2 Misleading Learners: Co-opting Your Spam Filter

Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, Kai Xia[1]

Abstract Using statistical machine learning for making security decisions introduces new vulnerabilities in large scale systems. We show how an adversary can exploit statistical machine learning, as used in the SpamBayes spam filter, to render it useless—even if the adversary's access is limited to only 1% of the spam training messages. We demonstrate three new attacks that successfully make the filter unusable, prevent victims from receiving specific email messages, and cause spam emails to arrive in the victim's inbox.

2.1 Introduction

Applications use statistical machine learning to perform a growing number of critical tasks in virtually all areas of computing. The key strength of machine learning is adaptability; however, this can become a weakness when an adversary manipulates the learner's environment. With the continual growth of malicious activity and electronic crime, the increasingly broad adoption of learning makes assessing the vulnerability of learning systems to attack an essential problem.

The question of robust decision making in systems that rely on machine learning is of interest in its own right. But for security practitioners, it is especially important, as a wide swath of security-sensitive applications build on machine learning technology, including intrusion detection systems, virus and worm detection systems, and spam filters [13, 14, 18, 20, 24].

Past machine learning research has often proceeded under the assumption that learning systems are provided with training data drawn from a natural distribution of inputs. However, in many real applications an attacker might have the ability to provide a machine learning system with maliciously chosen inputs that cause the system to infer poor classification rules. In the spam domain, for example, the adversary can send carefully crafted spam messages

[1] Comp. Sci. Div., Soda Hall #1776, University of California, Berkeley, 94720-1776, USA

that a human user will correctly identify and mark as spam, but which can influence the underlying machine learning system and adversely affect its ability to correctly classify future messages.

We demonstrate how attackers can exploit machine learning to subvert the SpamBayes statistical spam filter. Our attack strategies exhibit two key differences from previous work: traditional attacks modify attack instances to evade a filter, whereas our attacks interfere with the training process of the learning algorithm and *modify the filter itself*; and rather than focusing only on placing spam emails in the victim's inbox, we also present attacks that *remove legitimate emails* from the inbox.

We consider attackers with one of two goals: expose the victim to an advertisement or prevent the victim from seeing a legitimate message. Potential revenue gain for a spammer drives the first goal, while the second goal is motivated, for example, by an organization competing for a contract that wants to prevent competing bids from reaching their intended recipient.

An attacker may have detailed knowledge of a specific email the victim is likely to receive in the future, or the attacker may know particular words or general information about the victim's word distribution. In many cases, the attacker may know nothing beyond which language the emails are likely to use.

When an attacker wants the victim to see spam emails, a broad *dictionary attack* can render the spam filter unusable, causing the victim to disable the filter (Section 2.3.1.1). With more information about the email distribution, the attacker can select a smaller dictionary of high-value features that are still effective. When an attacker wants to prevent a victim from seeing particular emails and has some information about those emails, the attacker can target them with a *focused attack* (Section 2.3.1.2). Furthermore, if an attacker can send email messages that the user will train as non-spam, a *pseudospam attack* can cause the filter to accept spam messages into the user's inbox (Section 2.3.2).

We demonstrate the potency of these attacks and present a potential defense—the *Reject On Negative Impact (RONI) defense* tests the impact of each email on training and doesn't train on messages that have a large negative impact. We show that this defense is effective in preventing some attacks from succeeding.

Our attacks target the learning algorithm used by several spam filters, including SpamBayes (spambayes.sourceforge.net), a similar spam filter called BogoFilter (bogofilter.sourceforge.net), the spam filter in Mozilla's Thunderbird (mozilla.org), and the machine learning component of SpamAssassin (spamassassin.apache.org)—the primary difference between the learning elements of these three filters is in their tokenization methods. We target SpamBayes because it uses a pure machine learning method, it is familiar to the academic community [17], and it is popular with over 700,000 downloads. Although we specifically attack SpamBayes, the widespread use of its statistical learning algorithm suggests that other filters are also vulnerable to similar attacks[2].

Our experimental results confirm that this class of attacks presents a serious concern for statistical spam filters. A dictionary attack makes the spam filter unusable when controlling just 1% of the messages in the training set, and a well-informed focused attack removes the target email from the victim's inbox over 90% of the time. Our pseudospam attack causes the victim to see almost 90% of the target spam messages with control of less than 10% of the training data.

We explore the effect of the *contamination assumption*: the adversary can control some of the user's training data. Novel contributions of our research include:

- A detailed presentation of specific, successful attacks against SpamBayes.
- A discussion of how these attacks fit into a more general framework of attacks against machine learning systems.
- Experimental results showing that our attacks are effective in a realistic setting.
- A potential defense that succeeds empirically against the dictionary attack.

Below, we discuss the background of the training model (Section 2.2); we present three new attacks on SpamBayes (Section 2.3); we give experimental results (Section 2.4); and we propose a defense against these attacks together with further experimental results (Section 2.5).

A preliminary report on this work appeared in the First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET) [19].

2.2 Background

SpamBayes counts occurrences of tokens in spam and non-spam emails and learns which tokens are more indicative of each class. To predict whether a new email is spam or not, SpamBayes uses a statistical test to determine whether the email's tokens are sufficiently indicative of one class or the other, and returns its decision or *unsure*. In this section, we detail the statistical method SpamBayes uses to learn token scores and combine them in predic-

[2] We note that while some filters, such as SpamAssassin, use the learner only as one of several components of a broader filtering strategy, our attacks would still degrade the performance of SpamAssassin. Since other components of SpamAssassin are fixed rules, the only element that is trained is the learner. For SpamAssassin, our attacks will degrade the performance of this element in their system and thereby diminish its overall accuracy.

tion, but first we discuss realistic models for deploying SpamBayes and our assumption of adversarial control.

2.2.1 Training Model

SpamBayes produces a *classifier* from a *training set* of labeled examples of spam and non-spam messages. This classifier (or *filter*) is subsequently used to label future email messages as *spam* (bad, unsolicited email) or *ham* (good, legitimate email). SpamBayes also has a third label—when it isn't confident one way or the other, it returns *unsure*. We adopt this terminology: the true class of an email can be ham or spam, and a classifier produces the labels *ham*, *spam*, and *unsure*.

There are three natural choices for how to treat *unsure*-labeled messages: they can be placed in the spam folder, they can be left in the user's inbox, or they can be put into a third folder for separate review. Each choice can be problematic because the *unsure* label is likely to appear on both ham and spam messages. If *unsure* messages are placed in the spam folder, the user must sift through all spam periodically or risk missing legitimate messages. If they remain in the inbox, the user will encounter an increased amount of spam messages in their inbox. If they have their own "Unsure" folder, the user still must sift through an increased number of *unsure*-labeled spam messages to locate *unsure*-labeled ham messages. Too much *unsure* email is therefore almost as troublesome as too many false positives (ham labeled as *spam*) or false negatives (spam labeled as *ham*). In the extreme case, if every email is labeled *unsure* then the user must sift through every spam email to find the ham emails and thus obtains no advantage from using the filter.

Consider an organization that uses SpamBayes to filter incoming email for multiple users and periodically retrains on all received email, or an individual who uses SpamBayes as a personal email filter and regularly retrains it with the latest spam and ham. These scenarios serve as our canonical usage examples. We use the terms *user* and *victim* interchangeably for either the organization or individual who is the target of the attack; the meaning will be clear from context.

We assume that the user retrains SpamBayes periodically (*e.g.*, weekly); updating the filter in this way is necessary to keep up with changing trends in the statistical characteristics of both legitimate and spam email. Our attacks are not limited to any particular retraining process; they only require an assumption that we call the *contamination assumption*.

2.2.2 The Contamination Assumption

We assume that the attacker can send emails that the victim will use for training—the *contamination assumption*—but incorporate two significant restrictions: 1) attackers may specify arbitrary email bodies but cannot alter email headers; and 2) attack emails will always be trained as spam, not ham. In our pseudospam attack, however, we investigate the consequences of lifting the second restriction and allowing the attacker to have messages trained as ham.

It is common practice in security research to assume the attacker has as much power as possible, since a determined adversary may find unanticipated methods of attack—if a vulnerability exists, we assume it may be exploited. It is clear that in some cases the attacker can control training data. Here, we discuss realistic scenarios where the contamination assumption is justified; in the later sections, we examine its implications.

Adaptive spam filters must be retrained periodically to cope with the changing nature of both ham and spam. Many users simply train on all email received, using all *spam*-labeled messages as spam training data and all *ham*-labeled messages as ham training data. Generally the user will manually provide true labels for messages labeled *unsure* by the filter, as well as for messages filtered incorrectly as *ham* (false negatives) or *spam* (false positives). In this case, it is trivial for the attacker to control training data: any emails sent to the user are used in training.

The fact that users may manually label emails does not protect against our attacks: the attack messages are unsolicited emails from unknown sources and may contain normal spam marketing content. The *spam* labels manually given to attack emails are correct and yet allow the attack to proceed. When the attack emails can be trained as ham, a different attack is possible; our pseudospam attack explores the case where attack emails are trained as ham (see Section 2.3.2).

2.2.3 SpamBayes Learning Method

SpamBayes is a content-based spam filter that classifies messages based on the tokens (including header tokens) observed in an email. The spam classification model used by SpamBayes comes from Robinson [17, 22], based on ideas by Graham [8] together with Fisher's method for combining independent significance tests [7]. Intuitively, SpamBayes learns how strongly each token indicates *ham* or *spam* by counting the number of each type of email that token appears in. When classifying a new email, SpamBayes looks at all of its tokens and uses a statistical test to decide whether they indicate one label or the other with sufficient confidence; if not, SpamBayes returns *unsure*.

SpamBayes tokenizes each email E based on words, URL components, header elements, and other character sequences that appear in E. Each is treated as a unique token of the email. The SpamBayes algorithm only records whether or not a token occurs in the message, not how many times it occurs. Email E is represented as a binary vector $\mathbf{e}$ where

$$\mathbf{e}_i = \begin{cases} 1 & \text{the } i^{\text{th}} \text{ token occurs in } E \\ 0 & \text{otherwise} \end{cases} .$$

Below, we use $\mathbf{e}$ to refer to both the original message E and SpamBayes' representation of it since we are only concerned with the latter.

In training, SpamBayes computes a spam score vector $\mathbf{P}_{(S)}$ where the i^{th} component is a *token spam score* for the i^{th} token given by

$$\mathbf{P}_{(S,i)} = \frac{N_H N_S(i)}{N_H N_S(i) + N_S N_H(i)} \tag{1}$$

where N_S, N_H, $N_S(i)$, and $N_H(i)$ are the number of spam emails, ham emails, spam emails including the i^{th} token and ham emails including the i^{th} token, respectively. The quantity $\mathbf{P}_{(S,i)}$ is an estimate of $\Pr(E \text{ is spam} \mid \mathbf{e}_i)$ if the prior of ham and spam are equal, but for our purposes, it is simply a per-token score for the email. An analogous *token ham score* is given by $\mathbf{P}_{(H,i)} = 1 - \mathbf{P}_{(S,i)}$.

Robinson's method [22] smooths $\mathbf{P}_{(S,i)}$ through a convex combination with a prior belief x (default value of $x = 0.5$), weighting the quantities by $N(i)$ (the number of training emails with the i^{th} token) and s (chosen for strength of prior with a default of $s = 1$), respectively:

$$\mathbf{f}_i = \frac{s}{s + N(i)} x + \frac{N(i)}{s + N(i)} \mathbf{P}_{(S,i)} \quad . \tag{2}$$

Effectively, smoothing reduces the impact that low token counts have on the scores. For rare tokens, the score is dominated by the prior x. However, as more tokens are observed, the smoothed score gradually shifts to the score in Eq. (1). An analogous smoothed ham score is given by $1 - \mathbf{f}$.

After training, the filter computes the overall spam score $S(\mathbf{m})$ of a new message M using Fisher's method [7] for combining the scores of the tokens observed in M. SpamBayes uses at most 150 tokens from M with scores furthest from 0.5 and outside the interval [0.4,0.6]. Let $\delta(\mathbf{m})$ be the binary vector where $\delta(\mathbf{m})_i = 1$ if token i is one of these tokens, and 0 otherwise. The token spam scores are combined into a *message spam score* for M by

$$S(\mathbf{m}) = 1 - \chi^2_{2n}\left(-2(\log \mathbf{f})^{\mathrm{T}} \delta(\mathbf{m})\right) \quad , \tag{3}$$

where n is the number of tokens in M and $\chi^2_{2n}(\bullet)$ denotes the cumulative distribution function of the chi-square distribution with $2n$ degrees of freedom. A ham score $H(\mathbf{e})$ is similarly defined by replacing $\mathbf{f}$ with $1 - \mathbf{f}$ in Eq. (3). Finally, SpamBayes constructs an overall spam score for M by averaging $S(\mathbf{m})$ and $1 - H(\mathbf{m})$ (both being indicators of whether $\mathbf{m}$ is spam) giving the final score

$$I(\mathbf{m}) = \frac{1 + S(\mathbf{m}) - H(\mathbf{m})}{2} \tag{4}$$

for a message; a quantity between *0* (ham) and *1* (spam). SpamBayes predicts by thresholding $I(\mathbf{m})$ against two user-tunable thresholds θ_0 and θ_1, with defaults $\theta_0 = 0.15$ and $\theta_1 = 0.9$. SpamBayes predicts *ham*, *unsure*, or *spam* if $I(\mathbf{m})$ falls into the interval $[0,\theta_0]$, $(\theta_0,\theta_1]$, or $(\theta_1,1]$, respectively, and filters the message accordingly.

The inclusion of an *unsure* label in addition to *spam* and *ham* prevents us from purely using *ham-as-spam* and *spam-as-ham* misclassification rates (false positives and false negatives, respectively) for evaluation. We must also consider *spam-as-unsure* and *ham-as-unsure* misclassifications. Because of the practical effects on the user's time and effort discussed in Section 2.2.1, *ham-as-unsure* misclassifications are nearly as bad for the user as *ham-as-spam*.

2.3 Attacks

We examine several attacks against the SpamBayes spam filter. Unlike attacks that exploit flaws in an application's implementation or policies, our attacks take advantage of the learner's adaptive nature. Each attack embodies a particular insight about ways in which a machine learning system is vulnerable, which we can better understand in terms of several commonalities between the attacks.

In a previous paper, we categorize attacks against machine learning systems along three axes [1]. The axes of the taxonomy are:

Influence

- *Causative* attacks influence learning with control over training data.
- *Exploratory* attacks exploit misclassifications but do not affect training.

Security violation

- *Integrity* attacks compromise assets via false negatives.
- *Availability* attacks cause denial of service, usually via false positives.

Specificity

- *Targeted* attacks focus on a particular instance.
- *Indiscriminate* attacks encompass a wide class of instances.

The first axis of the taxonomy describes the capability of the attacker: whether (a) the attacker has the ability to influence the training data that is used to construct the classifier (a *Causative* attack) or (b) the attacker does not influence the learned classifier, but can send new emails to the classifier, and observe its decisions on these emails (an *Exploratory* attack).

The second axis indicates the type of security violation caused: (a) false negatives, in which spam slip through the filter (an *Integrity* violation); or (b) false positives, in which ham emails are incorrectly filtered (an *Availability* violation).

The third axis refers to how specific the attacker's intention is: whether (a) the attack is *Targeted* to degrade the classifier's performance on one particular type of email or (b) the attack aims to cause the classifier to fail in an *Indiscriminate* fashion on a broad class of email.

In the remainder of this section, we discuss three novel *Causative* attacks against SpamBayes' learning algorithm in the context of this taxonomy: one is an *Indiscriminate Availability* attack, one is a *Targeted Availability* attack, and the third is a *Targeted Integrity* attack.

A *Causative* attack against a learning spam filter proceeds as follows:

1. The attacker determines the goal for the attack.
2. The attacker sends attack messages to include in the victim's training set.
3. The victim (re-)trains the spam filter, resulting in a tainted filter.
4. The filter's classification performance degrades on incoming messages.

We consider two possible goals for the attacker: to cause spam emails to be seen by the victim or to prevent the victim from seeing legitimate emails. There are at least two motives for the attacker to cause legitimate emails to be filtered as spam. First, a large number of misclassifications will make the spam filter unreliable, causing users to abandon filtering and see more spam. Second, causing legitimate messages to be mislabeled can cause users to miss important messages. For example, an organization competing for a contract could block competing bids by causing them to be filtered as spam, thereby gaining a competitive advantage.

Based on these considerations, we can further divide the attacker's goals into four categories:

1. Cause the victim to *disable* the spam filter, thus letting all spam into the inbox
2. Cause the victim to *miss* a particular ham email filtered away as *spam*
3. Get a *particular* spam into the victim's inbox
4. Get *any* spam into the victim's inbox

In the remainder of this section, we describe attacks that achieve these objectives. Each of the attacks we describe consists of inserting emails into the training set that are drawn from a particular distribution; the properties of these distributions, along with other parameters, determine the nature of the attack. The *dictionary attack* sends email messages with tokens drawn from a broad distribution, essentially including every token with equal probability. The *focused attack* focuses the distribution specifically on one message or a narrow class of messages. If the attacker has the additional ability to send messages that will be trained as ham, a *pseudospam attack* can cause spam messages to reach the user's inbox.

2.3.1 Causative Availability Attacks

We first focus on *Causative Availability* attacks, which manipulate the filter's training data to increase the number of ham messages misclassified. We consider both *Indiscriminate* and *Targeted* attacks. In *Indiscriminate* attacks, enough false positives force the victim to disable the filter or frequently search in *spam*/*unsure* folders for legitimate messages erroneously filtered away. Hence, the victim is forced to view more spam. In *Targeted* attacks, the attacker does not disable the filter but surreptitiously prevents the victim from receiving certain messages.

Without loss of generality, consider the construction of a single attack message **a**. The victim adds it to the training set, (re-)trains on the contaminated data, and subsequently uses the tainted model to classify a new message **m**. The attacker also has some (perhaps limited) knowledge of the next email the victim will receive. This knowledge can be represented as a distribution **p**—the vector of probabilities that each token will appear in the next message.

The goal of the attacker is to choose the tokens for the attack message **a** to maximize the *expected spam score*:

$$\max_{\mathbf{a}} \mathbf{E}_{\mathbf{m}\sim\mathbf{p}}\left[I_{\mathbf{a}}(\mathbf{m})\right] \quad . \tag{5}$$

In other words, the attack goal is to maximize the expectation of $I_{\mathbf{a}}(\mathbf{m})$ (Eq. (4) with the attack message **a** added to the spam training set) of the next legitimate email **m** drawn from distribution **p**.

To describe the optimal attack under this criterion, we make two observations, which we detail in Appendix 2.A. First, $I_{\mathbf{a}}(\mathbf{m})$ is monotonically non-decreasing in $\mathbf{f}_i$ for all i. Therefore, increasing the score of any token in the attack message can only increase $I_{\mathbf{a}}$. Second, the token scores of distinct tokens do not interact; that is, adding the i^{th} token to the attack does not change the score $\mathbf{f}_j$ of some different token $j \neq i$. Hence, the attacker can simply choose which tokens will be most beneficial for their purpose. From this, we motivate two attacks, the *dictionary* and *focused* attacks, as instances of a common attack in which the attacker has different amounts of knowledge about the victim's email.

For this, let us consider specific choices for the distribution **p**. First, if the attacker has little knowledge about the tokens in target emails, we give equal probability to each token in **p**. In this case, we can optimize the expected message spam score by including *all possible tokens* in the attack email. Second, if the attacker has specific knowledge of a target email, we can represent this by setting $\mathbf{p}_i$ to 1 if and only if the i^{th} token is in the target email. This attack is also optimal with respect to the target message, but it is much more compact.

In reality, the attacker's knowledge usually falls between these two extremes. If the attacker has relatively little knowledge, such as knowledge that the victim's primary language is English, the attack can include all words in an English dictionary. This reasoning yields the *dictionary attack* (Section 2.3.1.1). On the other hand, the attacker may know *some* of the particular words to appear in a target email, though not all of the words. This scenario is the *focused attack* (Section 2.3.1.2). Between these levels of knowledge, an attacker could use information about the distribution of words in English text to make the attack more efficient, such as characteristic vocabulary or jargon typical of emails the victim receives. Any of these cases result in a distribution **p** over tokens in the victim's email that is more specific than an equal distribution over all tokens but less informative than the true distribution of tokens in the next message. Below, we explore the details of the dictionary and focused attacks, with some exploration of using an additional corpus of common tokens to improve the dictionary attack.

2.3.1.1 Dictionary Attack

The *dictionary attack*, an *Indiscriminate* attack, makes the spam filter unusable by causing it to misclassify a significant portion of ham emails so that the victim disables the spam filter, or at least must frequently search through *spam*/*unsure* folders to find legitimate messages that were filtered away. In either case, the victim loses confidence in the filter and is forced to view more spam achieving the ultimate goal of the spammer: the victim sees the attacker's spam. The result of this attack is denial of service, a higher rate of ham misclassified as *spam*.

The dictionary attack is an approximation of the optimal attack suggested in Section 3.1, in which the attacker maximizes the expected score by including all possible tokens. Creating messages with every possible token is infeasible in practice. Nevertheless, when the attacker lacks knowledge about the victim's email, this optimal attack can be approximated with an entire dictionary of the victim's native language—the *dictionary attack*. The dictionary attack increases the score of every token in a dictionary; *i.e.*, it makes them more indicative of spam.

The central idea that underlies the dictionary attack is to send attack messages containing a large set of tokens—the attacker's *dictionary*. The dictionary is selected as the set of tokens whose scores maximally increase the expected value of $I_{\mathbf{a}}(\mathbf{m})$ as in Eq. (5). Since the score of a token typically increases when included in an attack message except in unusual circumstances (see Appendix 2.A.2), the attacker simply needs to include any tokens that are likely to occur in future legitimate message. In particular, if the victim's language is known by the attacker, they can use that language's entire lexicon (or at least a large subset) as the attack dictionary. After training on the dictionary message, the victim's spam filter will have a higher spam score for every token in the dictionary, an effect that is amplified for rare tokens. As a result, future legitimate email is more likely to be marked as *spam* since it will contain many tokens from that lexicon.

A refinement uses a token source with a distribution closer to the victim's email distribution. For example, a large pool of *Usenet* newsgroup postings may have colloquialisms, misspellings, and other “words” not found in a proper dictionary; furthermore, using the most frequent tokens in such a corpus may allow the attacker to send smaller emails without losing much effectiveness. However, there is an inherent trade-off in choosing tokens. Rare tokens are the most vulnerable to attack since their scores will shift more towards spam (1.0 in the SpamBayes formulas) with fewer attack emails. However, the rare vulnerable tokens also are less likely to appear in future messages, diluting their usefulness.

2.3.1.2 Focused Attack

Our second *Causative Availability* attack is a *Targeted* attack—the attacker has some knowledge of a specific legitimate email to target for filtering. If the attacker has exact knowledge of the target email, placing all of its tokens in attack emails produces an optimal attack. Realistically, the attacker has partial knowledge about the target email and can guess only some of its tokens to include in attack emails. We model this knowledge by letting the attacker know a certain fraction of tokens from the target email, which are included in the attack message. (The attack email may also include additional tokens added by the attacker to obfuscate the attack message's intent.) When SpamBayes trains on the resulting attack email, the spam scores of the targeted tokens increase, so the target message is more likely to be filtered as *spam*. This is the focused attack.

Consider an example in which the attacker sends spam emails to the victim with tokens such as the names of competing companies, their products, and their employees. The bid messages may even follow a common template known to the attacker, making the attack easier to craft. As a result of the attack, the legitimate bid email may be filtered away as *spam*, causing the victim not to see it.

The focused attack is more concise than the dictionary attack because the attacker has detailed knowledge of the target email and no reason to affect other messages. This conciseness makes the attack both more efficient for the attacker and more difficult to detect for the defender. Further, the focused attack can be more effective because the attacker may know proper nouns and other non-word tokens common in the victim's email that are otherwise uncommon.

An interesting side-effect of the focused attack is that repeatedly sending similar emails tends to not only increase the spam score of tokens in the attack but also *reduce* the spam score of tokens not in the attack. To understand why, recall the estimate of the token posterior in Eq. (1), and suppose that the i^{th} token does not occur in the attack email. Then N_S increases with the addition of the attack email but $N_S(i)$ does not, so $\mathbf{P}_{(S,i)}$ decreases and therefore so does $\mathbf{f}_i$. In Section 2.4.3, we observe empirically that the focused attack can indeed reduce the spam score of tokens not included in the attack emails.

2.3.2 *Causative Integrity Attacks—Pseudospam*

We also examine *Causative Integrity* attacks, which manipulate the filter's training data to increase false negatives; that is, spam messages misclassified as *ham*. In contrast to the previous attacks, the *pseudospam attack* directly attempts to make the filter misclassify spam messages. If the attacker can choose messages arbitrarily that are trained as ham, the attack is similar to a focused attack with knowledge of 100% of the target email's tokens. However, there is no reason to believe a user would train on arbitrary messages as ham. We introduce the concept of a *pseudospam email*—an email that does not look like spam but that has characteristics (such as headers) that are typical of true spam emails. Not all users consider benign-looking, non-commercial emails offensive enough to mark them as spam.

To create pseudospam emails, we take the message body text from newspaper articles, journals, books, or a corpus of legitimate email. The idea is that in some cases, users may mistake these messages as ham for training, or may not be diligent about correcting false negatives before retraining, if the messages do not have marketing content. In this way, an attacker might be able to gain control of ham training data. This motivation is less compelling than the motivation for the dictionary and focused attacks, but in the cases where it applies, the headers in the pseudospam messages will gain significant weight indicating ham, so when future spam is sent with similar headers (*i.e.,* by the same spammer) it will arrive in the user's inbox.

2.4 Experiments

In this section, we present our empirical results. First, in Section 2.4.1, we outline our experimental setup. Then, we discuss the effect of each of our attacks in the remainder of the section.

2.4.1 Experimental Method

2.4.1.1 Datasets

In our experiments we use the Text Retrieval Conference (TREC) 2005 spam corpus [5], which is based on the Enron email corpus [12] and contains 92,189 emails (52,790 spam and 39,399 ham). This corpus has several strengths: it comes from a real-world source, it has a large number of emails, and its creators took care that the added spam does not have obvious artifacts to differentiate it from the ham.

We use two sources of tokens for attacks. First, we use the GNU Aspell English dictionary version 6.0-0, containing 98,568 words. We also use a corpus of English Usenet postings to generate tokens for our attacks. This corpus is a subset of a Usenet corpus of 140,179 postings compiled by the

University of Alberta's Westbury Lab [23]. An attacker can download such data and build a language model to use in attacks, and we explore how effective this technique is. We build a primary Usenet dictionary by taking the most frequent 90,000 tokens in the corpus (Usenet-90k), and we also experiment with a smaller dictionary of the most frequent 25,000 tokens (Usenet-25k).

The overlap between the Aspell dictionary and the most frequent 90,000 tokens in the Usenet corpus is approximately 26,800 tokens. The overlap between the Aspell dictionary and the TREC corpus is about 16,100 tokens, and the intersection of the TREC corpus and Usenet-90k is around 26,600 tokens.

2.4.1.2 Constructing Message Sets for Experiments

In constructing an experiment, we often need several non-repeating sequences of emails in the form of mailboxes. When we require a mailbox, we sample messages without replacement from the TREC corpus, stratifying our sampling to ensure the necessary proportions of ham and spam. For subsequent messages needed in any part of our experiments (target messages, headers for attack messages, and so on), we again sample emails without replacement from the messages remaining in the TREC corpus. In this way we ensure that no message is repeated in our experiments.

We construct attack messages by splicing elements of several emails together to make messages that are realistic under our model of the adversary's control. We construct our attack email bodies according to the specifications of the attack. We select the header for each attack email by choosing a random spam email from TREC and using its headers, taking care to ensure that the content-type and other Multipurpose Internet Mail Extensions (MIME) headers correctly reflect the composition of the attack message body. Specifically, we discard the entire existing multi- or single-part body and we set relevant headers (such as Content-Type and Content-Transfer-Encoding) to indicate a single plain-text body.

The tokens used in each attack message are selected from our datasets according to the attack method. For the dictionary attack, we use all tokens from the attack dictionary in every attack message (98,568 tokens for the Aspell dictionary and 90,000 or 25,000 tokens for the Usenet dictionary). For the focused and the pseudospam attacks, we select tokens for each attack message based on a fresh message sampled from the TREC dataset. The number of tokens in attack messages for the focused and pseudospam attacks varies, but all such messages are comparable in size to the messages in the TREC dataset.

Finally, to evaluate an attack, we create a control model by training SpamBayes once on the base training set. We incrementally add attack emails to the training set and train new models at each step, giving us a series of models tainted with increasing numbers of attack emails. (Because Spam-

Bayes is order-independent in its training, it arrives at the same model whether training on all messages in one batch or training incrementally on each email in any order.) We evaluate the performance of these models on a fresh set of test messages.

Parameter	Focused Attack	Pseudospam Attack	RONI defense
Training set size	2,000, 10,000	2,000, 10,000	2,000, 10,000
Test set size	200, 1,000	200, 1,000	200, 1,000
Spam percent	50, 75, 90	50, 75, 90	50
Attack percent	0.1, 0.5, 1, 2, 5, 10	0.1, 0.5, 1, 2, 5, 10	10
Validation folds	10	10	-
Target emails	20	-	-

Table 2.1 Parameters used in the experiments

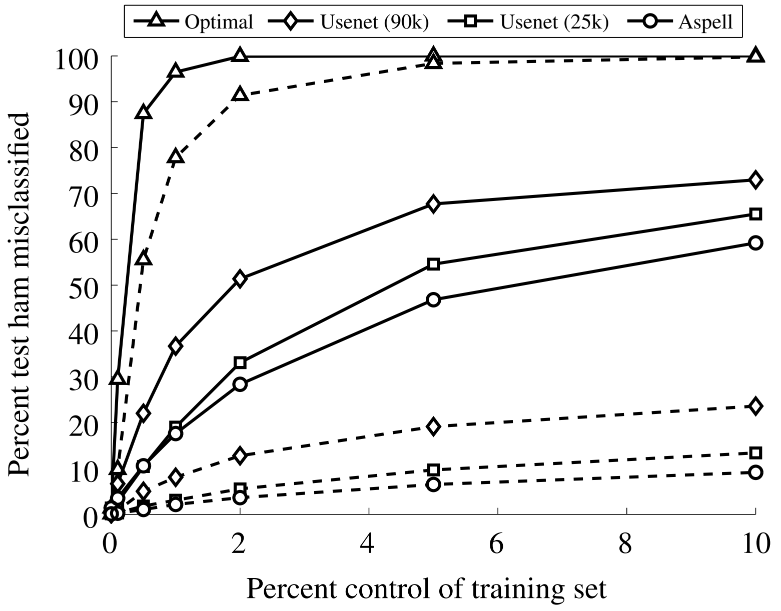

Fig. 2.1 Training on 10,000 messages (50% spam). Figures 2.1-2.4 show the effect of three dictionary attacks on SpamBayes in two settings. Figure 2.1 and Figure 2.2 have an initial training set of 10,000 messages (50% spam) while Figure 2.3 and Figure 2.4 have an initial training set of 2,000 messages (75% spam). Figure 2.2 and Figure 2.4 also depict the standard errors in our experiments for both of the settings. We plot percent of ham classified as *spam* (dashed lines) and as *spam* or *unsure* (solid lines) against the attack as percent of the training set. We show the optimal attack (Δ), the Usenet-90k dictionary attack (◊), the Usenet-25k dictionary attack (□), and the Aspell dictionary attack (○). Each attack renders the filter unusable with adversarial control over as little as 1% of the messages (101 messages).

2.4.1.3 Attack Assessment Method

We measure the effect of each attack by randomly choosing an inbox according to the parameters in Table 2.1 and comparing classification performance of the control and compromised filters using ten-fold cross-validation. In cross-validation, we partition the data into ten subsets and perform ten train-test epochs. During the *k*th epoch, the *k*th subset is set aside as a test set and the remaining nine subsets are combined into a training set. In this way, each email from the sample inbox functions independently as both training and test data.

In the following sections, we show the effect of our attacks on test sets of held-out messages. Because our dictionary and focused attacks are designed to cause ham to be misclassified, we only show their effect on ham messages; we found that their effect on spam is marginal. Likewise, for the pseudospam attack, we concentrate on the results for spam messages. Most of our graphs do not include error bars since we observed that the variation on our tests was small compared to the effect of our attacks (see Figure 2.2 and Figure 2.4). See Table 2.1 for our experimental parameters. We found that varying the size of the training set and spam prevalence in the training set had minimal impact on the performance of our attacks (for comparison, see Figure 2.1 and Figure 2.3), so we primarily present the results of 10,000-message training sets at 50% spam prevalence.

2.4.2 Dictionary Attack Results

We examine dictionary attacks as a function of the percent of attack messages in the training set. Figures 2.1-2.4 show the misclassification rates of three dictionary attack variants averaging over ten-fold cross-validation in two settings (Figure 2.1 and Figure 2.2 have an initial training set of 10,000 messages with 50% spam while Figure 2.3 and Figure 2.4 have an initial training set of 2,000 messages with 75% spam). First, we analyze the optimal dictionary attack discussed in Section 2.3.1 by simulating the effect of including every possible token in our attack emails. As shown in the figures, this optimal attack quickly causes the filter to mislabel all ham emails.

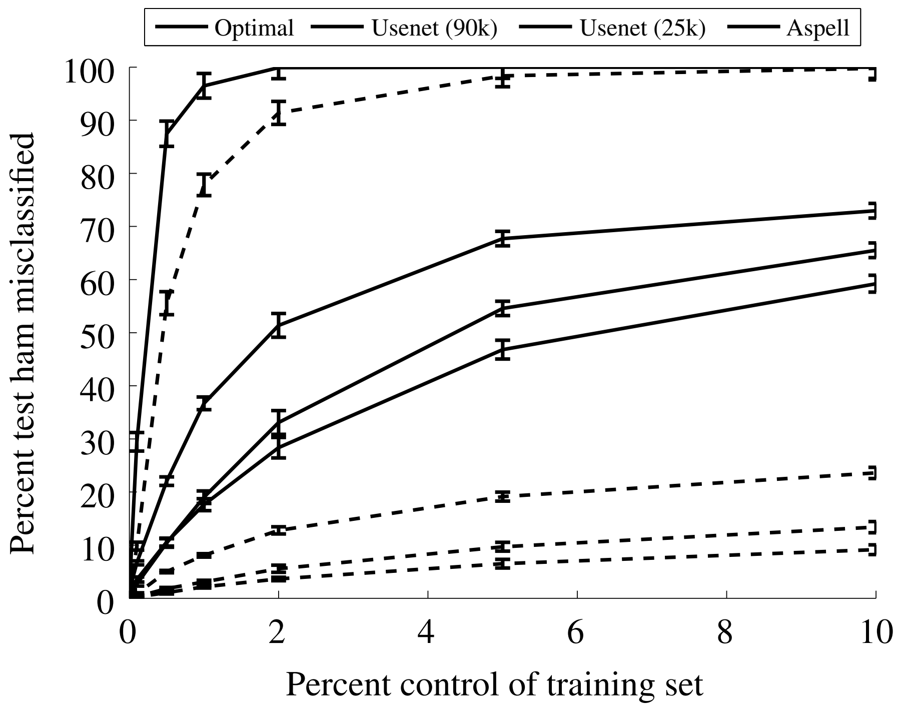

Fig. 2.2 Training on 10,000 messages (50% spam) with error bars. See Figure 2.1.

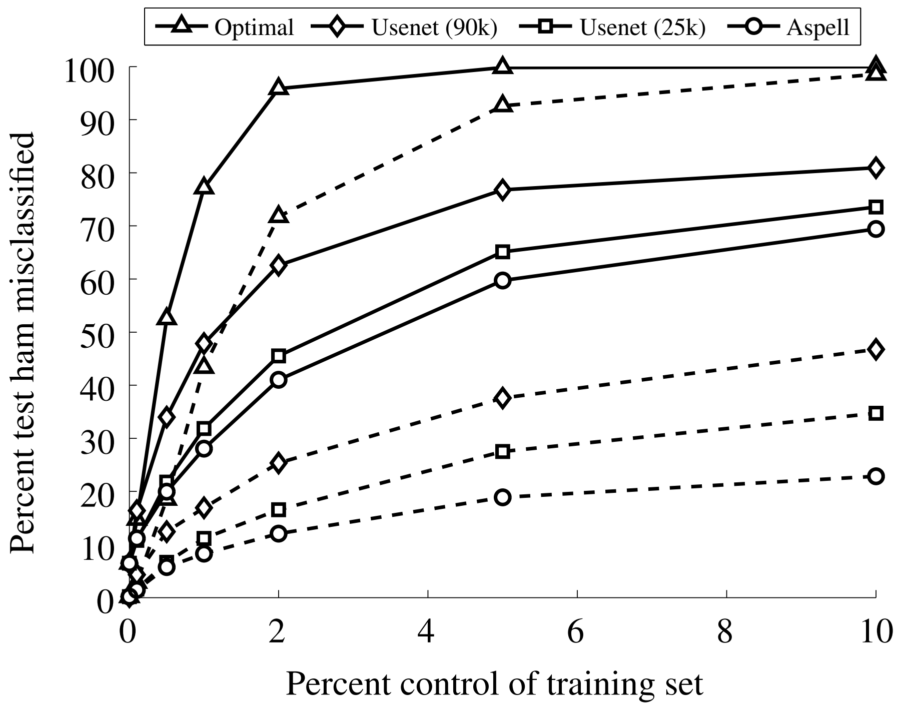

Fig. 2.3 Training on 2,000 messages (75% spam). See Figure 2.1.

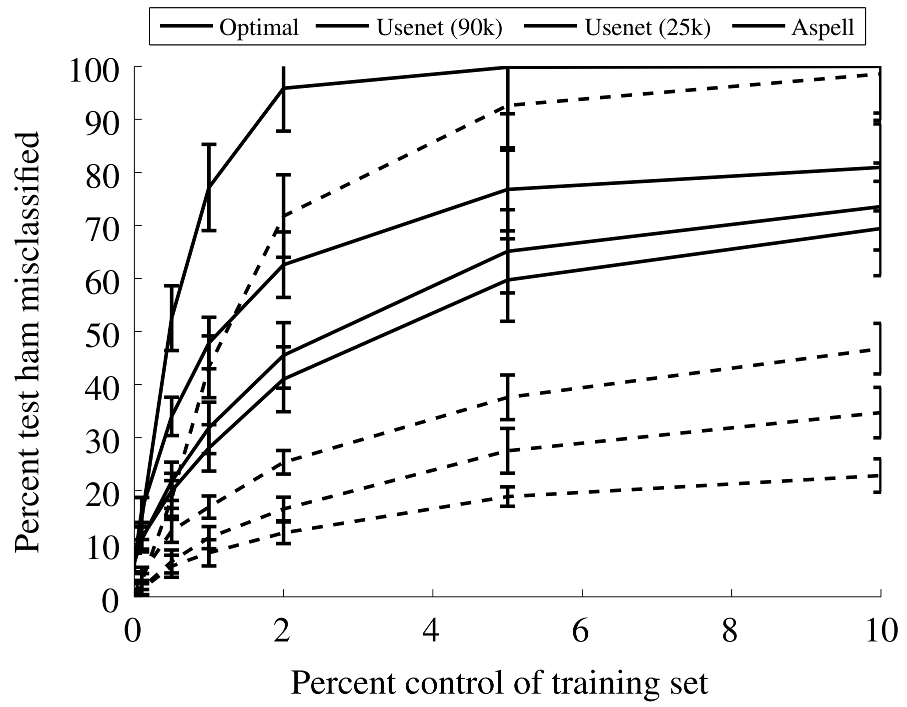

Fig. 2.4 Training on 2,000 messages (75% spam) with error bars. See Figure 2.1.

Dictionary attacks using tokens from the Aspell dictionary are also successful, though not as successful as the optimal attack. Both the Usenet-90k and Usenet-25k dictionary attacks cause more ham emails to be misclassified than the Aspell dictionary attack, since they contains common misspellings and slang terms that are not present in the Aspell dictionary. All of these variations of the attack require relatively few attack emails to significantly degrade SpamBayes' accuracy. After 101 attack emails (1% of 10,000), the accuracy of the filter falls significantly for each attack variation. Overall misclassification rates are 96% for optimal, 37% for Usenet-90k, 19% for Usenet-25k, and 18% for Aspell—at this point most users will gain no advantage from continued use of the filter.

It is of significant interest that so few attack messages can degrade a common filtering algorithm to such a degree. However, while the attack emails make up a small percentage of the *number of messages* in a contaminated inbox, they make up a large percentage of the *number of tokens*. For example, at 204 attack emails (2% of the training messages), the Usenet-25k attack uses approximately 1.8 times as many tokens as the entire pre-attack training dataset, and the Aspell attack includes 7 times as many tokens.

While it seems trivial to prevent dictionary attacks by filtering large messages out of the training set, such strategies fail to completely address this vulnerability of SpamBayes. First, while ham messages in TREC are relatively small (fewer than 1% exceeded 5,000 tokens and fewer than 0.01% of messages exceeded 25,000 tokens), this dataset may not be representative of actual messages. Second, an attacker can circumvent size-based thresholds. By fragmenting the dictionary, an attack can have a similar impact using more messages with fewer tokens per message. Additionally, informed token selection methods can yield more effective dictionaries as we demonstrate with the two Usenet dictionaries. Thus, size-based defenses lead to a trade-off between vulnerability to dictionary attacks and the effectiveness of training the filter. In Section 2.5, we present a defense that instead filters messages based directly on their impact on the spam filter's accuracy.

2.4.3 Focused Attack Results

In this section, we discuss experiments examining how accurate the attacker needs to be at guessing target tokens, how many attack emails are required for the focused attack to be effective, and what effect the focused attack has on the token scores of a targeted message. For the focused attack, we randomly select 20 ham emails from the TREC corpus to serve as the target emails before creating the clean training set. During each fold of cross-validation, we perform 20 focused attacks, one for each email, so our results average over 200 different trials.

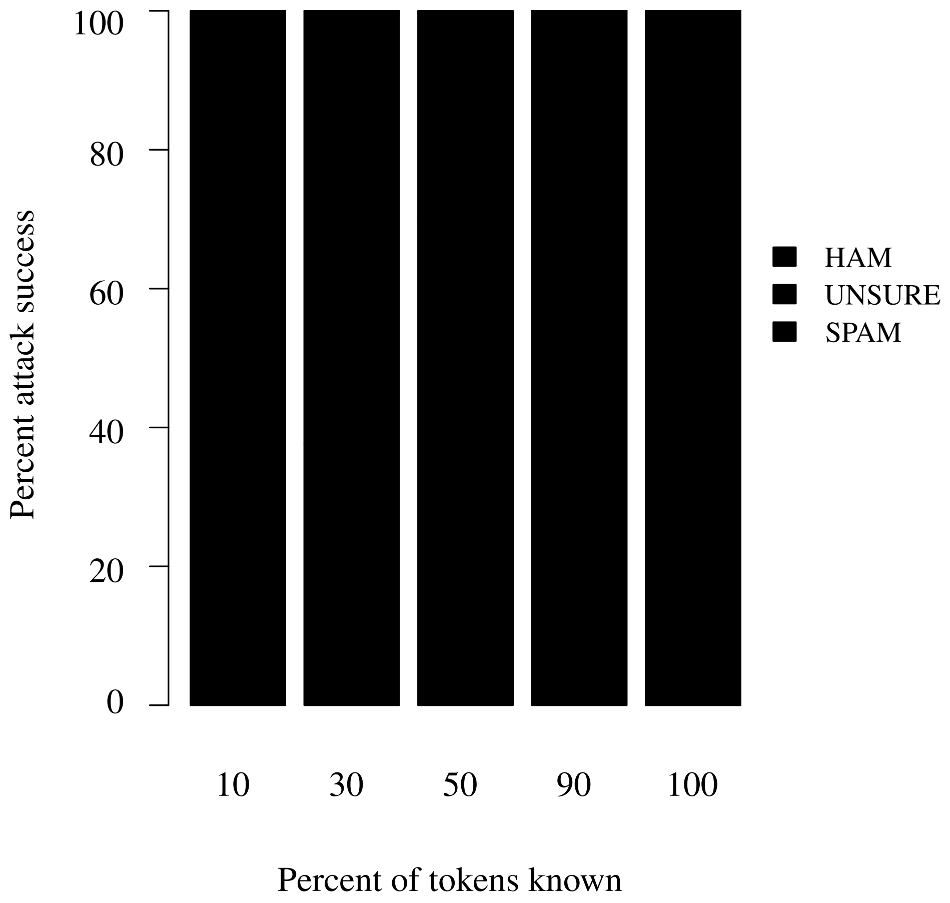

Fig. 2.5 Effect of the focused attack as a function of the percentage of target tokens known by the attacker. Each bar depicts the fraction of target emails classified as *spam*, *ham*, and *unsure* after the attack. The initial inbox contains 10,000 emails (50% spam).

These results differ from our preliminary focused attack experiments [19] in two important ways. First, here we randomly select a fixed percentage of tokens known by the attacker from each message instead of selecting each token with a fixed probability. The later approach causes the percentage of tokens known by the attacker to fluctuate from message to message. Second, here we only select messages with more than 100 tokens to use as target emails. With these changes, our results more accurately represent the behavior of a focused attack. Furthermore, in this more accurate setting, the focused attack is even more effective.

Figure 2.5 shows the effectiveness of the attack when the attacker has increasing knowledge of the target email by simulating the process of the attacker guessing tokens from the target email. We assume that the attacker knows a fixed fraction F of the actual tokens in the target email, with $F \in \{0.1, 0.3, 0.5, 0.9\}$ —the x-axis of Figure 2.5. The y-axis shows the percent of the 20 targets classified as *ham*, *unsure* and *spam*. As expected, the attack is increasingly effective as F increases. If the attacker knows 50% of the tokens in the target, classification changes to *spam* or *unsure* on *all* of the target emails, with a 75% rate of classifying as *spam*.

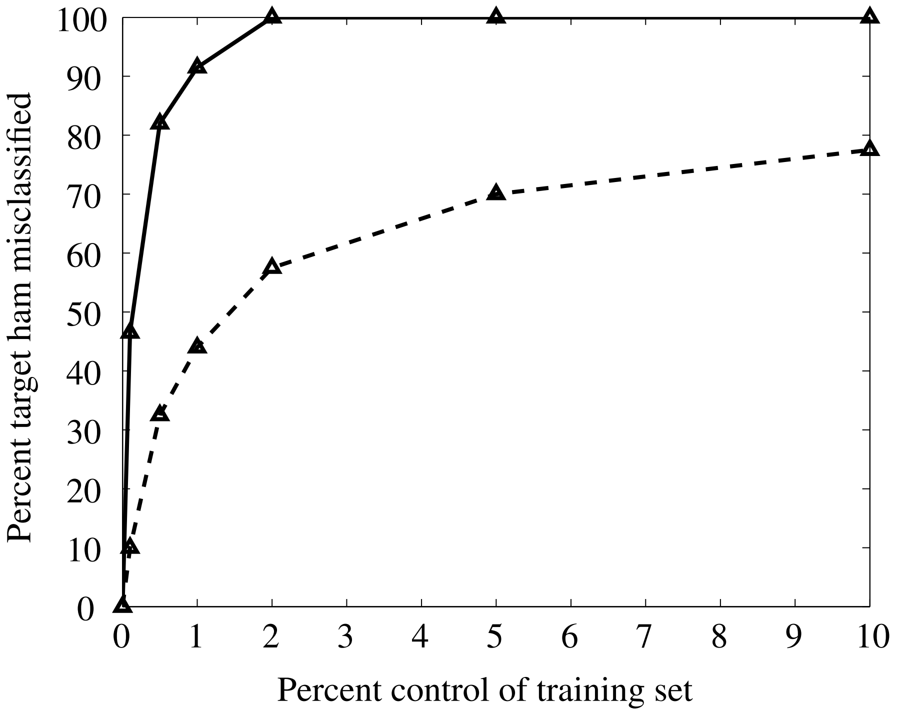

Fig. 2.6 Effect of the focused attack as a function of the number of attack emails with a fixed percentage (50%) of tokens known by the attacker. The dashed line shows the percentage of target ham messages classified as *spam* after the attack, and the solid line the percentage of targets that are *spam* or *unsure* after the attack. The initial inbox contains 10,000 emails (50% spam).

Figure 2.6 shows the attack's effect on misclassifications of the target emails as the number of attack messages increases. We fix the fraction of known tokens at 0.5. The x-axis shows the number of messages in the attack as a fraction of the training set, and the y-axis shows the fraction of target messages misclassified. With 101 attack emails inserted into an initial mailbox size of 10,000 (1%), the target email is misclassified as *spam* or *unsure* over 90% of the time.

Figures 2.7-2.9 show the attack's effect on three representative emails. Each of the figures represents a single target email from each of three attack results: ham misclassified as *spam* (Figure 2.7), ham misclassified as *unsure*

(Figure 2.8), and ham correctly classified as *ham* (Figure 2.9). Each point represents a token in the email. The x-axis is the token's spam score (from Eq. (2)) before the attack, and the y-axis is the token's score after the attack (0 means *ham* and 1 means *spam*). The ×'s are tokens included in the attack (known by the attacker) and the ○'s are tokens not in the attack. The histograms show the distribution of token scores before the attack (at bottom) and after the attack (at right).

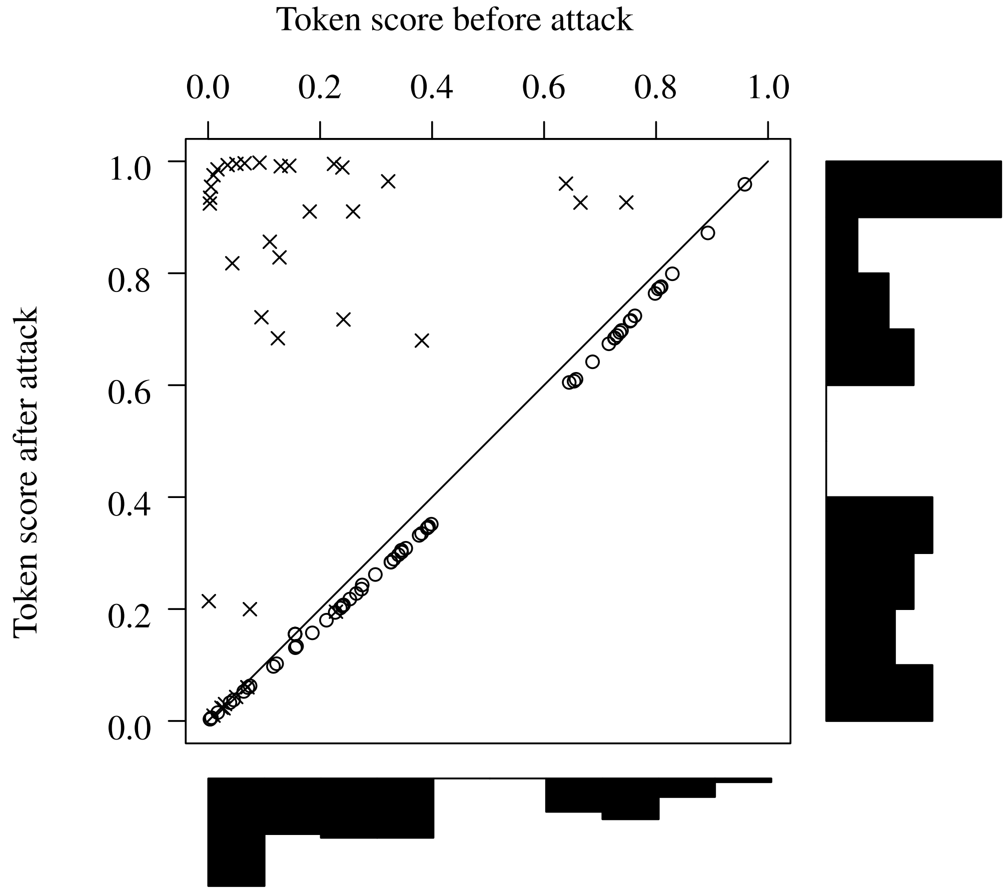

Fig. 2.7 Misclassified as *spam*. Figures 2.7-2.9 show the effect of the focused attack on three representative emails—one graph for each target. Each point is a token in the email. The x-axis is the token's spam score in Eq. (2) before the attack (0 means ham and 1 means spam). The y-axis is the token's spam score after the attack. The ×'s are tokens that were included in the attack and the ○'s are tokens that were not in the attack. The histograms show the distribution of spam scores before the attack (at bottom) and after the attack (at right).

Any point above the line $y=x$ is a token whose score increased due to the attack and any point below is a decrease. In these graphs we see that the score of the tokens included in the attack typically increase significantly while those not included decrease slightly. Since the increase in score is more significant for included tokens than the decrease in score for excluded tokens, the attack has substantial impact even when the attacker has a low probability of guessing tokens, as seen in Figure 2.5. Further, the before/after histograms in Figures 2.7-2.9 provide a direct indication of the attack's success. By shifting most token scores toward 1, we cause more misclassifications.

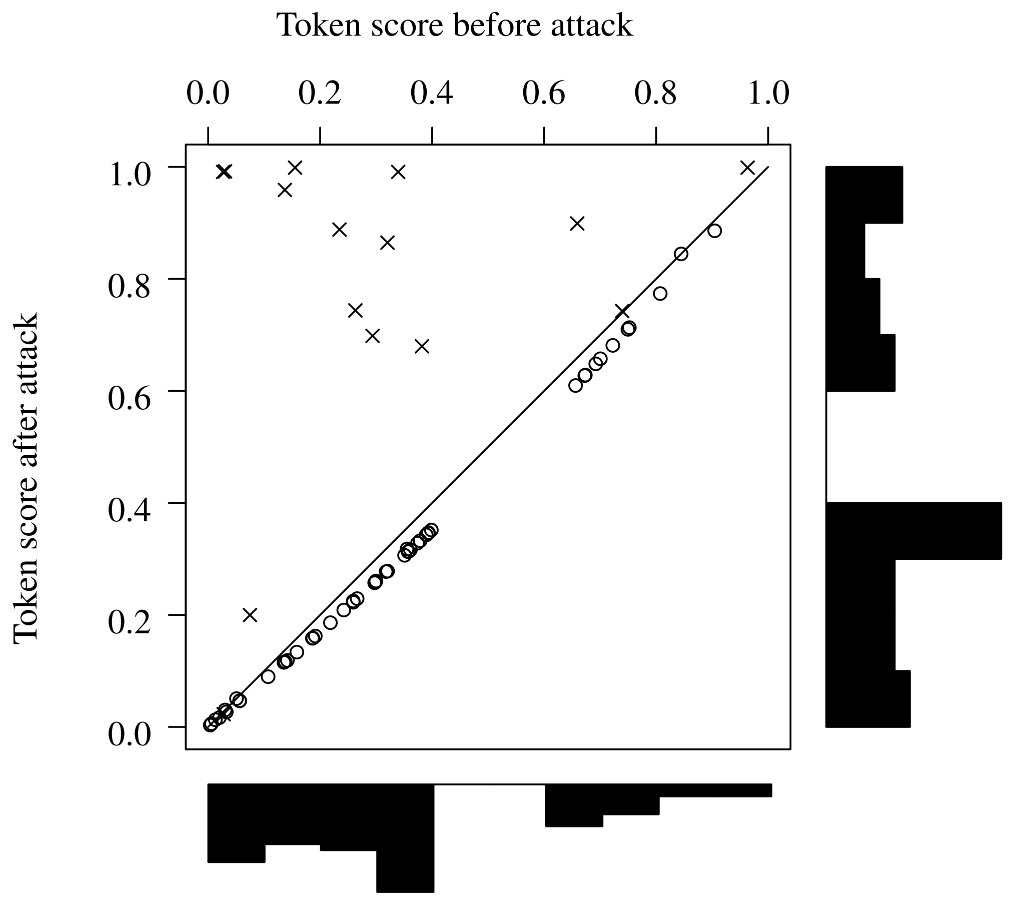

Fig. 2.8 Misclassified as *unsure*. See Figure 2.7.

2.4.4 Pseudospam Attack Experiments

In contrast to the previous attacks, for the pseudospam attack, we created attack emails that may be labeled as ham by a human as the emails are added into the training set. We setup the experiment for the pseudospam attack by first randomly selecting a target spam header to be used as the base header for the attack. We then create the set of attack emails that look similar to ham emails (see Section 2.3.2). To create attack messages, we combine each ham email with the target spam header. This is done so that the attack email has contents similar to other legitimate email messages. Header fields that may modify the interpretation of the body are taken from the ham email to make the attack realistic.

Figure 2.10 demonstrates the effectiveness of the *pseudospam attack*. We plot the percent of attack messages in the training set (x-axis) against the misclassification rates on the test spam email (y-axis). The solid line shows the fraction of target spam classified as *ham* or *unsure* spam while the dashed line shows the fraction of spam classified as *ham*. In the absence of attack, SpamBayes only misclassifies about 10% of the target spam emails (including those labeled *unsure*). If the attacker can insert a few hundred attack emails (1% of the training set), then SpamBayes misclassifies more than 80% of the target spam emails.

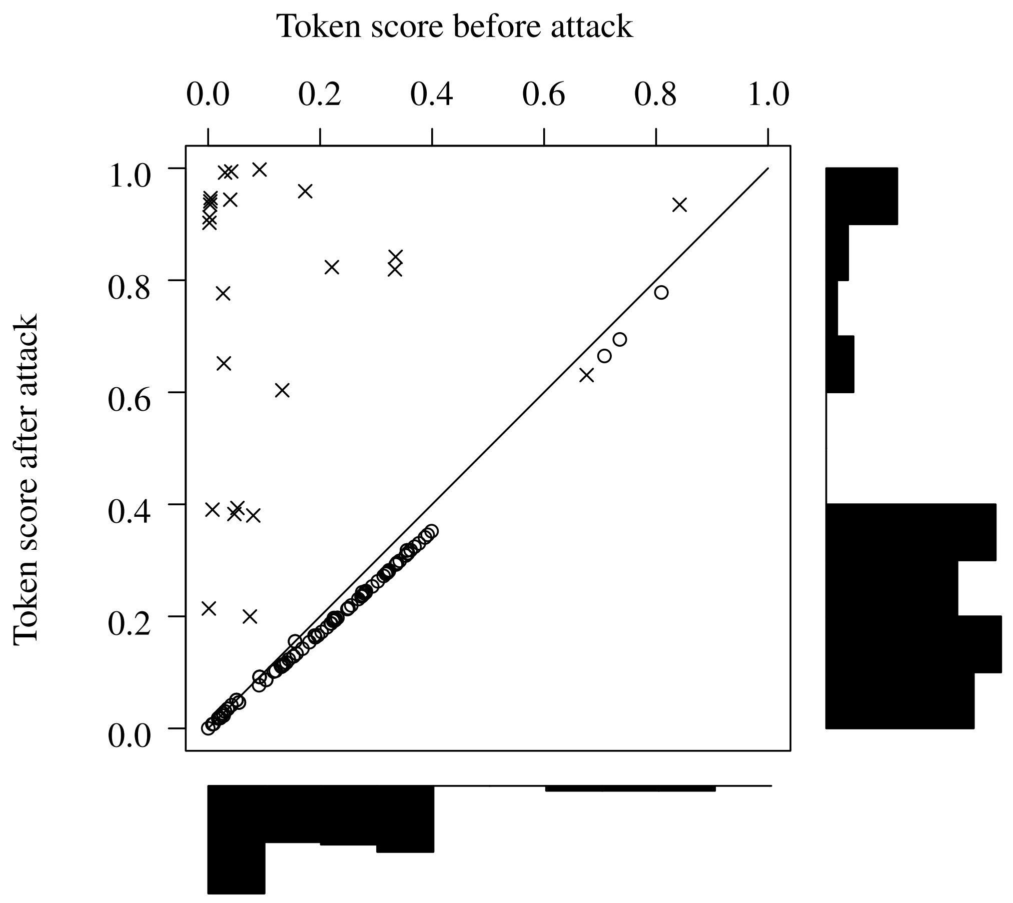

Fig. 2.9 Correctly classified as *ham*. See Figure 2.7.

Further, the attack has a minimal effect on regular ham and spam messages. Other spam email messages are still correctly classified since they do not generally have the same header fields as the adversary's messages. In fact, ham messages may have lower spam scores since they may contain tokens similar to those in the attack emails.

We also explore the scenario in which the pseudospam attack emails are labeled by the user as *spam* to better understand the effect of these attacks if the pseudospam messages fail to fool the user. The result is that, in general, SpamBayes classifies more spam messages incorrectly. As Figure 2.11 indicates, this variant causes an increase in spams mislabeled as either *unsure* or *ham* increases to nearly 15% as the number of attack emails increases. Further, this version of the attack does not cause a substantial impact on normal ham messages.

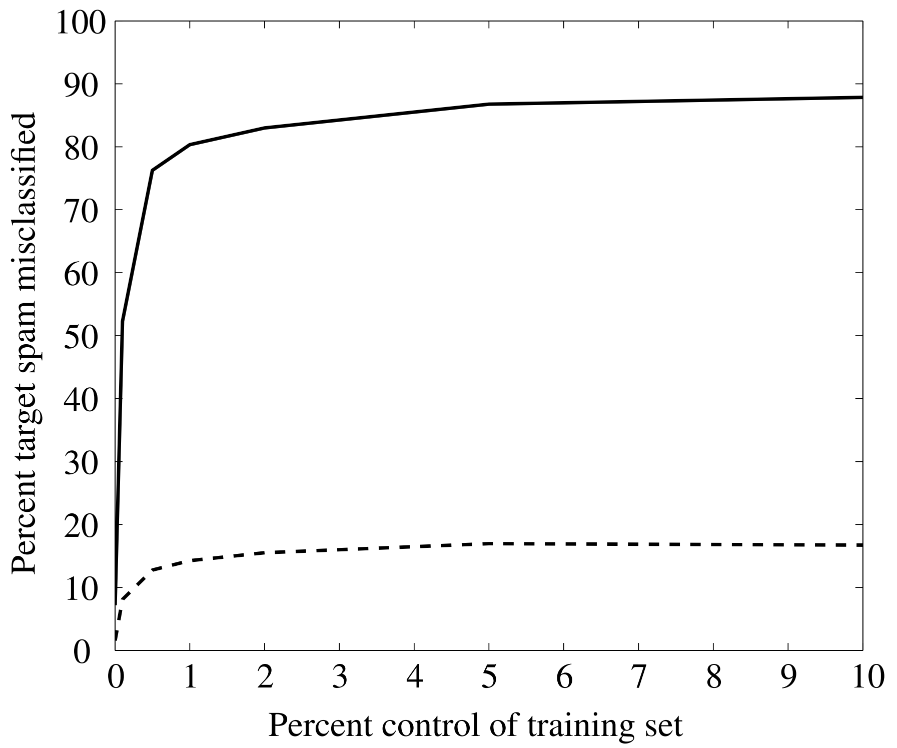

Fig. 2.10 Effect of the pseudospam attack when trained as ham as a function of the number of attack emails. The dashed line shows the percentage of the adversary's messages classified as *ham* after the attack, and the solid line the percentage that are *ham* or *unsure* after the attack. The initial inbox contains 10,000 emails (50% spam).

2.5 A Defense: Reject on Negative Impact (RONI)

Here we describe a potential defense against some of our attacks. Our *Causative* attacks succeed because training on attack emails causes the filter to learn incorrectly and misclassify emails. Each attack email contributes towards the degradation of the filter's performance; if we can measure each email's impact, then we can remove messages with a deleterious effect from the training set.

In the *Reject On Negative Impact* (*RONI*) *defense*, we measure the incremental effect of a *query email* Q by testing the performance difference with and without that email. We independently sample a 20-message training set T and a 50-message validation set V five times from the initial pool of emails given to SpamBayes for training, choosing a small training set so that the effect of a single email will be greater. We train on both T and $T \cup Q$ and measure the impact of Q as the average change in incorrect classifications on V over the five trials. We remove Q from the training pool if its impact is significantly harmful.

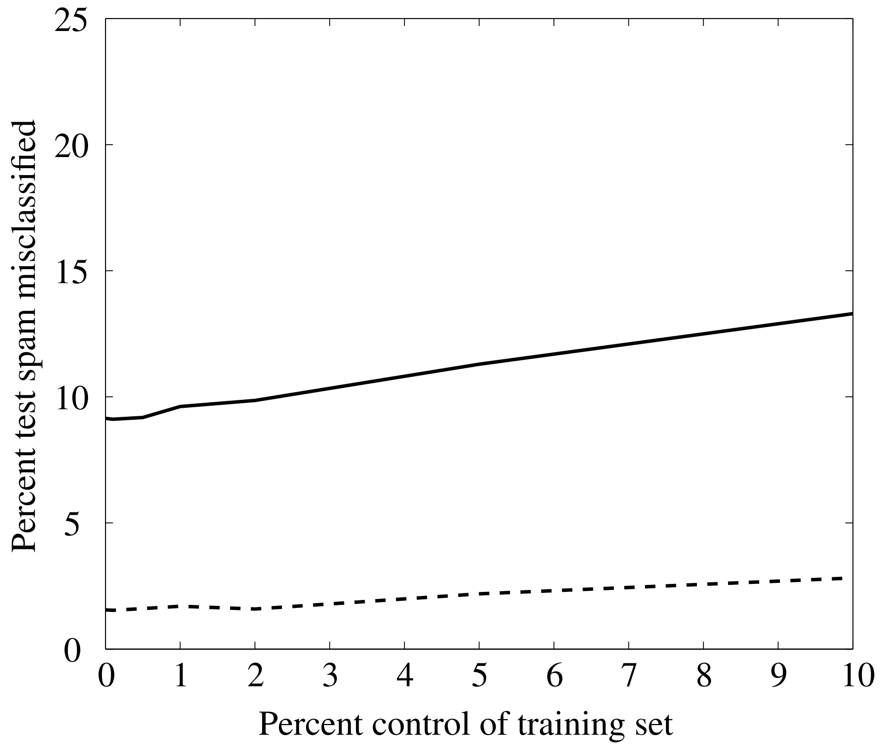

Fig. 2.11 Effect of the pseudospam attack when trained as spam, as a function of the number of attack emails. The dashed line shows the percentage of the normal spam messages classified as *ham* after the attack, and the solid line the percentage that are *unsure* after the attack. Surprisingly, training the attack emails as ham causes an increase in misclassification of normal spam messages. The initial inbox contains 10,000 emails (50% spam).

We test the effectiveness of this defense with 120 random non-attack spam messages and dictionary attack emails using both the Aspell and Usenet dictionaries. Our preliminary experiments show that the RONI defense is extremely successful against dictionary attacks, identifying 100% of the attack emails without flagging any non-attack emails. Each dictionary attack message causes an average decrease of *at least* 6.8 true negatives (ham-as-*ham* messages). In sharp contrast, non-attack spam messages cause *at most* an average decrease of 4.4 true negatives. Hence a simple threshold on this statistic is effective at separating dictionary attack emails from non-attack spam.

However, the RONI defense fails to detect focused attack emails because the focused attack targets a *future* message, so its effect on the training set is minute.

2.6 Related Work

Here we briefly review prior work related to the security of learning systems. A more detailed survey of this literature on appears in a related technical report [2].

Many authors have examined adversarial learning from a more theoretical perspective. For example, within the Probably Approximately Correct framework, Kearns and Li bound the classification error an adversary can cause with control over a fraction of the training set [10]. Dalvi et al. apply game theory to the classification problem [6]. They model the interactions between the classifier and attacker as a game and develop an optimal counter-strategy for an optimal classifier playing against an optimal opponent.

We focus on *Causative* attacks. Most existing attacks against content-based spam filters are *Exploratory* attacks that do not influence training but engineer spam messages so they pass through the filter. For example, Lowd and Meek explore reverse-engineering a spam classifier to find high-value messages that the filter does not block [15, 16], Karlberger et al. study the effect of replacing strong spam words with synonyms [9], and Wittel and Wu study the effect of adding common words to spam to get it through a spam filter [25].

Several others have recently developed *Causative* attacks against learning systems. Chung and Mok [3, 4] present a *Causative Availability* attack against the Autograph worm signature generation system [11], which infers blocking rules from the traffic of suspicious nodes. The main idea is that the attack node first sends traffic that causes Autograph to mark it suspicious, then sends traffic similar to legitimate traffic, resulting in rules that cause denial of service.

Newsome, Karp, and Song [21] present attacks against Polygraph [20], a polymorphic virus detector that uses machine learning. They primarily focus on conjunction learners, presenting *Causative Integrity* attacks that exploit certain weaknesses not present in other learning algorithms (such as that used by SpamBayes). They also suggest a *correlated outlier attack*, which attacks a naive-Bayes-like learner by adding spurious features to positive training instances, causing the filter to block benign traffic with those features (a *Causative Availability* attack). They speculate briefly about applying such an attack to spam filters; however, several of their assumptions about the learner are not appropriate in the case of SpamBayes, such as that the learner uses only features indicative of the positive class. Furthermore, although they present insightful analysis, they do not evaluate the correlated outlier attack against a real system. Our attacks use similar ideas, but we develop and test them on a real system. We also explore the value of information to an attacker, and we present and test a defense against the attacks.

2.7 Conclusion

Above, we show that an adversary can effectively disable the SpamBayes spam filter with relatively little system state information and relatively limited control over training data. Using the framework presented in Section 2.3, we have demonstrated the effectiveness of *Causative* attacks against SpamBayes if the adversary is given realistic control over the training process of SpamBayes, by limiting the header fields the attacker can modify and the amount of control over the training set. Our Usenet-25k dictionary attack causes misclassification of 19% of ham messages with only 1% control over the training messages[3], rendering SpamBayes unusable. Our focused attack changes the classification of the target message virtually 100% of the time with knowledge of only 30% of the target's tokens. Our pseudospam attack is able to cause almost 90% of the target spam messages to be labeled as either *unsure* or *ham* with control of less than 10% of the training data.

We also demonstrate a defense against some attacks. We explore the RONI defense, which filters out dictionary attack messages with complete success based on how a message impacts the performance of our classifier. Focused attacks are especially difficult; defending against an attacker with extra knowledge of future events remains an open problem.

Our attacks and defense should also work for other spam filtering systems based on similar learning algorithms, such as BogoFilter, Thunderbird's spam filter and the Bayesian component of SpamAssassin, although their effects may vary (and SpamAssassin uses more components than just the learning algorithm, so the effect of our attacks may be smaller). These techniques may also be effective against other learning systems, such as those used for worm or intrusion detection.

2.A Appendix: Analysis of an Optimal Attack

In this appendix, we justify our claims in Section 2.3.1 about optimal attacks.

[3] While the dictionary attack messages are larger than most typical email, we have demonstrated the effect of these attacks and that one can reduce the attack message size by using a more informed dictionary.

2.A.1 Properties of the Spam Score

The key to understanding heuristic attacks and constructing optimal attacks on SpamBayes is characterizing conditions under which the SpamBayes score $I(\mathbf{m})$ increases when the training corpus is injected with attack spam messages.

Lemma A.1 *The* $I(\mathbf{m})$ *score defined in Eq.* (*4*) *is non-decreasing in* $\mathbf{f}_i$ *for all i.*

Proof. We show that the derivative of $I(\mathbf{m})$ with respect to $\mathbf{f}_k$ is non-negative. By rewriting, Eq. (3) as $S(\mathbf{m}) = 1 - \chi^2_{2n}\left(-2\log\Pi_{i:\delta(\mathbf{m})_i=1}[\mathbf{f}_i]\right)$, we can use the chain rule. Let $x(\mathbf{f}) = \Pi_{i:\delta(\mathbf{m})_i=1}[\mathbf{f}_i]$ and we have

$$\frac{d}{dx}\left[1-\chi^2_{2n}\left(-2\log(x)\right)\right] = \frac{1}{(n-1)!}\left(-\log(x)\right)^{n-1} \quad ,$$

which is non-negative since $0 \le x \le 1$. We have $\partial x(\mathbf{f})/\partial \mathbf{f}_k$ is $\Pi_{i\neq k:\delta(\mathbf{m})_i=1}[\mathbf{f}_i] \ge 0$ if $\delta(\mathbf{m})_k=1$ or *0* otherwise. Combining these derivatives, we have

$$\frac{\partial S(\mathbf{m})}{\partial \mathbf{f}_k} \ge 0 \quad .$$

By an analogous derivation, replacing $\mathbf{f}_i$ by *1* - $\mathbf{f}_i$, we have

$$\frac{\partial H(\mathbf{m})}{\partial \mathbf{f}_k} \le 0 \quad .$$

Finally, we obtain the result

$$\frac{\partial I(\mathbf{m})}{\partial \mathbf{f}_k} = \frac{1}{2}\frac{\partial S(\mathbf{m})}{\partial \mathbf{f}_k} - \frac{1}{2}\frac{\partial H(\mathbf{m})}{\partial \mathbf{f}_k} \ge 0 \quad .$$

Remark A.2 *Given a fixed number of attack spam messages,* $\mathbf{f}_j$ *is independent of the number of those messages containing the* i^{th} *token for all* $j \neq i$.

This remark follows from the fact that the inclusion of the i^{th} token in attack spams affects $N_S(i)$ and $N(i)$ but not $N_H(i)$, N_S, N_H, $N_S(j)$, $N_H(j)$, $N(j)$ for all $j \neq i$ (see Eq. (1) and Eq. (2) in Section 2.2.3).

After an attack of a fixed number of spam messages, the score $I(\mathbf{m})$ of an incoming test message **m** can be maximized by maximizing each $\mathbf{f}_i$ separately. This motivates our attacks—intuitively, we increase the $\mathbf{f}_i$ of tokens appearing in **m**.

2.A.2 Effect of Poisoning on Token Scores

We have not yet established how email spam scores change as the result of an attack message in the training set. One might assume that the i^{th} score $\mathbf{f}_i$ should increase when the i^{th} token is added to the attack email. This would be the case, in fact, if the token score in Eq. (1) were computed according to Bayes' Rule. However, the score in Eq. (1) is derived by applying Bayes' Rule with an additional assumption that the prior of spam and ham is equal. As a result we show that $\mathbf{f}_i$ can be smaller if the i^{th} token is included in the attack email.

We consider a single attack spam message, after which the counts become

$$\begin{aligned} N_S &\mapsto N_S + 1 \\ N_H &\mapsto N_H \\ N_S(i) &\mapsto \begin{cases} N_S(i) + 1 & \text{if } \mathbf{a}_i = 1 \\ N_S(i) & \text{otherwise} \end{cases} \\ N_H(i) &\mapsto N_H(i) \quad . \end{aligned}$$

Using these count transformations, we assess the effect on the smoothed SpamBayes score $\mathbf{f}_i$ of training on an attack spam message **a**. If the i^{th} token is included in the attack (*i.e.,* $\mathbf{a}_i = 1$), then the new score for the i^{th} token (from Eq. (1)) is

$$\mathbf{P}_{(S,i)}^{(1)} = \frac{N_H\left(N_S(i)+1\right)}{N_H\left(N_S(i)+1\right)+\left(N_S+1\right)N_H(i)} \quad .$$

If the token is not included in the attack (*i.e.,* $\mathbf{a}_i = 0$), then the new token score is

$$\mathbf{P}^{(0)}_{(S,i)} = \frac{N_H N_S(i)}{N_H N_S(i) + (N_S + 1) N_H(i)} \quad .$$

We use the notation $\mathbf{f}_i^{(1)}$ and $\mathbf{f}_i^{(0)}$ to denote the smoothed spam score after the attack depending on whether or not the ith token was used in the attack message.

We wish to analyze the quantity

$$\Delta\mathbf{f}_i = \mathbf{f}_i^{(1)} - \mathbf{f}_i^{(0)} \quad .$$

One might expect this difference to always be non-negative, but we show that in some scenarios $\Delta\mathbf{f}_i < 0$. After some algebra, we expand $\Delta\mathbf{f}_i$ as follows:

$$\begin{aligned}\Delta\mathbf{f}_i &= \frac{s}{(s + N(i) + 1)(s + N(i))}\left(\mathbf{P}^{(1)}_{(S,i)} - x\right) \\ &\quad + \frac{N_H N(i)}{(s + N(i))(N_H N_S(i) + (N_S + 1) N_H(i))}\mathbf{P}^{(1)}_{(H,i)} \\ &= \frac{1}{(s + N(i) + 1)(s + N(i))} \cdot \alpha(i) \quad ,\end{aligned}$$

where $\mathbf{P}^{(1)}_{(H,i)} = 1 - \mathbf{P}^{(1)}_{(S,i)}$ is the altered ham score of the ith token. The first factor in the above expression is positive and the second is defined as

$$\alpha(i) = s(1 - x) + P^{(1)}_{(H,i)} \frac{N_H N(i)(N(i) + 1) + s N_H N_H(i) - s(N_S + 1) N_H(i)}{N_H N_S(i) + (N_S + 1) N_H(i)} \quad .$$

When we combine these two terms, only the numerator can be negative. Focusing on conditions under which this can happen leads to the (weakest) conditions that $N_S(i) = 0$, $s > 0$, $N_H(i) > 0$ and

$$x > \frac{N_H (N_H(i) + s)(N_H(i) + 1)}{s(N_H(i)(N_S + 1) + N_H)} \quad .$$

If $N_S(i) > 0$, the resulting bound would be larger.

Under SpamBayes' default values of $s = 1$ and $x = 1/2$, we can rewrite the above condition as $N_S(i) = 0$, $N_H(i) > 0$, and

$$N_S > N_H \left[\frac{2\left(N_H(i)+1\right)^2 - 1}{N_H(i)} \right] - 1 \quad .$$

Since $N_H(i) \geq 1$, we have the weakest possible condition for $\Delta\mathbf{f}_i$ to be negative under default settings: $N_S \geq 7N_H$, $N_H(i) = 1$, and $N_S(i) = 0$. Thus, when the number of spam, N_S, in the training set is sufficiently larger than the number of ham, N_H, it is possible that the score of a token will be lower if it is *included* in the attack message than if it were excluded. This is a direct result of the assumption made by SpamBayes that $N_S = N_H$. Such aberrations will occur most readily in tokens with low initial values of $N_H(i)$ and $N_S(i)$.

Acknowledgments We would like to thank Satish Rao and Carla Brodley for their useful comments and suggestions on this research. This work was supported in part by the Team for Research in Ubiquitous Secure Technology (TRUST), which receives support from the National Science Foundation (NSF award #CCF-0424422), the Air Force Office of Scientific Research (AFOSR #FA9550-06-1-0244), Cisco, British Telecom, ESCHER, Hewlett-Packard, IBM, iCAST, Intel, Microsoft, ORNL, Pirelli, Qualcomm, Sun, Symantec, Telecom Italia, and United Technologies; in part by California state Microelectronics Innovation and Computer Research Opportunities grants (MICRO ID#06-148 and #07-012) and Siemens; and in part by the cyber-DEfense Technology Experimental Research laboratory (DETERlab), which receives support from the Department of Homeland Security Homeland Security Advanced Research Projects Agency (HSARPA award #022412) and AFOSR (#FA9550-07-1-0501). We also gratefully acknowledge the support of the NSF through grants DMS-0434383 and DMS-0707060. The opinions expressed here are solely those of the authors and do not necessarily reflect the opinions of any funding agency, the State of California, or the U.S. government.

References

[1]. Barreno M, Nelson B, Sears R, Joseph AD, Tygar JD (2006) Can machine learning be secure? In: Proceedings of the ACM Symposium on InformAtion, Computer, and Communications Security (ASIACCS), pp 16-25

[2]. Barreno M, Nelson B, Joseph AD, Tygar JD (2008) The security of machine learning. Tech. Rep. UCB/EECS-2008-43, EECS Department, University of California, Berkeley, URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-43.html

[3]. Chung SP, Mok AK (2006) Allergy attack against automatic signature generation. In: Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID), pp 61-80

[4]. Chung SP, Mok AK (2007) Advanced allergy attacks: Does a corpus really help? In: Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID), pp 236-255

[5]. Cormack G, Lynam T (2005) Spam corpus creation for TREC. In: Proceedings of the Conference on Email and Anti-Spam (CEAS)

[6]. Dalvi N, Domingos P, Mausam, Sanghai S, Verma D (2004) Adversarial classification. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 99-108
[7]. Fisher RA (1948) Question 14: Combining independent tests of significance. American Statistician 2(5):30-30J
[8]. Graham P (2002) A plan for spam. http://www.paulgraham.com/spam.html
[9]. Karlberger C, Bayler G, Kruegel C, Kirda E (2007) Exploiting redundancy in natural language to penetrate Bayesian spam filters. In: Proceedings of the USENIX Workshop on Offensive Technologies (WOOT), pp 1-7
[10]. Kearns M, Li M (1993) Learning in the presence of malicious errors. SIAM Journal on Computing 22(4):807-837
[11]. Kim HA, Karp B (2004) Autograph: Toward automated, distributed worm signature detection. In: Proceedings of the USENIX Security Symposium, pp 271-286
[12]. Klimt B, Yang Y (2004) Introducing the Enron corpus. In: Proceedings of the Conference on Email and Anti-Spam (CEAS)
[13]. Lazarevic A, Ertöz L, Kumar V, Ozgur A, Srivastava J (2003) A comparative study of anomaly detection schemes in network intrusion detection. In: Barbará D, Kamath C (eds) Proceedings of the SIAM International Conference on Data Mining, pp 25-36
[14]. Liao Y, Vemuri VR (2002) Using text categorization techniques for intrusion detection. In: Proceedings of the USENIX Security Symposium, pp 51-59
[15]. Lowd D, Meek C (2005) Adversarial learning. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 641-647
[16]. Lowd D, Meek C (2005) Good word attacks on statistical spam filters. In: Proceedings of the Conference on Email and Anti-Spam (CEAS)
[17]. Meyer T, Whateley B (2004) SpamBayes: Effective open-source, Bayesian based, email classification system. In: Proceedings of the Conference on Email and Anti-Spam (CEAS)
[18]. Mukkamala S, Janoski G, Sung A (2002) Intrusion detection using neural networks and support vector machines. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN), pp 1702-1707
[19]. Nelson B, Barreno M, Chi FJ, Joseph AD, Rubinstein BIP, Saini U, Sutton C, Tygar JD, Xia K (2008) Exploiting machine learning to subvert your spam filter. In: Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)
[20]. Newsome J, Karp B, Song D (2005) Polygraph: Automatically generating signatures for polymorphic worms. In: Proceedings of the IEEE Symposium on Security and Privacy, pp 226-241
[21]. Newsome J, Karp B, Song D (2006) Paragraph: Thwarting signature learning by training maliciously. In: Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID 2006), pp 81-105
[22]. Robinson G (2003) A statistical approach to the spam problem. Linux Journal
[23]. Shaoul C, Westbury C (2007) A USENET corpus (2005-2007)
[24]. Stolfo SJ, Li WJ, Hershkop S, Wang K, Hu CW, Nimeskern O (2004) Detecting viral propagations using email behavior profiles. ACM Transactions on Internet Technology (TOIT) pp 187-221
[25]. Wittel GL, Wu SF (2004) On attacking statistical spam filters. In: Proceedings of the Conference on Email and Anti-Spam (CEAS)