Web Server Security

Cross Site Scripting

- Three main categories;
  - Non-persistent or reflected vulnerability, (probably the most common type)
  - DOM-based or Local cross-site scripting, similar to non-persistent, but includes extra JavaScript vulnerabilities in older versions of Internet Explorer
  - Stored, persistent, or second-order vulnerability, where the vulnerability is both stored and displayed back without validation

Non-persistent vulnerability

1. Alice often visits a particular website, which is hosted by Bob. Bob's website allows Alice to log in with a username/password pair and store sensitive information, such as billing information.
2. Mallory observes that Bob's website contains a reflected XSS vulnerability.
3. Mallory crafts a URL to exploit the vulnerability, and sends Alice an email, making it look as if it came from Bob (i.e., the email is spoofed).
4. Alice visits the URL provided by Mallory while logged into Bob's website.
5. The malicious script embedded in the URL executes in Alice's browser, as if it came directly from Bob's server. The script steals sensitive information (authentication credentials, billing info, etc) and sends this to Mallory's web server without Alice's knowledge.

DOM based

1. Mallory sends a URL to Alice (via email or another mechanism) of a maliciously constructed web page.
2. Alice clicks on the link.
3. The malicious web page's JavaScript opens a vulnerable HTML page installed locally on Alice's computer.
4. The vulnerable HTML page contains JavaScript which executes in Alice's computer's local zone.
5. Mallory's malicious script now may run commands with the privileges Alice holds on her own computer.

Stored or Persistent Vulnerability

1. Bob hosts a web site which allows users to post messages and other content to the site for later viewing by other members.
2. Mallory notices that Bob's website is vulnerable to a type 2 XSS attack.
3. Mallory posts a message, controversial in nature, which may encourage many other users of the site to view it.
4. Upon merely viewing the posted message, site users' session cookies or other credentials could be taken and sent to Mallory's webserver without their knowledge.
5. Later, Mallory logs in as other site users and posts messages on their behalf.

Impact of XSS

- Stealing and continuing the session of the (authenticated) victim
- Manipulating files on the victim's computer or the network she has access to
- Recording all keystrokes the victim makes in a web application and sending them to the hacker

Impact of XSS (cont)

- Stealing files from the attacked user's computer or the network he has access to
- Probing a company's intranet (where the victim is located) for further vulnerabilities
- Launching other attacks against systems the victim can reach with her browser (on the intranet)
- Performing brute force password cracking through the attacked user's compromised browser

Mitigating the Risk

- Filter All Input – you must, without fail, filter all input. Inspect all input, and only allow valid data into your application.
- Escape All Output – you should also escape all output. For data that is meant to be displayed as raw data and not interpreted as HTML, it must be escaped for the context of HTML.

Mitigating the Risk (cont)

- Only Allow Safe Content – instead of trying to predict what malicious data you want to reject, define your criteria for valid data, and force all input to abide by your guidelines.
  - For example, if a user is supplying a last name, you might start by only allowing alphabetic characters and spaces, as these are safe. If you reject everything else, Berners-Lee and O'Reilly will be rejected, despite being valid last names. However, this problem is easily resolved. A quick change to also allow single quotes and hyphens is all you need to do. Over time, your input filtering techniques will be perfected.

Mitigating the Risk (cont)

- Use Mature Solutions – when possible, use mature, existing solutions instead of trying to create your own. Functions like strip_tags() and htmlentities() are good choices.
- Use a Naming Convention – there are many naming conventions that you can use to identify whether a particular variable is tainted. Choose whichever convention is most intuitive to you, and use it consistently in all of your development. A simple example is to initialize an array called $clean, and only store data in $clean once it has been filtered.

Interesting Reads

- http://shiflett.org/articles/foiling-cross-site-attacks - slow to load, but lots of extended examples
- Monster and MySpace attacks (Wikipedia)