An Open Digest-based Technique for Spam Detection*

E. Damiani[1], **S. De Capitani di Vimercati**[1], **S. Paraboschi**[2], **P. Samarati**[1]
(1) DTI - Università di Milano - 26013 Crema, Italy
(2) DIGI - Università di Bergamo - 24044 Dalmine, Italy
{damiani, decapita, samarati}@dti.unimi.it, parabosc@unibg.it

Abstract

A promising anti-spam technique consists in collecting users opinions that given email messages are spam and using this collective judgment to block message propagation to other users. To be effective, this strategy requires a way to identify similarity among email messages, even if the program used by the spammer to generate the messages may try to obfuscate their common origin.
In this paper, we investigate the issues arising in the design of a digest-based spam detection mechanism, which has to satisfy many conflicting requirements: protect message confidentiality, be public, and prove difficult or expensive to fool by obfuscation techniques that automatically introduce differences into the same base spam message. We show that an open digest function is able to satisfy the above requirements and contribute to the fight against spam.

1 Introduction

The problem of *spam* or Unsolicited Bulk Email (UBE) is becoming a pressing issue [1, 3]. In spite of the development of many anti-spam techniques, the war against spam is far from being successful. This is partly due to several characteristics of spam that make it a difficult problem:

- *spam heterogeneity:* the CAUCE (Coalition Against Unsolicited Commercial Email) organization lists the following examples of spam: chain letters; pyramid schemes (including multilevel marketing), make-money-fast schemes, phone sex lines and ads for pornographic web sites, and so on;
- *spam definition:* one of the problems of designing a system against spam is defining what a spam message is. There are many borderline cases where what is spam for a user could be useful information for another (e.g., an opening for a job position or a mail from a vendor notifying thousands of customers of a defect in its products)
- *spam evolution:* spam evolves as rapidly as anti-spam techniques improve.

While no drastic solution to the spam problem is currently available, and none is likely to appear soon[1], several moderately successful anti-spam techniques have been proposed, each operating along a different line. Here, we briefly describe three main families of techniques. The first technique fights against the mail servers that are responsible for the generation of spam messages. This technique has a good potential and is typically realized using *blacklists* of misbehaving servers that are collected by several sites. The main drawback of black lists is that they are prone to misconfigurations that have often led to denial of service, with legitimate servers unable to exit from a list where they had been incorrectly inserted. *White lists* are complementary to black lists, and contain addresses of servers which are trusted not to propagate spam. A major drawback of white lists and, generally speaking, of restrictions to SMTP traffic is that they contradict the main original design goal of SMTP, namely the open paradigm of the Internet's global email system. A second family of solutions considers the content of messages and exploits the fact that spam typically falls within predefined categories and it is possible to distinguish spam based on its content. Depending on how email messages are filtered, these methods can be classified as *rule-based filters* and *Bayesian word distribution filters*. Rule-based filters, like SpamAssassin, assign a spam "score" to each message based on whether the message contains features typical of spam,

*This work was supported in part by the EU within the PRIME Project under contract IST-2002-507591 and by the Italian MIUR within the KIWI and MAPS projects.

[1]See `http://www.rhyolite.com/anti-spam/you-might-be.html` for a humorous view of self-proclaimed anti-spam "silver bullets".

such as keywords, URLs or, in the case of HTML messages, background colors.

Bayesian tools, like SpamProbe (http://spamprobe.sourceforge.net), assign a frequency-based probability to tokenized words (or pairs/triples) as spam indicators based on previous experiences. The Bayesian approach has been rather successful, but its success strictly depends on the fact that spam typically aims at a limited variety of messages. A creative use of spam would not find an obstacle on spam filters trained on the basis of previous experience. A third approach is focused on the fact that the same information is sent to many users, though spammers try to disguise it by creating a specific version of the message for each user. *Collaborative spam filters* collect information about what is spam throughout the network [2, 4]. In such a context, each mail server queries the community (or a central repository working as a collector) about every suspicious message. An important requirement in collaborative spam filtering is not disclosing neither spam messages' content. A widespread solution is *digest-based hashing*, where content of the electronic mail messages is hashed in order to identify content previously found in known spam. Recently, concerns have been raised about patents covering digest-based antispam systems.[2]

Our work is aimed at demonstrating that the requirement of avoiding message disclosure can be better satisfied by using an *open digest function* designed with special consideration of countering attacks that spammers can enact. We consider digests at the granularity of whole messages, but the approach could also be adapted to finer levels of granularities.

The contributions of this paper can be summarized as follows. First, we illustrate the different requirements that a digest for spam detection should satisfy (Section 2). Second, we present an open digest technique, that we have adapted to take into account disguising attacks and illustrate its resiliency with some experiments (Sections 3 and 4). Third, we propose the use of a multi-hash approach to maximize the cost of attacks to the filter on the part of the spammers (Section 5).

2 Requirements

As noted before, if a generic hash function like MD5 (or SHA-1) is used to produce the digest of a message, the spammer can easily fool this protection measure by inserting into each message, in an arbitrary position, a few random characters (called *hash busters*) that will immediately make the message unique, with practically no impact on the user perception of the message. This observation originates a first functional requirement:

- **Requirement 1**: *the digest identifying each message should not vary significantly for changes that can be produced automatically.*

What is needed is a localized hashing function such as those applied in information retrieval systems. However, the techniques designed for information retrieval have to be carefully adapted since they were not designed to tolerate malicious behavior which must instead be considered in our environment. For instance, *MIME encoding* is often used by spammers to disguise message content while it is in transit, thereby allowing it to sneak past content-based spam filtering.[3] Less often, spammers will use HTML character entity codes (in the form "&#nnn;", where "nnn" is the decimal code of the character being escaped) to disguise selected characters in an HTML message body. This originates our second requirement:

- **Requirement 2**: *the encoding must be robust against intentional attacks.*

A solution to the spam problem must assume that spammers are technically competent and after an analysis of the characteristics of the filtering solutions they may spend resources to design tools that are able to automatically produce spam that bypasses the checks put in place.

Finally an encoding suitable for use in the framework of an anti-spam filter should not put the user at risk of accidental deletion of important messages. Therefore, we have a third requirement:

- **Requirement 3**: *the encoding should support an extremely low risk of false positives.*

Note that the risk of classifying a legitimate email as spam (false positive), is far more costly than the risk of missing one spam (*false negative*).

The following section presents an open-source digest technique aimed at satisfying the three requirements above.

[2]For instance, Network Associates Inc. (NAI) has been granted a broad U.S. patent covering digest-based anti-spam techniques.

[3]A related technique, *URI encoding*, is often used by spammers in HTML message bodies to disguise selected characters in website URLs (indeed, any type of URL); it works in the same way as MIME.

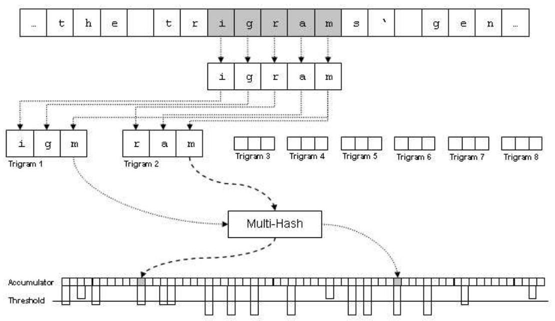

Figure 1: Nilsimsa working

3 A spam-identifier digest

Our proposal is a variation of the open source Nilsimsa digest technique (http://lexx.shinn.net/cmeclax/nilsimsa.html), adapted to provide undisguisability of spam messages. For simplicity, while presenting our solution we will continue to use the term Nilsimsa, pointing out the differences from the original version when needed. Nilsimsa satisfies our first requirement inasmuch it is a simple *local sensitive hash function*[4] that takes a document or a text string as input and provides as output a 32-byte code representing the distribution of the *trigrams* in the text.

Nilsimsa operates by using a window of 5 characters that slides along the text of the message one character at a time (see Figure 1). When a new character enters the window, the algorithm generates the trigrams associated with the window and passes each of them to a hash function $h()$. The hash function $h()$ computes a value $i = h(trigram)$ between 0 and 255 that corresponds to the i-th counter in an array of integers of size 255, called *accumulator*, and whose value is increased by 1. After the text analysis, the *accumulator* will present in the i-th cell the number of trigrams that have been found in the text producing i by the application of the hash function. The relative frequency of each bucket is compared with the average bucket frequency observed for a large collection of messages (typically, all the messages received by the user) and the value representing this ratio is associated with the bucket. Then, the ratio of each bucket is considered and if the i-th ratio is greater than the median, the i-th bit of the Nilsimsa code is set to 1; it is set to 0, otherwise. In this way a 32-byte code is produced.

[4]A local sensitive hash function is a function producing similar digests for similar documents.

The original version of Nilsimsa used a simple method to compute the bits in the digest, comparing the cardinality of the trigrams of each bucket with the average for all the buckets. This technique however, is not robust enough against aimed addition (Section 4.4); therefore, to increase robustness, we introduced the usage of the median as the term of comparison.

To determine if two messages present the same textual content, their Nilsimsa digests are compared, checking the number of bits in the same position that have the same value. The *Nilsimsa Compare Value* is the number of bits that are equal in two digests minus 128. For two randomly chosen 256-bit sequences, we expect an average of 128 equal bits, that is, a Nilsimsa Compare Value equal to zero. The maximum value of the Nilsimsa Compare Value is 128, for two identical digests. The original designer of Nilsimsa proposes to use a threshold of 24 for the Nilsimsa Compare Value, above which the two digests are considered as referring to the same message. The original threshold, equal to 24, does not satisfy our third requirement (low probability of false positives), since it corresponds to a probability of conflict between two random messages equal to $p = 1.35 \cdot 10^{-4}$. With the hypothesis that the system knows 10^6 distinct digests of spam messages, if we want a probability at most equal to 10^{-5} that a non-spam message produces a digest which corresponds to that of a spam, we propose to use a Nilsimsa Compare Value equal to 54, which, under the assumption of a uniform distribution of Nilsimsa codes, corresponds to a probability of conflict between the digests of two random messages equal to $p = 7.39 \cdot 10^{-12}$, that fully satisfies our third requirement. As a partial confirmation of the above analysis, we applied the Nilsimsa digest to a large collection of both legitimate messages and spam messages, 2500 messages of each kind[5] and with a threshold equal to 54 we observed no false positives.

4 Digest evaluation

We now evaluate the robustness of our technique against disguising attacks that spammers can enact to fool anti-spam filters based on the digest. For each attack we provide some considerations on why our technique is resilient to it and report the results of our experiments which confirm them. We have identified the following four kinds of attacks:

[5]We have taken the legitimate messages from comp.risks (`www.usenet.org`) and the spam ones from SpamArchive (`www.spamarchive.org`).

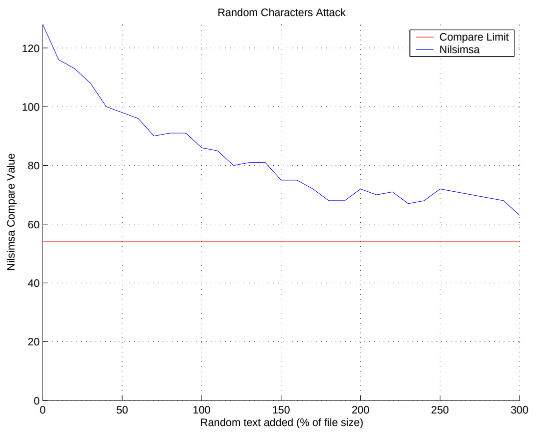

Figure 2: Random addition

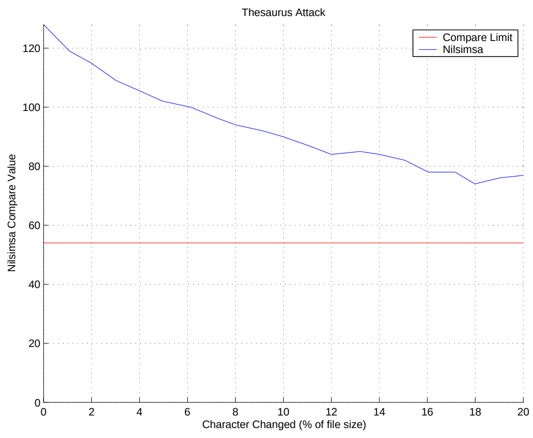

Figure 3: Thesaurus substitutions

- *Random addition:* random characters are added at the end of the original text;
- *Thesaurus substitution:* words of the original text are substituted by synonyms defined in a thesaurus;
- *Perceptive substitution:* characters of the original text are substituted without changing the text perception;
- *Aimed addition:* sequences of characters are added at the end of the original text to modify the trigram distribution of some specific buckets.

4.1 Random addition

This is the simplest and common attack that spammers use to overcome anti-spam filters. It is easy to see that the attack effectiveness against our digest will be limited, since the random text added generates a uniform trigram distribution that equally increases each bucket in the same way. The Nilsimsa Compared Value will then decrease slowly.

In our simulations, we considered the addition at the end, since it is certainly a position where it generates the least impact on user perception and, for how Nilsimsa computes the digest, the position within the text is essentially irrelevant. We added a random sequence of characters whose length ranged from 0 to 250% of the original text length. The latter figure is much higher than the percentage of noise normally added by spam generators. Figure 2 shows Nilsimsa's high robustness against this attack.

4.2 Thesaurus substitution

With this attack, words are replaced with synonyms defined in a Thesaurus. As an example, the word "`disappear`" could be substituted by "`vanish`". The thesaurus substitution will change the trigram distribution by: *i)* erasing some trigrams and also *ii)* adding new trigrams due to synonym substitution. The defense against this attack relies on the fact that synonymy is never perfect and therefore only a small percentage of text can be changed without modifying the semantics of the message.

In our simulation, the percentage of replaced text ranges from 0 to 20%. The reason for limiting the replacement to 20% of text is that it is not simple to identify a rich set of synonyms and therefore changing more than 20% of the text would eventually alter the meaning of a sentence. As Figure 3 shows, the Thesaurus attack is more effective than the Random one, but Nilsimsa is however able to resist this attack.

4.3 Perceptive substitution

This attack is realized through the substitution of characters with other ones preserving text perception on the part of the human reader. As an example, the word "`security`" could become "`s3cur1ty`".

We expect that this attack will be more effective than the previous one because a sparse substitution changes more trigrams than a word substitution. An analysis of the Nilsimsa code shows that a single char substitution will change exactly 24 trigrams. So, 120 trigrams are altered by a substitution of 5 characters, whereas a substitution of a 5-character word will change only 66 trigrams.

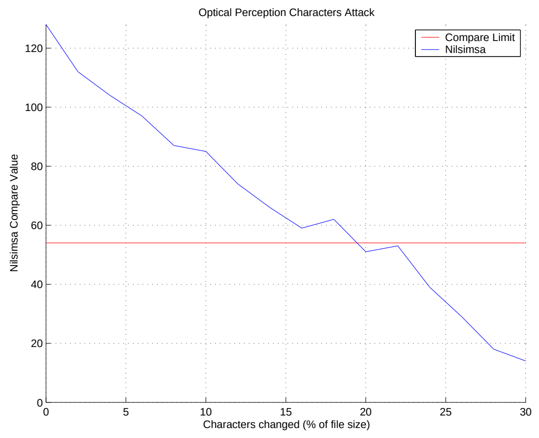

Figure 4: Perceptive substitution

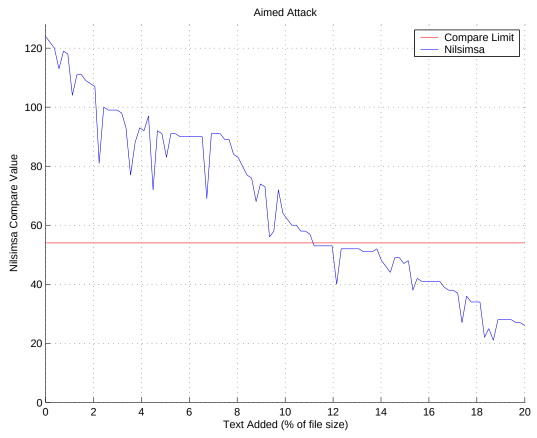

Figure 5: Aimed attack's to median defense

As the experiment in Figure 4 shows, the digest is robust against this attack until the percentage reaches a value around 20%. This is completely safisfactory, since HCI research shows that the impact on text readability of this strategy is considerable even with a low percentage of character substitutions; an excessive use of it makes messages virtually unreadable and therefore ineffective from the spammer's point of view. We can therefore conclude that our digest is robust as long as the readability of the message must be preserved.

4.4 Aimed addition

In the previous subsections, we evaluated the strength of attacks that are already used by spammers. All these attacks are relatively easy to implement and may be effective against several anti-spam solutions. An aimed attack is instead an attack directed specifically against the digest function. This attack requires a higher computational cost and a greater technical sophistication, but the reward may be a compact modification to each email message able to make the digest of an email message distinct from that of another perceptually-identical message. The main idea is to increase the cardinality of a chosen bucket set. This goal is attained with the generation of strings of characters that will be splitted into trigrams increasing the buckets under attack.

The aimed addition attack operates in two steps. The first step requires to find the buckets associated in the message with a low trigram cardinality; they will become the target buckets. The second step builds text strings to add at the end of the message, one character at a time, choosing at each iteration the character that produces the greatest number of trigrams that

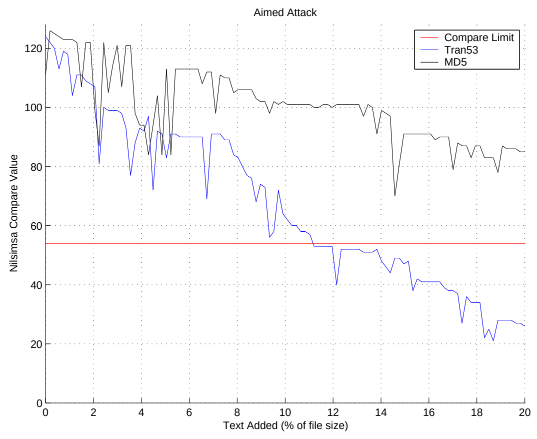

Figure 6: Aimed attack's to multihash-median defense

are mapped by the hash function into the target buckets that have yet to reach a value above the threshold. This way the cardinality of each of the target buckets increases and the bit in the digest changes from 0 to 1. Figure 5 shows a graphic representation of the effect of an aimed attack. We can observe that even a modest addition of a 10% prologue can make the digest distinct from that of the original message. Therefore an aimed attack is 20 times more efficient than a random addition. To contrast this attack, we propose the use of *multiple hash functions*.

5 Multihashing

An increased protection against the addition of a sequence of characters aimed at corrupting the digest can be obtained by using, instead of a single digest, of a family of digests, each characterized by a distinct hash function mapping trigrams into buckets. The hash functions that we have used for our experiments are the standard Nilsimsa hash function, called *Tran53*, and the well-known *MD5* hash function. Figure 6 shows the impact that an aimed trailer has on the aimed Nilsimsa function and the impact on another digest of the same message using MD5. As we can see, the attack aimed at a particular hash is not effective against the other, even if the size of the added text is more than doubled with respect to the minimum required by the aimed attack.

We can identify a continuum of solutions exploiting this strategy. The two extremes are represented respectively: *i)* by the solution where a single hash function is used, and *ii)* by a system where the hash function receives as parameter a nonce that makes the mapping between trigrams and buckets unpredictable. Both these extreme solutions have some drawbacks: the first one is weak against aimed addition attacks, while the second reduces system efficiency, as the evaluation of the digest can only occur when the nonce is specified, creating a serious obstacle to the efficient storage of digests. To produce a solution that is at the same time robust against attacks and efficient, we propose to use a definite number of distinct hash functions known in advance by all the participants in the cooperative spam detection network.

We assume that each mail server has a catalog, called *Spam_catalog*, which is a matrix with n columns (one for each hash function) and *num_spam* rows (one for each spam message). Entry $\mathit{Spam\_catalog}[i][j]$ stores the digest of the i-th spam message computed with the hash function $h_j()$. Upon reception of a message m, a user u_j can report the fact that m is spam to its own mailer ms_k. The mail server, at email checking time, generates two random numbers r_1 and r_2 that are used to index the required hash function. This way we reduce the spammer's likelihood to predict the hash that will be used in a given exchange. The mail server then computes the digests $d_1 = h_{r_1}(m)$ and $d_2 = h_{r_2}(m)$ and compares them with the corresponding digests stored in *Spam_catalog*. If the comparison produces a value greater than a specified threshold *Threshold*, the mail server considers the message m as spam, and tags it as spam. Note that to increase the mechanism's security more than two random hash comparisons could be used. Of course there is a trade-off between security and computational cost.

Each time mail server ms_k receives from a user a spam report about a message m, it updates the catalog *Spam_Catalog* by adding a row and storing the corresponding digests computed by applying the hash functions $h_1(m) \ldots h_n(m)$. The cost of the antispam system increases, in terms of both storage and network bandwidth, as it has to manage many digests. On the other hand, if the attacker now wants to automatically create instances of the same message that appear as distinct in their digest, she will have to attack all the different hash functions and our experiments indicate that this imposes a significant increase in the size of the message.

6 Conclusions

The use of digests for identifying spam messages in a privacy-preserving way is a fundamental technique for collaborative filtering. It is worth noticing that while for the sake of simplicity we considered digests at the level of whole messages, our approach can be readily adapted to finer levels of granularity such as paragraphs, short sentences or even words. This is particularly promising toward the use of open source hashing in the framework of hybrid approaches integrating digest-based and Bayesian techniques.

Acknowledgments

We thank Andrea Tironi and Luca Zaniboni for their contribution to the development of the experiments demonstrating the validity of the digest technique.

References

[1] L.F. Cranor and B.A. LaMacchia. Spam! *Communications of the ACM*, 41(8):74–83, August 1998.

[2] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. P2P-based collaborative spam detection and filtering. In *Proc. 4th IEEE Conf. on P2P*, Zurich, Switzerland, August 2004.

[3] J. Graham-Cumming. The spammer's compendium. In *Proc. of the 2003 Spam Conference*, Cambridge, January 2003.

[4] F. Zhou, L. Zhuang, B.Y. Zhao, L. Huang, A.D. Joseph, and J. Kubiatowicz. Approximate object-location and spam filtering on P2P systems. In *Proc. USENIX Middleware Conf.*, June 2003.