Know thy enemy

Created 21 January 2007

Spam sucks. Any site which allows unauthenticated users to submit forms will have a problem with spamming software (spambots) submitting junk content. A common technique to prevent spambots is CAPTCHA, which requires people to perform a task (usually noisy text recognition) that would be extremely difficult to do with software. But CAPTCHAs annoy users, and are becoming more difficult even for people to get right.

Rather than stopping bots by having people identify themselves, we can stop the bots by making it difficult for them to make a successful post, or by having them inadvertently identify themselves as bots. This removes the burden from people, and leaves the comment form free of visible anti-spam measures.

This technique is how I prevent spambots on this site. It works. The method described here doesn't look at the content at all. It can be augmented with content-based prevention such as Akismet, but I find it works very well all by itself.

By watching how spammers fail to create spam on my site, there seem to be three different types of spam creators: Playback spambots, form-filling spambots, and humans.

Playback bots

These are bots which have recorded POST data which they replay back to the form submission URL. A person visits the form the first time, and records the form data. Certain fields are marked as slots to be filled in with randomized spam later, but the structure of the form is played back verbatim each time. This includes the names of the fields, and the contents of hidden fields.

These bots don't even bother looking at the form as served by the site, but blindly post their canned data to the submission URL. Using unusual field names to avoid these bots will only work for a week or so, when they will then record the new field name, and begin posting with it.

A playback bot can be stopped by varying the hidden data on the form so that it will not be valid forever. A timestamp is a simple way to do this, making it possible to detect when old data is being replayed. The timestamp can be made tamper-proof by hashing it with a

secret and including the hash in the hidden data of the form. Replaying can be further hindered by including the client's IP address in the hash, so that data can't even be immediately replayed across an army of spambots.

Form-filling bots

These bots read the form served by the site, and mechanically fill data into the fields. I don't know if they understand common field names (email, name, subject) or not. On my site, I've observed bots that look at the type of the field, and fill in data based on the type. Single-line edit controls (type=text) get name, email, and subject, while textareas get the comment body. Some bots will fill the same data into all the fields of the same type, while others will enter (for example) different first names into each of the single-line fields.

Form-filling bots can be stopped by including editable fields on the form that are invisible to people. These fields are called honeypots and are validated when the form data is posted. If they contain any text, then the submitter must be a bot, and the submission is rejected.

Using randomized obscured field names, and strict validation can also stop these bots. If the email field must have an @-sign, and the name field must not, and the bot can't tell which field is email and which is name, then the chances it will make a successful post have been greatly reduced.

Humans

These are actual people using your form. There's nothing you can do to stop them, other than to remove the incentive. They want link traffic. Use the rel="nofollow" attribute on all links, and be clear that you are doing it.

Building the bot-proof form

The comment form has four key components: timestamp, spinner, field names, and honeypots.

The **timestamp** is simply the number of seconds since some fixed point in time. For example, the PHP function time() follows the Unix convention of returning seconds since 1/1/1970.

The **spinner** is a hidden field used for a few things: it hashes together a number of values that prevent tampering and replays, and is used to obscure field names. The spinner is an MD5 hash of:

- The timestamp,
- The client's IP address,
- The entry id of the blog entry being commented on, and
- A secret.

The **field names** on the form are all randomized. They are hashes of the real field name, the spinner, and a secret. The spinner gets a fixed field name, but all other fields on the form, including the submission buttons, use hashed field names.

Honeypot fields are invisible fields on the form. Invisible is different than hidden. Hidden is a type of field that is not displayed for editing. Bots understand hidden fields, because hidden fields often carry identifying information that has to be returned intact. Invisible fields are ordinary editable fields that have been made invisible in the browser.

The invisibility of the honeypot fields is a key way that bots reveal themselves. Because bots do not process the entirety of the HTML, CSS, and Javascript in the form, and because they do not build a visual representation of the page, and because they do not perceive the form as people do, they cannot distinguish invisible fields from visible ones. They will put data into honeypot fields because they don't know any better.

The form is built as usual, including:

- editable fields for all of the information we want to collect from the user,
- hidden fields for identifying information, including the timestamp, the spinner, and the entry id,
- invisible honeypot fields of all types, including submission buttons.

Processing the post data

When the form is posted back to the server, a number of checks are made to determine if the form is valid. If any validation fails, the submission is rejected.

First the spinner field is read, and is used to hash all of the real field names into their hashed counterparts so that we can find data on the form.

The timestamp is checked. If it is too far in the past, or if it is in the future, the form is invalid. Of course a missing or non-integer timestamp is also a deal-breaker.

The value of the spinner is checked. The same hash that created it in the first place is re-computed to see that the spinner hasn't been tampered with. (Note that this check isn't actually necessary, since if the spinner had been modified, it wouldn't have successfully hashed the timestamp field name and the timestamp verification would already have failed, but the extra check is harmless and reassuring.)

Check the honeypots. If any of them have any text in them, the submission is rejected.

Validate all the rest of the data as usual, for example, name, email, website, and so on.

At this point, if all of the validation succeeded, you know that you have a post from a human. You can also apply content-based spam prevention, but I have not found it to be necessary.

Making honeypots invisible

This is the essence of catching the bots. The idea here is to do something to keep the honeypot fields from being visible (or tempting) to people, but that bots won't be able to pick up on. There are lots of possibilities. As you can see from looking at my comment form, I've simply added a style attribute that sets display:none, but there are lots of other ideas:

- Use CSS classes (randomized of course) to set the fields or a containing element to display:none.
- Color the fields the same (or very similar to) the background of the page.
- Use positioning to move a field off of the visible area of the page.
- Make an element too small to show the contained honeypot field.
- Leave the fields visible, but use positioning to cover them with an obscuring element.
- Use Javascript to effect any of these changes, requiring a bot to have a full Javascript engine.
- Leave the honeypots displayed like the other fields, but tell people not to enter anything into them.

Criticisms

Let me address a few common criticisms.

Defeatability

In theory, it is possible for a spambot to defeat any of these measures. But in practice, bots are very stupid, and the simplest trick will confuse them. Spam prevention doesn't have to make it theoretically impossible to post spam, it just has to make it more difficult than most of the interesting forms on the internet. Spammers don't make software that can post to any form, they make software that can post to many forms.

A relevant joke:

> Jim and Joe are out hiking in the forest, when in the distance, they see a huge bear. The bear notices them, and begins angrily running toward them. Jim calmly checks the knots of his shoes and stretches his legs.
>
> Joe asks incredulously, "What are you doing? Do you think you can outrun that bear!?"
>
> Jim replies, "I don't have to outrun the bear, I just have to outrun you."

In any case, yes, spammers may eventually write spambots sophisticated enough to navigate honeypots properly. If and when they do, we can switch back to CAPTCHAs. In the meantime, honeypots work really well, and there are lots of ways to make them invisible we haven't even needed to use yet.

Accessibility

Users that don't use CSS or Javascript will be exposed to all of the honeypot fields. A simple solution is to label the fields so that these users will leave them untouched. As long as no text is entered into them, the form will submit just fine.

It works

This technique has been keeping spam off my site for a year now, and works really well. I have not had problems with false positives, as Akismet has had. I have not had problems with false negatives, as keyword-based filtering has had. Spambots may get more sophisticated, but their software complexity will have to increase orders of magnitude before they can break this method.

See also

- Negative CAPTCHA, Damien's post about a similar technique. The comments on that post got me energized to write up my technique.
- My blog, where many software engineering topics are discussed.