Today

• We need to cover same-origin policy, cookie policy, CSRF and XSS, but do not need to cover web injection
• Scribe: Dayeol
• Presenter: Rohan, Michael
HTTP
(Hypertext Transfer Protocol)
A common data communication protocol on the web

HTTP REQUEST:
GET /account.html HTTP/1.1
Host: www.safebank.com

HTTP RESPONSE:
HTTP/1.0 200 OK
<HTML> . . . </HTML>
URLs

Global identifiers of network-retrievable resources

Example:

http://safebank.com:81/account?id=10#statement
HTTP

HTTP REQUEST:
GET /account.html HTTP/1.1
Host: www.safebank.com

HTTP RESPONSE:
HTTP/1.0 200 OK
<HTML> . . . </HTML>
HTTP Request

GET: no side effect
POST: possible side effect

GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap,
       image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Chrome/21.0.1180.75 (Macintosh;
            Intel Mac OS X 10_7_4)
Host: www.safebank.com
Referer: http://www.google.com?q=dingbats

Data – none for GET
HTTP

HTTP REQUEST:
GET /account.html HTTP/1.1
Host: www.safebank.com

HTTP RESPONSE:
HTTP/1.0 200 OK
<HTML> . . . </HTML>
HTTP Response

HTTP version  Status code  Reason phrase

HTTP/1.0  200  OK
Date:  Sun, 12 Aug 2012  02:20:42  GMT
Server:  Microsoft-Internet-Information-Server/5.0
Connection:  keep-alive
Content-Type:  text/html
Last-Modified:  Thu, 9 Aug 2012  17:39:05  GMT
Set-Cookie:...
Content-Length:  2543

Data

<HTML>  This is web content formatted using html  </HTML>

Can be a webpage
HTML

A language to create structured documents
One can embed images, objects, or create interactive forms

index.html
<html>
<body>
  <div>
    foo
    <a href="http://google.com">Go to Google!</a>
  </div>
  <form>
    <input type="text" />
    <input type="radio" />
    <input type="checkbox" />
  </form>
</body>
</html>
CSS (Cascading Style Sheets)

Style sheet language used for describing the presentation of a document

```css
.index.css

p.serif {
font-family: "Times New Roman", Times, serif;
}
p.sansserif {
font-family: Arial, Helvetica, sans-serif;
}
```
Javascript

Programming language used to manipulate web pages. It is a high-level, untyped and interpreted language with support for objects.

Supported by all web browsers

```html
<script>
function myFunction() {
document.getElementById("demo").innerHTML = "Text changed.";
}
</script>

Very powerful!
HTTP REQUEST:
GET /account.html HTTP/1.1
Host: www.safebank.com

HTTP RESPONSE:
HTTP/1.0 200 OK
<HTML> . . . </HTML>
DOM (Document Object Model)

a cross-platform model for representing and interacting with objects in HTML

HTML

```html
<html>
  <body>
    <div>foo</div>
    <form>
      <input type="text" />
      <input type="radio" />
      <input type="checkbox" />
    </form>
  </body>
</html>
```

DOM Tree

```
|-> Document
  |-> Element (<html>)
    |-> Element (<body>)
      |-> Element (<div>)
        |-> text node
        |-> Form
          |-> Text-box
          |-> Radio Button
          |-> Check Box
```
Javascript is very powerful

Almost anything you want to the DOM!

A JS script embedded on a page can modify in almost arbitrary ways the DOM of the page. The same happens if an attacker manages to get you load a script into your page.

w3schools.com has nice interactive tutorials
Example of what Javascript can do...

Can change HTML content:

```html
<p id="demo">JavaScript can change HTML content.</p>

<button type="button" onclick="document.getElementById('demo').innerHTML = 'Hello JavaScript!'">Click Me!</button>

DEMO from w3schools.com
Other examples

Can change images
Can change style of elements
Can hide elements
Can unhide elements
Can change cursor
Other example: can access cookies

Read cookie with JS:
```javascript
var x = document.cookie;
```

Change cookie with JS:
```javascript
document.cookie = "username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```
Frames
Frames

- Enable embedding a page within a page

```html
<iframe src="URL"></iframe>
```
Frames

- Modularity
  - Brings together content from multiple sources
  - Client-side aggregation

- Delegation
  - Frame can draw only on its own rectangle
Frames

- Outer page can specify only sizing and placement of the frame in the outer page
- Frame isolation: Our page cannot change contents of inner page, inner page cannot change contents of outer page
Web security
Same-origin policy
Same-origin policy

- Each site in the browser is isolated from all others.
Same-origin policy

- Multiple pages from the same site are not isolated.
Origin

- Granularity of protection for same origin policy
- Origin = protocol + hostname + port

http://coolsite.com:81/tools/info.html

- It is **string matching**! If these match, it is same origin, else it is not. Even though in some cases, it is logically the same origin, if there is no match, it is not
Same-origin policy

One origin should not be able to access the resources of another origin

Javascript on one page cannot read or modify pages from different origins
Same-origin policy

- The origin of a page is derived from the URL it was loaded from.
- Javascript runs with the origin of the page that loaded it.

http://en.wikipedia.org

http://www.google-analytics.com
Origins of other components

• `<img src=""">` the image is “copied” from the remote server into the new page so it has the origin of the embedding page (like JS) and not of the remote origin

• iframe: origin of the URL from which the iframe is served, and not the loading website.
## Exercises

<table>
<thead>
<tr>
<th>Originating document</th>
<th>Accessed document</th>
</tr>
</thead>
</table>
Cross-origin communication

• Allowed through a narrow API: `postMessage`
• Receiving origin decides if to accept the message based on origin (whose correctness is enforced by browser)
Chromodo
Private Internet Browser
Fast and versatile Internet Browser based on Chromium, with highest levels of speed, security and privacy!

Issue 704: Comodo: Comodo "Chromodo" Browser disables same origin policy, Effectively turning off web security.
13 people starred this issue, and may be notified of changes.

Reported by tav@google.com, Jan 21, 2016

When you install Comodo Internet Security, by default a new browser called Chromodo is installed and set as the default browser. Additionally, all shortcuts are replaced with Chromodo links and all settings, cookies, etc are imported from Chrome. They also hijack DNS settings, among other shady practices.


Chromodo is described as "highest levels of speed, security and privacy", but actually disables all web security. Let me repeat that, they ***disable the same origin policy***... ?!?..
HTTP cookies
Outrageous Chocolate Chip Cookies

Recipe by: Joan

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

Ingredients

1/2 cup butter
1/2 cup white sugar
Market Pantry Granulated Sugar - 4lbs
$2.59
SEE DETAILS

1 cup all-purpose flour
1 teaspoon baking soda
1/4 teaspoon salt
1/2 cup rolled oats

1/3 cup packed brown sugar
1 cup semisweet chocolate chips

25 m
18 servings
207 cals
Cookies

• A way of maintaining state

Browser

GET ...

http response contains

Server

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

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

```
document.cookie
```
to see the cookie for that site
Cookie scope

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

HTTP Header:
Set-cookie: NAME=VALUE ;
don domain = (when to send) ;
path = (when to send)
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 (privacy and integrity)
Cookie scope

HTTP Header:
Set-cookie: NAME=VALUE ;
  domain = (when to send) ;
  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
Cookie scope

- Scope of cookie might not be the same as the URL-host name of the web server setting it
  - Different from same-origin policy!!

Rules on:
1. What scopes a URL-host name is allowed to set
2. When a cookie is sent to a host
What scope a server may set for a cookie

The browser checks if the 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’]

<table>
<thead>
<tr>
<th>allowed domains</th>
<th>disallowed domains</th>
</tr>
</thead>
<tbody>
<tr>
<td>login.site.com</td>
<td>user.site.com</td>
</tr>
<tr>
<td>.site.com</td>
<td>othersite.com</td>
</tr>
<tr>
<td>.com</td>
<td>.com</td>
</tr>
</tbody>
</table>

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

**path:** can be set to anything
We discussed the semantics of HTTP cookies in Chapter 3, but that discussion left out one important detail: the security rules that must be implemented to protect cookies belonging to one site from being tampered with by unrelated pages. This topic is particularly interesting because the approach taken here predates the same-origin policy and interacts with it in a number of unexpected ways.

Cookies are meant to be scoped to domains, and they can’t be limited easily to just a single hostname value. The domain parameter provided with a cookie may simply match the current hostname (such as foo.example.com), but this will not prevent the cookie from being sent to any eventual subdomains, such as bar.foo.example.com. A qualified right-hand fragment of the hostname, such as example.com, can be specified to request a broader scope, however.

Amusingly, the original RFCs imply that Netscape engineers wanted to allow exact host-scoped cookies, but they did not follow their own advice. The syntax devised for this purpose was not recognized by the descendants of Netscape Navigator (or by any other implementation for that matter). To a limited extent, setting host-scoped cookies is possible in some browsers by completely omitting the domain parameter, but this method will have no effect in Internet Explorer.

Table 9-3 illustrates cookie-setting behavior in some distinctive cases.

The only other true cookie-scoping parameter is the path prefix: Any cookie can be set with a specified path value. This instructs the browser to send the cookie back only on requests to matching directories; a cookie scoped to domain of example.com and path of /some/path/ will be included on a request to http://foo.example.com/some/path/subdirectory/hello_world.txt.

This mechanism can be deceptive. URLs paths are not taken into account during same-origin policy checks and, therefore, do not form a useful security boundary. Regardless of how cookies work, JavaScript code can simply hop between any URLs on a single host at will and inject malicious payloads into

<table>
<thead>
<tr>
<th>domain</th>
<th>Whether it will be set, and if so, where it will be sent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(value omitted)</td>
<td>foo.example.com (exact)</td>
</tr>
<tr>
<td>bar.foo.example.com</td>
<td></td>
</tr>
<tr>
<td>foo.example.com</td>
<td>*.foo.example.com</td>
</tr>
<tr>
<td>baz.example.com</td>
<td></td>
</tr>
<tr>
<td>example.com</td>
<td></td>
</tr>
<tr>
<td>ample.com</td>
<td></td>
</tr>
<tr>
<td>.com</td>
<td></td>
</tr>
</tbody>
</table>
We discussed the semantics of HTTP cookies in Chapter 3, but that discussion left out one important detail: the security rules that must be implemented to protect cookies belonging to one site from being tampered with by unrelated pages. This topic is particularly interesting because the approach taken here predates the same-origin policy and interacts with it in a number of unexpected ways.

Cookies are meant to be scoped to domains, and they can't be limited easily to just a single hostname value. The `domain` parameter provided with a cookie may simply match the current hostname (such as `foo.example.com`), but this will not prevent the cookie from being sent to any eventual subdomains, such as `bar.foo.example.com`. A qualified right-hand fragment of the hostname, such as `example.com`, can be specified to request a broader scope, however.

Amusingly, the original RFCs imply that Netscape engineers wanted to allow exact host-scoped cookies, but they did not follow their own advice. The syntax devised for this purpose was not recognized by the descendants of Netscape Navigator (or by any other implementation for that matter). To a limited extent, setting host-scoped cookies is possible in some browsers by completely omitting the `domain` parameter, but this method will have no effect in Internet Explorer.

Table 9-3 illustrates cookie-setting behavior in some distinctive cases.

<table>
<thead>
<tr>
<th>domain</th>
<th>Whether it will be set, and if so, where it will be sent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(value omitted)</td>
<td><code>foo.example.com</code> (exact)</td>
</tr>
<tr>
<td><code>bar.foo.example.com</code></td>
<td>Cookie not set: domain more specific than origin</td>
</tr>
<tr>
<td><code>foo.example.com</code></td>
<td><code>*.foo.example.com</code></td>
</tr>
<tr>
<td><code>baz.example.com</code></td>
<td>Cookie not set: domain mismatch</td>
</tr>
<tr>
<td><code>example.com</code></td>
<td><code>*.example.com</code></td>
</tr>
<tr>
<td><code>ample.com</code></td>
<td>Cookie not set: domain mismatch</td>
</tr>
<tr>
<td><code>.com</code></td>
<td>Cookie not set: domain too broad, security risk</td>
</tr>
</tbody>
</table>

When browser sends cookie

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

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

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

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

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

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

https://login.site.com/  
cookie: userid=u1; userid=u2  
(arbitrary order)
Client side read/write:  

- Setting a cookie in Javascript:
  
  document.cookie = “name=value; expires=…; ”

- Reading a cookie:
  - `alert(document.cookie)`: prints string containing all cookies available for document (based on [protocol], domain, path)

- Deleting a cookie:
  
  document.cookie = “name=; expires= Thu, 01-Jan-70”

`document.cookie` often used to customize page in Javascript
Sessions
Sessions

• A sequence of requests and responses from one browser to one (or more) sites
  – Session can be long (Gmail - two weeks) or short (e.g., banking)
  – without session mgmt:
    users would have to constantly re-authenticate

• Session management:
  – Authorize user once;
  – All subsequent requests are tied to user
Session tokens

Browser

GET /index.html

set anonymous session token

GET /books.html

anonymous session token

POST /do-login

Username & password

elevate to a logged-in session token

POST /checkout

logged-in session token

Web Site

check credentials

Validate token
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">

Can you see problems with these?
Storing session tokens: problems

• Browser cookie:
  browser sends cookie with every request, even when it should not (see CSRF attack)

• Embed in all URL links:
  token leaks via HTTP Referer header (your browser tells a site which previous site it visited last in the Referer header, which may contain session tokens)

• In a hidden form field: short sessions only

Best answer: a combination of all of the above.
Questions?

• Same-origin policy
• DOM model
• Cookie policy
• Sessions