Information Security
CS 526
Topic 11
Web Security Part 1
Readings for This Lecture

• Wikipedia
  – HTTP Cookie
  – Same Origin Policy
  – Cross Site Scripting
  – Cross Site Request Forgery
Background

• Many sensitive tasks are done through web
  – Online banking, online shopping
  – Database access
  – System administration

• Web applications and web users are targets of many attacks
  – Cross site scripting
  – SQL injection
  – Cross site request forgery
  – Information leakage
  – Session hijacking
Browser sends requests
Web site sends response pages, which may include code
Interaction susceptible to network attacks
Web Security/Privacy Issues

- Secure communications between client & server
  - HTTPS (HTTP over Secure Socket Layer)
- User authentication & session management
  - Cookies & other methods
- Active contents from different websites
  - Protecting resources maintained by browsers
- Web application security
- Web site authentication (e.g., anti-phishing)
- Privacy concerns
HTTP: HyperText Transfer Protocol

- Browser sends HTTP requests to the server
  - Methods: GET, POST, HEAD, ...
  - GET: to retrieve a resource (html, image, script, css, …)
  - POST: to submit a form (login, register, …)
  - HEAD

- Server replies with a HTTP response

- **Stateless** request/response protocol
  - Each request is independent of previous requests
  - Statelessness has a significant impact on design and implementation of applications
Use Cookies to Store State Info

- Cookies
  - A cookie is a name/value pair created by a website to store information on your computer.

Http is stateless protocol; cookies add state.
Cookies Fields

- An example cookie from my browser
  - Name: session-token
  - Content: "s7yZiOvFm4YymG...."
  - Domain: .amazon.com
  - Path: /
  - Send For: Any type of connection
  - Expires: Monday, September 08, 2031 7:19:41 PM
Cookies

- Stored by the browser
- Used by the web applications
  - used for authenticating, tracking, and maintaining specific information about users
    - e.g., site preferences, contents of shopping carts
  - data may be sensitive
  - may be used to gather information about specific users

- Cookie ownership
  - Once a cookie is saved on your computer, only the website that created the cookie can read it
Web Authentication via Cookies

• HTTP is stateless
  – How does the server recognize a user who has signed in?

• Servers can use cookies to store state on client
  – After client successfully authenticates, server computes an authenticator and gives it to browser in a cookie
    • Client cannot forge authenticator on his own (session id)
  – With each request, browser presents the cookie
  – Server verifies the authenticator
A Typical Session with Cookies

Client

1. POST /login.cgi
2. Set-Cookie: authenticator

Server

3. GET /restricted.html
   Cookie: authenticator

Client

Restricted content

Verify that this client is authorized
Check validity of authenticator

Authenticators must be unforgeable and tamper-proof
(malicious clients shouldn’t be able to modify an existing authenticator)

How to design it?
Cross Site Scripting
Client Side Scripting

• Web pages (HTML) can embed dynamic contents (code) that can be executed on the browser
  • JavaScript
    – embedded in web pages and executed inside browser
  • Java applets
    – small pieces of Java bytecodes that execute in browsers
• Browser extensions (plug-ins) provide further client-side programming abilities
  – E.g., Flash
HTML and Scripting

```html
<html>
  ...
  <p>
  <script>
    var num1, num2, sum
    num1 = prompt("Enter first number")
    num2 = prompt("Enter second number")
    sum = parseInt(num1) + parseInt(num2)
    alert("Sum = " + sum)
  </script>
  ...
  </html>

Browser receives content, displays HTML and executes scripts
```
Scripts are Powerful

- Client-side scripting is powerful and flexible, and can access the following resources
  - Local files on the client-side host
    - read / write local files
  - Webpage resources maintained by the browser
    - Cookies
    - Domain Object Model (DOM) objects
      - steal private information
      - control what users see
      - impersonate the user
  - Communicating with websites (via XMLHttpRequest)
Domain Object Model (DOM)

- Object-oriented model to represent webpages that allow programming access in Javascript
Browser as an Operating System

- Web users visit multiple websites simultaneously
- A browser serves web pages (which may contain programs) from different web domains
  - i.e., a browser runs programs provided by mutually untrusted entities
  - Running code one does not know/trust is dangerous
  - A browser also maintains resources created/updated by web domains
- Browser must confine (sandbox) these scripts so that they cannot access arbitrary local resources
- Browser must have a security policy to manage/protect browser-maintained resources and to provide separation among mutually untrusted scripts
Sandbox

• A security mechanism for separating/limiting running programs
  – Running untrusted programs.
    • E.g., javascripts in webpages, mobile apps
  – Running programs that are likely to be exploited.
    • E.g., network daemon programs
• Implementation: Clearly identify what resources a program needs and cut off the rest
  – Examples include operating system–level virtualization (such as Unix chroot), virtual machine monitors (VMMs), Java applets,
Same Origin Policy

- The basic security model enforced in the browser
- SoP isolates the scripts and resources downloaded from different origins
  - E.g., evil.org scripts cannot access bank.com resources
- Use origin as the security principal
  - Note that the concept of user accounts does not apply here as security principals
- Origin = domain name + protocol + port
  - all three must be equal for origin to be considered the same
Same Original Policy: What it Controls

- Same-origin policy applies to the following accesses:
  - manipulating browser windows
  - URLs requested via the XmlHttpRequest
  - manipulating frames (including inline frames)
  - manipulating documents (included using the object tag)
  - manipulating cookies
Problems with S-O Policy

• Poorly enforced on some browsers
  – Particularly older browsers
• Limitations if site hosts unrelated pages
  – Example: Web server often hosts sites for unrelated parties
    • http://www.example.com/account/
    • http://www.example.com/otheraccount/
  – Same-origin policy allows script on one page to access properties of document from another
• Can be bypassed in Cross-Site-Scripting attacks

• Usability: Sometimes prevents desirable cross-origin resource sharing
Browser Architecture: One Process versus Multiple Processes

- Most processes (e.g., Firefox, Internet Explorer) use one process for a web browser
  - Multiple threads are used for rendering different webpages
- Chrome uses multiple processes
  - Use OS protection mechanism to ensure that webpages from different sites cannot easily interact
    - Because they run in different processes
  - Reliability advantage: crashing in rendering one website doesn’t affect another
  - Security advantage: vulnerability in rendering does not compromise other sites; isolate plug-ins
  - Uses 3 types of processes: browser, renderers, plug-ins
What will the following program output?

```c
#include <stdio.h>
void main() {
    int x;
    int y;
    y = (x = 3) + (x = 4);
    printf("%d %d\n", x, y);
}
```
Cross Site Scripting (XSS)

• Recall the basics
  – scripts embedded in web pages run in browsers
  – scripts can access cookies
    • get private information
  – and manipulate DOM objects
    • controls what users see
  – scripts controlled by the same-origin policy

• Why would XSS occur
  – Web applications often take user inputs and use them as part of webpage (these inputs can have scripts)
How XSS Works on Online Blog

- Everyone can post comments, which will be displayed to everyone who view the post.
- Attacker posts a malicious comment that includes scripts (which reads local authentication credentials and send of to the attacker).
- Anyone who view the post can have local authentication cookies stolen.
- Web apps will check that posts do not include scripts, but the check sometimes fail.
Effect of the Attack

- Attacker can execute arbitrary scripts in browser

- Can manipulate any DOM component on victim.com
  - Control links on page
  - Control form fields (e.g. password field) on this page and linked pages.

- Can infect other users: MySpace.com worm.
MySpace.com  (Samy worm)

- Users can post HTML on their pages
  - MySpace.com ensures HTML contains no `<script>`, `<body>`, `onclick`, `<a href=javascript://>`
  - However, attacker find out that a way to include Javascript within CSS tags:
    `<div style="background:url('javascript:alert(1)')">`
    And can hide “javascript” as “java\nscript”

- With careful javascript hacking:
  - Samy’s worm: infects anyone who visits an infected MySpace page … and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.

Avoiding XSS bugs (PHP)

- Main problem:
  - Input checking is difficult --- many ways to inject scripts into HTML.

- Preprocess input from user before echoing it

- **PHP:** `htmlspecialchars(string)`

  ```
  & → &amp; ; " → &quot; ; ' → &#039;
  < → &lt; ; > → &gt;
  ```

  - `htmlspecialchars(
    "<a href='test'>Test</a>";
    ENT_QUOTES);

Outputs:

```
&lt;a href=&#039;test&#039;Test&lt;/a&gt;
```
Avoiding XSS bugs \hspace{1cm} (ASP.NET)

- ASP.NET 1.1:
  - `Server.HtmlEncode(string)`
    - Similar to PHP htmlspecialchars
  - `validateRequest`: (on by default)
    - Crashes page if finds `<script>` in POST data.
    - Looks for hardcoded list of patterns.
  - Can be disabled:
    
```
<%@ Page validateRequest="false" %>
```

Cross site request forgery
Cross site request forgery (abbrev. CSRF or XSRF)

• Also known as **one click attack** or **session riding**

• **Effect**: Transmits unauthorized commands from a user who has logged in to a website to the website.

• **Recall that a browser attaches cookies set by domain X to a request sent to domain X; the request may be from another domain**
  
  – Site Y redirects you to facebook; if you already logged in, the cookie is attached by the browser
**CSRF Explained**

- **Example:**
  - User logs in to bank.com. Forgets to sign off.
  - Session cookie remains in browser state
  
  - Then user visits another site containing:
    
    ```html
    <form name=F action=http://bank.com/BillPay.php>
    <input name=recipient value=badguy> ...
    <script> document.F.submit(); </script>
    ```
  
  - Browser sends user auth cookie with request
    - Transaction will be fulfilled

- **Problem:**
  - The browser is a confused deputy; it is serving both the websites and the user and gets confused who initiated a request
Real World CSRF Vulnerabilities

- Gmail
- NY Times
- ING Direct (4\textsuperscript{th} largest saving bank in US)
- YouTube
- Various DSL Routers
- Purdue WebMail
- PEFCU
- Purdue CS Portal
- …
Prevention

• Server side:
  – use cookie + hidden fields to authenticate a web form
    • hidden fields values need to be unpredictable and user-specific; thus someone forging the request need to guess the hidden field values
  – requires the body of the POST request to contain cookies
    • Since browser does not add the cookies automatically, malicious script needs to add the cookies, but they do not have access because of Same Origin Policy

• User side:
  – logging off one site before using others
  – selective sending of authentication tokens with requests (may cause some disruption in using websites)
Coming Attractions …

• More Web Security Issues
  – SQL injection
  – Side channel information leakage
  – Cookie privacy issues