Real Web 2.0: Battling Web spam, Part 1
Assess visitor behavior and control workflow to reduce spam

Uche Ogbuji
December 02, 2008

Spam on the Web is one of the biggest threats to a modern Web developer. The "bad guys" become more and more sophisticated every year in how to vandalize and proliferate ads over any Web 2.0 page they can grasp. To make matters worse, spam is increasingly used to distribute malware. The arms race is on, and Web developers need to know what basic tools are available to battle spam on their Web sites. This two-part installment provides a thorough guide to anti-spam techniques. This first article explains how to assess whether a visitor is a spammer and how to organize site workflow to discourage spam.

View more content in this series

In 1994 the National Science Foundation ended its prohibition of commercial speech on the Internet. At that time, e-mail and Usenet were the main forums for communication, and simple publishing systems such as Gopher were trying to establish a wider user base. The Web had barely emerged. That year a law firm named Canter & Siegel posted the first mass commercial spam on Usenet, hiring a Perl programmer to generate advertisements for its "Green card lottery" services, then blasting these to over 6,000 newsgroups. They became celebrities as well as villains, and promptly launched themselves as an outfit producing spam for others, promoting the benefits of spamming, and writing books on "Internet marketing". Since then no online forum has been safe from inappropriate commercial advertising. As the Web has become a more and more important venue for online discussion, and as Web 2.0 techniques have opened up venues for people to write to the Web as much as to read it, the problem of spam has grown with a vengeance.

Sometimes Web spam is a minor nuisance, but more often it's a pernicious problem. Spammers will often not be content to post one or two messages, but will often flood forums with their messages, until it overwhelms the desired subject matter. Sometimes the spam includes pornographic or antisocial messages that discourage participation. Most search engines will devalue pages with such messages, or with links to sites associated with spam, which means that spam can reduce your search engine optimization. And the final injury is that Web publishers end up wasting resources on spam-fighting, taking time way from other tasks.

Web spam comes in many forms, including:
• Spam articles and vandalized articles on wikis
• Comment spam on Weblogs
• Spam postings on forums, issue trackers, and other discussion sites
• Referrer spam (when spam sites pretend to refer users to a target site that lists referrers)
• False user entries on social networks

Dealing with Web spam is very difficult, but a Web developer neglects spam prevention at his or her peril. In this article, and in a second part to come later, I present techniques, technologies, and services to combat the many sorts of Web spam.

**Spammer behavior**

Online contributions by people come in irregular patterns, of timing and content. Spam is generally created by programs, like the Perl script that posted to 6,000 Usenet groups for Canter & Siegel. Sometimes you can use the mechanical patterns of these programs against the spammers. In cases where registration is required, you can use a variety of warning signs to flag an account for further review. For example, if the account is from a top-level domain such as .ru, .br, .biz, that's strongly associated with spammers. If an account is flagged, you might put a hold on the first few postings by that account and review them before releasing the hold.

**Flood control**

Wiki, Weblog, and forum spammers often send dozens of requests within seconds, and you can at least limit damage by limiting the number of requests one user or IP address can send within an interval. This technique is called flood control. You should make sure you control requests across the entire site in this way, and not just one page.

You can apply these behavior assessment techniques, and others in the next section, to prevent other types of abuse on the Web besides spam. For example, if you host a Webmail service, spammers might try to mass-create accounts from which they can send spam, or if you host an online auction site, people might write programs to manipulate the auction process. In general, anywhere you might have a system for registering new users, some of those users might try to find automated registrations for mischievous ends. In general, these techniques help Web developers ensure that people using their services are who they say they are.

**Workflow control**

Most spam comes from robots, so they have some characteristic quirks in their usage of a site that you can exploit. Figure 1 summarizes the difference between typical human and typical robot workflow in submitting a message or comment for addition to your site.

**Figure 1. Typical human versus robot Web workflow**
You can stop a great deal of spam by detecting typical robot workflow, which goes straight to the POST request.

**Form variation**

The first question you ask is whether the main form was loaded by the requester. One way to check is by varying aspects of the form in visible or invisible ways (to the user). The usual invisible trick is to vary the name of form fields. You might send the text area with the main content to one user with a field name of `content_081010_68_45_76_45`, using dates, user IP, and other such modifiers on a base name. You might not want to allow a variant field name to be reused. You can also make visible variations by requiring some users to check a box when posting, perhaps if other patterns indicate that the IP being used is suspicious.

**The nonce test**

A nonce is a hard-to-guess value generated for each page view that contains a form. Then you require that nonce as one of the fields in form submission. If the robot goes straight to POSTing the form submission, it will not have loaded the HTML page, and so it will not know the expected value for the nonce. There are many ways you can tune the specifics of a nonce test. In all cases, you want to be sure that a nonce cannot be reused, otherwise spammers will easily circumvent it. You might want to use the IP address and the date or full time at which the page request was made in order to generate and validate the nonce.

**Detecting JavaScript**

Some robots try to fool nonce tests by loading the page and reading the nonce. A careless spammer will just have the robot immediately send the POST, now that they have the required information. You might want to flag submissions that were made immediately after the form was loaded, with an interval that makes it improbable that a human was able to fill out the form inputs so quickly. This is related to flood-control tests. Even when robots are more careful, they often do not run any JavaScript on the form page. You can use this in several ways. First of all, you might not generate the nonce in the actual content of the form page, but rather through a secondary JavaScript request from the form page. You can set an event handler for the first time the user enters data into the main content field. Figure 2 illustrates.

**Figure 2. Using JavaScript to improve the nonce test**

![Diagram showing the process of using JavaScript to improve the nonce test](image)

The main problem with this is that some users disable JavaScript, and in fact some corporate policies require this, so you should provide secondary means for users to authenticate themselves, perhaps by using form variation only on users not invoking JavaScript. On the other hand, some
spammers launch their attacks using scripts within browser engines, so they do invoke JavaScript. You might use JavaScript tests as a weighting factor in spam detection, rather than as an absolute determinant.

**Test the H.Q. (human quotient)**

One very popular approach to fighting Web spam is a variation on the nonce test. You include with the post form a visual test of some sort that's easy for most people but tricky for a robot. The most popular such method is CAPTCHA. In this test you present an image representing some alphanumeric characters that the user must read and type into a form field. Spammers use optical character recognition to defeat CAPTCHA, so there is usually significant distortion to the images. Figure 3 is an example of a CAPTCHA image.

**Figure 3. A CAPTCHA image requiring the user to answer "smwm"**

![CAPTCHA Image](image)

Even with the mild distortion in this example, spammers have been able to defeat such CAPTCHA, so their adversaries have resorted to greater and greater levels of distortion, as in figure 4.

**Figure 4. A CAPTCHA image requiring the user to answer "following finding"**

![CAPTCHA Image](image)

There are several problems here. For one thing, as CAPTCHA gets more distorted to defeat spammers, it gets harder for people to read. And some people, such as the visually impaired, cannot see CAPTCHA images at all, so this technique exhibits poor accessibility, and can even be illegal in some circumstances. Despite these difficulties, CAPTCHA has become one of the most popular spam prevention devices.

**Textual confirmation**

A similar technique that doesn't suffer such accessibility problems is to present the user a random textual question. If the site has a specific domain, your questions might require basic knowledge of that domain. So on a medical news and information site, you might ask a question such as, "What is the primary organ associated with breathing?", expecting an answer of "lungs". With this technique it's important to have a wide variation of such questions, and to be careful with the wording so that people can generally answer, but robots cannot generally guess.

**Wrap up**

You can add subtlety to all the techniques I discussed in this article by forcing people to preview their postings. Just by adding this extra workflow stage you'll catch some spammers, and if you do it carefully, for example using JavaScript to automate preview for some users, the level of
inconvenience shouldn't stifle contribution. You could apply CAPTCHA, form variation, nonces, and more, depending on workflow, which makes things even more complicated for spammers in a manner that most legitimate users won't even notice.

Assessing behavior and managing workflow are enough to reduce spam, but usually not enough to eliminate it. For example, some spammers hire people to circumvent all the controls discussed in this article (sometimes called a "mechanical turk" attack). They seek people in places where labor costs are cheap and pay them to go to a target site and leave a spam message by hand. To address mechanical turks, as well as the most sophisticated spam robots, you need to harness the power of the large community of people who hate spam as much as you do. I'll explain how you do so in the next installment of this series.

© Copyright IBM Corporation 2008
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)