Adaptive Filtering of Comment Spam in Multi-user Forums and Blogs

Marco Ramilli and Marco Prandini

University of Bologna - DEIS
Viale Risorgimento 2, 40136 Bologna, Italy

Abstract. The influence of web-based user-interaction platforms, like forums, wikis and blogs, has extended its reach into the business sphere, where comments about products and companies can affect corporate values. Thus, guaranteeing the authenticity of the published data has become very important. In fact, these platforms have quickly become the target of attacks aiming at injecting false comments. This phenomenon is worrisome only when implemented by automated tools, which are able to massively influence the average tenor of comments. The research activity illustrated in this paper aims to devise a method to detect automatically-generated comments and filter them out. The proposed solution is completely server-based, for enhanced compatibility and user-friendliness. The core component leverages the flexibility of logic programming for building the knowledge base in a way that allows continuous, mostly unsupervised, learning of the rules used to classify comments for determining whether a comment is acceptable or not.

1 Introduction

One of the most interesting developments within the World Wide Web begun with the appearance of real collaborative authoring platforms like Forums, Blogs, Wikis and so on. Each of these applications essentially implements a variation of the same concept: a central subject is published (as a forum topic, a blog post, or a wiki page) and the user community can provide corrections, integrations and useful links. The importance of these innovative meeting platforms has quickly come to the attention of business players, since they allow both to get direct feedback useful for product development and placement, and to enable viral marketing of good products by means of recommendations. Unfortunately, also corporate attackers know the value of these tools as targets, and often they try to modify the authentic meaning of the community-provided feedback, by adding false knowledge to the system through fake comments. In the same way as *spam* hinders e-mail convenience, by burying useful communications under overwhelming amounts of unsolicited ones, the insertion of malicious additional information on knowledge-exchange applications can hide the correct items; hence the name of *comment spam* [24, 14, 16]. This paper illustrates a research activity aimed at mitigating the problem of comment spam, which is regarded as potentially very dangerous [25, 15, 18]. As it will be better explained in the following sections, the embraced approach tries to optimize effectiveness without requiring the limiting operational assumptions,

Ramilli M. and Prandini M. (2008).
Adaptive Filtering of Comment Spam in Multi-user Forums and Blogs.
In *Proceedings of the 6th International Workshop on Security in Information Systems*, pages 122-131
DOI: 10.5220/0001740901220131

especially regarding the client side, quite commonly afflicting many of the presently used systems. The paper is organized as follows: section 2 clarifies the importance of targets, the attack methods, and points to related work; section 3 states the design goals that have been considered essential in the quest for a new solution; section 4 illustrates a behavior of the proposed system suitable for attaining the previously stated goals; section 5 shows the architecture chosen to implement the illustrated behavior; section 6 gives some preliminary considerations on implementation and testing issues; finally, conclusions are drawn in section 7.

2 Problem Definition and Related Work

Many companies leverage the potential expressed by user communities in various ways. Customer feedback is useful to decide how to make a product more successful and to test new ideas. User behavior can suggest market opportunities. On independent communities, user ratings can deeply affect the reputation of a product and its maker; the bigger a company, the higher the number of comments and their dispersion over both internal and independent platforms. Of course companies cannot appoint a feedback manager to read every single comment posted on the web, so they build some special filter able to grab the ”meaning” of the comment. Understanding the meaning of a comment is extremely inaccurate if the easiest, word-based pattern matching filter techniques are exploited, as in reality many companies are doing.

Consequently, attackers can easily inject fake comments in order to change, for instance, the perceived satisfaction related to a product [17, 13]. Since companies usually assume that their filters are not precise but the basic data is safe, if the attacker is able to inject a fake comment without being detected, seeing the product public perception change could urge the company to adopt costly strategic decisions.

Spammers usually exploit three basic techniques for targeting their victims:

- If a community web site is powered by some common application, for example Wordpress [4], then a spammer can easily identify this by scanning for known URLs, and then exploiting potentially known vulnerabilities.
- Field names like ”name” or ”email” in HTML forms can usually be identified as spam targets because of their relation to user identification; spammers submit messages to any site providing this kind of form, losing in accuracy but potentially reaching a broader set of targets more quickly.
- Accepting an even worse trade-off between accuracy and search effort, spammers can simply try and submit spam to every form they can find regardless of the likeliness of the the field names.

Once a target has been identified, spammers typically perform two different kind of attacks:

- Self Replacing Contents. This attack begins with a malicious comment posted on the feedback manager by the attacker. Afterward the attacker runs a software able to repeat the previous message, randomizing the time lapses and the source.

– Smart Comments Generator. This attack begins with a malicious sentences database pre-built by the attacker. Afterward the attacker runs a daemon software able to retrieve sentences from the database, composing them to create messages and sending them to the target site.

Spammers almost always use automated tools in both phases of their attack: first, to scan the net for vulnerable web applications, then to submit comment spam to the interesting targets. As for any other attack, tracing and stopping comment spam at the network level can be made more difficult by exploiting compromised hosts as stepping stones and other well-known evasive techniques, and hence the defensive strategies must involve the specificity of the application layer.

The first attempt at solving this problem, then, was to address the methods spammers use to automate their activity. Normally a spammer would attempt to directly submit the spam without the use of a real, interactive web client. This peculiarity is worked to the defender's advantage by most of the currently proposed solutions, which exploit some kind of client-side-based approach to discriminate between human- and software-submitted comments.

A possible method requires passing a shared secret that a spammer cannot acquire without executing client-side code during a "real" session initialization. The shared secret is constructed for example by creating random client and server side code blocks; the result of these code blocks are then used as a shared secret. Without the support for executing, for example, Javascript, it becomes very difficult for a spammer to successfully acquire the shared secret because of the random construction of the blocks. However, many users too have legitimate reasons to disable the execution of Javascript, and consequently they will automatically be identified as spammers, so this cannot be considered a viable solution.

Another popular method is based on CAPTCHAs, i.e. images containing text that is impossible to automatically extract. Supposedly, then, the ability to type the text identifies a human intelligence on the client side. However, these approaches are regarded as only mildly effective [26, 22, 20, 8, 7], and exhibit obvious, significant drawbacks in terms of user-friendliness [2].

Akismet [1] takes the approach of a centralized spam identification system and licenses api keys to charge users for using the service. It does also have a "free for personal use" option which enables to protect a blog for free as long as it only has a small amount of visitors. Akismet parses blog comments and compares them to previously held spam comments in order to identify spam. There are also many free plug-ins developed by user communities on different engines for instance: spambam on WordPress, MOD for phpBB, phrase spam moderation for blojsom, Spam-X for GeekLog, spamkiller, spamcheck and AntiSpam for Nucleus.

All of these solutions are characterized by a fixed logic, i.e. the knowledge base and parameters used for message classification can evolve, but the underlying filtering algorithm that uses them remains the same. What differentiates our solution is the ease of reprogramming the classification engine itself, by leveraging the peculiarity of logic programming.

3 The Proposed Solution

Consequently to the analysis presented in the preceding sections, the aim of the research was set to meeting the design goals summarized hereinafter.

Transparency. Users should not be aware of the filtering system to make it work. This specification implies that any decision about the admissibility of messages must be taken without the help of interactive or automated means of telling ”real” clients from automated ones. There is also an additional, quite important advantage of concentrating all the burden on a server: any lightweight client, for instance as it is commonly found on mobile devices, will work, paving the way for the extension of the proposed system to text messaging.

Evolution. The criteria for sorting out bad messages should evolve during the system's lifetime, taking advantage of what the system sees. This property of course doesn't rule out the need for a statically-provided initial knowledge base (in fact, it is required), but the ability to continuously, correctly track the adaptive evasive measures used by the attackers is deemed as very important.

Accuracy. The system should take advantage of the present knowledge about the commonly-used spam composition techniques in order to optimize the rate of correct classification, at the same time being open for the possible integration of novel knowledge in the future.

Efficiency. The system should be able to carry out its tasks introducing acceptable delays within reasonable hardware constraints.

The first and foremost design choice was to follow a quite classical approach, basing the system on the concept of computing a score for each processed comment, and discriminating between spam and ham depending on the score crossing a given threshold or not.

The comparison to Bayesian filtering commonly used against e-mail spam is quite natural. However, the design of the proposed approach must take into account some advantages and disadvantages peculiar of the different context. Among the advantages, it is useful to notice that many content-hiding methods (like the use of images or obfuscated links), which are commonly found in e-mail spam, are typically disallowed in comment platforms, allowing filtering systems to deal with natural text only. Among the disadvantages, for example, there is the difficulty of enhancing scoring accuracy by means of blacklists or whitelists, because comment spam is less massive, more targeted than its e-mail counterpart, making the distributed collection of evidence indicting possible spam sources harder. Another differentiating feature is that while e-mail spammers work with the only goal of evading filters, even when this means sending blatantly incoherent messages, comment spammers have much less freedom in constructing their messages, which must be credible enough to trick human readers into accepting them as authentic comments.

Following these observations, another key choice has been made regarding how to represent the knowledge that allows to decide whether a message is spam or ham (and,

at the same time, how to to extract this information from the messages in a way that allows efficient and effective processing.) Presently, as already anticipated, the most commonly used technique for mass-submission of automatically generated spam is mixing sentences conveying the desired meaning under different forms. The devised system, then, was designed according to a general feature extraction paradigm, but currently adopts a very simple and efficient algorithm using punctuation analysis to split each message in simple sentences. The single sentence is the base element of the knowledge base, and is associated with a score representing the probability of finding it in a spam message. Furthermore, instead of separating the components taking care of the sentence storage and of comment processing, exploiting a standard database for the latter function, the knowledge base and the scoring algorithm are integrated as an expandable prolog theory. The resulting system is characterized by a pragmatic approach to the usage of the powerful computing models which are typical of the artificial intelligence field; a twofold advantage is achieved: on the one hand, being able to quickly store and effectively leverage the knowledge needed for the subsequent classification task, as detailed in the following sections, on the other hand paving the way to the possible future integration with more powerful classification algorithms, for example taking into account multiple sentences at once during the scoring process.

4 Operation

From the functional point of view, the system we are describing has three distinct modes of operation: Initial Training, Query Processing, and Learning. The first mode of operation is needed at system startup or if novel spam building techniques appear, while the second and third ones are actually strictly linked and together represent the normal state of the system.

4.1 Initial Training

Every score based automaton needs a training phase during which the administrator (the automaton administrator) teaches it the most important pieces of knowledge. In this phase typical sentences are fed to the system, each one associated to a score that can be positive or negative.

Negative scores are associated to innocuous sentences that are known to appear inside spam messages, and are useful to avoid that during the learning phase these sentences receive a strong spam connotation, possibly leading to a high rate of false positives. Positive scores are associated to sentences typical of comment spam. In both cases, the higher the absolute value, the higher the confidence in the sentence classification.

During the training process, the administrator submits a purposely crafted form through the feedback manager to initiate the training phase of the anti spam engine, sending sentences and their scores. The anti spam engine, following the training request transforms the sentence and the associated score in a theory-rule, including it in its knowledge. The longer the administrator coaches the anti spam engine, the more accurate the anti spam engine becomes in discriminating legitimate sentences from spam.

4.2 Querying Phase

After the training phase, the anti-spam engine should be able to reply correctly to most of the queries . In order to classify a message, it is divided in sentences, which are individually matched against the prolog theory representing the knowledge base. A score is associated to each sentence, and the sum of all the sentence-scores is compared to the decision threshold. Of course, the threshold will be chosen as a positive value, so as to be crossed when a comment exhibits a dominance of spam clues.

4.3 Learning

The system has to update its knowledge base after the detection of a spam message.

Learning from past history is a normal behavior for humans but it is less obvious for automata. The meaning of machine learning has been well discussed in the past [23, 6]; for this reason saying that our system attains this goal is a maybe too strong claim. In our model, learning means that the anti spam engine's knowledge is growing up in function of past detected spam. Every message is evaluated from the core engine; evaluating messages means dividing messages into sentences and then evaluate each sentence. If a message is considered spam, the probability that an automated spamming tool will reuse its own sentences to build another spam message is high, and consequently adding these sentences to the knowledge makes sense.

The system is purposely unbalanced towards the learning of new spam-indicating sentences, in order to be as effective as possible at filtering. This bias, as previously said, can be counterbalanced during the initial training phase, but false positives can arise in the long run. We claim that this behavior is preferable to having a higher rate of false negatives silently slipping through the system. A false positive is easily spotted by the legitimate user whose comment is blocked, and consequently with a little cooperation can be brought to the attention of the system administrator, who can adjust the knowledge base accordingly.

5 Architecture

The system's architecture can be deduced from the complete use case diagram shown in Figure 1; it is modeled according to a structure which differentiates three logical blocks depending on the communication side the lie on. The first block is located on the client browser and represents what the end user sees. It is a purely logical component, i.e. no software is installed on the client to make the proposed system work. The second block is on the commenting platform web server and represents where the user wants to publish her comments (Feedback Manager). The third block, the main anti spam engine, can be co-located with the Feedback Manager, but is designed to be accessible through remote web service.

When a user tries to post a comment from her browser, the message reaches the feedback manager which queries the anti-spam engine. The reply summarizes the probability of the comment of being spam with a numeric score: when the total score is higher than a given threshold, the feedback manager drops the comment. On the other

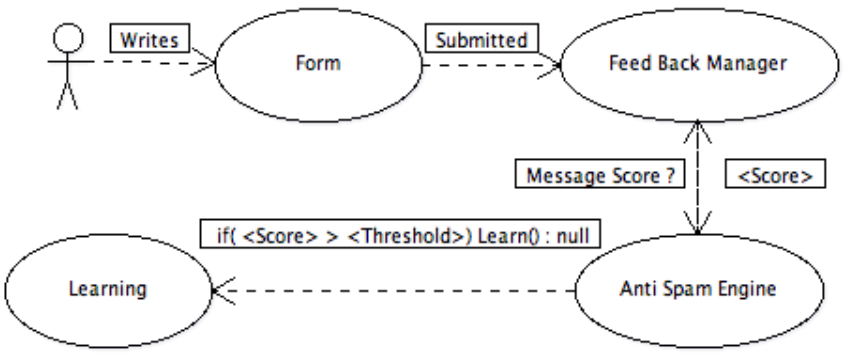

Fig. 1. Use Case Diagram.

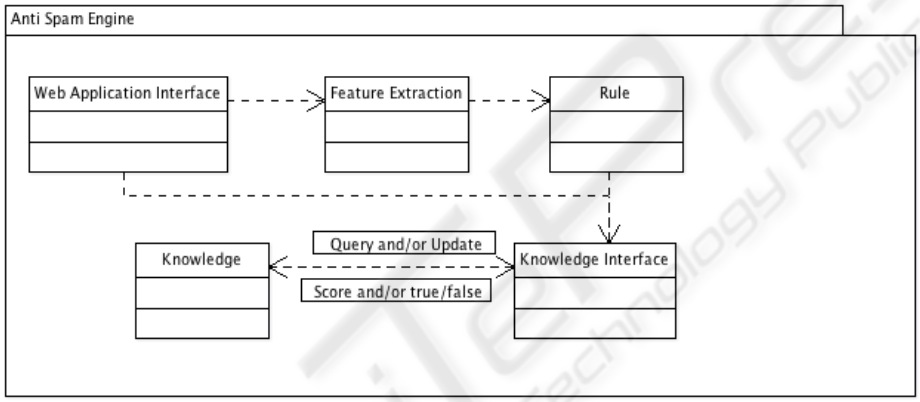

Fig. 2. Anti Spam Engine Internal Architecture.

hand if the the comment doesn't reach the threshold, the feedback manager is allowed to publish it and to store it on its own database. The threshold is configurable by the feedback manager's administrator and is stored on the feedback manager side. The main computational load is placed on the anti-spam engine that must elaborate its knowledge in order to compute a sensible score.

Comments are received by the engine through the Web Application Interface (Figure 2). Upon receiving a query, the interface starts a feature extraction phase, initially splitting the entire message into single sentences. In some cases it is not trivial understanding where a sentence ends and another begins. This leads to another category of research problems altogether, which various other groups are working on [19, 5]. Our current implementation is able to distinguish the sentences from punctuation marks, but we left an open door to more flexible implementations based on the general feature extraction model.

The score of each sentence is evaluated according to the rules specified in the knowledge base, which receives the sentences through a Knowledge Interface (KI). Presently,

the KI invokes the solution of a Prolog goal (representing the query) by means of a Java-Prolog engine based on tuProlog [12].

The Feedback Manager informs the Anti Spam Engine of rejected messages upon receiving the score and comparing it to the threshold. This notification causes the Engine to update the knowledge base with the offending sentences, through a similar tuProlog interface.

Finally, it should be noted that the knowledge base could either be shared among different Feedback Managers (which keep the possibility of differentiating their thresholds), or kept separate for each one. A shared knowledge base has the advantage of being useful for many "customers" at the cost of only one initial training session, and being more frequently used, it gathers further knowledge more quickly. On the other hand, the accuracy could be compromised if the updates and the queries regard too many different subjects. A separate knowledge base, furthermore, could be integrated with the Feedback Manager if there is no interest in taking advantage of the more flexible, two-components architecture, thus providing a single package which is easier to install and configure.

6 Implementation and Testing

The core of the proposed system is fully implemented and functional, whereas its practical usability is still limited due to the alpha stage in the development of appropriate front-ends for the integration within comment platforms. In order to foster the diffusion of the devised system, especially aiming at real-world validation, a WordPress plug-in is being completed. The alpha version can be requested to the authors, and will be publicly available shortly. It is organized in three different functional areas: (1) configuration area, which allows the user to insert the fixed threshold, the URL where the remote Anti Spam Engine (ASE) can be reached, the username and password protecting it from unauthorized access; (2) Training area, where the user may insert the word/sentence and the relative score that she wants to teach to the remote ASE; (3) the WordPress API hook, that is user-transparent bridge to the WP's comment-handling engine that grabs the comments to classify them and decide their destination.

We found that a realistic test, measuring the effectiveness of the proposed approach, is very difficult to perform. The main reasons are related both to the testing methodology and to the availability of a suitable data set. In both fields, e-mail spam has received a great deal of attention, with entire workshops dedicated to the definition of meaningful metrics, standard evaluation procedures, and the collection of reference databases [10, 11]. Some of the results are in the process of being ported to the field of short messages [9], which exhibit features more similar to comments than e-mail, and the related tools [3] could thus be exploited for anti-comment-spam solutions testing, but this adaptation proved to be far from straightforward. There is only one database known to the authors that was collected with the goal of testing this kind of systems. However, it was used with the aim of vaildating a comment spam filter targeted at a rather different kind of problem, namely link spam [21], and in the words of the authors is quite limited: "a small collection of 50 blog pages, with 1024 comments"; since the classification procedure manually tagged each comment as spam or non-spam according to the specific

meaning of link spam, it is not even directly usable for comment spam in the sense used in this paper.

Some experimental results have been collected during both manual train-and-test sessions based both on data gathered from forums that were known to have failed at rejecting comment spam in the past, and on the aforementioned blog-pages dataset; the outcome is very encouraging, yet too preliminary to publish as an irrefutable proof of strenght of the proposed system.

7 Conclusions and Future Work

In this work we described a system to fight the problem of comment spam. The proposed approach overcomes the limitations of CAPTCHA- and Javascript-based known techniques, which, according to the literature, are only partially effective and can cause accessibility problems. The architecture of the filtering system is centered on a completely server-based classification engine implemented as a dynamic filters, whose learning curve can be controlled by means of a web-based interface for maximum convenience. The first very important result, consequently, is having designed a system which exhibits excellent compatibility with any client commonly used to send comments (even on mobile platforms) and requires moderate efforts for its administration. The modular construction of the classification engine, composed of a feature extractor followed by the scoring system proper, allows experimenting different methods to represent the meaning associated with the analyzed comment. Currently, the tested feature extractor tries to isolate the different sentences composing the comment. Notwithstanding its simplicity, this method yield satisfactory preliminary results; future work will be directed towards the definition of a more effective and robust algorithm, and comprehensive experimental validation within realistic environments.

References

1. Akismet comment spam and trackback spam stopper. http://akismet.com/.
2. Inaccessibility of captcha - alternatives to visual turing tests on the web. w3c working group note. http://www.w3.org/TR/turingtest/.
3. Trec 2006 spam evaluation kit. http://plg.uwaterloo.ca/ gvcormac/jig/.
4. Wordpress - blog tool and weblog platform. http://wordpress.org/.
5. Sentence recognition through hybrid neuro-markovian modeling. In *ICDAR '01: Proceedings of the Sixth International Conference on Document Analysis and Recognition*, page 731, Washington, DC, USA, 2001. IEEE Computer Society.
6. F. Belanger and C. V. Slyke. Abuse or learning? *Commun. ACM*, 45(1):64–65, 2002.
7. J. W. Brian Chess, Yekaterina Tsipenyuk O'Neil. Javascript hijacking - fortify software white paper. http://www.fortifysoftware.com/servlet/downloads/public/JavaScript_Hijacking.pdf, 2007.
8. J. C. Brustoloni and R. Villamarín-Salomón. Improving security decisions with polymorphic and audited dialogs. In *SOUPS '07: Proceedings of the 3rd symposium on Usable privacy and security*, pages 76–85, New York, NY, USA, 2007. ACM.
9. G. V. Cormack, J. M. G. Hidalgo, and E. P. Sánz. Spam filtering for short messages. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 313–320, New York, NY, USA, 2007. ACM.

10. G. V. Cormack and T. R. Lynam. Trec 2005 spam track overview. In *In Proc. 14th Text REtrieval Conference (TREC 2005*, 2005.
11. G. V. Cormack and T. R. Lynam. Online supervised spam filter evaluation. *ACM Trans. Inf. Syst.*, 25(3):11, 2007.
12. E. Denti, A. Omicini, and A. Ricci. tuProlog: A light-weight Prolog for Internet applications and infrastructures. In I. Ramakrishnan, editor, *Practical Aspects of Declarative Languages*, volume 1990 of *LNCS*, pages 184–198. Springer, 2001. 3rd International Symposium (PADL 2001), Las Vegas, NV, USA, 11–12 Mar. 2001. Proceedings.
13. E. T. B. Dimitri do B. DeFigueiredo and S. F. Wu. Trust is in the eye of the beholder. UCDavis Technical Report CSE 2007-09.
14. D. Fetterly, M. Manasse, and M. Najork. Spam, damn spam, and statistics: using statistical analysis to locate spam web pages. In *WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 1–6, New York, NY, USA, 2004. ACM.
15. M. Hu, A. Sun, and E.-P. Lim. Comments-oriented blog summarization by sentence extraction. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 901–904, New York, NY, USA, 2007. ACM.
16. N. Jindal and B. Liu. Review spam detection. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1189–1190, New York, NY, USA, 2007. ACM.
17. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 640–651, New York, NY, USA, 2003. ACM.
18. Y.-R. Lin, H. Sundaram, Y. Chi, J. Tatemura, and B. L. Tseng. Splog detection using self-similarity analysis on blog temporal dynamics. In *AIRWeb '07: Proceedings of the 3rd international workshop on Adversarial information retrieval on the web*, pages 1–8, New York, NY, USA, 2007. ACM.
19. R. D. Lins and P. Gonçalves. Automatic language identification of written texts. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 1128–1133, New York, NY, USA, 2004. ACM.
20. D. Lopresti. *Leveraging the CAPTCHA Problem*, pages 97–110. 2005.
21. G. Mishne, D. Carmel, and R. Lempel. Blocking blog spam with language model disagreement. In *Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web - AIRWeb 2005*, pages 1–6. Lehigh University, Bethlehem, PA USA, 2005.
22. G. Mori and J. Malik. Recognizing objects in adversarial clutter: Breaking a visual captcha. In *CVPR (1)*, pages 134–144. IEEE Computer Society, 2003.
23. D. Nguyen and B. Widrow. Neural networks for self-learning control systems. *Control Systems Magazine, IEEE*, 10(3):18–23, Apr 1990.
24. A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting spam web pages through content analysis. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 83–92, New York, NY, USA, 2006. ACM.
25. D. Sculley and G. M. Wachman. Relaxed online svms for spam filtering. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 415–422, New York, NY, USA, 2007. ACM.
26. J. Yan and A. S. E. Ahmad. Breaking visual captchas with naive pattern recognition algorithms. *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 279–291, 10-14 Dec. 2007.