Cross-Site Scripting

- What is XSS?
- How XSS Works
- Mitigating XSS

What is Cross-Site Scripting?

- **Cross-Site Scripting** (XSS) is an an attack in which an attacker attempts to compromise a victim by tricking the victim into running a JavaScript
- Web applications that accept input or request parameters, generate output HTML based on the input or parameter, and fail to **escape** special characters are vulnerable to XSS

- We refer to a Web site that fails to guard against XSS as a **vulnerable site**

Though XSS attacks are not limited to JavaScript, that is the most common vector.

XSS Attack Categories

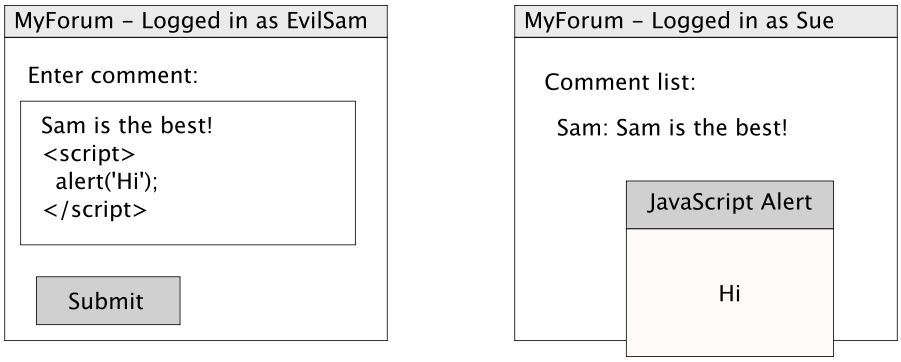
- **Persistent** attacks involve vulnerable Web sites that take user input and display it to other users
- **Non-persistent** or **reflected** attacks occur when vulnerable sites accept request parameters and use the parameter values to generate a page to a single user

For more details on these definitions, see:

`http://en.wikipedia.org/wiki/Cross-site_scripting`

XSS Persistent Attacks

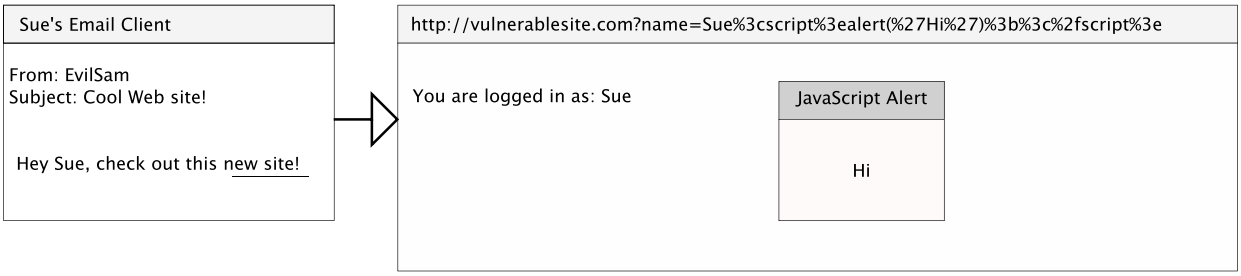
- Suppose an application takes user input and then simply redisplays it, perhaps as part of a user-generated content Web site:
  - Message boards and forums
  - User-review pages
  - Blogs
- In this example, an attacker entered a harmless JavaScript **alert** box invocation script



These attacks are considered "persistent" since the vulnerable Web site often stores such user-entered content into a database and redisplays it for all other users of the site.

XSS Non-Persistent Attacks

- In a non-persistent attack, attackers usually trick the victim into clicking on a link to the vulnerable Web site, typically delivered by email or on another Web site
- The link contains request parameters which the attacker crafts to contain scripts
- The vulnerable site then **reflects** the request parameters back to the victim's browser, where the malicious scripts run



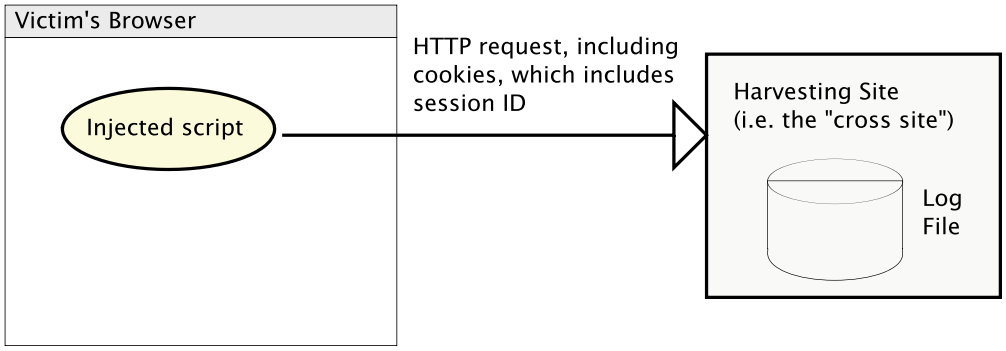
In this example, the vulnerable application accepted a request parameter, "name" and simply outputted its value to the Web page. And since the value contained a script, the script runs in Sue's browser, which trusts the script to the extent that it trusts the Web site. So if Sue was logged-in to the vulnerable site, the script has access to browser cookies. In this case, the script displayed a harmless JavaScript alert.

Note that the since the script was passed as a request parameter, the attacker used HTTP encoding on the request value as is normal for such parameters.

Instead of using a hyperlink, attackers can also use HTML forms.

How Does an XSS Attack Work?

- The attack begins when a logged-in victim's browser executes the malicious script
- The script makes a request to another URL - the "cross site" or **harvesting site**
- The harvesting site can capture the victim's request and session information



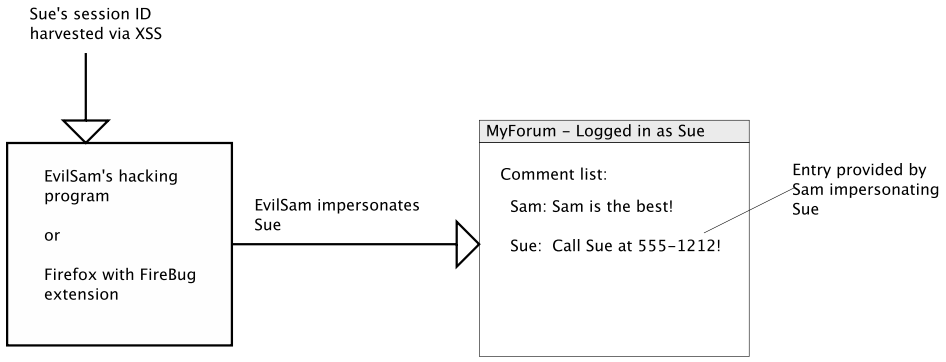
The malicious script runs as from within the victim's browser and can access cookies, which in many cases include a session identifier.

The script can also make an HTTP request to a harvesting site that the attacker creates, passing the cookies. The harvesting site can then log the information, and perhaps send an email notifying the attacker that the log has an entry. The attacker then can use the harvested session ID to impersonate the victim.

Note that since sessions generally have a timeout, the attacker has a limited time window in which they can impersonate the victim.

What Can an Attacker Do?

- Once the attacker has the victim's session ID, the attacker can impersonate the victim, perhaps using a custom hacking program or a browser add-in, for example, the Firebug plugin for Firefox
- The attacker can then access the victim's information on the vulnerable Web site
- XSS attacks can also contain links that send victims to authentic-looking, malicious Web sites that trick the user or install malware



Mitigating XSS

- The best way to avoid XSS is to not trust user input or request parameters
- Strategies:
    - Do not accept user input/request parameters and redisplay
    - Allow input and redisplay, but only allowing a non-HTML syntax (e.g. BBCode)
    - Sanitize the input/request parameters by removing characters such as < - this is sometimes referred to as **escaping**

Escaping is also referred to as "output encoding", or just "encoding".

You should also carefully consider how the application responds if it detects an XSS attack. It's a good idea to use some sort of a generic error page that doesn't give the attacker any information about why their request was rejected or filtered.

For more information on Bulletin Board Code (BBCode), see:

`http://en.wikipedia.org/wiki/BBCode`

JSP Strategies for Mitigating XSS

- Use the JSTL **c:out** tag, which by default "escapes" HTML characters such as <
- Use the JSTL **fn:escapeXML()** function within **JSP expressions**
- Use JSF components like **<h:outputText>**, which escapes HTML by default

```
Output JSP
----------

Bad:  ${param.name}
Good: ${fn:escapeXml(param.name)}
Good: <c:out value="${param.name}" />
```


Note that the JSP expression language does NOT escape XML by default, so applications that use it to redisplay user input are vulnerable to XSS.

More XSS Mitigation Strategies

- Always specify the document's **encoding** in the HTML *head* element - that prevents an attacker from entering special characters in a different Unicode encoding
- Consider using an encoding library such as OWASP's **Enterprise Security API** (ESAPI)

```
1    <head>
2     <meta http-equiv="Content-Type"
3       content="text/html; charset=ISO-8859-1">
4    </head>
```


In the HTML fragment shown here, we only allow input using the IS-8859-1 encoding, which comprises the characters used by Western languages such as English.

You can find out more about ESAPI at:

`https://www.owasp.org/index.php/ESAPI`

If you have a servlet-based application that generates HTML based on user input, you should write escaping code yourself or consider using a library like ESAPI.

Chapter Summary

In this chapter, you learned:

- What XSS attacks are and how they work
- How to mitigate against XSS attacks