

## Penetration Testing for Spam Filters

Yugesh Madhavan      João W. Cangussu  
Department of Computer Science  
University of Texas at Dallas  
{yugesh,cangussu}@utdallas.edu

Ram Dantu  
Department of Computer Science  
University of North Texas  
rdantu@unt.edu

### Abstract

*Despite all the advances on techniques to block spam e-mail messages we still receive them on a frequent basis. This is due mainly to the ability of the spammers to modify the message and pass the filters. Therefore a testing technique that could resemble the behavior of spammers would improve the number of scenarios tested and allow filters to be developed based on the potential changes made by the spammers. An approach based on Natural Language Processing (NLP) for the penetration of spam filters is proposed here. Preliminary results using SpamAssassin are provided indicating the feasibility of the proposed approach.*

### 1. Introduction

It is easy to see that the Internet has been congested with spam traffic; despite the use of filters we still receive spam messages. Spam filters are reported as having accuracy as high as 99.9% being able to detect almost all spam messages. If that is the case why do we still receive spam messages daily? Spammers are aware of the filters and how they work and are able to change their e-mail messages to confuse the filters.

Testing/experiments is one of the major tasks used to determine the accuracy of spam filters. However most of the testing done is based on existing e-mail corpuses. Some of the corpus have been reported to account for a million of mixed legitimate and spam e-mail messages [15]. Part of the corpus is used to train the spam filter and the remaining is used to verify the accuracy. While the use of such approach is extremely helpful to understand and test the past behavior of spammers it does not help much on testing their adaptive behavior. That is, the corpus allows for testing how the spam e-mail messages have been changed but not how they could be changed in the future. The sole use of e-mail corpuses for testing will mostly lead to a situation where the developers of spam filters are chasing spammers; this scenario must be changed. What is needed is a

testing technique that resembles the behavior of spammers and can consequently be used to improve the adaptiveness of spam filters prior to deployment. Most of the effort has been concentrated on the collection of massive e-mail corpuses but not much effort has been done in the development of a testing technique that dynamically generates new spam messages to penetrate the filters.

Here we propose *SPoT* (Spam Penetration Testing) as one step towards a more comprehensive testing technique to the penetration test of spam filters. *SPoT* uses existing spam messages to generate a series of mutant messages. Keyword extraction based on Natural Language Processing techniques is used to extract the keywords from the existing messages and then a type of thesaurus is used to replace the keywords by alternative words that will increase the chances of the message to pass the filter. Although an e-mail corpus is still used by the *SPoT* framework such corpus does not need to be large. Indeed, the technique proposed here can be used to generate more than a million new e-mail spam messages starting with only five or ten original messages. Additionally, traditional techniques such as spoofing the header of the e-mail and obfuscating words are combined to improve the chances of penetrating the filter.

The rest of this paper is organized as follows. The *SPoT* framework is presented in Section 2 followed by preliminary results in Section 3. Existing works relevant to the approach proposed here are presented in Section 4. Conclusions along with future work are addressed in Section 5.

### 2. *SPoT*: Spam Penetration Testing

The approach proposed here has the major goal of developing an automated tool for the penetration testing of spam filters. The tool must be able to resemble the behavior of spammers and automatically change spam messages to pass the filters. Any spam message can be changed up to the point that it becomes almost a legitimate message. In this case any filter will fail to detect the message. The goal is not only to verify if the filter can be penetrated but how much effort is required to achieve that. A filter that after only one

change can no longer detect the spam message is not as effective as a filter that requires many changes for a message to be considered as not spam. The less effort required to change the message and penetrate the filter the more likely these changes can be automated. This is a huge advantage to spammers that are now using “bots” to conduct their campaigns. The effectiveness of a spammer reduces considerably as the effort required to change a spam message and pass a filter increases. The number of new spam messages reaching our e-mail boxes will reduce if the message has to be manually changed (based on human intelligence).

The overall structure of the *SPoT* framework is presented in Figure 1. The main idea is to use an existing, possibly small, e-mail corpus of spam e-mail messages. The filter under test should be able to identify all the messages in the corpus as spam or should be tuned for that. That is, without changing the messages all of them should be filtered. There are two major goals for the proposed approach. The first is to make enough changes in the spam e-mail message in such a way it passes (penetrates) the filter. The second goal is to define a measurement of how much a message needs to be changed to pass a specific filter. The less a message needs to be changed the easier it is to penetrate the filter; the less effective is the filter.

Given a spam e-mail message  $s_m$  from the corpus, *SPoT* can make a series of modifications that can be used separately or in a combined way. First, using Natural Language Processing techniques a list of keywords ( $K_1, K_2, \dots, K_j$ ) is automatically extracted from the e-mail message. These keywords can be either directly obfuscated or replaced by a synonymous word and then obfuscated. The Spam Generator will use a series of  $n \geq 1$  changes to create a new version of the spam message. The Spam Generator will also use a new header created by the “Header Spoofer” before sending the message. The components of the approach presented in Figure 1 are described in Sections 2.1 through 2.4.

## 2.1. Keyword Extractor

The “Keyword Extractor” will initially use existing techniques [13, 16] to find the most relevant words for a given set of spam e-mail messages. Later they will be analyzed to verify if customizations are needed to improve their accuracy. The preliminary results presented in Section 3 are an indication that even simple techniques will lead to reasonably good accuracy. One aspect to be concerned is the size of the message and the size of the e-mail corpus. Our goal is to avoid the use of large e-mail corpuses and use only sample spam messages to automatically generate new messages. This should not be a problem as there exists techniques to extract keywords from a small number or even a short single document [8, 4]. The term “keyword” is also used in this paper to represent a “keyphrase”.

The extraction of proper keywords is crucial as many spam filters use Bayesian techniques to compute the probability of an e-mail message  $S$  being a spam. For example, Eqn. 1 computes the spam probability of a message  $S$  given the occurrences of keywords  $W_1, W_2, \dots, W_n$ .

$$P(S|W_1W_2 \dots W_n) = \frac{P(W_1W_2 \dots W_n|S) \times P(S)}{P(W_1W_2 \dots W_n)} \quad (1)$$

## 2.2. Spam Thesaurus

Once the keywords from the spam message body have been extracted a list of words with similar meanings is selected based on an existing dictionary/thesaurus. The “Spam Thesaurus” does not contain only words that are synonymous of the keyword but mainly alternative words with similar meanings. For example, “inherited” could be replaced by actual synonymous such as “acceded” and “be granted” and also by “be awarded” depending on the context. Some context can be drawn from the target domain. For example the famous “Nigeria” spam or its variations says that someone has “inherited” a “fortune”. These two words can be easily extracted by the “Keyword Extractor”. Assume now that the target of the spammer are college students; these are easily identified by the “.edu” extension. The “Spam Thesaurus” can then replace “inherited” by “being awarded” and “fortune” by “scholarship” and most likely the new message will pass the spam filter. This change would be targeting students but replacing “fortune” by “research grant” would target the faculty body. The user profile may be extracted in many different ways. It could be based on the domain as mentioned above and it could be also based on the company type of business extracted from a webpage.

## 2.3. Obfuscator

Many times spam filters look for exact matching or small variations of specified keywords. For example, the words “save” or “savings” may be listed in one of the criteria to detect the spam. However, aware of this technique spammers are obfuscating the keywords by replacing or adding characters to the keywords in an attempt to pass the filter. The words “save” and “savings” could be replaced respectively by “\$ave” and “\$avings”. Alternatively, characters could be added turning the keywords into “sa-ve” and “savi\_ngs” for example. The goal is not to create a orthographically correct word but to change it in such a way that the reader will easily guess what would have been the correct word. This helps the message to pass the filter while not changing the meaning of the message.

The obfuscation can be applied directly to the keywords extracted from the message body or to the replacements obtained from the “Thesaurus” component.

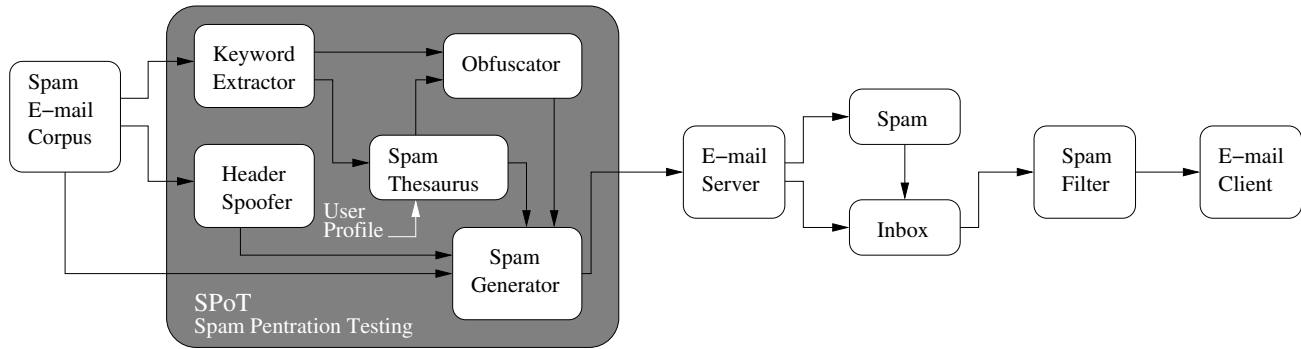


Figure 1. Proposed Approach for the *SPoT* Framework.

## 2.4. Header Spoofer

Most of spam filters allows for the specification of a “black list”; a list of e-mail or domain addresses that should be always classified as sending spam e-mail messages. Additionally filters do not always check for the entire header of the message focusing mostly on the senders. The work done by Palla et. al. is an exception [11] as it considers not only the sender but also the entire path traversed by the e-mail; their approach is restricted to header analysis without verifying the message body. As before, spammers know that after sometime the e-mail addresses used to send the first messages will be added to the “black list” and to circumvent this problem they create a faked header; that is, they spoof the header of the e-mail.

To make matters worse, spam is now being launched by ‘Bots’, which send large volume of spam to any single domain from a given IP address. Spammers are sending most of the email spam using zombie PCs (“Botnets”) which are used to launch massive spam campaigns. In general, a machine is used as a “Bot” without the user knowledge [3, 12]. This problem further complicates the matter of identifying spam messages based on header analysis as a non-spammer machine may be used to send spam messages. The “Header Spoofer” block has therefore the goal of resembling spammers behavior and modify the header to avoid detection.

## 2.5. Spam Generator

Given an original spam e-mail message  $SM^0$  the Spam Generator will make consecutive changes in the message to generate new mutant spam messages. We referred to the new messages as  $SM_j^i$  where  $i$  is the order of the mutant and  $j$  is the index of the new message. The order of the mutant indicates how many changes have been made to the original message  $SM^0$ . Here, the replacement of all occurrences of a specific keyword/keyphrase is considered as one single change. Up to this point the only mutant operator

we have used is the replacement of a keyword from another keyword from either the “Spam Thesaurus” or the “Obfuscator”. Notice that keywords from the “Obfuscator” also contain words from the thesaurus. Aside from the header analysis based on a blacklist, most spam filters are based on the computation of the probability of a spam message given the occurrence of certain words (refer to Eq. 1 for an example). Therefore, for the preliminary results presented here we have restrict the generation of new spam messages to the replacement of keywords were all keywords are treated equally. We have deferred the definition of mutant operators based on the type/class of words for future work. For example, all medicine drugs may be treated as one class, all countries from a continent as another class, etc. and the operators would be applied based on each class.

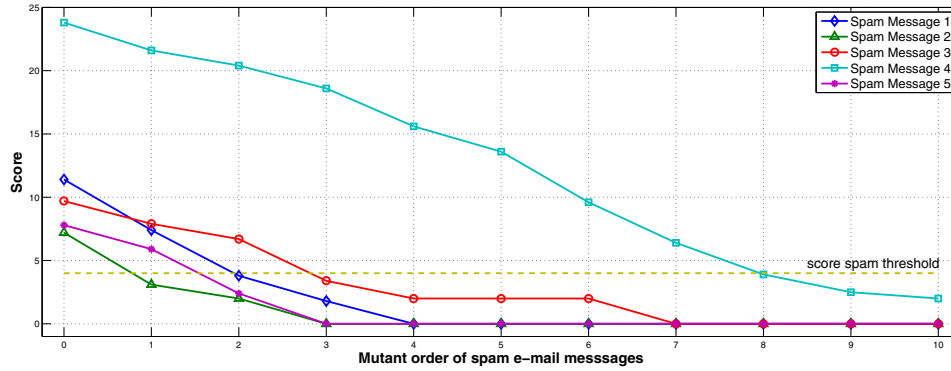
## 3. Preliminary Results

The results of applying *SPoT* to test SpamAssassin [7], an instance of a spam filter, are presented in this section. A brief description of Spam Assassin and how the experiment has been conducted is given in Section 3.1. Section 3.2 presents the preliminary results.

### 3.1. Experiment Setup

SpamAssassin is a powerful open source spam filter available in the market today. As the main goal was to test the complete effectiveness of SpamAssassin, the Use of Non-Local network tests were disabled. Another reason to do this was that our main aim was not to check whether the black lists available on the Internet was effective, as it was one of the main purposes of the Non-Local network tests. Further, as the UTDallas SMTP server is used to send the mails, and the server is listed in most of the safe lists available, the safe list tests were also disabled.

An application was developed in .NET to achieve the purpose of spoofing. As mentioned above, the goal was not



**Figure 2. Spam score reduction for five original spam messages and corresponding mutant messages. The values of the x axis represent the order of the mutant; 0 represents the original spam message, 1 represents a first order mutant, 2 a second order mutant and so on.**

to spoof the entire header. The sender’s domain and email addresses were the mainly spoofed parts. This application, in collaboration with the UTDallas SMTP server was used to send the mails. A small set of five spam messages have been selected to serve as seeds for the generation of new messages. The five messages are typical spam and should be detected by any existing filter. Mails that were selected had to do with selling of drugs including Viagra, winning of an online lottery, a bank employee persuading us to share a fortune, people asking us to help them get their money back in exchange for a commission and other instances of popular Nigerian scams. The idea is to selected a small number of easy detectable messages and show that they can be easily modified to create many other messages that will be able to penetrate the filter.

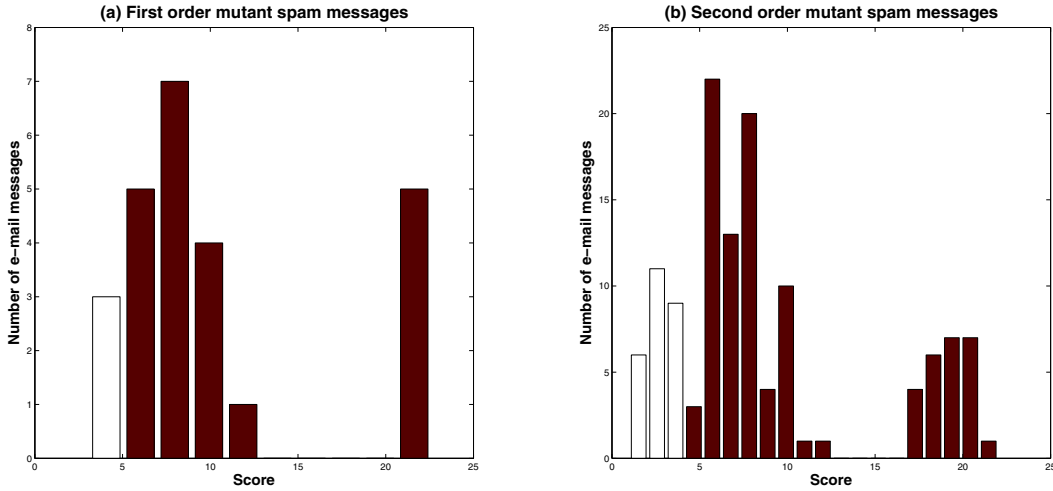
After selecting the spam e-mail messages above Tropes [1] was used to extract the keywords. Tropes is a statistical text analysis tool that extract the keywords based not only on the frequency but also on how they are used. A thesaurus of a smaller size was also created for the extracted keywords. The creation of the thesaurus has been done manually but the ultimate goal is to draw pertinent words from an existing data base or even from an overall web search. The application that was developed was then used to send spam mails.

The spam message that is generated uses the UTDallas SMTP server to reach the Internet. A separate gmail account was created for the purpose of receiving these spam emails. Using the POP feature available in gmail, these mails were downloaded and processed by SpamAssassin before being passed on as spam or ham to the email clients. These emails were then extensively studied to understand which rules had passed and which rules had given way to the changes. The results are presented next.

### 3.2. Experimental Results

As stated earlier one of the major goals of the *SPoT* framework is to detect the amount of effort required to penetrate a filter. Figure 2 provides an indication of such effort. The x axis in the figure represent the original spam message and the associated order of the mutated message. For example  $SM1_0^0$  represents the original spam message 1 and  $SM1_i^1$  for  $i = 1, \dots, 10$  represents the mutant message of order  $i$ . The y axis in the figure represents the score, as given by Spam Assassin, associated with each message. The score threshold to determine if a message is spam or not has been specified to four (dashed horizontal line in Figure 2). The results in the figure show that message SM4 is the hardest one to change in order to pass the filter. The message contains all elements of traditional spam (lottery, African countries, large sum of money, etc.) which justifies the initial higher score given by Spam Assassin. Therefore many changes were required to lower the score below the specified threshold. With respect to the other messages, it can be seen that the filter is penetrated using mutant messages of maximum third order (three changes in the message). In most cases the scores dropped to zero indicating a completely legitimate message as recognized by SpamAssassin.

The results of the classification of all the messages created using first and second order mutation are presented in the histograms in Figure 3. Automation of this process is already under way. Since there were initially five spam messages, twenty five were generated for the first order mutants and 125 for the second order mutants. The clear bars in the figure represent e-mails that have score less than four and have consequently penetrated the filter. As it can be seen in Figure 3(a), only three first order messages have penetrate the filter. For the second order messages in Figure 3(b) a



**Figure 3. Histograms for the scores of the messages generated by the *SPoT* framework. Part (a) presents the messages created using first order mutation and part (b) the messages using second order mutation. The clear bars on the histograms represent the messages that have score less than four and have consequently penetrate the filter.**

total of twenty six messages achieved score less than four. These results are quite impressive if compared to the report accuracy of more than 90% for SpamAssassin and other filters. That is, 12% of the first order mutant spam messages have been able to penetrate the filter; this number goes to 20.8% for the second order messages. For example, Yerazunis [17] reports figures where less than 2% of the messages were not properly identified as spam. Other sources have also reported accuracies over 95% [15, 6].

Although the results presented here cannot be directly extrapolated to other spam filters our conjecture is that the same trend will be observed for them. That is, the number of messages penetrating the filter will be much larger than the ones purely based on e-mail corpuses. We also believe that the accuracy of the filters will drop significantly as the order of the mutant messages increase. The higher the mutant order the lower the accuracy of spam filters. We are aware that a much larger experiment needs to be conducted to confirm our conjectures but the results presented here are a preliminary indication of them. We also plan to add many more spam filters, including commercially available ones, as subject of future experiments.

#### 4. Related Work

A lot of work have been done with respect to penetration or security testing [2, 14, 9, 10, 5] but penetrating spam filters have not been the major focus. Next a brief description of works that are more related to *SPoT* are presented and compared to *SPoT*.

Yerazunis [17] provides some insights on the analysis of spam filters. A corpus of 4417 e-mail messages was used and shuffled 10 times to produce a larger testbed and also a different sequence for the training period. The last 500 messages from each shuffle were used to verify the accuracy of the filters. The shuffled did have an impact on the training and consequently on the performance of the filters but penetration was not the goal on his work. The spam messages were not altered, just “sent” into a different order. In our case the order does not matter as the messages are altered to resemble the actual behavior of a spammer.

The work done by Webb et al. [15] focuses on camouflage the spam message. That is, the messages are augmented by a large amount of legitimate text in order to confuse the filters. Filters are based not only on spam text but also on legitimate text. The more “legitimate” text in the message the lower the chance of classifying it as spam. They have done an analysis of the performance of the filters based on the fact that a false positive is more costly to the user than a false negative (missing a legitimate message could be more harmful than having to read and delete a spam message). Their results show that after training the filters with enough camouflage messages the performance increases substantially and most spam messages are detected. The approach proposed here differs substantially from the work of Webb et al [15] as it aims to keep changing the spam messages in order to pass the filter. Their work relies on a large e-mail corpora while we automatically generate the new spam messages. The initial results of using *SPoT* show that the performance of a spam filter considerably de-

creases with consecutive changes in the message body.

The work presented by Graham-Cumming [6] does an analysis for different reasons spam is not detected by the filters. It shows that “word salad” (adding random words to the spam message to confuse the filter) is ineffective as most messages are still detected by the filters (only 0.04% got through the filter) and the ones that passed the filter needed hundreds of words to be added to the original spam message. The work also presented the use of an “evil” filter to generate new spam messages. Spam messages that have passed the “good” filter are used to train the “evil” filter and send the modified message with 5 additional random words. Using this approach a set of “kryptonite” words were identified; these words would turn a spam message into a ham message. The work proposed here is similar but presents a major distinction. While Graham-Cumming’s work relied on feedback (which he recommends should never be given to a spammer) to extract the “kryptonite” words from the messages that got through the good filter *SPoT* is an open loop with no feedback from the filter. Also, the goal is not to add words to the spam messages but replace them by meaningful interesting words based on a potential user profile extracted from the web.

## 5. Conclusions and Future Work

As most security issues, spam detection is not a easy problem to solve. For every new technology developed to detect spam there is a response by spammers with new ingenious spam messages to penetrate the filters. One can say that most of the time developers are always behind in this race. One of the reasons for that being the way spam filters are tested. *SPoT*, a new approach based on the extraction of keywords from spam messages (the same technique used to create filters) for the generation of new messages have been proposed here. *SPoT* allows the computation of the effort required to penetrate a filter. The higher the effort the more effective the filter. The proposed approach does not need a large e-mail corpus and millions of new spam messages can be generated from a very small corpus. The preliminary results using *SPoT* to penetrate the SpamAssassin filter are an indication that the proposed approach does improve the testing effectiveness. The results show that not much effort (represented by a lower order mutant spam message) is required to penetrate the filter.

Although encouraging, the results presented here are based on a small sample of original spam messages; also the number of new generated messages was kept low since parts of the process were still done manually. A larger experiment will be conducted after the entire automation of the process. The experiment is going to be augmented not only with respect to the number of messages but it will also include the testing of other existing spam filter solutions. Al-

ternative ways to generate new spam messages is also going to be investigated. Finally, more advanced techniques will be analyzed for the extraction of keywords.

## References

- [1] Acetic Semantic-Knowledge. *Tropes, Version 7.0- Reference Manual*, 2006. www.semantic-knowledge.com.
- [2] B. Arkin, S. Stender, and G. McGraw. Software penetration testing. *IEEE Security and Privacy*, 3(1):84–87, 2005.
- [3] D. Dagon, C. Zou, and W. Lee. Modeling botnet propagation using time zones. In *In Proceedings of the 13th Network and Distributed System Security Symposium NDSS*, February 2006.
- [4] W.-W. Deng and H. Peng. Research on a naive bayesian based short message filtering system. In *Proceedings of the Fifth International Conference on Machine Learning and Cybernetics*, Dalian, August 2006. IEEE.
- [5] D. Geer and J. Harthorne. Penetration testing: a duet. *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 185–195, 2002.
- [6] J. Graham-Cummin. How to beat a bayesian spam filter. In *MIT Spam Conference*, 2004.
- [7] J. Mason. Filtering spam with spamassassin. In *proceedings of HEANet Annual Conference*, 2002.
- [8] Y. Matsuo and M. Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(1):157–169, 2004.
- [9] G. McGraw. Software security. *Security & Privacy, IEEE*, 2(2):80–83, Mar-Apr 2004.
- [10] G. McGraw and B. Potter. Software security testing. *IEEE Security and Privacy*, 2(5):81–85, 2004.
- [11] S. Palla, R. Dantu, and J. W. Cangussu. Spam classification based on email path analysis. *International Journal of Information Security and Privacy (IJISP)*, 2(2):46–69, April-June 2008.
- [12] A. Ramachandran, N. Feamster, and D. Dagon. Revealing botnet membership using dnsbl counter-intelligence. In *USENIX, SRUTI*, 2006.
- [13] W. N. Sado, D. Fontaine, and P. Fontaine. A linguistic and statistical approach for extracting knowledge from documents. In *Proceedings of the 15th International Workshop on Database and Expert Systems Applications (DEXA'04)*. IEEE Computer Society, 2004.
- [14] H. H. Thompson. Application penetration testing. *IEEE Security and Privacy*, 3(1):66–69, 2005.
- [15] S. Webb, S. Chitti, and C. Pu. An experimental evaluation of spam filter performance and robustness against attack. *International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 0:8 pp., 2005.
- [16] I. H. Witten, G. W. Paynter, E. Frank, C. Gutwin, and C. G. Nevill-Manning. Kea: practical automatic keyphrase extraction. In *DL '99: Proceedings of the fourth ACM conference on Digital libraries*, pages 254–255, New York, NY, USA, 1999. ACM.
- [17] W. Yeraunus. The spam-filtering accuracy plateau at 99.9% accuracy and how to get past it. In *MIT Spam Conference*, Cambridge, MA, 2004.