Comment Spam Injection Made Easy

Marco Ramilli, Marco Prandini
DEIS - University of Bologna, Italy
Motivation

- User communities are steadily gaining importance in terms of business impact, e.g.
  - Product promotion
  - Brand reputation
- Large communities suffer from an asymmetry of forces
  - Interested users read malicious comments in detail
  - Centralized monitors are quickly overwhelmed by the number of comments
- Automated tools are exploited on both sides
  - To inject malicious comments
  - To detect suspicious comments... but what's suspicious and what's not?
Not only one kind of spam

The kind of undesired messages on a forum, blog, or wiki can be roughly categorized as:

- **Link spam**
  - Tries to catch the attention of the reader to convey advertisements or other messages, not related to the topic of the discussion board, and usually including the URL of an external site
  - Not only annoying, but potentially dangerous (leading to malware-carrying sites)

- **Comment spam**
  - Subtler, apparently innocuous messages that are injected to false the “real” feeling of the community on the topic of the discussion board
Is comment spam important?

I'll give a controversial reply to the question: it's important because very few players realize it's important.

- The institutional victims rarely realize they are, and can't do much to curb the problem without risking significant image backlashes.

However, from a different viewpoint...

- The underground is actively engaged in developing injection tools.

So, probably, silence is not such a good measure of the absence of the problem (but everyone decide).

If STV communities are going to open up, will the same problems now faced by social networks be likely to appear there?
Related work

The focus of available countermeasures is placed on automatic tools; they work by:

1) Impeding the access to the submission interface
2) Scanning the messages for undesirable content

Category (1) is usually implemented by placing non-automatically-computable items in the posting forms, like CAPTCHA, Challenge/Response functions, ...

– Usually effective, but recently cracked
– Invasive (especially in highly interactive contexts)

Category (2) typically exploit the same techniques used for e-mail spam, like Bayesian filtering, blacklists, ...

– Mostly based on the assumption that the undesired message is strikingly different from the “average good message”
An experiment

- Supposing that access-limiting countermeasures like CAPTCHA are not deployed, what's the effectiveness of content-filtering countermeasures?
- Our testbed:
  - Target: a WordPress-powered blog
  - Tested filters: Akismet, Defensio
  - Attack tool: a very simple message composer and sender
An experiment - results

- We put 60 message parts in the DB – 20 introductions, 20 bodies, 20 conclusions: up to 8000 different messages can be generated.
- If the delay between posts is too small (less than 15s), WP’s anti-DoS timeouts kick in and the percentage of accepted posts is lower than 50%
- When the overall number of sent messages approaches the size of the available set, messages identical to previously accepted ones start to be generated. WordPress is able to detect and reject “perfect” duplicates.
- But... all the distinct messages have been successfully delivered without neither Akismet nor Defensio being able to interfere!
What should we look for?

- The only filter that showed some effectiveness was the standard anti-duplication feature of WordPress
- Our idea is that of extending the same approach to “too similar” messages, instead of limiting it to identical ones only
- So... what's the meaning of “similar”?
  - Hypothesis #1: messages are made of sentences
  - Hypothesis #2: sentences are easy to detect
  - Hypothesis #3: automated tools will reuse sentences
  - Thesis: extracting and storing sentences, instead of storing whole messages, provides a reliable database of what should not be accepted again
Prototype design guidelines

- In designing the proposed system, we followed the principle that our hypotheses are probably no so easy to satisfy in the real world.
- Consequently, the architecture should easily accommodate changes in terms of sentence recognition within the submitted messages and matching algorithm between the sentence under analysis and the stored body of knowledge.
Architecture - overview

- From the broadest perspective, the proposed solution works according to a very simple model:
Instead of storing the sentences in a database, we chose to represent them as clauses in a Prolog theory.

The whole theory can automatically:

- associate a score to a sentence
- provide sentence recognition
  - In this first prototype, we implemented this feature as simply splitting the message according to punctuation
- compute the total message score

```
check(Message,Punteggio) :- message(Message,Punteggio).
message(['spam'|T],Punteggio) :- message(T,P), (Punteggio is P + 6).
message(['work with us'|T],Punteggio) :- message(T,P), (Punteggio is P + 6).
message([],0).
```
Architecture – supervised learning

Two sources of sentences contribute to the body of knowledge

- (1) An initial set provided by a system trainer
  - Initial training is very useful to speed up the recognition of well-known spam sentences, if any
  - Static knowledge can compensate for common but innocuous sentences, whose frequency would otherwise be mistaken for a spam indicator
    - Negative or null scores can be associated to sentences
Architecture – supervised learning

- Two sources of sentences contribute to the body of knowledge
  - (2) Automatically added sentences
    - Unknown sentences can be
      - Ignored if not associated to any other known spam indicator
      - Automatically added to the body of knowledge if present in a message that's been classified as spam
      - Submitted to the trainer for a score assignment in dubious cases
    - Known sentences can have their score updated, for example multiplied by a fixed value when they are found again and again.
      - Multiplying instead of adding a constant allows to keep null the score associated to manually-inserted innocuous sentences
In this prototype, we didn't put much effort
- neither in modeling a complex theory encompassing all the possible modes of combining sentences into messages,
  - Presently, a preprocessor reduces every message to a fixed form as a sequence of separated sentences
- nor in trying to abstract from the slight variations that could make two very similar sentences perceived as totally different
  - We are working to integrate a general concept of feature extraction, hopefully within the main theory
Implementation

- The system is implemented according to the client-server model
  - The client is typically a plug-in for the message engine to protect
  - The server does everything, receiving the message from the client and returning a decision (spam or ham)
  - The two components can be packaged to form a self-contained solution, or a server can be shared among many clients in order to leverage the same knowledge base
Implementation

- The prototype is written in Java, and makes use of the tuProlog engine

```java
public int query_theory(String message) {
    String address = censure(message);

    // set new theory to query
    refreshTheory();
    out.println("core as refreshed the theory !!");
    int point = 0;
    String goal = "check(['" + message + "]",X).";

    System.out.println(goal);
    try{
        out.println("I'M SOLVING YOUR GOAL !!!!! it's very hard acc..");
        SolveInfo info = this.engine.solve(goal);
        point = Integer.parseInt(info.getTerm("X").toString());
    }
    catch(Exception ew){ out.println("Error Query theory " + ew); }
    // detach the theory..... for update.
    this.clearTheory();
    return point;
}
```
Some screenshots – WP integration

![Screenshot of WordPress admin panel](image.png)

**Plugin Management**

Plugins extend and expand the functionality of WordPress. Once a plugin is installed, you may activate it or deactivate it here.

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Version</th>
<th>Description</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Akismet</td>
<td>2.1.3</td>
<td>Akismet checks your comments against the Akismet web service to see if they look like spam or not. You need a WordPress API key to use it. You can review the spam it catches under &quot;Comments.&quot; To show off your Akismet stats just put <code>&lt;php akismet_counter(); ?&gt;</code> in your template. See also: WP Stats plugin. By Matt Mullenweg.</td>
<td>Activate</td>
</tr>
<tr>
<td>Defensio Anti-Spam</td>
<td>1.5.2</td>
<td>Defensio is an advanced spam filtering web service that learns and adapts to your behaviors and those of your readers. To use this plugin, you need to obtain a free API key. Tell the world how many spam Defensio caught! Just put <code>&lt;php defensio_counter(); ?&gt;</code> in your template. By Karabunga, Inc.</td>
<td>Activate</td>
</tr>
<tr>
<td>Hello Dolly</td>
<td>1.5</td>
<td>This is not just a plugin, it symbolizes the hope and enthusiasm of an entire generation summed up in two words sung most famously by Louis Armstrong: Hello, Dolly. When activated you will randomly see a lyric from Hello, Dolly in the upper right of your admin screen on every page. By Matt Mullenweg.</td>
<td>Activate</td>
</tr>
<tr>
<td>SpamBam</td>
<td>2.3.2</td>
<td>A plugin that hopefully eliminates comment spam By Gareth Hayes for BlogSecurity.net.</td>
<td>Activate</td>
</tr>
<tr>
<td>SpamStress</td>
<td>0.1</td>
<td>A plugin that hopefully eliminates comment spam thank basic Artificial Intelligent Step By Marco Ramilli.</td>
<td>Deactivate</td>
</tr>
</tbody>
</table>
Some screenshots – plugin connection to the engine
Some screenshots – training
Conclusions / Future work

- The work was motivated by an “instinctive” suspicion about the impact of comment spam
- Ironically, we were quicker at proving the feasibility of the attack than at developing the countermeasure
- The proposed system works well in the outlined scenario:
  - we are aware that this could be the result of a too narrow definition of the problem
  - we tried to design a filtering architecture, or meta-algorithm, rather than a parametric filter, flexible enough to adapt to structures of undesired messages different from that we hypothesized
  - we are working to generalize even more the recognition capabilities of the core theory, to evolve from the model “message as a concatenation of sentences” to “message as a combination of concepts” (maybe a good application for text mining techniques?)
Thank you.