Discouraging Wiki Spam with Proof-of-Work

Justin Bailey (justinb@cs.pdx.edu)
Dr. Wu-chang Feng, CS510, Winter 2012

March 13, 2012

My project uses a proof-of-work scheme to discourage wiki spam. I implemented and tested a prototype to be used with the open-source DokuWiki project. DokuWiki can be extended through plugins; my plugin requires wiki authors to solve a computational problem before they can submit content. I used several heuristics to determine the difficulty of the puzzle presented to the author. The source code for my work can be downloaded at `https://github.com/m4dc4p/dokuwiki-kapow`.

Wiki Software

"Wiki" software enables collaborative content creation and editing on the internet. Authors do not need to use HTML to create content, and articles written will generally be published immediately. Wikipedia, one of the top 10 most visited sites on the internet,[1] hosts over 3 million articles,[2] all authored by volunteers and published using wiki software.

The popularity of some wiki sites and the ease with which content can be created gives rise to "wiki spam." Wiki spam consists of links to spam sites, with no real content. By placing links to spam sites on various wikis, the spammer attempts to raise their spam site's ranking in search engine results,[3] ultimately resulting in more traffic (and revenue) for the spammer.

Wiki software supports a variety of anti-spam techniques. The software might only allow content creation from certain IP addresses, limit content creation to registered users, or require that all content be approved by a moderator before being published.[4] Each

[1] According to `http://www.google.com/adplanner/static/top1000`, accessed March 13, 2012.

[2] From `http://en.wikipedia.org/wiki/Special:Statistics`.

[3] See `http://en.wikipedia.org/wiki/Pagerank` for more.

[4] Though not specifically for wiki software, the article `http://en.wikipedia.org/wiki/Spam_in_blogs` discusses many of these techniques.

technique makes wiki spam harder to create, but also makes it harder for legitimate authors to create content.

Proof of Work

Client puzzles, or "proof-of-work," attempt to thwart spam (and other malicious behavior) by changing the economics of spam creation. Proof-of-work requires authors to solve a computationally difficult problem before their content will be published.

The difficulty of the puzzle can be scaled according to heuristics that determine the "spamminess" of the particular author. For non-spammers, the server gives an easy problem that can be solved nearly instantaneously. For spammers, the server gives a problem that may not be solved for years.

Spammers raise their search engine ranking by posting a large number of links on different sites. The more links they can post, the more their ranking will rise. Making it hard to post links changes the economics for spammers — if they cannot post links quickly and cheaply, they will not make money. Therefore, by using "proof-of-work" to raise the cost for spammers, we can reduce the profit potential of wiki spam.

DokuWiki

The DokuWiki[5] project provides a free, open-source wiki implementation written in PHP. DokuWiki offers a number of mechanisms to combat spam: the administrator of a given DokuWiki installation can choose to limit content creation only to registered users; content can be blocked if it contains certain words (such as "Viagra"); external links can be modified so they do not contribute to search engine ranking. DokuWiki itself can by customized via plugins, an number of which offer other anti-spam tools.[6]

A Proof of Work Plugin

For my project, I created a proof-of-work plugin for DokuWiki.[7] My plugin extends DokuWiki's editing page so that authors must solve a client puzzle before the wiki

[5]Hosted at `http://www.dokuwiki.org`.

[6]See `http://www.dokuwiki.org/plugins?plugintag=spam#extension__table`.

[7]Source code available at `https://github.com/m4dc4p/dokuwiki-kapow`. I targeted the "Angua" (2012-01-25) version of DokuWiki.

software will accept their content updates. My plugin uses several heuristics to determine the difficulty of the puzzle presented to the author.

I based my work on the guest book prototype developed by Tien Le, Wu-chang Feng, and Ed Kaiser.[8] They provide a PHP and JavaScript implementation of a hard cryptography problem with adjustable difficulty and a a DNS blacklisting heuristic, both of which I re-used entirely.

Heuristics

My prototype uses the four heuristics shown in Figure 1 to determine the puzzle's difficulty. Each heuristic evaluates to an integer score. The plugin calculates difficulty by raising 128 to the sum of all the heuristic scores:[9]

$$Dc = \lfloor 128^{score} \rfloor \tag{1}$$

where Dc represents the difficulty.

Le, Feng and Kaiser implemented Rule (1), a DNS "blacklist" look-up that takes a client IP and determines its threat level. Dc quickly blows up with even the least threatening IPs. In practice, this heuristic may be too aggressive. If so, it could be scaled to give a result between 0 and 3 rather than 0 and 255.

Rule (2) checks if the author authenticated themselves. If so, it evaluates to 0, otherwise 1. This rule assumes that a spammer will not bother to register an account before posting spam.

The breadcrumb heuristic, Rule (3), checks if the author navigated anywhere else on the wiki. If the author browsed at least two other pages, the rule evaluates to 0. Otherwise, the rule calculates 2 minus the number of pages navigated. This rule assumes a spammer will go to one page and begin posting spam, whereas a normal user would browse the wiki before adding content.

Rule (4) checks if the author browses from the same subnet as the wiki. This rule evaluates to 0 when the author originates from the same subnet; otherwise, it evaluates to 1. This rule gives more trust to local traffic. For a wiki on the open internet it would probably not be appropriate.

Alone, none of these rules (except the DNS blacklist) will generate a very difficult

[8] See `http://kapow.cs.pdx.edu/kapow/guestbook`.

[9] I did not investigate the exponential behavior of the puzzle deeply, but scores above $16,384$ (i.e., 128^2) result in puzzles that are not solved in the time I have patience for.

	Heuristic	Range	Motivation
(1)	DNS Blacklist	0 – 255	Calculates spam "threat" of the author's IP using `http://httpbl.org`.
(2)	User Logged In	0 – 1	We give more trust to authenticated users.
(3)	Empty Breadcrumbs	0 – 2	Spammers unlikely to browse the site, therefore breadcrumb trail will be empty.
(4)	IP Not in Subnet	0 – 1	Spammers not likely to be from the same subnet.

Figure 1: Heuristics used to determine the client puzzle's difficulty.

puzzle. However, if the author fails any three of the rules, they will find they effectively cannot post content.

Discouraging Spam vs. Detecting Spam

Early on I attempted to use the "spam" score calculated by SpamAssassin[10] to determine if the content submitted was spam or not. This approach came directly from the webmail prototype of Le, Feng and Spencer.[11] Necessarily, authors needed to submit content they could be assigned a puzzle. In contrast, I setled on an approach that assigns a puzzle to each author immediately, as they compose content.[12]

This approach attempts to discourage spam from being posted at all. When an author begins to create or edit a page, my plugin uses the heuristics given above and assigns a puzzle to the author. Until the puzzle is solved, the author cannot submit their content. Legitimate authors will probably take a little time composing their content, so they may not even notice the wait. Even a legitimate user given a hard puzzle may not mind so

[10]Found at `http://spamassassin.apache.org`.

[11]See `http://kapow.cs.pdx.edu/kapow/mail`.

[12]The discussion found at `http://wiki.apache.org/spamassassin/BlogSpamAssassin` inspired my approach.

much if they only create or edit one piece of content. Spammers, on the other hand, will be discouraged from posting content and editing a large number of pages.

In contrast, the detection approach makes all users wait to solve a puzzle. The system will only generate a puzzle after the author submits their content. The author must then wait to solve the puzzle before they can proceed. While this approach may do a better job of detecting spam in the long run, I believe the legitimate authors pay too high a price.

Future Work

My prototype only implements the most rudimentary detection rules, and does not lend itself to re-use by other plugin authors. DokuWiki supports an event system by which plugins can communicate and react. My plugin could expose events for calculating the difficulty score, allowing other plugins to implement their own heuristics for detecting spammers.

My plugin always (and only) modifies DokuWiki's editing form. Proof-of-work might be useful elsewhere in the site, though. The plugin could be architect ed to expose its proof-of-work implementation for re-use in other contexts.

Conclusion

My project addresses the problem of wiki spam. I created a plugin for the open-source DokuWiki project. My plugin used a proof-of-work scheme to discourage spam content from being posted. The plugin could be extended in a number of ways, such as by implementing an event-based system for extending the heuristics used to determine puzzle difficulty. Source code for the plugin can be downloaded from `https://github.com/m4dc4p/dokuwiki-kapow`.