

Web Security: Session management

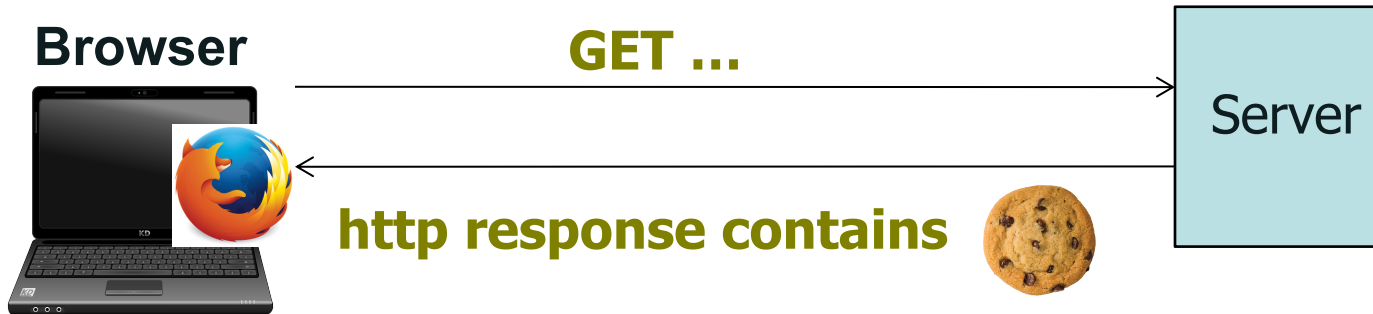
CS 161: Computer Security

Prof. Raluca Ada Popa

April 10, 2020

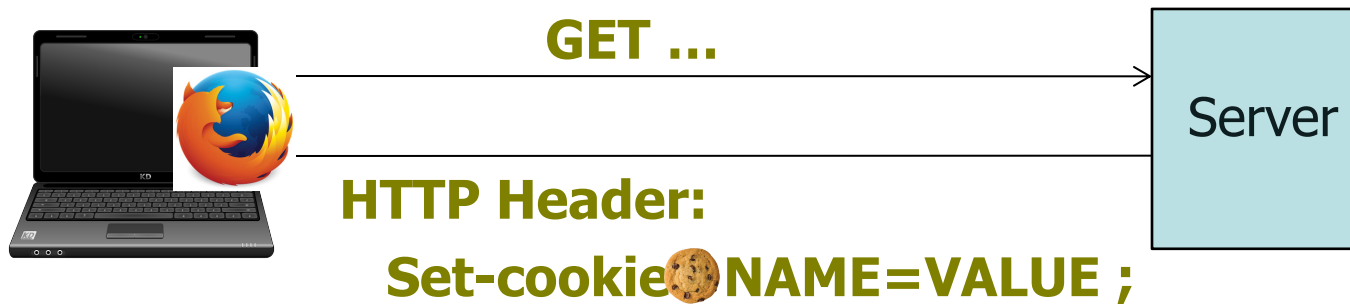
Cookies

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

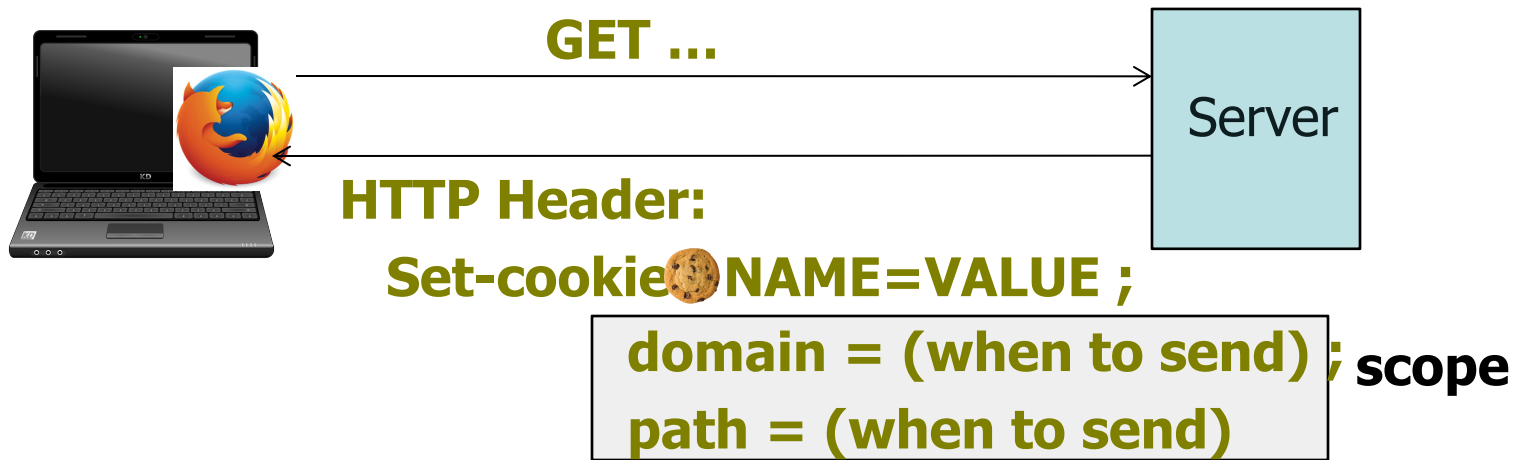
`document.cookie`

to see the cookie for that site

Each name=value is one cookie.

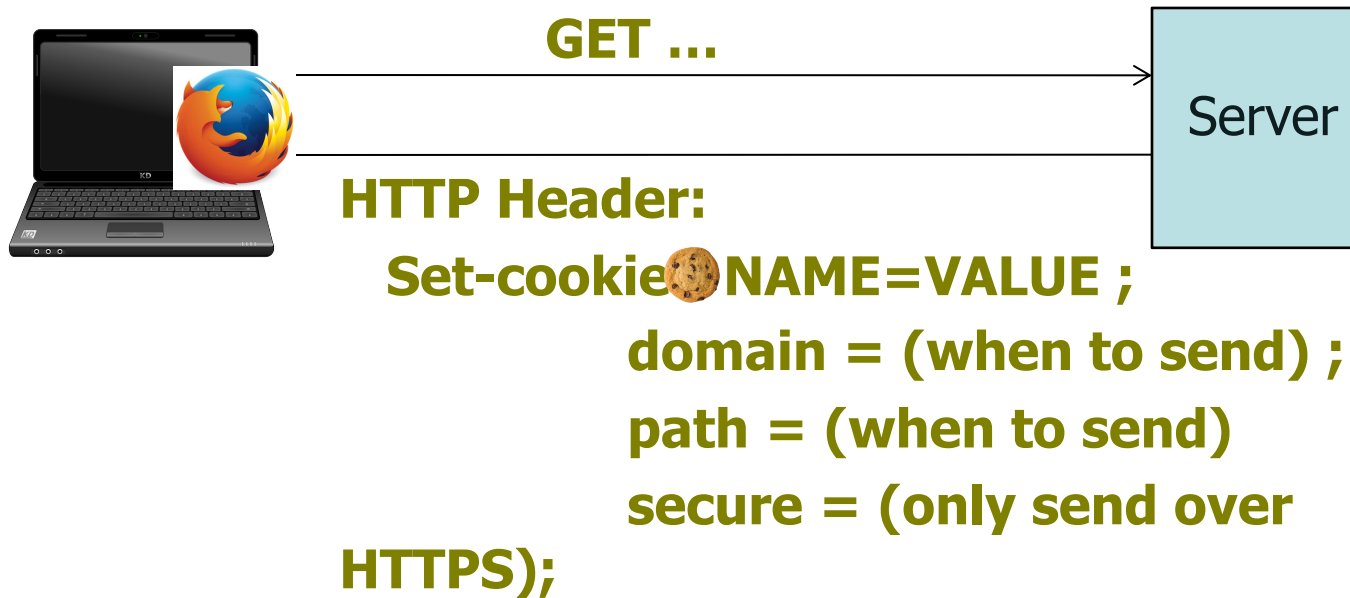
`document.cookie` lists all cookies **in scope for document**

Cookie scope



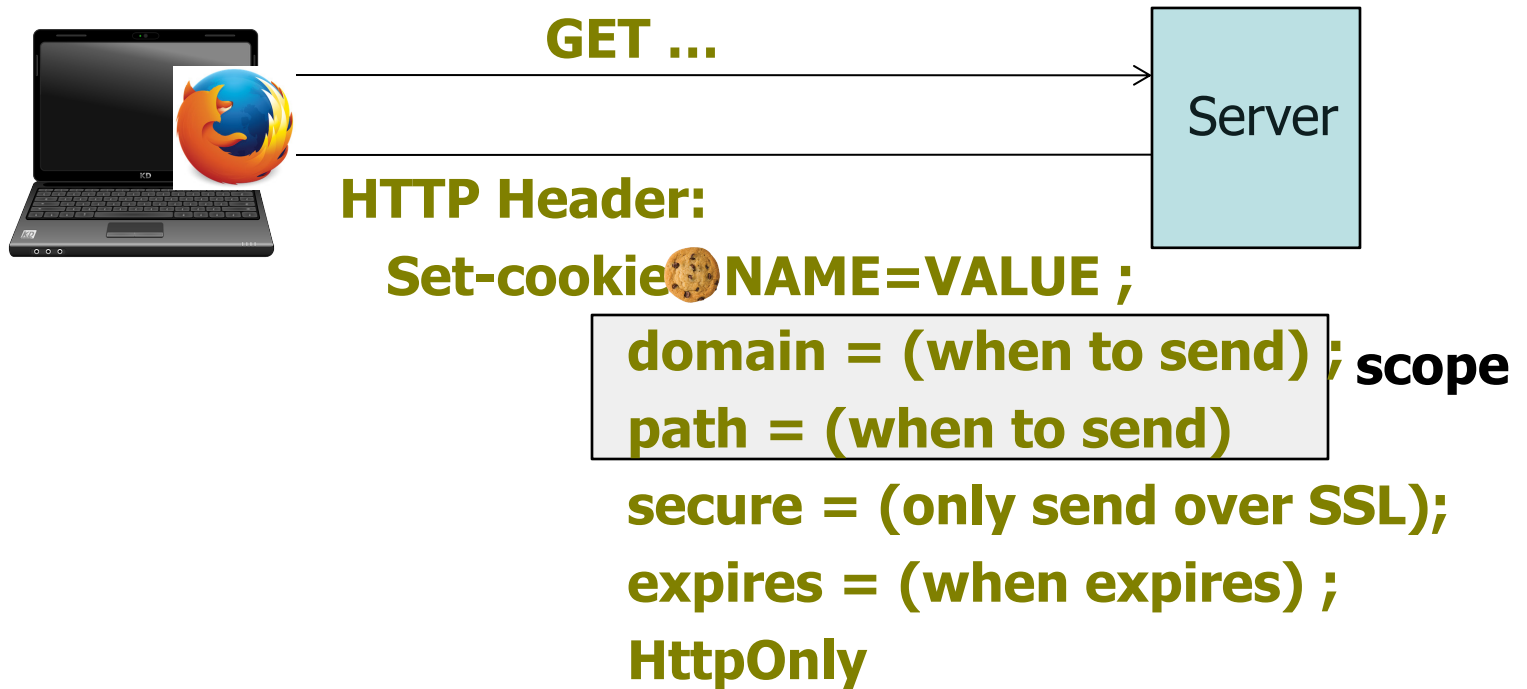
- When the browser connects to the same server later, it **automatically attaches** the cookies in scope: header containing the name and value, which the server can use to connect related requests.
- Domain and path inform the browser about which sites to send this cookie to

Cookie scope



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

Cookie scope



- **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

login.site.com

.site.com

disallowed domains

user.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**

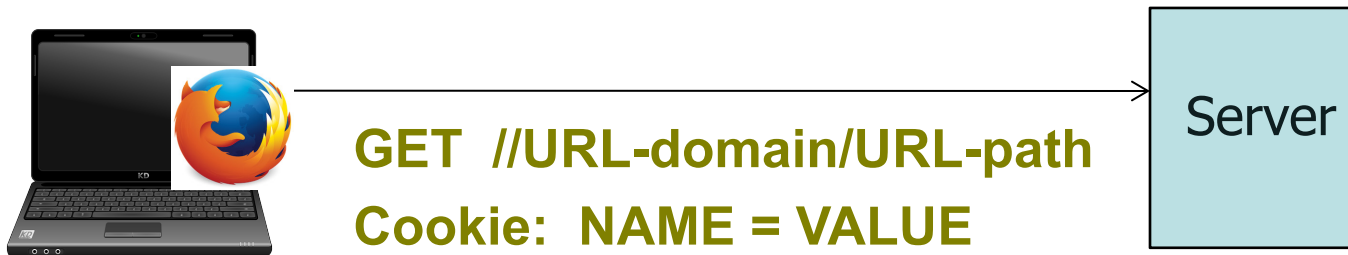
path: can be set to anything

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)	<i>foo.example.com</i> (exact)
<i>bar.foo.example.com</i>	Cookie not set: domain more specific than origin
<i>foo.example.com</i>	*. <i>foo.example.com</i>
<i>baz.example.com</i>	Cookie not set: domain mismatch
<i>example.com</i>	*. <i>example.com</i>
<i>ample.com</i>	Cookie not set: domain mismatch
<i>.com</i>	Cookie not set: domain too broad, security risk

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



GET //URL-domain/URL-path
Cookie: NAME = VALUE

Server

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?

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

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)	<code>foo.example.com</code> (exact) ?
<code>bar.foo.example.com</code>	Cookie not set: domain more specific than origin
<code>foo.example.com</code>	?
<code>baz.example.com</code>	Cookie not set: domain mismatch
<code>example.com</code>	?
<code>ample.com</code>	Cookie not set: domain mismatch
<code>.com</code>	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)	<code>foo.example.com</code> (exact)	<code>*.foo.example.com</code>
<code>bar.foo.example.com</code>	Cookie not set: domain more specific than origin	
<code>foo.example.com</code>	<code>*.foo.example.com</code>	
<code>baz.example.com</code>	Cookie not set: domain mismatch	
<code>example.com</code>	<code>*.example.com</code>	
<code>ample.com</code>	Cookie not set: domain mismatch	
<code>.com</code>	Cookie not set: domain too broad, security risk	

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/

http://login.site.com/

https://login.site.com/

cookie: userid=u2

cookie: userid=u2

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)

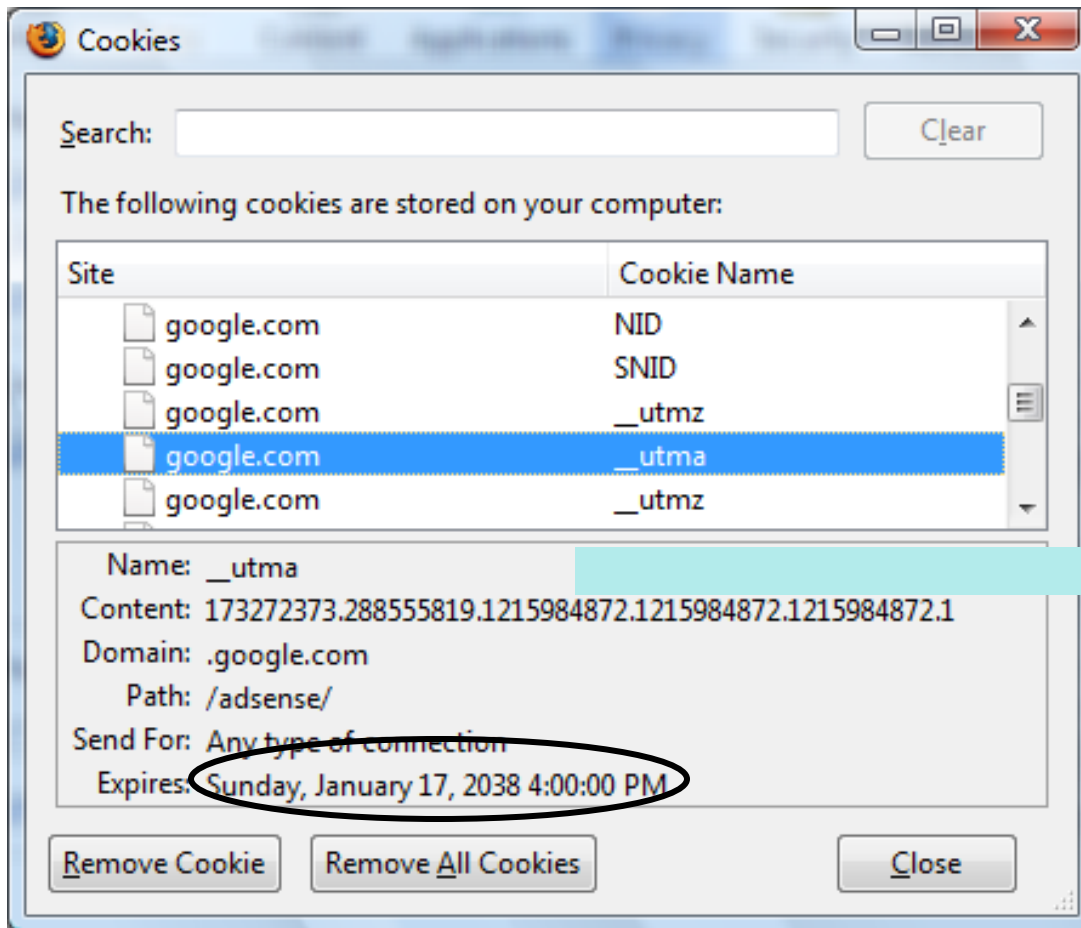
- Deleting a cookie:

```
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



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

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

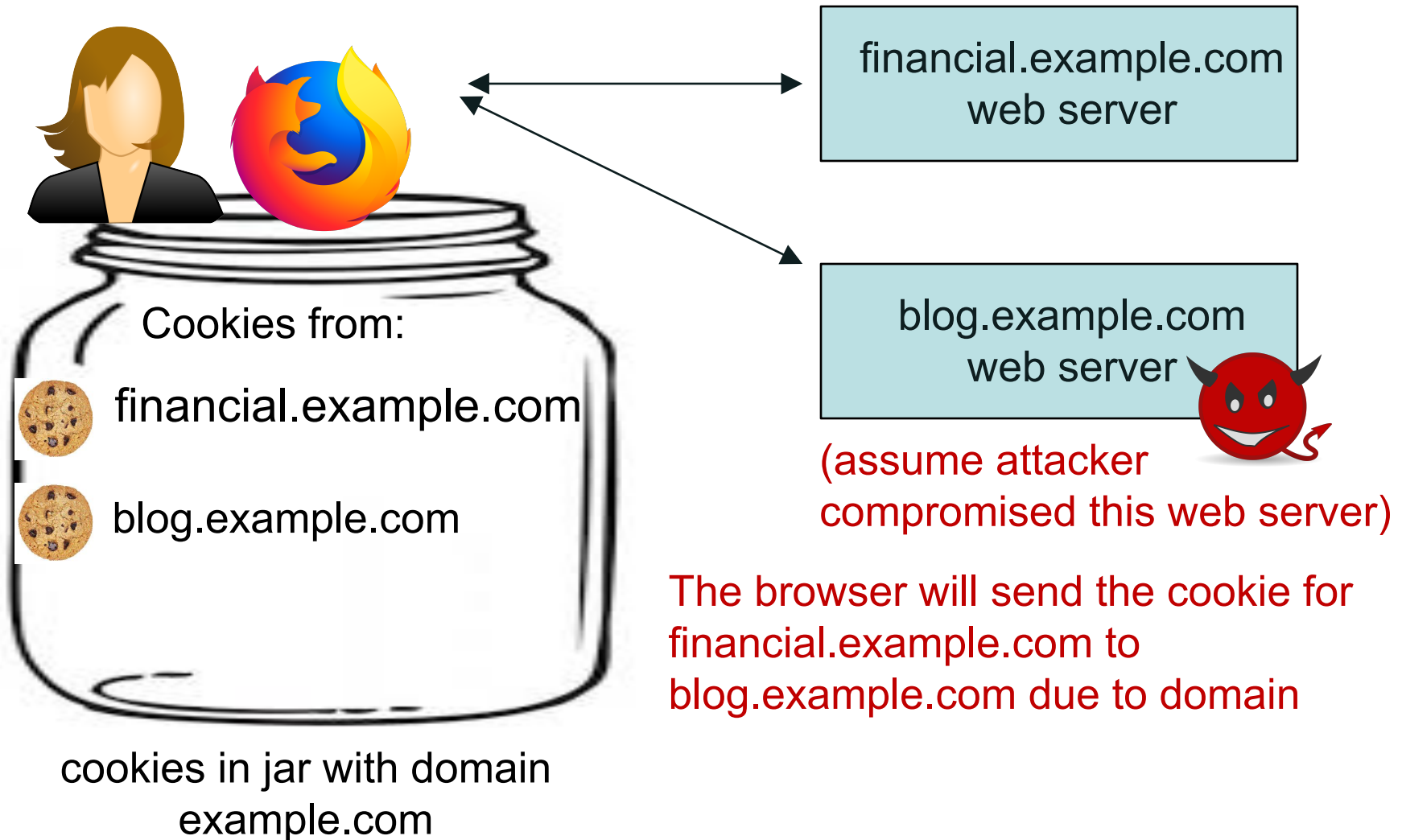
JS on each of these URLs can access the corresponding cookies even if the domains are not the same

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?

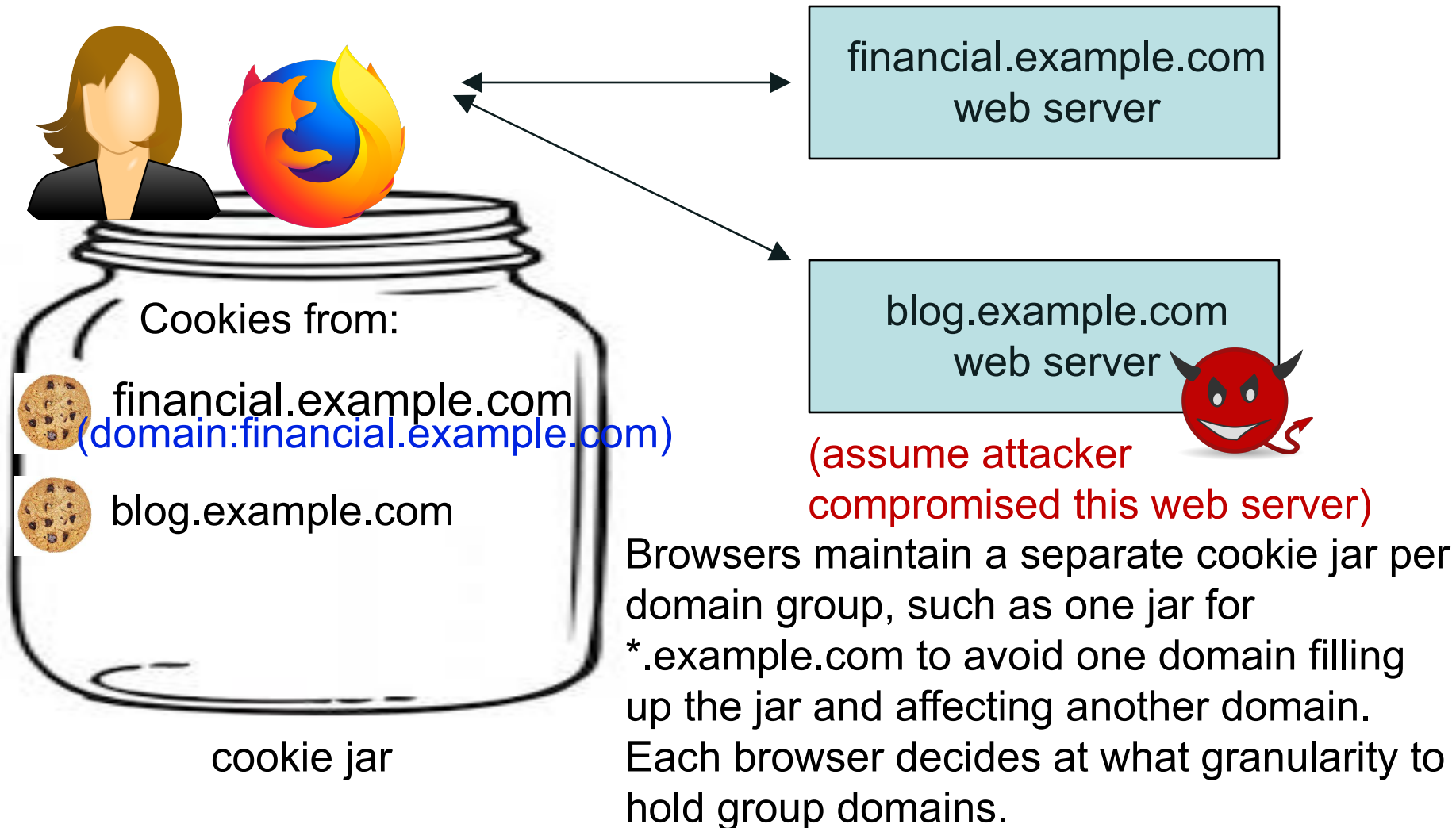
Example

Victim user browser



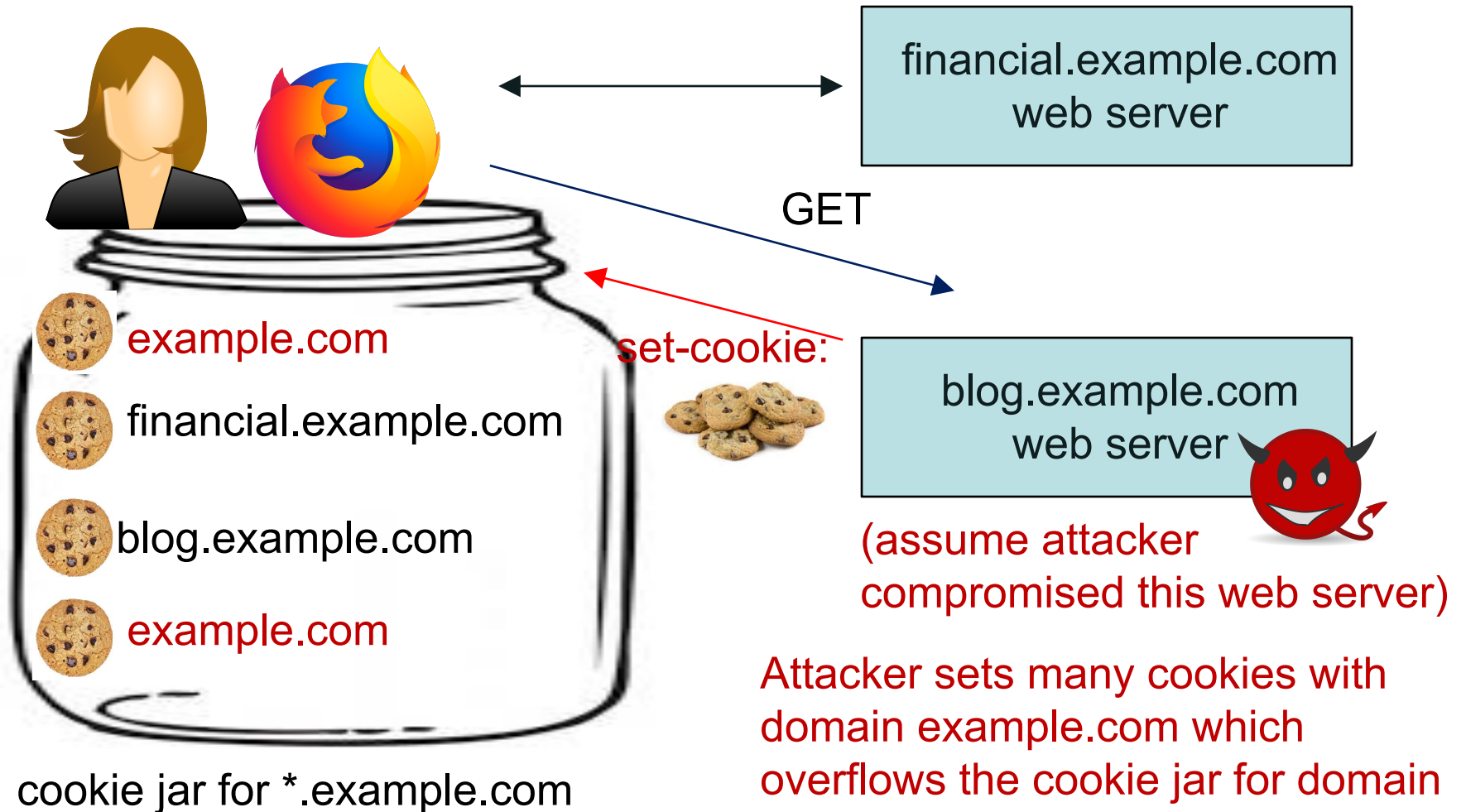
Example

Victim user browser



Example

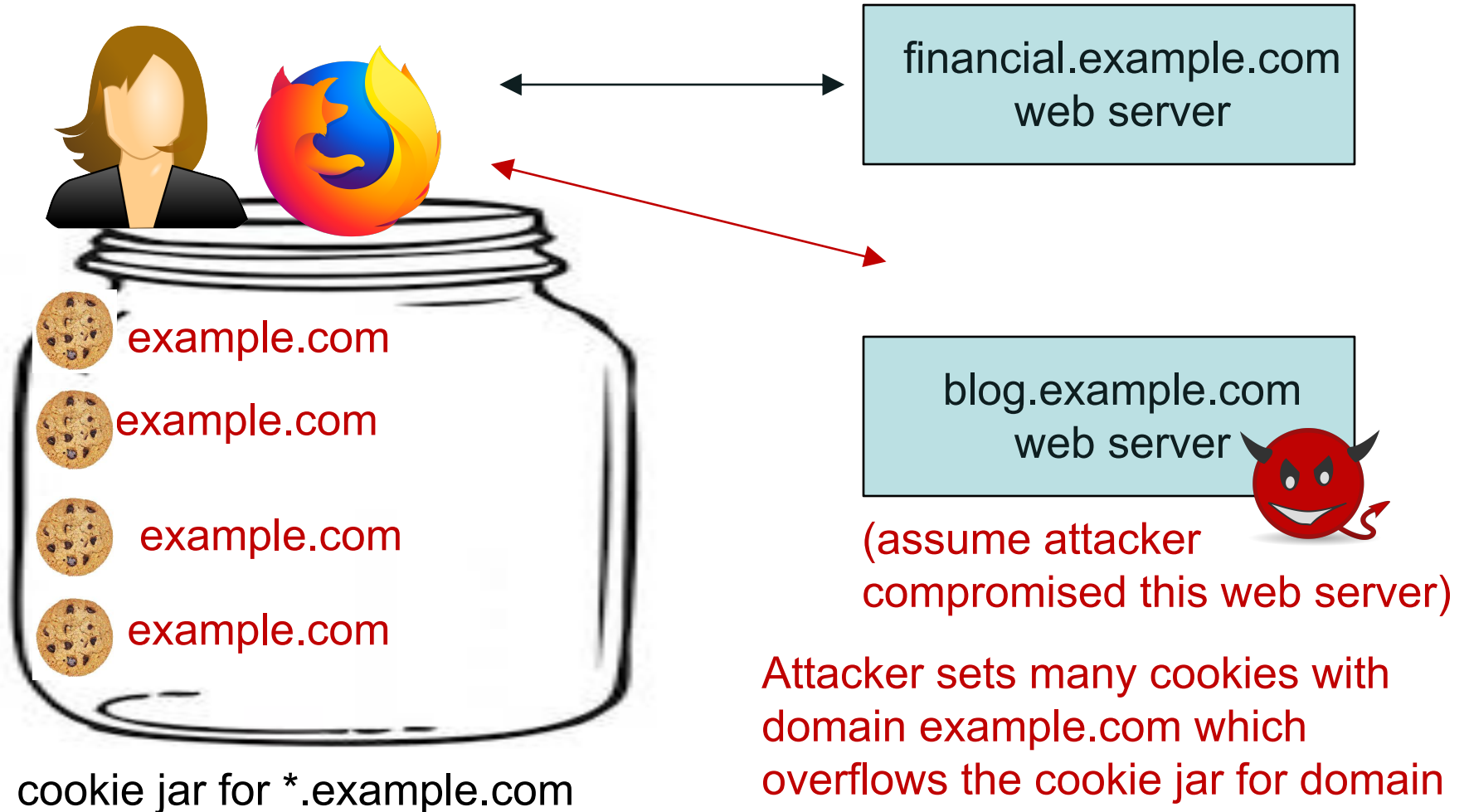
Victim user browser



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

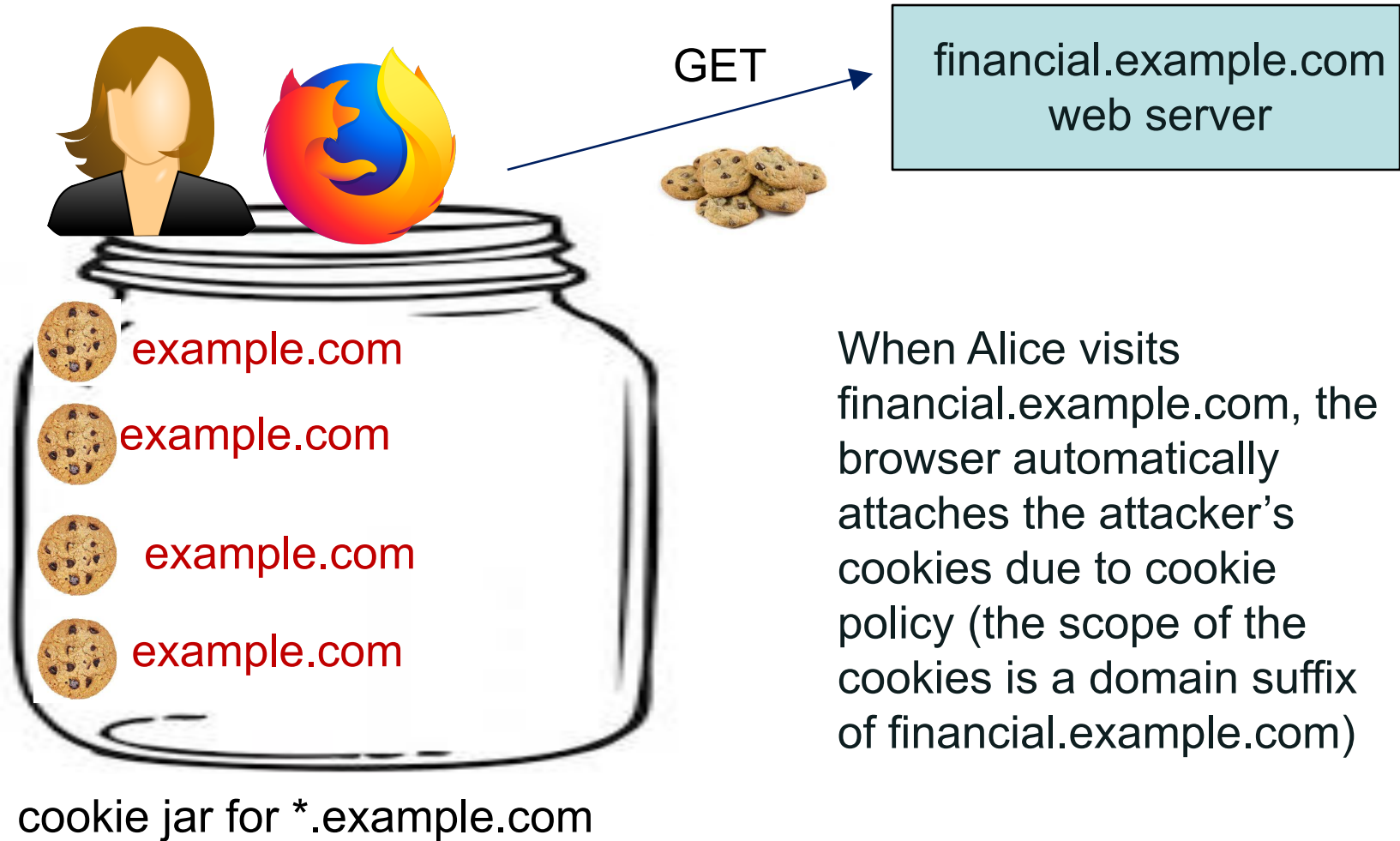
Victim user browser



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



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

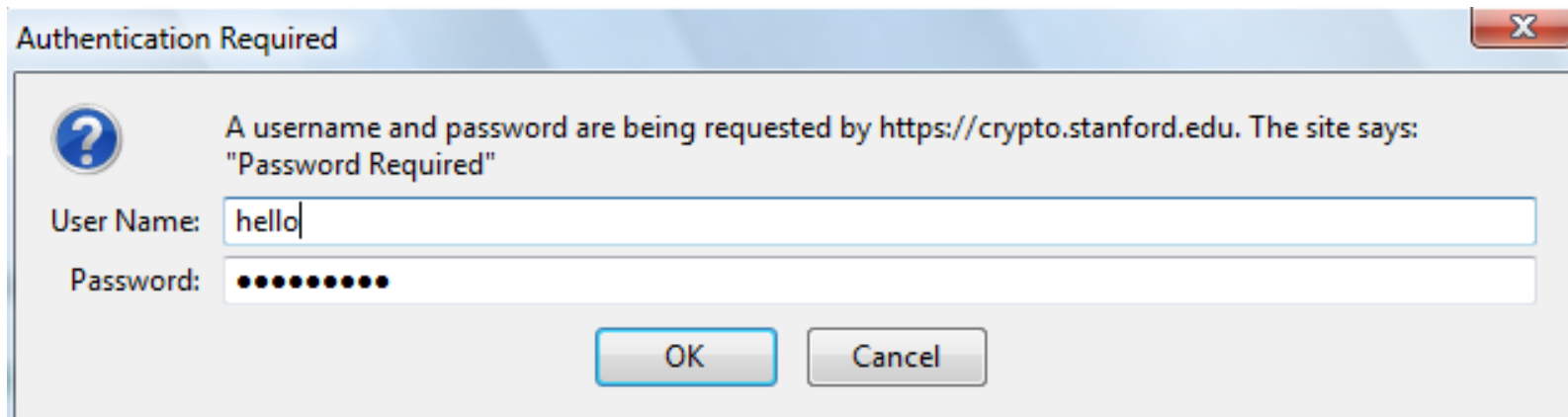
Pre-history: HTTP auth

One username and password for a group of users

HTTP request: `GET /index.html`

HTTP response contains:

`WWW-Authenticate: Basic realm="Password Required"`



Browsers sends hashed password on all subsequent HTTP requests:

`Authorization: Basic ZGFdfibzsdfgkjheczi1NXRIeHQ=`

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

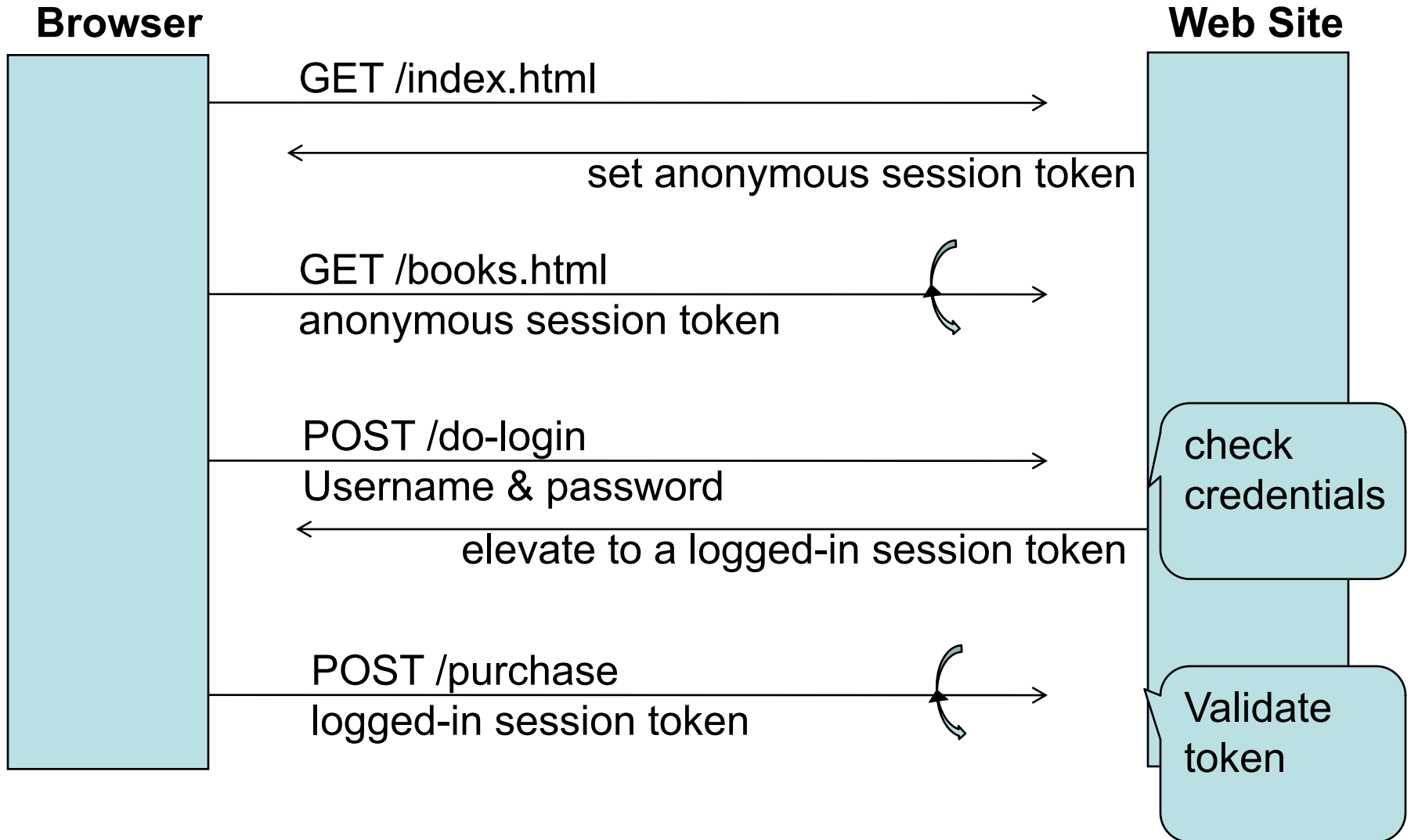
Actual Web

- 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)