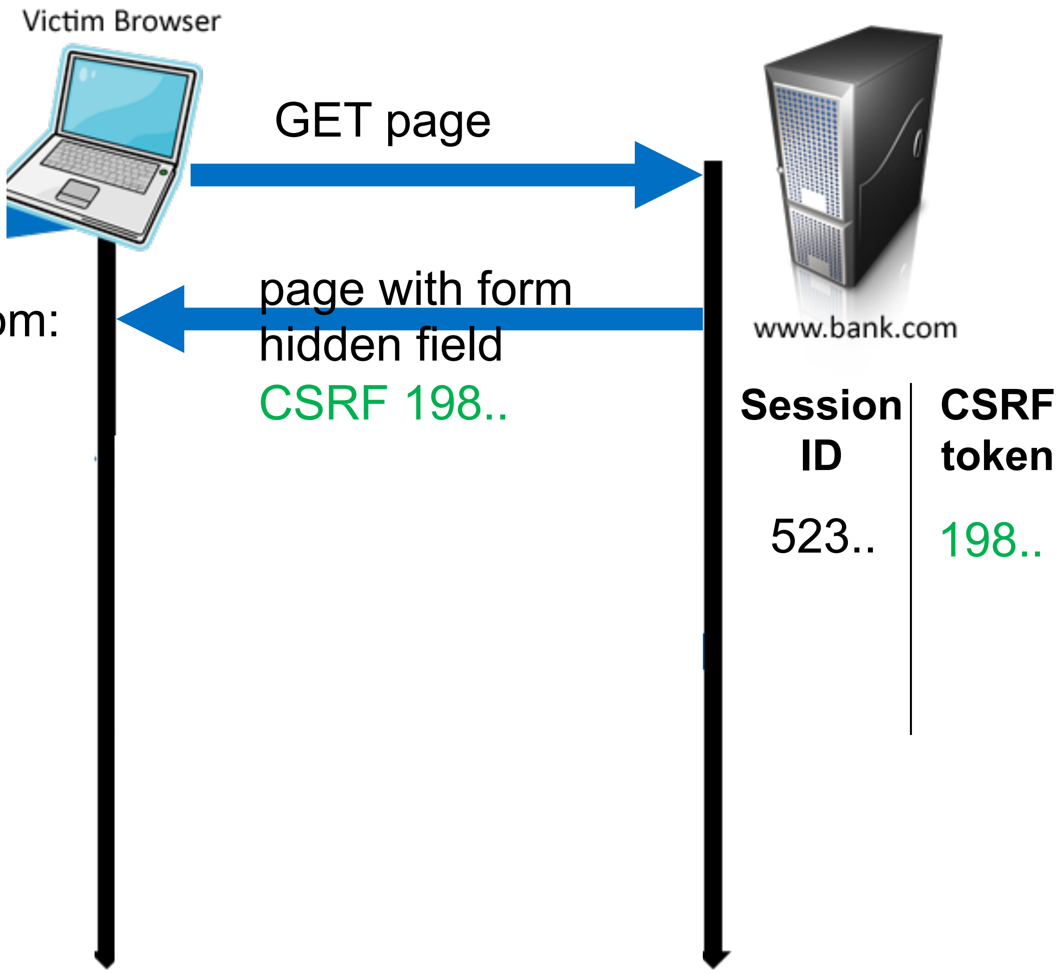
- **123456**
- **Dateofbirth**

CSRF token must be hard to guess by the attacker

How the token is used

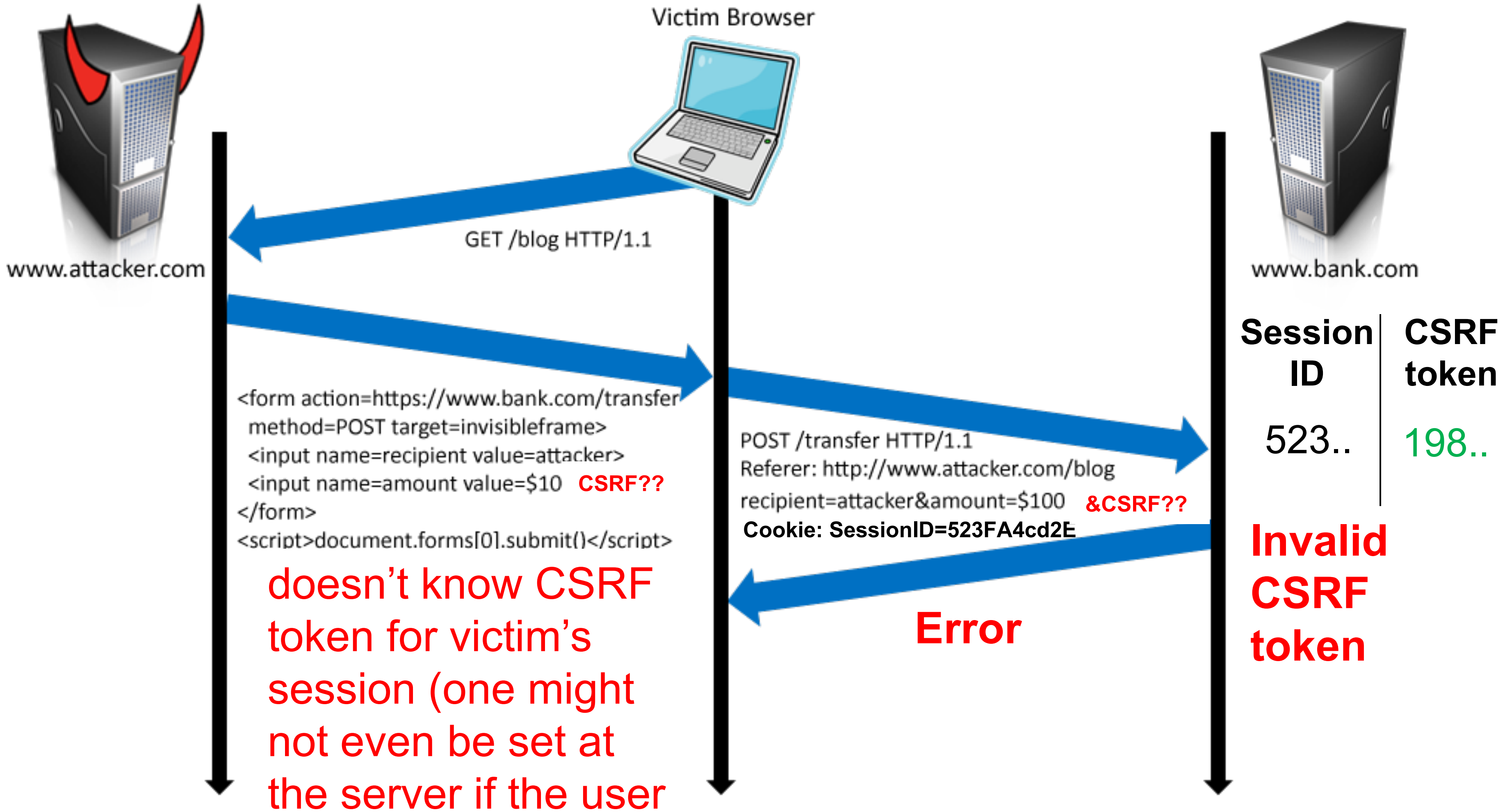
- The server stores state that binds the user's CSRF token to the user's session token
- Embeds a fresh CSRF token in every form
- On every request the server validates that the supplied CSRF token is associated with the user's session token
- Disadvantage is that the server needs to maintain a large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

Regular use



cookie for bank.com:
SessionID =
523FA4cd2E

Attack attempt



```
<form action=https://www.bank.com/transfer
  method=POST target=invisibleframe>
  <input name=recipient value=attacker>
  <input name=amount value=$10   CSRF??
</form>
<script>document.forms[0].submit()</script>
```

```
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100   &CSRF??
Cookie: SessionID=523FA4cd2E
```

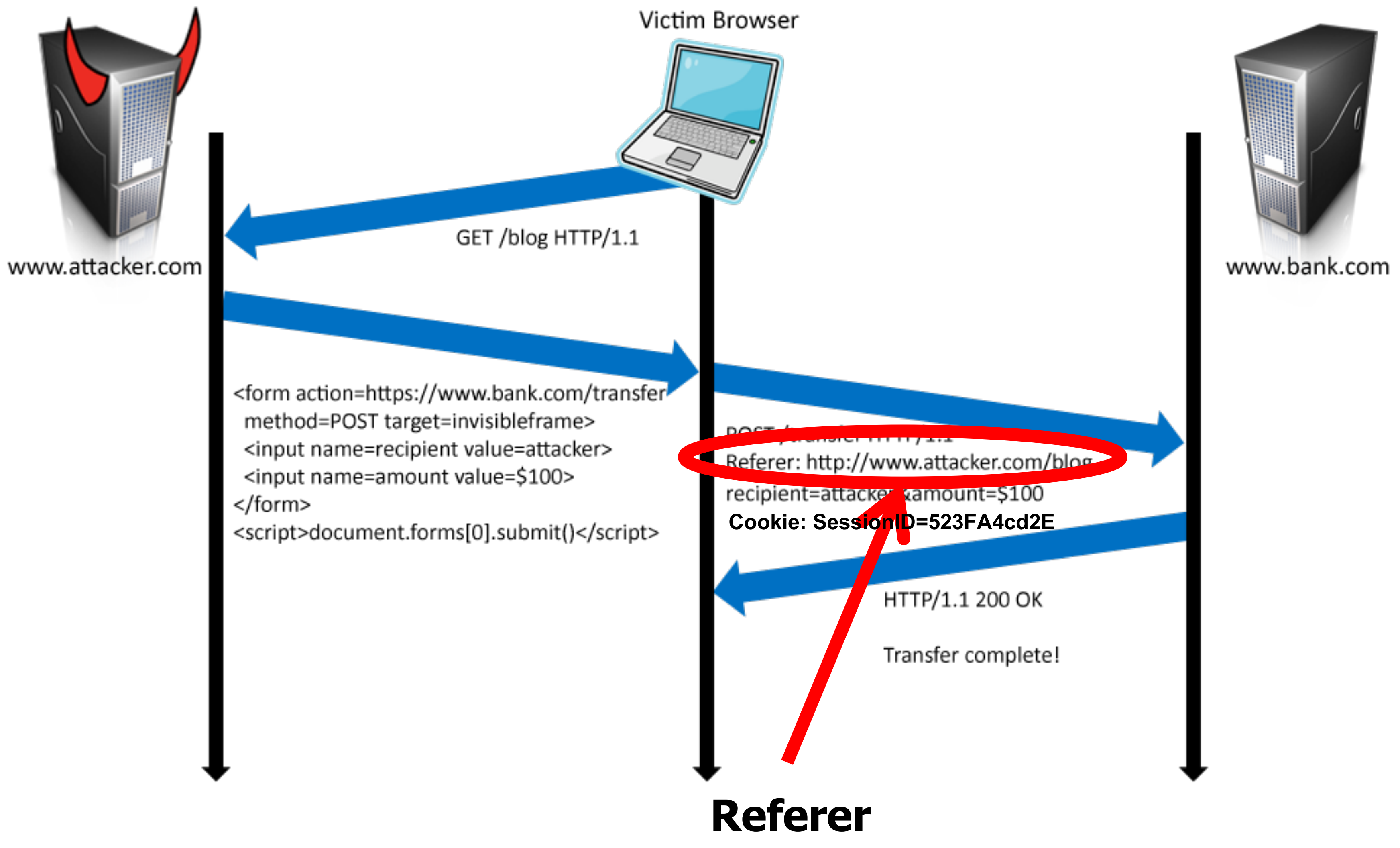
Session ID	CSRF token
523..	198..

doesn't know CSRF token for victim's session (one might not even be set at the server if the user did not request the form recently)

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

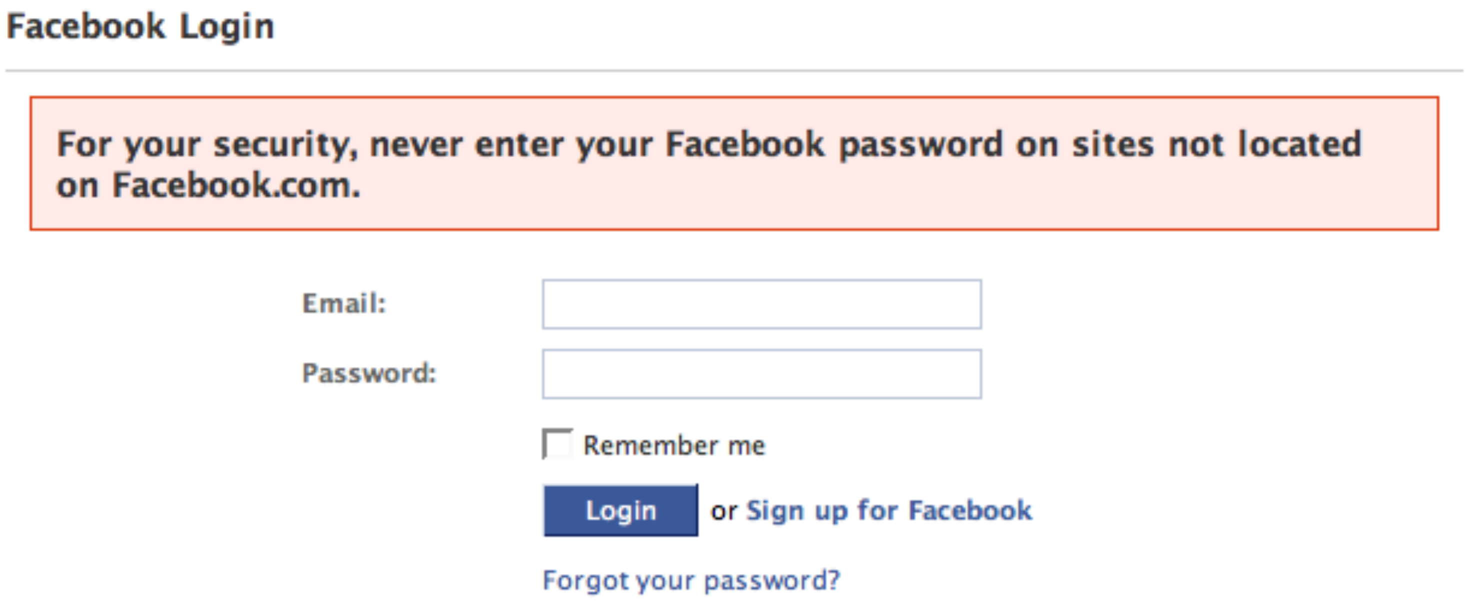
Refer header



```
<form action=https://www.bank.com/transfer
  method=POST target=invisibleframe>
  <input name=recipient value=attacker>
  <input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>
```

```
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100
Cookie: SessionID=523FA4cd2E
```

Referer Validation



Facebook Login

For your security, never enter your Facebook password on sites not located on Facebook.com.

Email:

Password:

Remember me

Referer Validation Defense

- HTTP Referer header
  - Referer: http://www.facebook.com/ ✓
  - Referer: http://www.attacker.com/evil.html ✗
  - Referer: [empty] ?
    - 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

Summary: CSRF

- CSRF attacks execute request on benign site because cookie is sent automatically
- Defenses for CSRF:
  - embed unpredictable token and check it later
  - check referer header in addition as defense in depth