Web Defenses

CS155 Computer and Network Security
Review: CSRF Attacks
Cross-Site Request Forgery (CSRF)

Cross-site request forgery (CSRF) attacks are a type of web exploit where a website transmits unauthorized commands as a user that the server trusts.

In a CSRF attack, a user is tricked into submitting an unintended (often unrealized) web request to a website — generally takes advantage of session cookies.

You need to actively build defenses into web apps to protect against CSRF attacks.
Options for Preventing CSRF Attacks

Do not trust cookies to indicate whether an authorized application submitted request since they’re included in every (in-scope) request.

We need another mechanism that allows us to ensure that a request is authentic (coming from a trusted page).

Three commonly used techniques to validate intent:

- Referer Header Validation
- Secret Validation Token
- Custom HTTP Header (forces CORS Pre-Flight Permissions Check)

Or, simply, don't send cookies:

- sameSite Cookies
Options for Preventing CSRF Attacks

Do not trust cookies to indicate whether an authorized application submitted request since they’re included in every (in-scope) request

We need another mechanism that allows us to ensure that a request is authentic (coming from a trusted page)

Three commonly used techniques to validate intent:

- Referer Header Validation
- Secret Validation Token
- Custom HTTP Header (forces CORS Pre-Flight Permissions Check)

Or, simply, don't send cookies:

- sameSite Cookies
When to use each method?

**Custom HTTP Header** — Generally used when accessing REST APIs (since header can only be set using Javascript anyway)

**Secret Validation Token** — Used for any conventional HTML interactions (e.g., login form that POSTs to a URL when user clicks submit)
sameSite Cookies

Cookies that match the domain of the current site, i.e. what's currently displayed in the browser's address bar, are referred to as first-party cookies.

Cookies from domains other than the current site are third-party cookies.

Cookies marked as sameSite are only sent if first party.

Will not be sent for image, form post if URL bar ≠ origin of resource.
Two Modes

sameSite cookie setting can be in two modes:

**Strict Mode (SameSite=Strict):** The cookie will only be sent if the site for the cookie matches the site currently shown in the browser's URL bar.

Problem: If you're on Site A, click on a link to Site B, then Site B won't receive cookie because when you clicked on the link, URL bar said Site A (or, if you simply typed the site into the URL bar)

**Lax Mode (SameSite=Lax):** Allows cookie to be sent with these top-level navigations.
Review: XSS Attacks
Cross Site Scripting (XSS)

**Cross Site Scripting:** Attack occurs when application takes untrusted data and sends it to a web browser without proper validation or sanitization.

- **Command/SQL Injection**
  - Attacker’s malicious code is executed on app’s server

- **Cross Site Scripting**
  - Attacker’s malicious code is executed on victim’s browser
Cookie Theft!

https://google.com/search?q=<script>...</script>
Where can injection come from?

- HTTP request from user
  - Query parameters, form fields, headers, cookies, file uploads
- Data from a database
- Third-party services
Many Frameworks Support Filtering

EJS template:

```html
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

**Server code:**

```javascript
res.render('template-name', { user })
```
Filtering is **Really** Hard

Large number of ways to call Javascript and to escape content

- URI Scheme: `<img src="javascript:alert(document.cookie);"`

- On{event} Handers: onSubmit, OnError, onSyncRestored, … (there’s ~105)

Samy Worm: CSS

Tremendous number of ways of encoding content

`<IMG SRC="#0000106#000097#000118#000097#000115#000099#0000114#0000112#0000116#000058#000097#0000108#0000101#000114#0000116#000040#000039#000088#000083#000083#000039#000041>`

Google XSS Filter Evasion!
Content Security Policies (Prevents XSS)
Content Security Policy (CSP)

You’re always safer using a whitelist- rather than blacklist-based approach

**Content-Security-Policy** is an HTTP header that servers can send that declares which dynamic resources (e.g., Javascript) are allowed

**Good News:** CSP eliminates XSS attacks by whitelisting the origins that are trusted sources of scripts and other resources and preventing all others

**Bad News:** CSP headers are complicated and folks frequently get the implementation incorrect.
Example CSP — Javascript

Policies are defined as a set of directives for where different types of resources can be fetched. For example:

**Content-Security-Policy**: `script-src 'self'`

→ Javascript can only be loaded from the same domain as the page
→ No Javascript from any other origins will be executed
→ no inline `<script></script>` will be executed
Example CSP — Javascript

Policies are defined as a set of directives for where different types of resources can be fetched. For example:

**Content-Security-Policy**: `script-src '*'`

- Javascript can only be loaded from any external domain
- no inline `<script></script>` will be executed
**Example CSP — Default**

The **default-src** directive defines the default policy for fetching resources such as JavaScript, images, CSS, fonts, AJAX requests, frames, HTML5 media.

**Content-Security-Policy:** `default-src 'self' cdn.com;`

- Dynamic resources can only be loaded from the same domain and CDN.
- No content from any other origins will be executed.
- No inline `<script></script>` or `<style>` will be executed.
Multiple Directives

**Content-Security-Policy:** default-src 'self';
          img-src *; script-src cdn.jquery.com

→ content can only be loaded from the same domain as the page, except
  → images can be loaded from any origin
  → scripts can only be loaded from cdn.jquery.com
→ no inline `<script>` will be executed
→ no inline `<style>` will be executed
Other Directives

CSP provides a whole list of different directives for locking down scripts:

- script-src
- style-src
- img-src
- connect-src
- font-src
- object-src
- media-src
- frame-src
- report-uri
- ..

Look at https://content-security-policy.com/
Mozilla Recommended Default

This policy allows images, scripts, AJAX, form actions, and CSS from the same origin, and does not allow any other resources to load (e.g., object, frame, media, etc). Also no inline scripts.

It is a good starting point for many sites.

```
default-src 'none'; script-src 'self';
connect-src 'self'; img-src 'self'; style-src 'self';
base-uri 'self'; form-action 'self'
```
If you're worried a new policy might break your site, there's a soft enforce mode that just reports violations. Great starting point.

Content-Security-Policy-Report-Only:
default-src 'self';
report-uri https://example.com/report
Real-World Breaks CSP

Content-Security-Policy:
default-src: 'self';
script-src: 'self' https://www.google-analytics.com

<script>
window.GoogleAnalyticsObject = 'ga'
function ga () { window.ga.q.push(arguments) }
window.ga.q = window.ga.q || []
window.ga.l = Date.now()
window.ga('create', 'UA-XXXXXXX-XX', 'auto')
window.ga('send', 'pageview')
</script>
<script async src='https://www.google-analytics.com/analytics.js'></script>
Strict Dynamic

Content-Security-Policy: script-src 'strict-dynamic' 'nonce-abc123...' 

Website HTML:

<script src='https://trusted.com/good.js' nonce='abc123'></script>
<script nonce='abc123'>foo()</script>

Specifies that the trust explicitly given to a script present in the markup, by accompanying it with a nonce, shall be propagated to all the scripts loaded by that root script.
Similar Protection for iFrames

HTML5 Sandboxes allow further privilege separation even if iFrame is from the same origin.

<iframe src="untrusted.html" sandbox></iframe>

- Plugins are disabled.
- Script execution is blocked
- Form submission is blocked
- The content is treated as if it was from a globally unique origin. Meaning, all APIs which require same-origin (such as localStorage, XMLHttpRequest, and access to the DOM of other documents) are blocked.
- The content is blocked from navigating the top level window or other frames
- Popup windows are blocked

<iframe src="demo_iframe_sandbox_form.htm" sandbox="allow-forms"></iframe>

https://www.w3schools.com/tags/att_iframe_sandbox.asp
Clickjacking Attacks
Clickjacking

Attacker uses a transparent frame to trick a user into clicking on a button or link on another page when they were intending to click on the top level page.
Incorrect solution: framebusting

if (top != self) { top.location = self.location; }

Easy for parent to intercept and block call to change URL of page
Correct Solution: CSP

HTTP response from server:
HTTP/1.1 200 OK
...
Content-Security-Policy: frame-ancestors 'none';
...

<iframe src='example.com'> will cause an error

frame-ancestors ‘self’ ;
means only example.com can frame page
Sub-Resource Integrity
Third-Party Content Safety

**Question:** how do you safely load an object from a third party service?

```html
<script src="https://code.jquery.com/jquery-3.4.0.js"></script>
```

If `code.jquery.com` is compromised, your site is too!
MaxCDN Compromise

2013: MaxCDN, which hosted bootstrapcdn.com, was compromised.

MaxCDN had laid off a support engineer having access to the servers where BootstrapCDN runs. The credentials of the support engineer were not properly revoked. The attackers had gained access to these credentials.

Bootstrap JavaScript was modified to serve an exploit toolkit.
Sub-Resource Integrity (SRI)

SRI allows you to specify expected hash of file being included

```html
<script
    src="https://code.jquery.com/jquery-3.4.0.min.js"
    integrity="sha256-BJeo0qm959uMBGb65z40ejJYGSgR1fNKwOg="
/>
```
Sub-Resource Integrity (SRI)

```html
<script src="https://code.jquery.com/jquery-3.5.1.min.js"
     integrity="sha256-9/aliU8dGd2tb6OSuZixeV4y/faTqgFtohetphb0j0="
     crossorigin="anonymous">
</script>
```

Browser: (1) load sub-resource, (2) compute hash of contents, (3) compare value to the integrity attribute.

- if hash mismatch: script or stylesheet are not executed and an error is raised.
Enforce SRI with CSP

HTTP response from server:
HTTP/1.1 200 OK
...
Content-Security-Policy: require-sri-for script style;
...

Requires SRI for all scripts and style sheets on page
Securely Using Cookies
Cookies have no integrity

Users can change and delete cookie values
* Edit cookie database (FF: cookies.sqlite)
  * Modify Cookie header (FF: TamperData extension)

Shopping cart software
  Set-cookie: shopping-cart-total = 150 ($)

User edits cookie file (cookie poisoning):
  Cookie: shopping-cart-total = 15 ($)

Similar problem with localStorage and hidden fields:
  <INPUT TYPE="hidden" NAME=price VALUE="150">
Sign Cookies if Data

Goal: data integrity

Requires server-side secret key $k$ unknown to browser

**Generate tag:** $T \leftarrow \text{MACsign}(k, (\text{SID, name, value}))$

**Verify tag:** $\text{MACverify}(k, (\text{SID, name, value}), T)$

Binding to session-id (SID) makes it harder to replay old cookies
Authentication and Session Management
Pre-history: HTTP auth

HTTP request: GET /index.html

HTTP response contains:

WWW-Authenticate: Basic realm="Password Required"

Browsers sends hashed password on all subsequent HTTP requests:

Authorization: Basic ZGFddfibzsdfgkjheczI1NXRIeHQ=
HTTP auth problems

Hardly used in commercial sites:

• User cannot log out other than by closing browser
  – What if user has multiple accounts?
  multiple users on same machine?

• Site cannot customize password dialog

• Confusing dialog to users

• Easily spoofed

   Do not use ...
Session Management Today

GET / HTTP/1.1
cookies: []

HTTP/1.0 200 OK
cookies: [session: e82a7b92]

<html><h1>Welcome!</h1></html>
Session Management Today

GET / HTTP/1.1
cookies: []

HTTP/1.0 200 OK
cookies: [session: e82a7b92]
<html><h1>Welcome!</h1></html>

GET /loginform HTTP/1.1
cookies: []

HTTP/1.0 200 OK
cookies: [session: e82a7b92]
<html><form>...</form></html>

Create Anonymous Session ID
Session Management Today

GET  /loginform  HTTP/1.1
cookies: []

POST  /login  HTTP/1.1
cookies: []
username: zakir
password: stanford

GET  / HTTP/1.1
cookies: []

GET  /loginform  HTTP/1.1
cookies: []

HTTP/1.0 200 OK
cookies: [session: e82a7b92]

<html><form>...</form></html>

HTTP/1.0 200 OK
cookies: [session: e82a7b92]

<!-- Welcome! -->

HTTP/1.0 200 OK
cookies: [session: e82a7b92]

<html><h1>Login Success</h1></html>
Session Management Today

GET / HTTP/1.1
cookies: []

HTTP/1.0 200 OK
cookies: [session: e82a7b92]

GET /loginform HTTP/1.1
cookies: []

HTTP/1.0 200 OK
cookies: [session: e82a7b92]

POST /login HTTP/1.1
cookies: []
username: zakir
password: stanford

HTTP/1.0 200 OK
cookies: [session: e82a7b92]

GET /account HTTP/1.1
cookies: [session: e82a7b92]
Session Tokens

**Example 1:** counter

⇒ user logs in, gets counter value,
   can view sessions of other users

**Example 2:** weak MAC. \( \text{token} = \{ \text{userid}, \text{MAC}_k(\text{userid}) \} \)

- Weak MAC exposes \( k \) from few cookies.

Session tokens must be unpredictable to attacker

To generate: use underlying framework (e.g. ASP, Tomcat, Rails)

Rails: \( \text{token} = \text{SHA256}(\text{current time, random nonce}) \)
Implementing Logout

Web sites must provide a logout function:

- **Functionality:** let user to login as different user
- **Security:** prevent others from abusing account

What happens during logout:

1. Delete SessionToken from client
2. Mark session token as expired on server

Problem: many web sites do (1) but not (2)!!

⇒ Especially risky in case of XSS vulnerability
Authenticating Users

Plain Text Passwords (Terrible)
- Store the password and check match against user input
- Don’t trust anything that can provide you your password
Authenticating Users

Plain Text Passwords (Terrible)
- Store the password and check match against user input
- Don’t trust anything that can provide you your password

Store Password Hash (Bad)
- Store SHA-1(pw) and check match against SHA-1(input)
- Weak against attacker who has hashed common passwords
Authenticating Users

Plain Text Passwords (Terrible)
- Store the password and check match against user input
- Don’t trust anything that can provide you your password

Store Password Hash (Bad)
- Store SHA-1(pw) and check match against SHA-1(input)
- Weak against attacker who has hashed common passwords

Store Salted Hash (Better)
- Store \((r, \text{SHA-1}(pw \mid |r))\) and check against \(\text{SHA-1}(input \mid |r)\)
- Prevents attackers from pre-computing password hashes
**Authenticating Users**

**Store Salted Hash (Best)**
- Store \((r, H(pw \| r))\) and check match against \(H(\text{input} \| r)\)
- Prevents attackers from pre-computing password hashes

Making sure to choose an \(H\) that’s expensive to compute:

- **SHA-512**: 3,235 MH/s
- **SHA-3 (Keccak)**: 2,500 MH/s
- **BCrypt**: 43,551 H/s

Use one of bcrypt, scrypt, or pbkdf2 when building an application
Phishing and U2F
Phishing Attacks

Attacker sends a fraudulent message that tricks user into revealing sensitive data (e.g., login, credit card)

Almost all phishing attacks take place over the web — difficult to know if you're in the right place as a user.

SMS-based 2FA does little good. Mostly protects against stolen credentials.
U2F + Physical Security Keys

Goals:
- **Browser malware cannot steal user credentials**
- U2F should not enable tracking users across sites
- U2F uses counters to defend against token cloning

![U2F token (holds user credentials)](image1)
![Browser](image2)
![Service (github.com)](image3)
Physical Security Keys

U2F Device

challenge

Client

signature(challenge)

Relying Party

challenge

$s$

Lookup $k_{pub}$

Check $s$ using $k_{pub}$
Physical Security Keys
Physical Security Keys

U2F Device

- Lookup the $k_{priv}$ associated with $h$
- $h$, $a$; challenge, origin, channel id, etc.
- signature($a$, $c$)
- $s$

Client

- Check app id
- handle, app id, challenge
- $c$, $s$

Server

- Lookup the $k_{pub}$ associated with $h$
- $h$, $a$
- Check $s$ using $k_{pub}$
- Verify origin and channel id
Build a Secure Web Application
Many Steps Involved

**Best Advice:** Use a modern web framework — many security precautions are built in today — but don't assume!

**Protect Against CSRF:** Never depend on cookies to signal user intent! Use CORS Pre-Flight or CSRF Tokens. Set cookies as `sameSite` and `secure`.

**Protect Against XSS:** Set a **Content Security Policy** and do not use any inline scripts. Use `httpOnly` cookies.

**Protect Against SQL Injection:** Use **Parameterized SQL** or Object Relational Mapper (ORM)
Many More Steps Involved

**Protect Against Data Breach:** Use modern hashing algorithm like BCRYPT and salt passwords

**Protect Against Clickjacking:** Set *Content Security Policy* that prevents you from being shown in an IFRAME

**Protect Against Malicious Third Parties:** Use Iframes, CSP, and HTML5 Sandboxes

**Protect Against Compromised Third Parties:** Use Sub-Resource Integrity Headers

**Protect Against Credential Compromise and Phishing:** Use U2F
Third Party Cookies
Third Party Cookies

- Site A’s page requests a third-party resource (image, script, iframe)
  - Normally, browser sends cookie associated with that third-party in that request

Cookie: ID=784c39
Referer: cnn.com/
Third Party Cookies

- Site A’s page requests a third-party resource (image, script, iframe)
  - Normally, browser sends cookie associated with that third-party in that request

Cookie: ID=784c39
Referer: reddit.com/
Cookie: ID=784c39
Referer: cnn.com/
1. First of all, define "cat lady." Does one cat = cat lady? Two cats = cat lady? Does joking about being a cat lady à la sparkling, outgoing multimillionaire Taylor Swift automatically make one a cat lady? It is my personal belief that most female cat owners below the age of 40 fall into the "not a cat girl, not yet a cat lady" category.

2. Cat ladies mostly look like ... normal ladies. You know, like regular women. Not like the old lady who sits in front of your local Shoe Rite with aluminum foil on her head.
Facebook, DoubleClick, etc. know much more about you than actual website does because they can track you across websites.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Top 1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>google-analytics.com</td>
<td>67.8%</td>
</tr>
<tr>
<td>gstatic.com</td>
<td>50.1%</td>
</tr>
<tr>
<td>fonts.googleapis.com</td>
<td>42.8%</td>
</tr>
<tr>
<td>doubleclick.net</td>
<td>40.5%</td>
</tr>
<tr>
<td>facebook.com</td>
<td>33.7%</td>
</tr>
<tr>
<td>google.com</td>
<td>33.2%</td>
</tr>
<tr>
<td>facebook.net</td>
<td>27.4%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain</th>
<th>Top 1M</th>
</tr>
</thead>
<tbody>
<tr>
<td>ajax.googleapis.com</td>
<td>23.1%</td>
</tr>
<tr>
<td>googleapis.com</td>
<td>19.6%</td>
</tr>
<tr>
<td>googlesyndication.com</td>
<td>19.6%</td>
</tr>
<tr>
<td>googleadservices.com</td>
<td>14.1%</td>
</tr>
<tr>
<td>twitter.com</td>
<td>12.8%</td>
</tr>
<tr>
<td>fbcdn.net</td>
<td>10.7%</td>
</tr>
<tr>
<td>adnxs.com</td>
<td>10.5%</td>
</tr>
</tbody>
</table>
You’ve gone incognito

Pages you view in Incognito tabs won’t stick around in your browser’s history, cookie store, or search history after you’ve closed all of your incognito tabs. Any files you download or bookmarks you create will be kept.

However, you aren’t invisible. Incognito doesn’t hide your browsing from your employer, your internet service provider, or the websites you visit.

LEARN MORE
Ghostery found 13 trackers

- ClickTale
  - Analytics, Analytics, Behavior Tracking
- DoubleClick
  - Advertising
- Facebook Connect
  - Widgets, Social
- Gravity Insights
  - Analytics
- Krux Digital
  - Beacons
- Livefyre
  - Widgets, Commenting System
Do Not Track

Mozilla is a global, nonprofit organization dedicated to making the Web better. We emphasize principle over profit, and believe that the Web is a shared public resource to be cared for, not a commodity to be sold. We answer to no one but you and believe it is crucial to put you in control of your online experience. We are aiming to give you better insight and control into the ways your personal information is collected, used, stored and shared online.

Mozilla Firefox offers a Do Not Track feature that lets you express a preference not to be tracked by websites. When the feature is enabled, Firefox will tell advertising networks and other websites and applications that you want to opt-out of tracking for purposes like behavioral advertising.