Securing AJAX Applications

Charlie Reis

Writing Web Apps is Hard

- AJAX is cool, but also complicated & quirky
- Hard enough just to make it work
- **Securing against attacks is even harder**

Web Apps are Easy Targets

- Can directly attack server
- Can attack via users' web browsers

Outline

1. Client Side Logic and State

2. Cross Site Scripting

3. Cross Site Request Forgery

4. JavaScript Hijacking

(5. Advanced Topics)

1. Client-Side Logic & State

- **Do NOT trust the client, ever**
  - Don’t trust the user
  - Don’t trust the client hardware
  - Don’t trust the client OS
  - Definitely don’t trust the client browser

Clients are Still Useful

- AJAX puts more content, logic, & state at client
- **Implications**:
  - Check any user-supplied data on the server
  - Know that any client-side check may be skipped
  - Be careful when exposing AJAX APIs on server
    - *e.g., all public methods of objects*

Outline

1. Client Side Logic and State

2. Cross Site Scripting

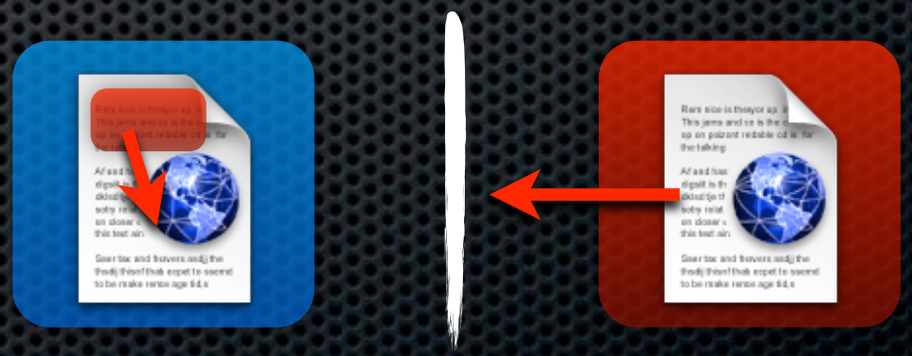
3. Cross Site Request Forgery

4. JavaScript Hijacking

(5. Advanced Topics)

2. Cross-Site Scripting

- One of the most common web app vulnerabilities
- Web client security boils down to **Same Origin Policy**
- XSS gets around it
  - Attacker runs scripts inside your web page



XSS Consequences

- With XSS, attackers can:
  - Steal cookies (then take over user account)
  - Phish users (e.g., fake login dialog)
  - Control almost ALL content/code in entire origin

Types of XSS

- **Reflected XSS**
  - Some input echoed on the page (e.g., search query)
  - Victim must follow an evil link
- **Stored XSS**
  - Some input stored in a DB (e.g., blog comment)
  - All later visitors become victims
- **DOM-Based XSS**
  - Client-side JS code echoes input into page

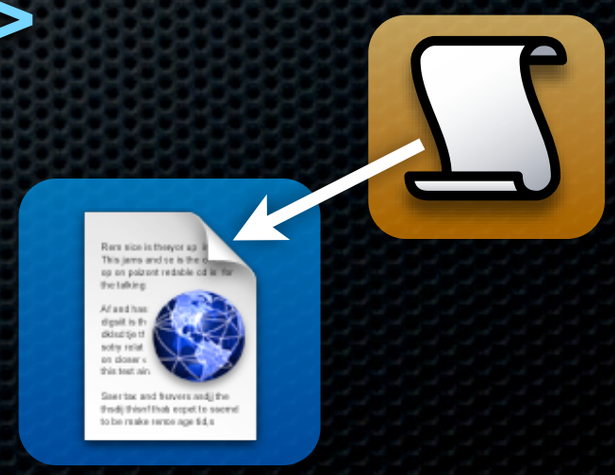
Defense: Data Sanitization

- Don’t ever trust user input
  - Escape all **output data** to prevent scripts
- Important: escaping must be **context sensitive**
  - In HTML? Within a tag? A script? A URL?
- Don’t invent your own sanitizer. It will be wrong.
  - Use template systems or frameworks if possible

    See: ClearSilver, Google CTemplate, GWT, etc

Related Threats:

- **SQL Injection** (older news, but similar issue)
- **JSON data**
  - Bad idea: *running* someone else's code to get data
  - Good idea: passing it to a JSON parser
- **<script src=http://differentsite.com>**
  - Be careful: do you trust that site?



Outline

1. Client Side Logic and State

2. Cross Site Scripting

3. Cross Site Request Forgery

4. JavaScript Hijacking

(5. Advanced Topics)

3. Cross-Site Request Forgery

- Who's logged in? The user, or his browser?
- Pages in background can make requests to your site
  - Browser sends user's cookies
- Evil pages can take actions in user's name
  - Transfer funds, delete data, change state

Non-Defenses

- Check referrer of request
  - Often blocked by proxies, can sometimes be faked
- Require POST, not GET, for state-changing requests
  - Do this anyway! (important for crawlers, etc)
  - Won’t stop CSRF via JavaScript, though

Defense: Prevent Forging

- Make state-changing requests hard to guess
  - Avoid predictable URLs if possible
- **Tokens**: verify request comes from a page you trust
  - Put a hard-to-forge token in your page
  - Verify its presence in the request

Related Threats:

- **Session Fixation**
  - User gets logged in as the attacker
  - Queries, credit card entries, etc, go to attacker
  - Prevent using CSRF tokens on login pages
- **Clickjacking**
  - Attacker puts your page as a frame under mouse
  - Frame-busting defense, but tough to get right

Outline

1. Client Side Logic and State

2. Cross Site Scripting

3. Cross Site Request Forgery

4. JavaScript Hijacking

(5. Advanced Topics)

4. JavaScript Hijacking

- Many JS files have sensitive data (e.g., JSON)
- **Any web site can embed them with <script src=>**
- Can't *read* their contents, but can *run* them
- That's enough for attackers to get the data (e.g., Gmail contacts)

Defense: Don’t let data run

- Put something at the top to prevent execution
  - e.g., **while(1);** or unparseable characters
  - Remove before parsing the JSON
- Don’t use predictable URLs

Outline

1. Client Side Logic and State

2. Cross Site Scripting

3. Cross Site Request Forgery

4. JavaScript Hijacking

(5. Advanced Topics)

5. Advanced Topics

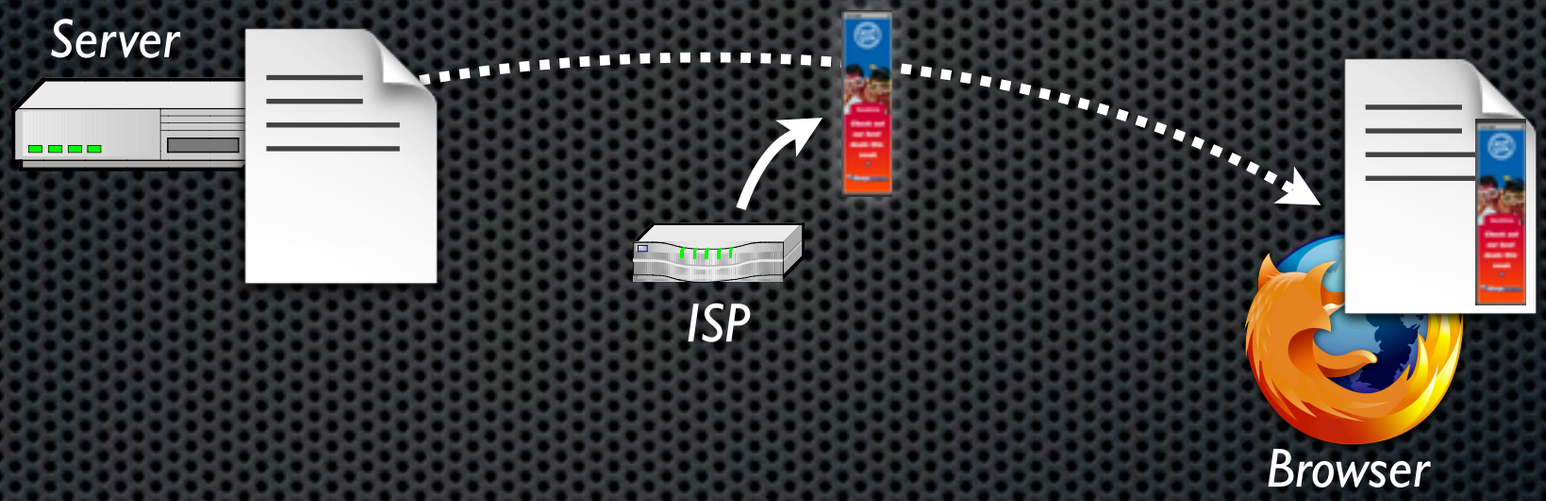
- Content Hosting
- Don’t trust the network
- Cutting-edge security features

Content Hosting

- Be careful before allowing users to upload content
  - Browsers guess content types poorly
    - e.g., GIFAR: both a valid GIF and a JAR (applet)
    - Flash cross-domain policy files loosely parsed
- Leads to XSS

Don’t Trust the Network

- ISPs and client proxies modify web pages (1%)



- Have observed XSS vulns as a result
- Use HTTPS whenever possible

Cutting Edge Security

- **Caja**: JS subset for untrusted gadgets
- **HTTP Only Cookies**: harder to get via XSS
- **Strict Transport Security**: only allow HTTPS
- **Mozilla Content Security Policies**:
  ways to tell browser how to help with security
- **Reflective XSS Filters** (IE8, Chrome)

Summary

- Be wary of your users, think like an attacker
- Use security-minded frameworks whenever possible
  - Encapsulate the hard stuff

References & Tools

- **Browser Security Handbook**
  http://code.google.com/p/browsersec/wiki/Main
- **Foundations of Security: What Every Web Programmer Needs to Know**
  (Daswani, Kern, Kesavan)
- **RatProxy**
  http://code.google.com/p/ratproxy/