CSE 127: Computer Security
Web security model

Nadia Heninger and Deian Stefan

Some slides adopted from Zakir Durumeric, Dan Boneh, and Kirill Levchenko
Lecture objectives

• Basic understanding of how the web works
• Understand relevant attacker models
• Understand browser same-origin policy
HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

```
https://cseweb.ucsd.edu/classes/fa19/cse127-ab/pa/pa1/#part-2-echo-in-x86-10-pts
```

Part 2: echo in x86 (10 pts)

Files for this sub-assignment are located in the `x86` subdirectory of the `student` user's home directory in the VM image; that is, `/home/student/x86`. SSH into the VM and `cd` into that directory to begin working on it.

For this part, you will be implementing a simplified version of the familiar `echo` command, using raw x86 assembly code. The goal of this assignment is to familiarize you with writing programs directly in x86.

Your `echo` command must behave as follows:

- When run with a single command line argument (e.g., `./echo Hello`):
HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):
HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

https://cseweb.ucsd.edu:443/classes/fa19/cse127-ab/lectures?nr=7&lang=en#slides
HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

  https://cseweb.ucsd.edu:443/classes/fa19/cse127-ab/lectures?nr=7&lang=en#slides
HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):
HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

```
https://cseweb.ucsd.edu:443/classes/fa19/cse127-ab/lectures?nr=7&lang=en#slides
```
HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

```
https://cseweb.ucsd.edu:443/classes/fa19/cse127-ab/lectures?nr=7&lang=en#slides
```
HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):

```
https://cseweb.ucsd.edu:443/classes/fa19/cse127-ab/lectures?hr=7&lang=en#slides
```
HTTP protocol

- Protocol from 1989 that allows fetching of resources (e.g., HTML documents)
- Resources have a uniform resource location (URL):
HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).
HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).
HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).
HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).
HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).

http://example.com
HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).
HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).
HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).
HTTP protocol

- Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).
• Clients and servers communicate by exchanging individual messages (as opposed to a stream of data).
Anatomy of a request

GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats
Anatomy of a request

method

GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats
Anatomy of a request

method | path
--- | ---
GET | /index.html

HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats
Anatomy of a request

**method** | **path** | **version**
--- | --- | ---
GET | /index.html | HTTP/1.1

Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats
Anatomy of a request

---

**method**

**GET**

**path**

/index.html

**version**

HTTP/1.1

**headers**

- **Accept:** image/gif, image/x-bitmap, image/jpeg, */*
- **Accept-Language:** en
- **Connection:** Keep-Alive
- **User-Agent:** Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
- **Host:** www.example.com
- **Referer:** http://www.google.com?q=dingbats
Anatomy of a request

```
GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats

(body (empty))
```
Anatomy of a response

HTTP/1.0  200 OK
Date:  Sun, 21 Apr 1996  02:20:42  GMT
Server:  Microsoft-Internet-Information-Server/5.0
Connection:  keep-alive
Content-Type:  text/html
Last-Modified:  Thu, 18 Apr 1996 17:39:05  GMT
Set-Cookie:  ...
Content-Length:  2543

<html>
  Some data... whatever ...  </html>
Anatomy of a response

status code

HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: ...
Content-Length: 2543

<html>Some data... whatever ... </html>
Anatomy of a response

HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: ...
Content-Length: 2543

<html>Some data... whatever ... </html>
Anatomy of a response

status code
HTTP/1.0 200 OK

headers
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: ...
Content-Length: 2543

body
<html>Some data... whatever ... </html>
Many HTTP methods

- **GET**: Get the resource at the specified URL.
- **POST**: Create new resource at URL with payload.
- **PUT**: Replace current representation of the target resource with request payload.
- **PATCH**: Update part of the resource.
- **DELETE**: Delete the specified URL.
In practice: it’s a mess

• GETs should NOT change server state; in practice, some servers do perform side effects

• Old browsers don’t send PUT, PATCH, and DELETE

  ➤ So, almost all side-effecting requests are POSTs; real method hidden in a header or request body
HTTP/2

- Major revision of HTTP released in 2015
- Based on Google SPDY Protocol
- No major changes in how applications are structured
  Major changes (mostly performance):
  - Allows pipelining requests for multiple objects
  - Multiplexing multiple requests over one TCP connection
  - Header compression
  - Server push
Making HTTP stateful: cookies

- HTTP cookie: small piece of data that a server sends to the browser, who stores it and sends it back with subsequent requests

- What is this useful for?
  - Session management: logins, shopping carts, etc.
  - Personalization: user preferences, themes, etc.
  - Tracking: recording and analyzing user behavior
Setting cookies in response

HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: trackingID=3272923427328234
Set-Cookie: userID=F3D947C2
Content-Length: 2543

<html>Some data... whatever ... </html>
Setting cookies in response

HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: trackingID=3272923427328234
Set-Cookie: userID=F3D947C2
Content-Length: 2543

<html>Some data... whatever ... </html>
Sending cookie with each request

GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Cookie: trackingID=3272923427328234
Cookie: userID=F3D947C2
Host: www.example.com
Referer: http://www.google.com?q=dingbats
Sending cookie with each request

GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Cookie: trackingID=3272923427328234
Cookie: userID=F3D947C2
Host: www.example.com
Referer: http://www.google.com?q=dingbats
Basic browser execution model

- Each browser window:
  - Loads content
  - Parses HTML and runs Javascript
  - Fetches sub resources (e.g., images, CSS, Javascript)
  - Respond to events like onClick, onMouseover, onLoad, setTimeout
Nested execution model

- Windows may contain frames from different sources
  - Frame: rigid visible division
  - iFrame: floating inline frame
Nested execution model

• Windows may contain frames from different sources
  ➤ Frame: rigid visible division
  ➤ iFrame: floating inline frame

• Why use frames?
  ➤ Delegate screen area to content from another source
  ➤ Browser provides isolation based on frames
  ➤ Parent may work even if frame is broken
Document object model (DOM)

- Javascript can read and modify page by interacting with DOM
  - Object Oriented interface for reading and writing website content
- Includes browser object model
  - Access window, document, and other state like history, browser navigation, and cookies
Modifying the DOM using JS

<html>
  <body>
    <ul id="t1">
      <li>Item 1</li>
    </ul>
  </body>
</html>
Modifying the DOM using JS

```html
<html>
  <body>
    <ul id="t1">
      <li>Item 1</li>
    </ul>
  
  ...
  
  </body>
</html>

<script>
  const list = document.getElementById('t1');
  const newItem = document.createElement('li');
  const newText = document.createTextNode('Item 2');
  list.appendChild(newItem);
  newItem.appendChild(newText);
</script>

• Item 1
Modifying the DOM using JS

```
<html>
  <body>
    <ul id="t1">
      <li>Item 1</li>
    </ul>
    ...  
  </body>
</html>

<script>
  const list    = document.getElementById('t1');
  const newItem = document.createElement('li');
  const newText = document.createTextNode('Item 2');
  list.appendChild(newItem);
  newItem.appendChild(newText)
</script>

- Item 1
- Item 2
Modern websites are complicated
Modern websites are complicated

The LA Times homepage includes 540 resources from nearly 270 IP addresses, 58 networks, and 8 countries. Many of these aren’t controlled by the main sites.
Modern websites are complicated

- Third party ad
- Google analytics
- Framed ad
- jQuery library
- Local scripts
- Extensions
Lecture objectives

- Basic understanding of how the web works
- Understand relevant attacker models
- Understand browser same-origin policy
Relevant attacker models

Network attacker

http://example.com
Relevant attacker models

Network attacker

Web attacker

https://evil.com

http://example.com
Relevant attacker models

Gadget attacker
Web attacker with capabilities to inject limited content into honest page
Relevant attacker models

Gadget attacker
Web attacker with capabilities to inject limited content into honest page
Relevant attacker models

Gadget attacker
Web attacker with capabilities to inject limited content into honest page

https://example.com
evil.com
example.com
Relevant attacker models

Gadget attacker
Web attacker with capabilities to inject limited content into honest page
Relevant attacker models

Gadget attacker
Web attacker with capabilities to inject limited content into honest page
Most of our focus: web attacker model
And variants of it
Web security

- Safely browse the web in the presence of web attackers
  ➤ The browser is the new OS analogy
Web security

- Safely browse the web in the presence of web attackers
  - The browser is the new OS analogy

Filesystem

Cookies/HTML5 local storage
Web security

- Safely browse the web in the presence of web attackers
  - The browser is the new OS analogy
Web security

- Safely browse the web in the presence of web attackers

➤ The browser is the new OS analogy
Web security

- Safely browse the web in the presence of web attackers

➤ The browser is the new OS analogy

Process 1
- skype
- VM + UIDs + seccomp-bpf

Process 2
- keypassx

Page 1
- 4chan.org
- SOP

Page 2
- bank.ch
- SOP

Filesystem
- UIDs + ACLs

Cookies/HTML5 local storage
Same origin policy (SOP)

- **Origin**: isolation unit/trust boundary on the web
  - (scheme, domain, port) triple derived from URL
- **SOP goal**: isolate content of different origins
  - Confidentiality: script contained in `evil.com` should not be able to read data in `bank.ch` page
  - Integrity: script from `evil.com` should not be able to modify the content of `bank.ch` page
SOP for the DOM

- Each frame in a window has its own origin
- Frame can only access data with the same origin
  ➤ DOM tree, local storage, cookies, etc.
SOP for the DOM

- Each frame in a window has its own origin
- Frame can only access data with the same origin
  - DOM tree, local storage, cookies, etc.
SOP for the DOM

- Each frame in a window has its own origin
- Frame can only access data with the same origin
  - DOM tree, local storage, cookies, etc.
SOP for the DOM

• Each frame in a window has its own origin
• Frame can only access data with the same origin
  ➤ DOM tree, local storage, cookies, etc.
How do you communicate with frames?

- Message passing via `postMessage` API
  
  **Sender:**

  ```javascript
  targetWindow.postMessage(message, targetOrigin);
  ```

  **Receiver:**

  ```javascript
  window.addEventListener("message", receiveMessage, false);
  function receiveMessage(event){
    if (event.origin !== "http://example.com")
      return;
    ...
  }
  ```
SOP for HTTP responses

- Pages can perform requests across origins
  - SOP does not prevent a page from leaking data to another origin by encoding it in the URL, request body, etc.

- SOP prevents code from directly inspecting HTTP responses
  - Except for documents, can often learn some information about the response
Documents

- Can load cross-origin HTML in frames, but not inspect or modify the frame content.
Documents

- Can load cross-origin HTML in frames, but not inspect or modify the frame content.
Documents

- Can load cross-origin HTML in frames, but not inspect or modify the frame content.
Documents

- Can load cross-origin HTML in frames, but not inspect or modify the frame content.
• Can load cross-origin HTML in frames, but not inspect or modify the frame content.
Scripts

- Can load scripts from across origins
- Scripts execute with privileges of the page
- Page can see source via `func.toString()`
Scripts

- Can load scripts from across origins
- Scripts execute with privileges of the page
- Page can see source via `func.toString()`
Scripts

- Can load scripts from across origins
- Scripts execute with privileges of the page
- Page can see source via `func.toString()`
Images

- Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels
- Page can see `img.width`
Images

- Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels
- Page can see `img.width`

```
https://a.com
(https://a.com, 443)
```

```
(https://fb.com, 443)
```

```
(https://a.com, 443)
```
Images

- Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels
- Page can see img.width

```javascript
if (loggedIn(user))
    then
else
```
Images

- Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels
- Page can see `img.width`

```javascript
if loggedIn(user) then       else
```
Images

- Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels
- Page can see `img.width`

```javascript
if (loggedIn(user))
    then
        else
            40px
            80px
```

(https://a.com, 443)

(https://fb.com, 443)

(https://a.com, 443)
Images

- Browser renders cross-origin images, but SOP prevents page from inspecting individual pixels.
- Page can see `img.width`.

```javascript
if (img.width > 40) { ... } else { ... }

if (loggedIn(user)) then
else
```
SOP for fonts and CSS are similar.
SOP for cookies

- Cookies allow server to store small piece of data on the client
- Client sends cookie back to server next time the client loads a page
- Sending cookies only to the right websites really important
  - Don’t send cookie for bank.com to attacker.com if authentication token
SOP for cookies

- Cookies use a separate definition of origins.
- DOM SOP: origin is a (scheme, domain, port)
- Cookie SOP: ([scheme], domain, path)
  - (https,cseweb.ucsd.edu, /classes/fa19/cse127-ab)
SOP: Cookie scope setting

- A page can set a cookie for its own domain or any parent domain, as long as the parent domain is not a public suffix.
- The browser will make a cookie available to the given domain including any sub-domains.

<table>
<thead>
<tr>
<th>Subdomain</th>
<th>Allowed</th>
<th>Disallowed</th>
</tr>
</thead>
<tbody>
<tr>
<td>login.site.com</td>
<td></td>
<td>other.site.com</td>
</tr>
<tr>
<td>site.com</td>
<td></td>
<td>com</td>
</tr>
<tr>
<td>othersite.com</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SOP: Cookie scope setting

- A page can set a cookie for its own domain or any parent domain, as long as the parent domain is not a public suffix.
- The browser will make a cookie available to the given domain including any sub-domains.

Example:
- \texttt{cseweb.ucsd.edu} can set cookies for \texttt{ucsd.edu} (unless \texttt{ucsd.edu} is on public suffix list)
SOP: Cookie scope setting

PUBLIC SUFFIX LIST

A "public suffix" is one under which Internet users can (or historically could) directly register names. Some examples of public suffixes are .com, .co.uk and pvt.k12.ma.us. The Public Suffix List is a list of all known public suffixes.

The Public Suffix List is an initiative of Mozilla, but is maintained as a community resource. It is available for use in any software, but was originally created to meet the needs of browser manufacturers. It allows browsers to, for example:

- Avoid privacy-damaging "supercookies" being set for high-level domain name suffixes
- Highlight the most important part of a domain name in the user interface
- Accurately sort history entries by site

We maintain a fuller (although not exhaustive) list of what people are using it for. If you are using it for something else, you are encouraged to tell us, because it helps us to assess the potential impact of changes. For that, you can use the psl-discuss mailing list, where we consider issues related to the maintenance, format and semantics of the list. Note: please do not use this mailing list to request amendments to the PSL's data.

It is in the interest of Internet registries to see that their section of the list is up to date. If it is not, their customers may have trouble setting cookies, or data about their sites may display sub-optimally. So we encourage them to maintain their section of the list by submitting amendments.
// ===BEGIN ICANN DOMAINS===

ac
cm.ac
edu.ac
gov.ac
net.ac
mil.ac
org.ac

// ad : https://en.wikipedia.org/wiki/.ad
ad
nom.ad

// ae : https://en.wikipedia.org/wiki/.ae
// see also: "Domain Name Eligibility Policy" at http://www.aeda.ae/eng/aepolicy.php
ae
c.o.ae
net.ae
org.ae
sch.ae
ac.ae
ac.ae
gov.ae
mil.ae

// aero : see https://www.information.aero/index.php?id=66
aero
accident-investigation.aero
accident-prevention.aero
aerobatic.aero
aeroclub.aero
aerodrome.aero
agents.aero
aircraft.aero
airline.aero
What cookies are sent?

- Browser always sends all cookies in a URL’s scope:
  - Cookie’s domain is domain suffix of URL’s domain
  - Cookie’s path is a prefix of the URL path
Cookie scoping example

**Cookie 1:**
- name = mycookie
- value = mycookievalue
- domain = login.site.com
- path = /

**Cookie 2:**
- name = cookie2
- value = mycookievalue
- domain = site.com
- path = /

**Cookie 3:**
- name = cookie3
- value = mycookievalue
- domain = site.com
- path = /my/home
<table>
<thead>
<tr>
<th></th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>login.site.com</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>login.site.com/my/home</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>site.com/my</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

**Cookie 1:**
- name = mycookie
- value = mycookievalue
- domain = login.site.com
- path = /

**Cookie 2:**
- name = cookie2
- value = mycookievalue
- domain = site.com
- path = /

**Cookie 3:**
- name = cookie3
- value = mycookievalue
- domain = site.com
- path = /my/home
Cookie scoping example

**Cookie 1:**
- name = mycookie
- value = mycookievalue
- domain = login.site.com
- path = /

**Cookie 2:**
- name = cookie2
- value = mycookievalue
- domain = site.com
- path = /

**Cookie 3:**
- name = cookie3
- value = mycookievalue
- domain = site.com
- path = /my/home

<table>
<thead>
<tr>
<th>checkout.site.com</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Cookie scoping example**

**Cookie 1:**
- name = mycookie
- value = mycookievalue
- domain = login.site.com
- path = /

**Cookie 2:**
- name = cookie2
- value = mycookievalue
- domain = site.com
- path = /

**Cookie 3:**
- name = cookie3
- value = mycookievalue
- domain = site.com
- path = /my/home

<table>
<thead>
<tr>
<th>Domain</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cookie scoping example

Cookie 1:
name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:
name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:
name = cookie3
value = mycookievalue
domain = site.com
path = /my/home

<table>
<thead>
<tr>
<th>Domain</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
### Cookie scoping example

<table>
<thead>
<tr>
<th>Cookie 1:</th>
<th>Cookie 2:</th>
<th>Cookie 3:</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = mycookie</td>
<td>name = cookie2</td>
<td>name = cookie3</td>
</tr>
<tr>
<td>value = mycookievalue</td>
<td>value = mycookievalue</td>
<td>value = mycookievalue</td>
</tr>
<tr>
<td>domain = login.site.com</td>
<td>domain = site.com</td>
<td>domain = site.com</td>
</tr>
<tr>
<td>path = /</td>
<td>path = /</td>
<td>path = /my/home</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URL</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cookie scoping example

<table>
<thead>
<tr>
<th></th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cookie scoping example

**Cookie 1:**
- name = mycookie
- value = mycookievalue
- domain = login.site.com
- path = /

**Cookie 2:**
- name = cookie2
- value = mycookievalue
- domain = site.com
- path = /

**Cookie 3:**
- name = cookie3
- value = mycookievalue
- domain = site.com
- path = /my/home

<table>
<thead>
<tr>
<th>Domain</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>


Cookie scoping example

Cookie 1:
name = mycookie
value = mycookievalue
domain = login.site.com
path = /

Cookie 2:
name = cookie2
value = mycookievalue
domain = site.com
path = /

Cookie 3:
name = cookie3
value = mycookievalue
domain = site.com
path = /my/home

<table>
<thead>
<tr>
<th>URL</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Cookie scoping example

<table>
<thead>
<tr>
<th>Domain</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com/my/home</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Cookie scoping example

<table>
<thead>
<tr>
<th>Domain</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com/my/home</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Cookie scoping example

**Cookie 1:**
- name = mycookie
- value = mycookievalue
- domain = login.site.com
- path = /

**Cookie 2:**
- name = cookie2
- value = mycookievalue
- domain = site.com
- path = /

**Cookie 3:**
- name = cookie3
- value = mycookievalue
- domain = site.com
- path = /my/home

<table>
<thead>
<tr>
<th>Domain</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com/my/home</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
### Cookie scoping example

**Cookie 1:**
- name = mycookie
- value = mycookievalue
- domain = login.site.com
- path = /

**Cookie 2:**
- name = cookie2
- value = mycookievalue
- domain = site.com
- path = /

**Cookie 3:**
- name = cookie3
- value = mycookievalue
- domain = site.com
- path = /my/home

<table>
<thead>
<tr>
<th>Domain</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com/my/home</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Cookie scoping example

<table>
<thead>
<tr>
<th>Domain</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com/my/home</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>site.com/my</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Cookie scoping example

<table>
<thead>
<tr>
<th>Domain</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com/my/home</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>site.com/my</td>
<td>No</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Cookie scoping example**

**Cookie 1:**
- name = mycookie
- value = mycookievalue
- domain = login.site.com
- path = /

**Cookie 2:**
- name = cookie2
- value = mycookievalue
- domain = site.com
- path = /

**Cookie 3:**
- name = cookie3
- value = mycookievalue
- domain = site.com
- path = /my/home

<table>
<thead>
<tr>
<th>Domain</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com/my/home</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>site.com/my</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
## Cookie scoping example

<table>
<thead>
<tr>
<th>Domain</th>
<th>Cookie 1</th>
<th>Cookie 2</th>
<th>Cookie 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkout.site.com</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>login.site.com/my/home</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>site.com/my</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Which cookies are sent?

https://evil.com

http://bank.ch

http://4chan.org

http://bank.ch
Which cookies are sent?

```
<html>
  <img src="https://bank.ch"/>
</html>
```
Which cookies are sent?

```html
<html>
  <img src="https://bank.ch"/>
</html>
```
Why is this bad?

```html
<html>
  <img src="https://bank.ch/transfer?amt=$1B&to=evil"/>
</html>
```

Cross-site request forgery (CSRF) attack!
SameSite cookies

Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; SameSite=Strict;

A same-site cookie is only sent when the request originates from the same site (top-level domain)
Cookies are always sent! So?

- Network attacker can steal cookies if server allows unencrypted HTTP traffic

- Don’t need to wait for user to go to the site; web attacker can make cross-origin request
Secure cookies

A secure cookie is only sent to the server with an encrypted request over the HTTPS protocol.
Does the cookie path give us finer-grained (than origin) isolation?
No!

- **Cookie SOP:**
  - cseweb.ucsd.edu/~dstefan does not see cookies for cseweb.ucsd.edu/~nadiah

- **DOM SOP:**
  - cseweb.ucsd.edu/~dstefan can access the DOM of cseweb.ucsd.edu/~nadiah
  - How can you access cookie?
No!

- **Cookie SOP:**
  - cseweb.ucsd.edu/~dstefan does not see cookies for cseweb.ucsd.edu/~nadiah

- **DOM SOP:**
  - cseweb.ucsd.edu/~dstefan can access the DOM of cseweb.ucsd.edu/~nadiah
  - How can you access cookie?

```javascript
const iframe = document.createElement("iframe");
iframe.src = "https://cseweb.ucsd.edu/~nadiah";
document.body.appendChild(iframe);
alert(iframe.contentWindow.document.cookie);
```
Bank uses Google analytics

• What happens when your bank includes Google Analytics Javascript? Can it access your Bank’s authentication cookie?
  ➤ Yes! Javascript is running with origin’s privileges. Can access `document.cookie`.

• SOP doesn’t prevent leaking data:

```javascript
const img = document.createElement("image");
document.body.appendChild(img);
```
HttpOnly cookies

Set-Cookie: id=a3fWa; Expires=Wed, 21 Oct 2015 07:28:00 GMT; HttpOnly;

Don’t expose cookie in document.cookie
Wednesday