5 Availability Attack Case Study: SpamBayes

Adversaries can also execute attacks designed to degrade the classifier's ability to distinguish between allowed and disallowed events. These *Causative Availability* attacks against learning algorithms cause the resulting classifiers to have unacceptably high false-positive rates; i.e., a successfully poisoned classifier will misclassify benign input as potential attacks, creating an unacceptable level of interruption in legitimate activity. This chapter provides a case study of one such attack on the SpamBayes spam detection system. We show that cleverly crafted attack messages—pernicious spam email that an uninformed human user would likely identify and label as spam—can exploit SpamBayes' learning algorithm, causing the learned classifier to have an unreasonably high false-positive rate. (Chapter 6 demonstrates *Causative* attacks that instead result in classifiers with an unreasonably high false-negative rate—these are *Integrity* attacks.) We also show effective defenses against these attacks and discuss the tradeoffs required to defend against them.

We examine several attacks against the SpamBayes spam filter, each of which embodies a particular insight into the vulnerability of the underlying learning technique. In doing so, we more broadly demonstrate attacks that could affect any system that uses a similar learning algorithm. The attacks we present target the learning algorithm used by the spam filter SpamBayes (spambayes.sourceforge.net), but several other filters also use the same underlying learning algorithm, including BogoFilter (bogofilter.sourceforge.net), the spam filter in Mozilla's Thunderbird email client (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; i.e., the learning algorithm is fundamentally identical, but each filter uses a different set of features. We demonstrate the vulnerability of the underlying algorithm for SpamBayes because it uses a pure machine learning method, it is familiar to the academic community (Meyer & Whateley 2004), and it is popular with over 700,000 downloads. Although here we only analyze SpamBayes, the fact that these other systems use the same learning algorithm suggests that other filters are also vulnerable to similar attacks. However, the overall effectiveness of the attacks would depend on how each of the other filters incorporates the learned classifier into the final filtering decision. For instance, filters such as Apache SpamAssassin (Apa n.d.), only use learning as one of several components of a broader filtering engine (the others are handcrafted non-adapting rules), so attacks against it would degrade the performance of the filter, but perhaps their overall impact would be lessened or muted entirely. In principle,

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

though, it should be possible to replicate these results in these other filters. Finally, beyond spam filtering, we highlight the vulnerabilities in SpamBayes' learner because these same attacks could also be employed against similar learning algorithms in other domains. While the feasibility of these attacks, the attacker's motivation, or the contamination mechanism presented in this chapter may not be appropriate in other domains, it is nonetheless interesting to understand the vulnerability so that it can be similarly assessed for other applications.

We organize our approach to studying the vulnerability of SpamBayes' learning algorithm based on the framework discussed in Chapter 3. Primarily, we investigated *Causative Availability* attacks on the filter because this type of attack was an interesting new facet that could actually be deployed in real-world settings. Here the adversary has an additive contamination capability (i.e., the adversary has exclusive control of some subset of the user's training data), but is limited to only altering the positive (spam) class; we deemed this contamination model to be the most appropriate for a crafty spammer. Novel contributions of our research include a set of successful principled attacks against SpamBayes, an empirical study validating the effectiveness of the attacks in a realistic setting, and a principled defense that empirically succeeds against several of the attacks. We finally discuss the implications of the attack and defense strategies and the role that attacker information plays in the effectiveness of these attacks.

In this chapter, we discuss the background of the training model (see Section 5.1); we present three new attacks on SpamBayes (see Section 5.3); we give experimental results (see Section 5.5); and we present a defense against these attacks together with further experimental results (see Section 5.4). This chapter builds on the work of Nelson et al. (2008, 2009).

5.1 The SpamBayes Spam Filter

SpamBayes is a content-based statistical spam filter that classifies email using token counts in a model proposed by Robinson (2003) as inspired by Graham (2002). Meyer & Whateley (2004) describe the system in detail. SpamBayes computes a spam score for each token in the training corpus based on its occurrence in spam and non-spam emails; this score is motivated as a smoothed estimate of the posterior probability that an email containing that token is spam. The filter computes a message's overall spam score based on the assumption that the token scores are independent, and then it applies Fisher's method (cf. Fisher 1948) for combining significance tests to determine whether the email's tokens are sufficiently indicative of one class or the other. The message score is compared against two thresholds to select the label *spam*, *ham* (i.e., non-spam), or *unsure*. In the remainder of this section, we detail the statistical method SpamBayes uses to estimate and aggregate token scores.

5.1.1 SpamBayes' Training Algorithm

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

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

SpamBayes was designed by Robinson (2003) and Meyer & Whateley (2004), based on ideas by Graham (2002), together with Fisher's method for combining independent significance tests (Fisher 1948). Intuitively, SpamBayes learns how strongly each token indicates *ham* or *spam* by counting the number of each type of email in which the token appears. When classifying a new email, SpamBayes considers all the message's tokens as evidence of whether the message is spam or ham 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 X based on words, URL components, header elements, and other character sequences that appear in X. Each is treated as a unique token of the email independent of their order within the message, but for convenience, we place an ordering on the tokens so that each unique token has a fixed position i among the entire alphabet of tokens. Further, SpamBayes only records whether or not a token occurs in the message, not how many times it occurs. Email X is thus represented as a binary (potentially infinite length) vector $\mathbf{x}$ where

$$x_i = \begin{cases} 1, & \text{if the } i^{\text{th}} \text{ token occurs in } X \\ 0, & \text{otherwise} \end{cases}.$$

This message vector representation records which tokens occur in the message independent of their order or multiplicity.

The training data used by SpamBayes is a dataset of message vector (representing each training message) and label pairs: $\mathbb{D}^{(\text{train})} = \left\{ \left(\mathbf{x}^{(1)}, y^{(1)}\right), \left(\mathbf{x}^{(2)}, y^{(2)}\right), \ldots, \left(\mathbf{x}^{(N)}, y^{(N)}\right) \right\}$ where $\mathbf{x}^{(i)} \in \{0, 1\}^D$ and $y^{(i)} \in \{ham, spam\}$. As in Section 2.2.1, this training data can be represented as a training matrix $\mathbf{X} = \left[\mathbf{x}^{(1)}\, \mathbf{x}^{(2)}\, \ldots\, \mathbf{x}^{(N)}\right]^\top \in \{0, 1\}^{N \times D}$ along with its label vector $\mathbf{y} = \left[y^{(1)}\, y^{(2)}\, \ldots\, y^{(N)}\right] \in \{0, 1\}^N$, using 1 to represent *spam* and 0 for *ham*. Using the training matrix, the token-counting statistics used by SpamBayes can be expressed as

$$\mathbf{n}^{(s)} \triangleq \mathbf{X}^\top \mathbf{y} \qquad \mathbf{n}^{(h)} \triangleq \mathbf{X}^\top (\mathbf{1} - \mathbf{y}) \qquad \mathbf{n} \triangleq \mathbf{n}^{(s)} + \mathbf{n}^{(h)}$$

which are vectors containing the cumulative token counts for each token in all, spam, and ham messages, respectively. We also define $N^{(s)} \triangleq \mathbf{y}^\top \mathbf{y}$ as the total number of training spam messages and $N^{(h)} \triangleq (\mathbf{1} - \mathbf{y})^\top (\mathbf{1} - \mathbf{y})$ as the total number of training ham messages (and, of course, $N = N^{(s)} + N^{(h)}$).

From these count statistics, SpamBayes computes a spam score for the i^{th} token by estimating the posterior $\Pr(X \text{ is spam} | x_i = 1)$. First, the likelihoods $\Pr(x_i = 1 | X \text{ is spam})$ and $\Pr(x_i = 1 | X \text{ is ham})$ for observing the i^{th} token in a spam/ham message are estimated using the maximum likelihood estimators yielding the likelihood vectors $L_i^{(s)} = \frac{1}{N^{(s)}} \cdot \mathbf{n}^{(s)}$ and $L_i^{(h)} = \frac{1}{N^{(h)}} \cdot \mathbf{n}^{(h)}$.

Second, using the likelihood estimates $\mathbf{L}^{(s)}$ and $\mathbf{L}^{(h)}$ and an estimate $\pi^{(s)}$ on the prior distribution $\Pr(X \text{ is spam})$, Bayes' Rule is used to estimate the posteriors as $\mathbf{P}^{(s)} \propto \frac{\pi^{(s)}}{N^{(s)}} \cdot \mathbf{n}^{(s)}$ and $\mathbf{P}^{(h)} \propto \frac{1-\pi^{(s)}}{N^{(h)}} \cdot \mathbf{n}^{(h)}$ along with the constraints $\mathbf{P}^{(s)} + \mathbf{P}^{(h)} = \mathbf{1}$. However, instead of using the usual naive Bayes maximum likelihood prior estimator $\pi^{(s)} = \frac{N^{(s)}}{N^{(s)} + N^{(h)}}$, SpamBayes uses the agnostic prior distribution $\pi^{(s)} = \frac{1}{2}$, a choice that gives the learner

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

unusual properties that we further discuss in Appendix C.2.1. Based on this choice of prior, SpamBayes then computes a *spam score vector* $\mathbf{P}^{(s)}$ specified for the i^{th} token as

$$P_i^{(s)} = \frac{N^{(h)} n_i^{(s)}}{N^{(h)} n_i^{(s)} + N^{(s)} n_i^{(h)}}; \tag{5.1}$$

i.e., this score is an estimator of the posterior $\Pr(X \text{ is spam} | x_i = 1)$. An analogous *token ham score* is given by $\mathbf{P}^{(h)} = \mathbf{1} - \mathbf{P}^{(s)}$.

Robinson's method (Robinson 2003) smooths $P_i^{(s)}$ through a convex combination with a prior distribution 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 the strength of the prior with a default of $s = 1$), respectively:

$$q_i = \frac{s}{s + n_i} x + \frac{n_i}{s + n_i} P_i^{(s)}. \tag{5.2}$$

Smoothing mitigates overfitting for rare tokens. For instance, if the token "floccinaucinihilipilification" appears once in a spam and never in a ham in the training set, the posterior estimate would be $P_i^{(s)} = 1$, which would make any future occurrence of this word dominate the overall spam score. However, occurrence of this word only in spam may have only been an artifact of its overall rarity. In this case, smoothing is done by adding a prior distribution that the posterior for every token is $x = \frac{1}{2}$ (i.e., an agnostic score). For rare tokens, the posterior estimate is dominated by this prior. However, when a token is more frequently observed, its smoothed score approaches the empirical estimate of the posterior in Equation (5.1) according to the strength given to the prior by s. An analogous smoothed ham score is given by $1 - \mathbf{q}$.

5.1.2 SpamBayes' Predictions

After training, the filter computes the overall spam score $I(\hat{\mathbf{x}})$ of a new message $\hat{X}$ using Fisher's method (Fisher 1948) for combining the scores of the tokens observed in $\hat{X}$. SpamBayes uses at most 150 tokens from $\hat{X}$ with scores furthest from 0.5 and outside the interval (0.4, 0.6) (see Appendix C.2.2 for more details). Let $\mathbb{T}_{\hat{\mathbf{x}}}$ be the set of tokens that SpamBayes incorporates into its spam score, and let $\boldsymbol{\delta}(\hat{\mathbf{x}})$ be the indicator function for this set. The token spam scores are combined into a *message spam score* for $\hat{X}$ by

$$S(\hat{\mathbf{x}}) = 1 - \chi^2_{2\tau_{\hat{\mathbf{x}}}}\left(-2(\log \mathbf{q})^\top \boldsymbol{\delta}(\hat{\mathbf{x}})\right), \tag{5.3}$$

where $\tau_{\hat{\mathbf{x}}} \triangleq |\mathbb{T}_{\hat{\mathbf{x}}}|$ is the number of tokens from $\hat{X}$ used by SpamBayes and $\chi^2_{2\tau_{\hat{\mathbf{x}}}}(\cdot)$ denotes the cumulative distribution function of the chi-square distribution with $2\tau_{\hat{\mathbf{x}}}$ degrees of freedom. A ham score $H(\hat{\mathbf{x}})$ is similarly defined by replacing $\mathbf{q}$ with $1 - \mathbf{q}$ in Equation (5.3). Finally, SpamBayes constructs an overall spam score for $\hat{X}$ by averaging $S(\hat{\mathbf{x}})$ and $1 - H(\hat{\mathbf{x}})$ (both being indicators of whether $\hat{X}$ is spam), giving the final score

$$I(\hat{\mathbf{x}}) = \tfrac{1}{2}\left(S(\hat{\mathbf{x}}) + 1 - H(\hat{\mathbf{x}})\right) \tag{5.4}$$

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

for a message: a quantity between 0 (strong evidence of ham) and 1 (strong evidence of spam). SpamBayes predicts by thresholding $I(\hat{\mathbf{x}})$ against two user-tunable thresholds $\theta^{(h)}$ and $\theta^{(s)}$, with defaults $\theta^{(h)} = 0.15$ and $\theta^{(s)} = 0.9$. SpamBayes predicts *ham*, *unsure*, or *spam* if $I(\hat{\mathbf{x}})$ falls into the interval $[0, \theta^{(h)}]$, $(\theta^{(h)}, \theta^{(s)}]$, or $(\theta^{(s)}, 1]$, 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 5.2.3, *ham-as-unsure* misclassifications are nearly as bad for the user as *ham-as-spam*.

5.1.3 SpamBayes' Model

Although the components of the SpamBayes algorithm (token spam scores, smoothing, and chi-squared test) were separately motivated, the resulting system can be described by a unified probability model for discriminating ham from spam messages. While Robinson motivates the SpamBayes classifier as a smoothed estimator of the posterior probability of spam, he never explicitly specifies the probabilistic model. We specify a discriminative model and show that the resulting estimation can be re-derived using empirical risk minimization. Doing so provides a better understanding of the modeling assumptions of the SpamBayes classifier and its vulnerabilities.

In this model, there are three random variables of interest: the spam label y_i of the i^{th} message, the indicator variable $X_{i,j}$ of the j^{th} token in the i^{th} message, and the token score q_j of the j^{th} token. We use the convention that a label is 1 to indicate *spam* or 0 to indicate *ham*. In the discriminative setting, given $\mathbf{X}_{i,\bullet}$ as a representation of the tokens in the i^{th} message and the token scores $\mathbf{q}$, the message's label y_i is conditionally independent of all other random variables in the model. The conditional probability of the message label given the occurrence of a single token $X_{i,j}$ is specified by

$$\Pr\left(y_i | X_{i,j}, q_j\right) = \left((q_j)^{y_i} \cdot (1 - q_j)^{1-y_i}\right)^{X_{i,j}} \left(\tfrac{1}{2}\right)^{1-X_{i,j}}, \tag{5.5}$$

i.e., in the SpamBayes model, each token that occurs in the message is an indicator of its label, whereas tokens absent from the message have no impact on its label. Because SpamBayes' scores only incorporate tokens that occur in the message, traditional generative spam models (e.g., Figure 5.1(b)) are awkward to construct, but the above discriminative conditional probability captures this modeling nuance. Further, there is no prior distribution for the token indicators $X_{i,j}$ but there is a prior on the token scores. Treating these as binomial parameters, each has a beta prior with common parameters α and β, giving them a conditional probability of

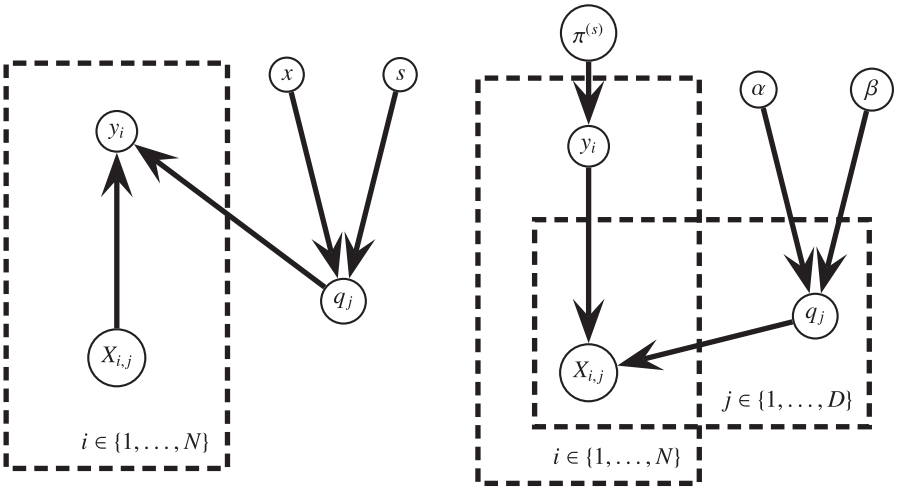
$$\Pr\left(q_j | \alpha, \beta\right) = \tfrac{1}{\mathrm{B}(\alpha,\beta)} \cdot (q_j)^{\alpha-1} \cdot (1 - q_j)^{\beta-1}, \tag{5.6}$$

where $\mathrm{B}(\alpha, \beta)$ is the beta function (see the Glossary). As earlier mentioned, Robinson instead used an equivalent parameterization with a strength parameter s and prior

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

parameter x for which $\alpha = s \cdot x + 1$ and $\beta = s\,(1 - x) + 1$. Using this parameterization, x specifies the mode of the prior distribution distribution. In SpamBayes, these parameters are fixed a priori rather than treated as random hyper-parameters. Their default values are $\pi^{(s)} = \frac{1}{2}$, $x = \frac{1}{2}$, and $s = 1$.

Together, the label's probability conditioned on the j^{th} token and the prior distribution on the j^{th} token score are used to derive a spam score for the message (based only on the j^{th} token). However, unlike a maximum likelihood derivation, SpamBayes' parameter estimation for q_j is not based on a joint probability model over all tokens. Instead, the score for each token is computed separately by maximizing the labels' likelihood within a per-token model as depicted in Figure 5.1(a); i.e., the model depicts a sequence of labels based solely on the presence of the j^{th} token. Based on the independence assumption of Figure 5.1(a), the conditional distributions of Equation (5.5) combine together to make the following joint log probability based on the j^{th} token (for



(a) SpamBayes' Discriminative Model for the j^{th} token

(b) Traditional Generative Model

Figure 5.1 Probabilistic graphical models for spam detection. **(a)** A probabilistic model that depicts the dependency structure between random variables in SpamBayes for a *single* token (SpamBayes models each token as a separate indicator of ham/spam and then combines them together assuming each is an independent test). In this model, the label y_i for the i^{th} email depends on the token score q_j for the j^{th} token if it occurs in the message; i.e., $X_{i,j} = 1$. The parameters s and x parameterize a beta prior distribution on q_j. **(b)** A more traditional generative model for spam. The parameters $\pi^{(s)}$, α, and β parameterize the prior distributions for y_i and q_j. Each label y_i for the i^{th} email is drawn independently from a Bernoulli distribution with $\pi^{(s)}$as the probability of *spam*. Each token score for the j^{th} token is drawn independently from a beta distribution with parameters α and β. Finally, given the label for a message and the token scores, $X_{i,j}$ is drawn independently from a Bernoulli. Based on the likelihood function for this model, the token scores q_j computed by SpamBayes can be viewed simply as the maximum likelihood estimators for the corresponding parameter in the model.

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

N messages):

$$\begin{aligned}\log \Pr\left(\mathbf{y}, \mathbf{X}_{\bullet,j} | \alpha, \beta\right) &= \log \Pr\left(q_j | \alpha, \beta\right) + \sum_{i=1}^{N} \log \Pr\left(y_i | X_{i,j}, q_j\right) \\ &= -\log\left(\mathrm{B}\left(\alpha, \beta\right)\right) + (\alpha - 1)\log\left(q_j\right) + (\beta - 1)\log\left(1 - q_j\right) \\ &\quad + \sum_{i=1}^{N}\left[y_i X_{i,j}\log\left(q_j\right) + (1 - y_i)X_{i,j}\log\left(1 - q_j\right)\right]\end{aligned}$$

Maximizing this joint distribution (nearly) achieves the token scores specified by SpamBayes. To solve for the maximum, differentiate the joint probability with respect to the j^{th} token score, q_j, and set the derivative equal to 0. This yields

$$\begin{aligned}q_j &= \frac{\sum_{i=1}^{N} y_i X_{i,j} + \alpha - 1}{\sum_{i=1}^{N} X_{i,j} + \alpha - 1 + \beta - 1} \\ &= \frac{\alpha - 1}{n_j + \alpha - 1 + \beta - 1} + \frac{n_j^{(s)}}{n_j + \alpha - 1 + \beta - 1},\end{aligned}$$

where the summations in the first equation are simplified to token counts based on the definitions of y_i and $X_{i,j}$. Using the equivalent beta parameterization with x and s and the usual posterior token score $P_i^{(s)} = \frac{n_i^{(s)}}{n_i^{(s)} + n_i^{(h)}}$ (which differs from the SpamBayes' token score used in Equation (5.1) unless $N^{(s)} = N^{(h)}$), this equation for the maximum-likelihood estimator of q_j is equivalent to the SpamBayes' estimator in Equation (5.2).

The above per-token optimizations can also be viewed as a joint maximization procedure by considering the overall spam and ham scores $S(\cdot)$ and $H(\cdot)$ for the messages in the training set (see Equation 5.3). These overall scores are based on Fisher's method for combining independent p-values and assume that each token score is independent. In fact, $S(\cdot)$ and $H(\cdot)$ are tests for the aggregated scores $s_{\mathbf{q}}(\cdot)$ and $h_{\mathbf{q}}(\cdot)$ defined by Equations (C.1) and (C.2)—tests that monotonically increase with $s_{\mathbf{q}}(\cdot)$ and $h_{\mathbf{q}}(\cdot)$, respectively. Thus, from the overall spam score $I(\cdot)$ defined by Equation (5.4), maximizing $s_{\mathbf{q}}(\cdot)$ for all spam and $h_{\mathbf{q}}(\cdot)$ for all ham is a surrogate for minimizing the prediction error of $I(\cdot)$; i.e., minimizing some loss for $I(\cdot)$. Hence, combining the individual tokens' conditional distributions (Equation 5.5) together to form

$$Q\left(y_i, \mathbf{X}_{i,\bullet}, \mathbf{q}\right) = -\log \prod_{j=1}^{D}\left((q_j)^{y_i} \cdot (1 - q_j)^{1 - y_i}\right)^{X_{i,j}},$$

can be viewed as the loss function for the score $I(\cdot)$, and the sum of the negative logarithm of the token score priors given by Equation 5.6 can be viewed as its regularizer.[1] Moreover, minimizing this *regularized empirical loss* again yields the

[1] This interpretation ignores the censoring function $\mathbb{T}$ in which SpamBayes only uses the scores of the most informative tokens when computing $I(\cdot)$ for a message. As discussed in Appendix C.1 this censoring action makes $I(\cdot)$ non-monotonic in the token scores q_j. Computing the token scores without considering $\mathbb{T}$ can be viewed as a tractable relaxation of the true objective.

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

SpamBayes' token scores from Equation (5.2). In this way, SpamBayes can be viewed as a regularized empirical risk minimization technique.

Unfortunately, the loss function Q is not a negative log-likelihood because the product of the scores is unnormalized. When the proper normalizer is added to Q, the resulting parameter estimates for q_j no longer are equivalent to SpamBayes' estimators. In fact, SpamBayes' parameter estimation procedure and its subsequent prediction rule do not appear to be compatible with a traditional joint probability distribution over all labels, tokens, and scores (or at least we were unable to derive a joint probability model that would yield these estimates). Nonetheless, through the loss function Q, SpamBayes can be viewed as a regularized empirical risk minimization procedure as discussed in Section 2.2.

By analyzing this model of SpamBayes, we now identify its potential vulnerabilities. First, by incorporating a prior distribution on the token scores for smoothing, Robinson prevented a simple attack. Without any smoothing on the token scores, all tokens that only appear in ham would have token scores of 0. Since the overall score $I(\cdot)$ is computed with products of the individual token scores, including any of these ham-only tokens would cause spam to be misclassified as *ham* (and vice versa for spam-only tokens), which the adversary could clearly exploit. Similarly, using the censor function $\mathbb{T}$ helps prevent attacks in which the adversary pads a spam with many *hammy* tokens to negate the effect of *spammy* tokens. However, despite these design considerations, SpamBayes is still vulnerable to attacks. The first vulnerability of SpamBayes comes from its assumption that the data and tokens are independent, for which each token score is estimated based solely on the presence of that token in ham and spam messages. The second vulnerability comes from its assumption that only tokens that occur in a message contribute to its label. While there is some intuition behind this assumption, in this model, it causes rare tokens to have little support so that their scores can be easily changed. Ultimately, these two vulnerabilities lead to a family of attacks that we call dictionary attacks that we present and evaluate in the rest of this chapter.

5.2 Threat Model for SpamBayes

In analyzing the vulnerabilities of SpamBayes, we were motivated by the taxonomy of attacks (see Section 3.3). Known real-world attacks that spammers use against deployed spam filters tend to be *Exploratory Integrity* attacks: either the spammer obfuscates the especially spam-like content of a spam email, or it includes content not indicative of spam. Both tactics aim to get the modified message into the victim's inbox. This category of attack has been studied in detail in the literature (e.g., see Lowd & Meek 2005*a*, 2005*b*, Wittel & Wu 2004; Dalvi et al. 2004). However, in this chapter we investigate the compelling threat of *Causative* attacks against spam filters, which are unique to machine learning systems and potentially more harmful since they alter the filter.

In particular, a *Causative Availability* attack can create a powerful denial of service. For example, if a spammer causes enough legitimate messages to be filtered by the user's spam filter, the user is likely to disable the filter and therefore see the spammer's

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

unwanted messages. Alternatively, an unscrupulous business owner may wish to use spam filter denial of service to prevent a competitor from receiving email orders from potential customers. In this chapter, we present two novel *Causative Availability* attacks against SpamBayes: the dictionary attack is *Indiscriminate* and the focused attack is *Targeted*.

5.2.1 Attacker Goals

We consider an attacker with one of two goals: expose the victim to an advertisement or prevent the victim from seeing a legitimate message. The motivation for the first objective is obviously the potential revenue gain for the spammer if its marketing campaign is widely viewed. For the second objective, 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 wants to prevent competing bids from reaching the intended recipient and so gain a competitive advantage. An unscrupulous company can achieve this by causing its competitors' messages to be filtered as spam.

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 Cause a *particular* spam to be delivered to the victim's inbox.
4 Cause *any* spam to be delivered into the victim's inbox.

These objectives are used to construct the attacks described next.

5.2.2 Attacker Knowledge

The knowledge the attacker has about a user's messages may vary in different scenarios and thus lead to different attack strategies. 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 (see Section 5.3.1.1). 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* (see Section 5.3.1.2). Furthermore, if an attacker can send email messages that the user will train

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

as non-spam, a pseudospam attack can cause the filter to accept spam messages into the user's inbox (see Section 5.3.2).

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. The 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 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 does not train on messages that have a large negative impact. We show that this defense is effective in preventing some attacks from succeeding.

5.2.3 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 is not confident one way or the other, the classifier returns *unsure*. We use the following 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, users must sift through all spam periodically or risk missing legitimate messages. If they remain in the inbox, users will encounter an increased amount of spam messages in the inbox. If they have their own "Unsure" folder, they 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 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 the 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. These attacks are not limited to any particular retraining process; they only require the following assumption about the attacker's control of data.

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

5.2.4 The Contamination Assumption

We assume that the attacker can send emails that the victim will use for training—the *contamination assumption*. 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. Since the attacker has limited control of the training data or a portion of it, our contamination assumption is reasonable, but we incorporate two significant restrictions: 1) the attacker may specify arbitrary email bodies, but cannot alter email headers; and 2) attack emails will always be trained as spam, not ham. 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 these 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. In this pseudospam attack, we remove the second restriction on the attacker's abilities and explore the case where attack emails are trained as ham (see Section 5.3.2).

5.3 Causative Attacks against SpamBayes' Learner

We present three novel *Causative* attacks against SpamBayes' learning algorithm in the context of the attack taxonomy from Section 5.2.1: an *Indiscriminate Availability* attack, a *Targeted Availability* attack, and a *Targeted Integrity* attack. These attacks are generally structured according to the following steps:

1 The attacker determines its 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 contaminated filter.
4 The filter's classification performance degrades on incoming messages in accordance with the attacker's goal.

In the remainder of this section, we describe attacks that achieve the objectives outlined earlier in Section 5.2. Each of the attacks consists of inserting emails into the training set that are drawn from a particular distribution (i.e., according to the attacker's knowledge discussed in Section 5.2.2). The properties of these distributions, along with other parameters, determine the nature of the attack. The *dictionary* attack sends email

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

messages with tokens drawn from a broad distribution, essentially including every token with equal probability. The *focused* attack focuses the distribution specifically on a single 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.

5.3.1 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, too many false positives force the victim to disable the filter or frequently search in *spam*/*unsure* folders for legitimate messages that have been erroneously filtered away. Hence, the victim is forced to view more spam. In a *Targeted* attack, the attack is not designed to disable the filter, but instead it 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 $\hat{X}$. The attacker also has some (perhaps limited) knowledge of the next email the victim will receive. This knowledge can be represented as a distribution $\mathbf{p}$—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 $\mathbf{a}$ to maximize the *expected spam score:*

$$\max_{\mathbf{a}} \mathrm{E}_{\hat{\mathbf{x}}\sim\mathbf{p}} \left[I_{\mathbf{a}} \left(\hat{\mathbf{x}} \right) \right] ; \tag{5.7}$$

that is, the attack's goal is to maximize the expectation of $I_{\mathbf{a}} (\hat{\mathbf{x}})$ (Equation (5.4) with the attack message $\mathbf{a}$ added to the spam training set of the next legitimate email $\hat{\mathbf{x}}$ drawn from distribution $\mathbf{p}$. However, in analyzing this objective, it is shown in Appendix C.2 that the attacker can generally maximize the expected spam score of any future message by including *all possible tokens* (words, symbols, misspellings, etc.) in attack emails, causing SpamBayes to learn that all tokens are indicative of spam—we call this an *Optimal* attack.[2]

To describe the optimal attack under this criterion, we make two observations, which we detail in Appendix C.2. First, for most tokens, $I_{\mathbf{a}} (\cdot)$ is monotonically nondecreasing in q_i. Therefore, increasing the score of any token in the attack message will generally increase $I_{\mathbf{a}} (\hat{\mathbf{x}})$. 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 q_j of some different token $j \neq i$. Hence, the attacker can simply choose which tokens will be most beneficial for its 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.

[2] As discussed in Appendix C.2 these attacks are optimal for a relaxed version of the optimization problem. Generally, optimizing the problem given by Equation 5.7 requires exact knowledge about future messages $\hat{\mathbf{x}}$ and is a difficult combinatorial problem to solve.

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

For this, let us consider specific choices for the distribution $\mathbf{p}$. First, if the attacker has little knowledge about the tokens in target emails, we give equal probability to each token in $\mathbf{p}$. In this case, one 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 practice, the optimal attack requires intractably large attack messages, but the attacker can exploit its knowledge about the victim (captured by $\mathbf{p}$) to approximate the effect of an optimal attack by instead using a large set of common words that the victim is likely to use in the future such as a dictionary—hence these are *dictionary attacks*. 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* (see Section 5.3.1.1). 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* (see Section 5.3.1.2). 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 $\mathbf{p}$ 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.

5.3.1.1 Dictionary Attack

The *dictionary* attack, an *Indiscriminate* attack, makes the spam filter unusable by causing it to misclassify a significant portion of ham emails (i.e., causing false positives) so that the victim loses confidence in the filter. As a consequence either the victim disables the spam filter, or at least must frequently search through *spam*/*unsure* folders to find legitimate messages that were incorrectly classified. 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 views desired spams while searching for legitimate mail. The result of this attack is denial of service; i.e., a higher rate of ham misclassified as *spam*.

The dictionary attack is an approximation of the optimal attack suggested in Section 5.3.1, 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 by the set of all tokens that the victim is likely to use such as a dictionary of the victim's native language—we call this a *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 $I_{\mathbf{a}}(\hat{\mathbf{x}})$ as in

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

Equation (5.7). Since the score of a token typically increases when included in an attack message (except in unusual circumstances as described in Appendix C), the attacker can simply include any tokens that are likely to occur in future legitimate messages according to the attacker's knowledge from the distribution **p**. In particular, if the victim's language is known by the attacker, it can use that language's entire lexicon (or at least a large subset of it) as the attack dictionary. After training on a set of dictionary messages, 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 of this attack instead uses a token source with a distribution closer to the victim's true 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 tradeoff in choosing tokens. Rare tokens are the most vulnerable to attack since their scores will shift more toward spam (a spam score of 1.0 given by the score in Equation (5.4)) with fewer attack emails. However, the rare vulnerable tokens also are less likely to appear in future messages, diluting their usefulness. Thus the attack must balance these effects in selecting a set of tokens for the attack messages.

In our experiments (Section 5.5.2), we evaluate two variants of the dictionary attacks: the first is based on the *Aspell* dictionary and the second on a dictionary compiled from the most common tokens observed in a *Usenet* corpus. We refer to these as the *Aspell* and *Usenet* dictionary attacks, respectively.

5.3.1.2 Focused Attack

The second *Causative Availability* attack is a *Targeted* attack—the attacker has some knowledge of a specific legitimate email it targets to be incorrectly filtered. If the attacker has exact knowledge of the target email, placing all of its tokens in attack emails produces an optimal targeted attack. Realistically, though, the attacker only 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 attacker constructs attack email that contain words likely to occur in the target email; i.e., the tokens known by the attacker. The attack email may also include additional tokens added by the attacker to obfuscate the attack message's intent since extraneous tokens do not influence the attack's effect on the targeted tokens. When SpamBayes trains on the resulting attack email, the spam scores of the targeted tokens generally increase (see Appendix C), so the target message is more likely to be filtered as *spam*. This is the focused attack.

For example, an unscrupulous company may wish to prevent its competitors from receiving email about a competitive bidding process, and it knows specific words that will appear in the target email, obviating the need to include the entire dictionary in their attacks. It attacks by sending spam emails to the victim with tokens such as the names of competing companies, their products, and their employees. In addition, if the bid messages follow a common template known to the malicious company, this further

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

facilitates the attack. As a result of the attack, legitimate bid emails may be filtered away as *spam*, causing the victim not to see them.

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 nonword tokens common in the victim's email that are otherwise uncommon in typical English text.

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 to *reduce* the spam score of tokens not in the attack. To understand why, recall the estimate of the token posterior in Equation (5.1), and suppose that the j^{th} token does not occur in the attack email. Then $N^{(s)}$ increases with the addition of the attack email but $n_j^{(s)}$ does not, so $P_j^{(S)}$ decreases and therefore so does q_j. In Section 5.5.3, we observe empirically that the focused attack can indeed reduce the spam score of tokens not included in the attack emails.

5.3.2 Causative Integrity Attacks—Pseudospam

We also study *Causative Integrity* attacks, which manipulate the filter's training data to increase the number of 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, noncommercial 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.

5.4 The Reject on Negative Impact (RONI) Defense

Saini (2008) studied two defense strategies for countering *Causative Availability* attacks on SpamBayes. The first was a mechanism to adapt SpamBayes' threshold

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

parameters to mitigate the impact of an *Availability* attack called the threshold defense. This defense did reduce the false-positive rate of *dictionary* attacks but at a cost of a higher false-negative rate. He also discussed a preliminary version of the reject on negative impact defense, which we describe here and evaluate in detail.

In Section 3.5.4.1, we summarized the reject on negative impact defense. As stated in that section, (RONI) is a defense against *Causative* attacks, which measures the empirical effect that each training instance has when training a classifier with it, identifies all instances that had a substantial negative impact on that classifier's accuracy, and removes the offending instances from the training set, $\mathbb{D}^{(\text{train})}$, before training the final classifier. To determine whether a candidate training instance is deemed to be deleterious or not, the defender trains a classifier on a base training set, then adds the candidate instance to that training set, and trains a second classifier with the candidate instance included. The defender applies both classifiers to a *quiz set* of instances with known labels, measuring the difference in accuracy between the two. If adding the candidate instance to the training set causes the second classifier to produce substantially more classification errors than were produced by the first classifier trained without it, that candidate instance is rejected from the training set due to its detrimental effect.

More formally, we assume there is an initial training set $\mathbb{D}^{(\text{train})}$ and a set $\mathbb{D}^{(\text{suspect})}$ of additional candidate training points to be added to the training set. The points in $\mathbb{D}^{(\text{suspect})}$ are assessed as follows: first a *calibration set* $\mathbb{C}$, which is a randomly chosen subset of $\mathbb{D}^{(\text{train})}$, is set aside. Then several independent and potentially overlapping training/quiz set pairs $(\mathbb{T}_i, \mathbb{Q}_i)$ are sampled from the remaining portion of $\mathbb{D}^{(\text{train})}$, where the points within a pair of sets are sampled without replacement. To assess the impact (empirical effect) of a data point $(x, y) \in \mathbb{D}^{(\text{suspect})}$, for each pair of sets $(\mathbb{T}_i, \mathbb{Q}_i)$ one constructs a *before* classifier f_i trained on $\mathbb{T}_i$ and an *after* classifier $\hat{f}_i$ trained on $\mathbb{T}_i + (x, y)$; i.e., the sampled training set with (x, y) concatenated. The reject on negative impact defense then compares the classification accuracy of f_i and $\hat{f}_i$ on the quiz set $\mathbb{Q}_i$, using the change in true positives and true negatives caused by adding (x, y) to $\mathbb{T}_i$. If either change is significantly negative when averaged over training/quiz set pairs, (x, y) is considered to be too detrimental, and it is excluded from $\mathbb{D}^{(\text{train})}$. To determine the significance of a change, the shift in accuracy of the detector is compared to the average shift caused by points in the calibration set $\mathbb{C}$. Each point in $\mathbb{C}$ is evaluated in a way analogous to evaluation of the points in $\mathbb{D}^{(\text{suspect})}$. The median and standard deviation of their true positive and true negative changes are computed, and the significance threshold is chosen to be the third standard deviation below the median.

5.5 Experiments with SpamBayes

5.5.1 Experimental Method

Here we present an empirical evaluation of the impact of *Causative Availability* attacks on SpamBayes' spam classification accuracy.

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

5.5.1.1 Datasets

In these experiments, we use the Text Retrieval Conference (TREC) 2005 spam corpus as described by Cormack & Lynam (2005), which is based on the Enron email corpus (Klimt & Yang 2004) and contains 92,189 emails (52,790 spam and 39,399 ham). By sampling from this dataset, we construct sample inboxes and measure the effect of injecting attacks into them. 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 the attacks. This corpus is a subset of a Usenet corpus of 140,179 postings compiled by the University of Alberta's Westbury Lab (Shaoul & Westbury 2007). 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.

5.5.1.2 Constructing Message Sets for Experiments

In constructing an experiment, we often require several nonrepeating sequences of emails in the form of mailboxes. When we require a mailbox, we sample messages without replacement from the TREC corpus, stratifying the sampling to ensure the necessary proportions of ham and spam. For subsequent messages needed in any part of the experiment (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 within the experiment.

We construct attack messages by splicing elements of several emails together to make messages that are realistic under a particular model of the adversary's control. We construct the 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 the 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

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

Table 5.1 Parameters used in the Experiments on Attacking SpamBayes

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	N/A
Spam prevalence	0.50, 0.75, 0.90	0.50, 0.75, 0.90	0.50
Attack fraction	0.001, 0.005, 0.01, 0.02, 0.05, 0.10	0.001, 0.005, 0.01, 0.02, 0.05, 0.10	0.10
Folds of validation	10	10	N/A
Target Emails	20	N/A	N/A

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, yielding a sequence of models tainted with increasing numbers of attack messages. (Because SpamBayes 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.

5.5.1.3 Attack Assessment Method

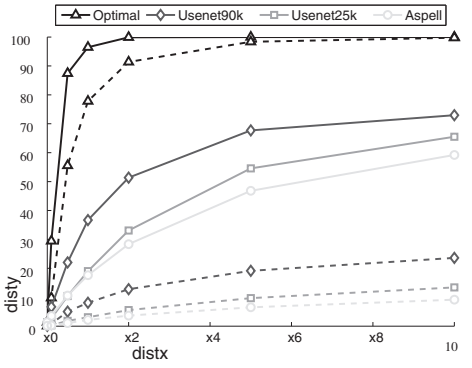
We measure the effect of each attack by randomly choosing an inbox according to the parameters in Table 5.1 and comparing classification performance of the control and compromised filters using 10-fold cross-validation. In cross-validation, we partition the data into 10 subsets and perform 10 train-test epochs. During the k^{th} epoch, the k^{th} subset is set aside as a test set, and the remaining 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 sequel, we demonstrate the effectiveness of attacks on test sets of held-out messages. Because the 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 in the tests was small compared to the effect of the attacks (see Figure 5.2(b) and (d)). See Table 5.1 for the parameters used in the experiments. We found that varying the size of the training set and spam prevalence in the training set had minimal impact on the performance of the attacks (for comparison, see Figure 5.2(a) and (c)), so we primarily present the results of 10,000-message training sets at 50% spam prevalence.

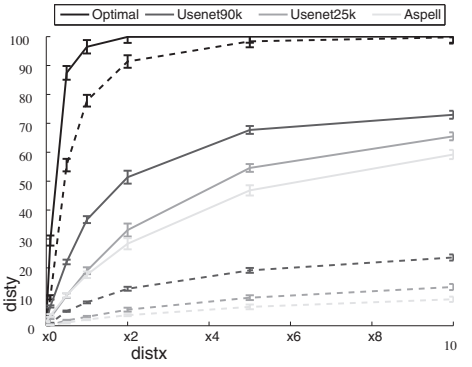
5.5.2 Dictionary Attack Results

We examine dictionary attacks as a function of the percent of attack messages in the training set. Figure 5.2 shows the misclassification rates of three dictionary attack

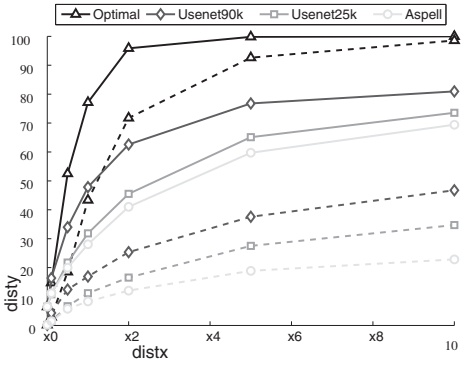
https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press



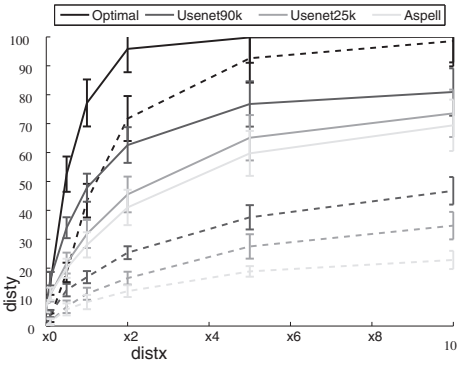
(a) Training on 10,000 messages (50% spam)



(b) Attacks (with error bars)



(c) Training on 2,000 messages (75% spam)



(d) Attacks (with error bars)

Figure 5.2 Effect of three dictionary attacks on SpamBayes in two settings. **(a)** and **(b)** have an initial training set of 10,000 messages (50% spam), while **(c)** and **(d)** have an initial training set of 2,000 messages (75% spam). **(b)** and **(d)** also depict the standard errors in the 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 (△), 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).

variants averaging over 10-fold cross-validation in two settings ((a) and (b) have an initial training set of 10,000 messages with 50% spam while (c) and (d) have an initial training set of 2,000 messages with 75% spam). First, we analyze the optimal dictionary attack discussed in Section 5.3.1 by simulating the effect of including every possible token in our attack emails. As shown in the figure, this optimal attack quickly causes the filter to mislabel all ham emails with only a minute fraction of control of the training set.

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,

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

since they contain 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 so the attack has succeeded.

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 has been redacted to remove many attachments and hence 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 tradeoff between vulnerability to dictionary attacks and the effectiveness of training the filter. In the next section, we present a defense that instead filters messages based directly on their impact on the spam filter's accuracy.

5.5.3 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 executed 20 focused attacks, one for each email, so the results average over 200 different trials.

These results differ from the focused attack experiments conducted in Nelson et al. (2008) 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 latter approach causes the percentage of tokens known by the attacker to fluctuate from message to message. Second, we only select messages with more than 100 tokens to use as target emails. With these changes, these results more accurately represent the behavior of a focused attack. Furthermore, in this more accurate setting, the focused attack is even more effective.

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

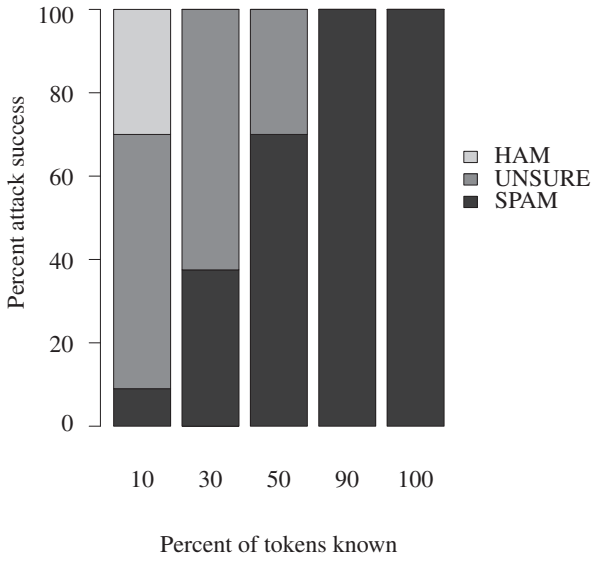


Figure 5.3 Effect of the focused attack as a function of the percentage of target tokens known by the attacker. Each bar depicts the fraction of target emails classified as *spam*, *ham*, and *unsure* after the attack. The initial inbox contains 10,000 emails (50% spam).

Figure 5.3 shows the effectiveness of the attack when the attacker has increasing knowledge of the target email by simulating the process by which the attacker guesses 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 5.3. 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*.

Figure 5.4 shows the attack's effect on misclassifications of the target emails as the number of attack messages increases with the fraction of known tokens fixed at 50%. 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.

Figure 5.5 shows the attack's effect on three representative emails. Each of the graphs in the figure represents a single target email from each of three attack results: ham misclassified as *spam* (a), ham misclassified as *unsure* (b), and ham correctly classified as *ham* (c). Each point represents a token in the email. The x-axis is the token's spam score (from Equation (5.2)) before the attack, and the y-axis is the token's score after the attack (0 indicates *ham* and 1 indicates *spam*). The $\times$'s are tokens included in the attack (known by the attacker) and the $\bigcirc$'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).

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

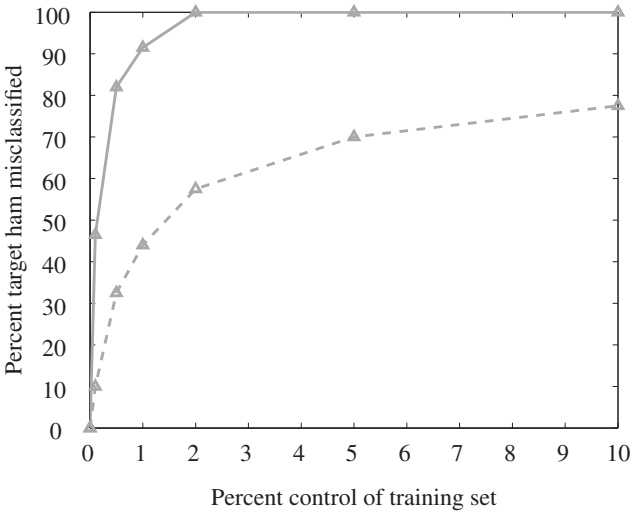


Figure 5.4 Effect of the focused attack as a function of the number of attack emails with a fixed fraction ($F = 0.5$) of tokens known by the attacker. The dashed line shows the percentage of target ham messages classified as *spam* after the attack, and the solid line the percentage of targets that are *spam* or *unsure* after the attack. The initial inbox contains 10,000 emails (50% spam).

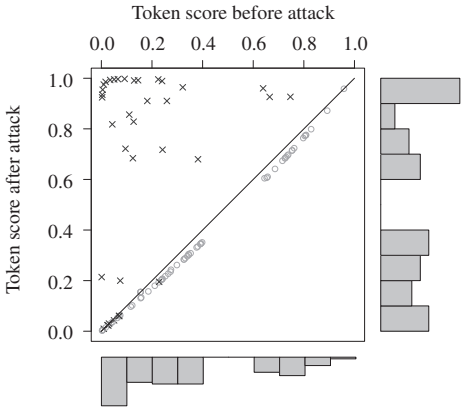
Any point above the line $y = x$ is a token whose score increased due to the attack, and any point below is a decrease. These graphs demonstrate that the scores of the tokens included in the attack typically increase significantly while those not included decrease slightly. Since the increase in score 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 5.3. Further, the before/after histograms in Figure 5.5 provide a direct indication of the attack's success. In shifting most token scores toward 1, the attack causes more misclassifications.

5.5.4 Pseudospam Attack Experiments

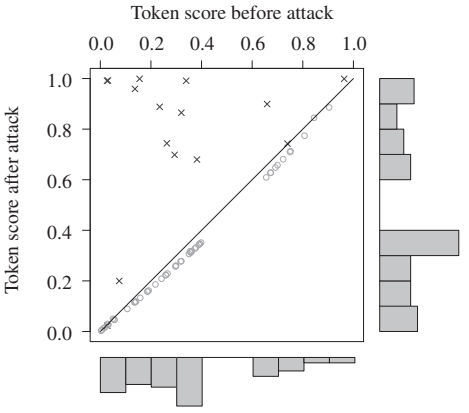
In contrast to the previous attacks, for the pseudospam attack, we created attack emails that may be labeled as ham by a human as the emails are added into the training set. We set up the experiment for the pseudospam attack by first randomly selecting a target spam header to be used as the base header for the attack. We then create the set of attack emails that look similar to ham emails (see Section 5.3.2). 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 5.6 demonstrates the effectiveness of the pseudospam attack by plotting the percent of attack messages in the training set (x-axis) against the misclassification rates

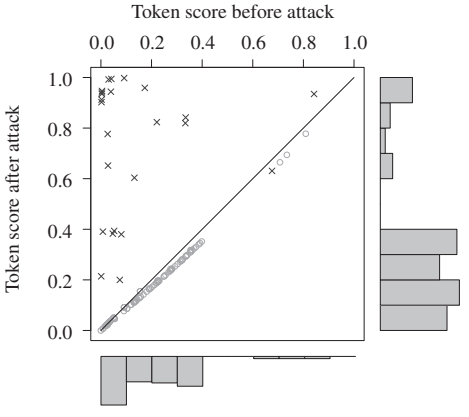
https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press



(a) Misclassified as *spam*



(b) Misclassified as *unsure*



(c) Correctly classified as *ham*

Figure 5.5 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 Equation (5.2) before the attack (0 indicates *ham* and 1 indicates *spam*). The y-axis is the token's spam score after the attack. The $\times$'s are tokens that were included in the attack, and the $\circ$'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).

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.

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

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

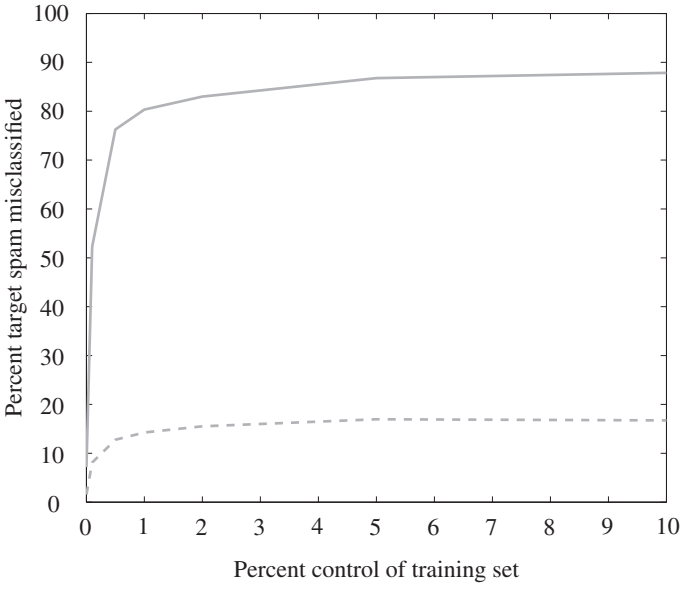


Figure 5.6 Effect of the pseudospam attack when trained as ham as a function of the number of attack emails. The dashed line shows the percentage of the adversary's messages classified as *ham* after the attack, and the solid line the percentage that are *ham* or *unsure* after the attack. The initial inbox contains 10,000 emails (50% spam).

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 5.7 indicates, this variant causes an increase in spams mislabeled as either *unsure* or *ham* 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.

5.5.5 RONI Results

Again to empirically evaluate the reject on negative impact defense, we sample inboxes from the TREC 2005 spam corpus. In this assessment, we use 20-fold cross validation to get an initial training inbox $\mathbb{D}^{(\text{train})}$ of about 1,000 messages (50% spam) and a test set $\mathbb{D}^{(\text{eval})}$ of about 50 messages. We also sample a separate set $\mathbb{D}^{(\text{suspect})}$ of 1,000 additional messages from the TREC corpus to test as a baseline. In each fold of cross-validation, we run five separate trials of RONI. For each trial, we use a calibration set of 25 ham and 25 spam messages and sample three training/quiz set pairs of 100 training and 100 quiz messages from the remaining 950 messages. We train two classifiers on each training set for each message in $\mathbb{D}^{(\text{suspect})}$, one with and one without the message, measuring

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

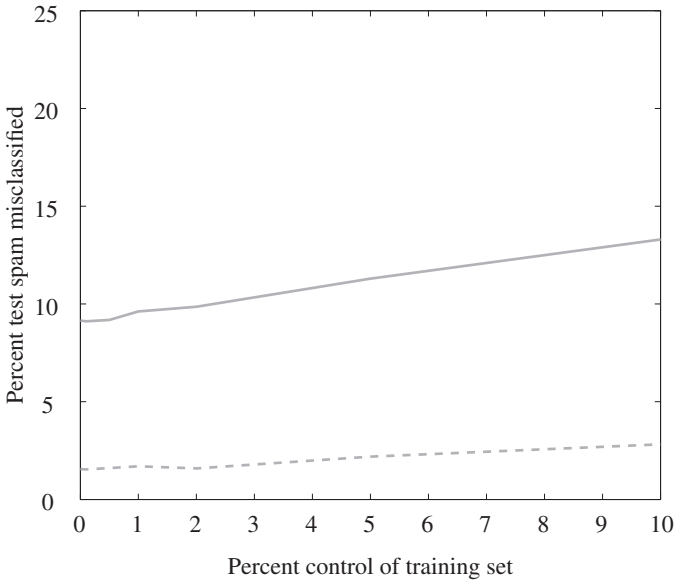


Figure 5.7 Effect of the pseudospam attack when trained as spam as a function of the number of attack emails. The dashed line shows the percentage of the normal spam messages classified as *ham* after the attack, and the solid line the percentage that are *unsure* after the attack. Surprisingly, training the attack emails as ham causes an increase in misclassification of normal spam messages. The initial inbox contains 10,000 emails (50% spam).

performance on the corresponding quiz set and comparing it to the magnitude of change measured from the calibration set.

We perform RONI evaluation for each message in $\mathbb{D}^{(\text{suspect})}$ as just described to see the effect on non-attack emails. We find that the reject on negative impact defense (incorrectly) rejects an average of 2.8% of the ham and 3.1% of the spam from $\mathbb{D}^{(\text{suspect})}$. To evaluate the performance of the post-RONI filter, we train a classifier on all messages in $\mathbb{D}^{(\text{suspect})}$ and a second classifier on the messages in $\mathbb{D}^{(\text{suspect})}$ not rejected by RONI. When trained on all 1,000 messages, the resulting filter correctly classifies 98% of ham and 80% of the spam. After removing the messages rejected by RONI and training from scratch, the resulting filter still correctly classifies 95% of ham and 87% of the spam. The overall effect of the reject on negative impact defense on classification accuracy is shown in Figure 5.8.

Since the RONI technique removes non-attack emails in this test, and therefore removing potentially useful information from the training data, SpamBayes' classification accuracy suffers. It is interesting to see that test performance on spam actually improves after removing some emails from the training set. This result seems to indicate that some non-attack emails confuse the filter more than they help when used in training, perhaps because they happen to naturally fit some of the characteristics that attackers use in emails.

Next we evaluate the performance of RONI where $\mathbb{D}^{(\text{suspect})}$ instead consists of attack emails from the attacks described earlier in Sections 5.3. RONI rejects every single

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

Before RONI

		Predicted Label		
		ham	spam	unsure
Truth	ham	97%	0.0%	2.5%
	spam	2.6%	80%	18%

After RONI

		Predicted Label		
		ham	spam	unsure
Truth	ham	95%	0.3%	4.6%
	spam	2.0%	87%	11%

Figure 5.8 Effect of the RONI defense on the accuracy of SpamBayes in the absence of attacks. Each confusion matrix shows the breakdown of SpamBayes's predicted labels for both ham and spam messages. **Left:** The average performance of SpamBayes on training inboxes of about 1,000 message (50% spam). **Right:** The average performance of SpamBayes after the training inbox is censored using RONI. On average, RONI removes 2.8% of ham and 3.1% of spam from the training sets. (Numbers may not add up to 100% because of rounding error.)

dictionary attack from any of the dictionaries (optimal, Aspell, and Usenet). In fact, the degree of change in misclassification rates for each dictionary message is greater than five standard deviations from the median, suggesting that these attacks are easily eliminated with only minor impact on the performance of the filter (see Figure 5.9).

A similar experiment with attack emails from the focused attack shows that the RONI defense is much less effective against focused attack messages. The likely explanation is simple: *Indiscriminate* dictionary attacks broadly affect many different messages with their wide scope of tokens, so its consequences are likely to be seen in the quiz sets. The focused attack is instead targeted at a single *future* email, which may not bear any significant similarity to the messages in the quiz sets. However, as the fraction of tokens correctly guessed by the attacker increases, the RONI defense identifies increasingly many attack messages: Only 7% are removed when the attacker guesses 10% of the tokens, but 25% of the attacks are removed when the attacker guesses 100% of the tokens. This

Dictionary Attacks (Before RONI)

		Predicted Label		
		ham	spam	unsure
Optimal				
True Label	ham	4.6%	83%	12%
	spam	0.0%	100%	0.0%
Aspell				
True Label	ham	66%	12%	23%
	spam	0.0%	98%	1.6%
Usenet				
True Label	ham	47%	24%	29%
	spam	0.0%	99%	0.9%

Dictionary Attacks (After RONI)

		Predicted Label		
		ham	spam	unsure
Optimal				
True Label	ham	95%	0.3%	4.6%
	spam	2.0%	87%	11%
Aspell				
True Label	ham	95%	0.3%	4.6%
	spam	2.0%	87%	11%
Usenet				
True Label	ham	95%	0.3%	4.6%
	spam	2.0%	87%	11%

Figure 5.9 We apply the RONI defense to dictionary attacks with 1% contamination of training inboxes of about 1,000 messages (50% spam) each. **Left:** The average effect of optimal, Usenet, and Aspell attacks on the SpamBayes filter's classification accuracy. The confusion matrix shows the breakdown of SpamBayes's predicted labels for both ham and spam messages after the filter is contaminated by each dictionary attack. **Right:** The average effect of the dictionary attacks on their targets after application of the RONI defense. By using RONI, all of these dictionary attacks are caught and removed from the training set, which dramatically improves the accuracy of the filter.

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

Focused Attacks (Before RONI)

	Target Prediction		
	ham	spam	unsure
10% guessed	78%	0.0%	22%
30% guessed	30%	5.2%	65%
50% guessed	5.8%	23%	71%
90% guessed	0.0%	79%	21%
100% guessed	0.0%	86%	14%

Focused Attacks (After RONI)

	Target Prediction		
	ham	spam	unsure
10% guessed	79%	2.7%	21%
30% guessed	36%	4.8%	59%
50% guessed	19%	20%	61%
90% guessed	20%	62%	19%
100% guessed	21%	66%	13%

Figure 5.10 The effectiveness of the RONI defense on focused attacks with 1% contamination of training inboxes of about 1,000 messages (50% spam) each. **Left:** The average effect of 35 focused attacks on their targets when the attacker correctly guesses 10, 30, 50, 90, and 100% of the target's tokens. **Right:** The average effect of the focused attacks on their targets after application of RONI. By using RONI, more of the target messages are correctly classified as ham, but the focused attacks largely still succeed at misclassifying most targeted messages.

is likely due to the fact that with more correctly guessed tokens, the overlap with other messages increases sufficiently to trigger RONI more frequently. However, the attack is still successful in spite of the increased number of detections (see Figure 5.10).

5.6 Summary

Motivated by the taxonomy of attacks against learners, we designed real-world *Causative* attacks against SpamBayes' learner and demonstrated the effectiveness of these attacks using realistic adversarial control over the training process of SpamBayes. Optimal attacks against SpamBayes caused unusably high false-positive rates using only a small amount of control of the training process (more than 95% misclassification of ham messages when only 1% of the training data is contaminated). Usenet dictionary attacks also effectively use a more realistically limited attack message to cause misclassification of 19% of ham messages with only 1% control over the training messages, rendering SpamBayes unusable in practice. We also show that an informed adversary can successfully target messages. The focused attack changes the classification of the target message virtually 100% of the time with knowledge of only 30% of the target's tokens. Similarly, the pseudospam attack is able to cause nearly 90% of the target spam messages to be labeled as either *unsure* or *ham* with control of less than 10% of the training data.

To combat attacks against SpamBayes, we designed a data sanitization technique called the reject on negative impact (RONI) defense that expunges any message from the training set if it has an undue negative impact on a calibrated test filter. RONI is a successful mechanism that thwarts a broad range of dictionary attacks—or more generally *Indiscriminate Causative Availability* attacks. However, the RONI defense also has costs. First, this defense yields a slight decrease in ham classification (from 98% to 95%). Second, RONI requires a substantial amount of computation—testing each message in $\mathbb{D}^{(\text{suspect})}$ requires us to train and compare the performance of several classifiers. Finally, RONI may slow the learning process. For instance, when a user correctly labels

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

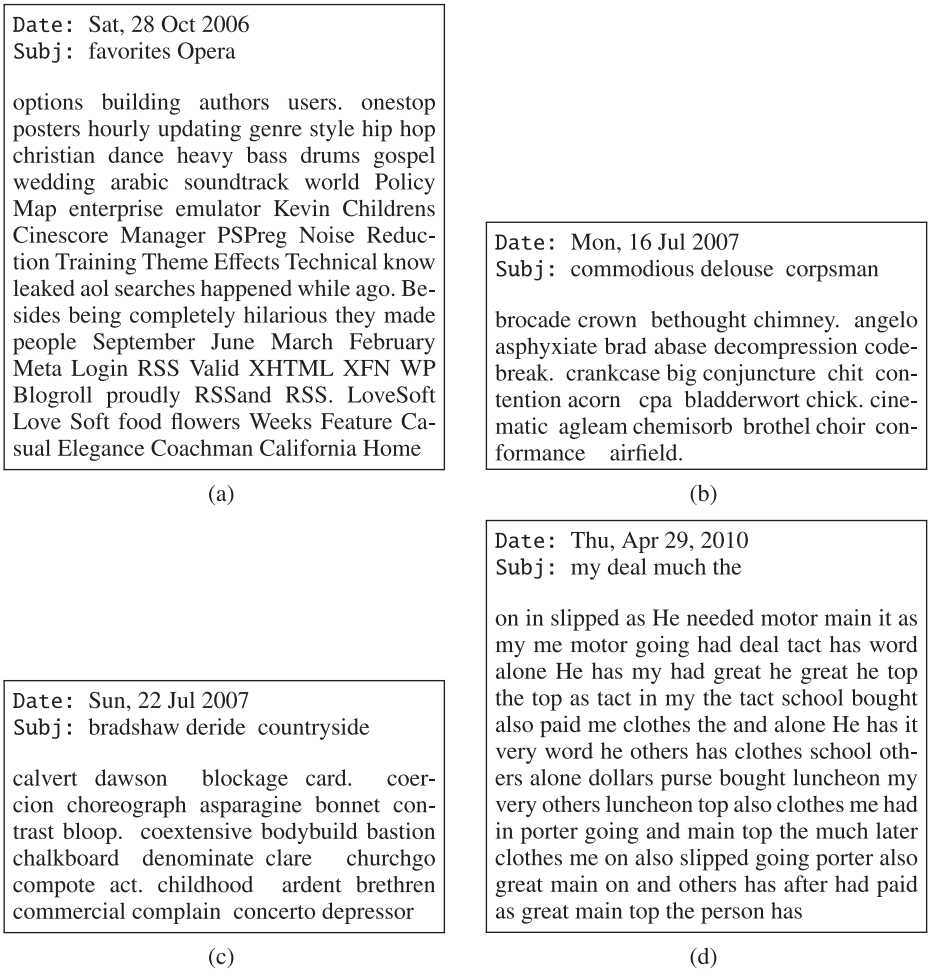


Figure 5.11 Real email messages that are suspiciously similar to dictionary or focused attacks. Messages **(a)**, **(b)**, and **(c)** all contain many unique rare words, and training on these messages would probably make these words into spam tokens. As with the other three emails, message **(d)** contains no spam payload, but has fewer rare words and more repeated words. Perhaps repetition of words is used to circumvent rules that filter messages with too many unique words (e.g., the *UNIQUE_WORDS* rule of Apache SpamAssassin (Apa n.d.)).

a new type of spam for training, RONI may reject those instances because the new spam may be very different from spam previously seen and more similar to some non-spam messages in the training set.

In presenting attacks against token-based spam filtering, there is a danger that spammers may use these attacks against real-world spam filters. Indeed, there is strong evidence that some emails sent to our colleagues may be attacks on their filter. Examples of the contents of such messages are included in Figure 5.11 (all personal information in these messages has been removed to protect the privacy of the message recipients).

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press

However, these messages were not observed at the scale required to poison a large commercial spam filter such as GMail, Hotmail, or Yahoo! Mail. It is unclear what, if any, steps are being taken to prevent poisoning attacks against common spam filters, but we hope that, in exposing the vulnerability of existing techniques, designers of spam filters will harden their systems against attacks. It is imperative to design the next generation of spam filters to anticipate attacks against them, and we believe that the work presented here will inform and guide these designs.

Although this work investigated so-called Bayesian approaches to spam detection, there are other approaches that we would like to consider. One of the more popular open-source filters, Apache SpamAssassin (Apa n.d.), incorporates a set of hand-crafted rules in addition to its token-based learning component. It assigns a score to each rule and tallies them into a combined spam score for a message. Other approaches rely exclusively on envelope-based aspects of an email to detect spam. For instance, the IP-based approach of Ramachandran, Feamster, & Vempala (2007) uses a technique they call *behavioral blacklisting* to identify (and blacklist) likely sources of spam. This diverse range of detection techniques require further study to identify their vulnerabilities and how spammers exploit multifaceted approaches to spam detection. Further, there is a potential for developing advanced spam filtering methods that combine these disparate detection techniques together; the online expert aggregation setting discussed in Section 3.6 seems particularly well suited for this task.

https://doi.org/10.1017/9781107338548.005 Published online by Cambridge University Press