University of Luxembourg
Faculty of Sciences, Technology and Communication

Bachelor's Thesis

Manipulation of online voting systems via exploitation of Bayesian spam filters

by

Tom Schmitz

Supervisors:

Prof. Dr. Sjouke Mauw (University of Luxembourg)
Prof. Dr. Peter Y.A. Ryan (University of Luxembourg)

Luxembourg, June, 2015

Abstract

This paper focuses on manipulating the outcome of Online Voting Systems, based on the example of The Helios Online Voting platform. These systems themselves allow for little manipulation since their main focus is on securing the voting process. This means guaranteeing to the voters that their voice is recorded correctly and goes untampered into the ballot. We will therefore focus on the environment to which these systems are bound. More precisely, we take advantage of Bayesian spam-filters to automatically get the emails from Helios or similar voting platforms blocked for a selected group of voters. This way our victims lose their opportunity to vote. If we are able to manipulate the spam-filters of enough eligible voters, this will have a significant impact on the outcome of a voting.

Contents

Chapter 1

Introduction

1.1 Background

A vote over the Internet can be simple and quick, which speaks for this method. The trend of online voting is on the rise and it is very likely that we will be giving our voice electronically in future elections. Voters need to be notified though. The easiest way to reach someone electronically is per Email. The latter have been used with nearly the same protocols since the 1990. There have been some additional features, which organize, redirect or even sign and encrypt emails but these are not part of the average user's email-tools.

A feature about emails we all have though is a spam filter, some without even taking note of it. Most of us think that we understand how spam filters work and that we are in control of what we see and what we don't. But with the user's trust comes responsibility for the developer. A software can be designed in a very intuitive way but its effectiveness will always be bound to the user interaction. This challenge stays hidden and is mostly underrated by most users. In this paper we will show the consequences bound to this challenge, while taking advantage of a users limited interaction with spam filters. More precisely we will focus on the action of marking incoming mails as Spam, which will trigger learning algorithms and influence which emails will be automatically transferred to the spam folder in the future. We will discuss how we could take advantage of this to influence the outcome of elections. The idea will be to push victims into unknowingly manipulating their spam filters in a way that they are going to miss targeted messages.

1.2 Structure

In this paper we will first of all introduce the reader to the technologies that were used for these manipulations, namely how online voting systems work and how they rely on emails. We then talk about spam filters, more precisely the Bayesian filtering methods, the vulnerabilities of such and how we can make use of these for our experiments.

Chapter 2

Online Voting Systems

Online voting systems form a large part within the domain of electronic voting technologies. Most online voting systems rely on a web-based platform which will be consulted by the users via their web browsers. The idea of such online voting systems is to simplify the voting process and the evaluation of the votes but just like for the normal voting methods on paper, the online platforms which collect and count the votes must be secure and trustworthy.

2.1 End-to-End verifiability

Most of the main online voting systems claim to be End-to-end auditable or end-to-end voter verifiable (E2E [1]), which means they strongly invested in integrity properties and tamper resistance. Cryptographic methods are the key to ensuring voters that their votes are legitimate and guarantee them a high level of privacy. There is a certain number of steps taken during an online voting process, which lead from the voter's intention to the election result. Each of these steps of this chain's integrity is to be guarded, which is achieved by end-to-end measures. Requirements therefore run down to voter auditing, enabling a voter to check whether his/her vote is correctly included electronically and universal verifiability, enabling anyone to determine the correctness of the ballots and the tally.

2.2 Vulnerabilities

As soon as a system is made available through the Internet, the chance that someone will be tempted to attack the system becomes significant. Since votings are crucial to democracy and can decide over very important matters, the temptation and gain for hackers is not to be neglected. Of course, online voting systems developers are aware of a certain amount of responsibility and highly interested in making sure their ballots are safe. Nevertheless there are always attacks, which are harder to prevent.

Some attacks against Online voting systems include browser rootkits, DDOS, Bot-Nets, Cross-Site-Scripting and hacking of personal computers and email accounts. Taking into account E2E verifiability, it becomes clear very fast, that

the vulnerabilities of such systems lie not within the software itself but rather the environment to which it is bound to. This environment includes browsers of personal and public computers and email-accounts. This environment is necessary to connect to the voters and publish the results.

2.3 Examples

Some example of online voting systems which are the most present on the web and focus on E2E and privacy are: *Scantegrity*, *Pret-a-Voter* [2], *STARVote* [3], *VoteHere* [4] and the *Helios Voting System* [5].
For this paper we are going to consider The Helios Voting System, which claims to be private, verifiable and proven. The project by Ben Adida is open source, which makes it ideal for research.

2.4 Helios

We start by explaining the general operation of Helios. The election/referendum is set up by a user who logs in with a Google-/Facebook- or Helios-account. This voting can be public or private. Considering the private mode, Helios will require a list containing names and email addresses of eligible people. After the poll is set up completely, Helios processes the list and sends out emails to the potential voters, to notify them that they can now give their vote and inform them on how they can do this. Such an email consists of a subject, message body which is to be written by the election initiator, an election URL and election fingerprint. For voter authentication, a voter ID and a password are included. Following the provided link, after the login with the provided credentials the vote is recorded, reviewed and submitted. Every voter obtains a ballot tracker which will be the guarantee that the ballot made it to the tally and more importantly that it was not tampered with in the process.

The structure of such a voting email is as follows:

Dear *voter.name*,

custom_message

Election URL: *election_vote_url*
Election Fingerprint: *voter.election.hash*

Your voter ID: *voter.voter_login_id*
Your password: *voter.voter_password*

Log in with your *voter.voter_type* account.

We have recorded your vote with smart tracker: *voter.vote_hash*
You may re-vote if you wish: only your last vote counts.

In order to protect your privacy, this election is configured
to never display your voter login ID, name, or email address to the public.
Instead, the ballot tracking center will only display your alias.
Your voter alias is *voter.alias*.

IMPORTANTLY, when you are prompted to log in to vote,
please use your *voter ID*, not your alias.

–
Helios

Chapter 3

Email

When it comes to communication of notifications, questions, messages and data, emails are indispensable. Interestingly, though, a huge number of emails are neither encrypted nor authenticated. Emails follow very simple protocols that don't include safety mechanisms by default and not a lot of users are aware of how to enhance this situation. An email consists of a sender-address, a receiver-address, the date, the current time-stamp and the subject forming all together the header information followed by the body, dedicated to text and data containing mostly HTML/plaintext.

3.1 Vulnerabilities

The vulnerabilities lie within confidentiality, authentication, integrity, non-repudiation, access control, and availability. There are numerous active and passive attacks targeted to email. While active attacks consist in modification of message contents, masquerade, replay and denial-of service, the passive ones target disclosure of information and traffic analysis. Since anyone can send emails to anybody, a huge percentage of all emails is devoted to Spam.

3.2 Spam

Spam describes any message that is unwanted in the receivers inbox. This may be due to the risk that this email contains harmful software, links to web-pages that are involved in identity theft or try to sell more or less legal products. Also many online services Spam their users with promotions and offers. In order to fight spam, the most effective way is to have a certain filter which acts on the mailbox, getting the spam-mails out of the way. Usually spam mails are put into a different folder than the inbox itself or deleted immediately.

3.3 Filtering

The main ideas of filtering involve black-listing, statistical approaches based on regular expressions or ominous character-sequences, check-sums and Bayes-filtering. Most of the tools for spam filtering combine some of the before men-

tioned methods to detect if an email is rather spam or ham. The main goal of course is to eliminate spam mails but a side problem consists of good emails being misclassified as spam. Since the hardness of this problem there have to be cut-offs meaning there is usually a third classification between *spam* and *ham*, namely *unsure*. Statistical approaches render a probability from 0 (ham) to 1 (spam) denoting the spammicity (probability that the email is spam), where the cut-off from ham is usually around 0.4 and the spam cut-off is at 0.9 or 0.99 depending on the accuracy of the tools used. [7]

Most spam filters are implemented closely within the mailbox. They fetch the new mails and insert certain headers to the emails usually containing the spam probability and a tag whether they think the document is spam, ham or unsure. These headers can be recognized either by the chosen email-program or handed over to a software which will deviate the mails in certain mail-directories/mailboxes.

3.4 Naive Bayes Filters

One particular type of email filtering is called the Bayesian method [6]. It has the ability to learn and classify spam and ham with a minimum user interaction. A Bayesian filter (or naive Bayes filter) evaluates the probability that an email is spam on the words which it contains. The tool consists of two parts. A learner and a classifier.

The learner takes as input usually two mailbox files (.mbox). One file containing spam and the other one ham. While processing, also called training, the learner takes every word into account and adds/updates it in the database. The database contains every word encountered during training, associated with a probability that it is spam, ham and the number of occurrences. These are the rules, that the classifier will take into account, while evaluating new emails. Basically every word of a new email will be compared to the existing ones in the database, which in combination will lead to the overall probability that the message is spam.

Let's imagine for example that the word "vote" is contained in a message. From the database we obtain:
- **P(W|S)** which expresses the conditional probability that the word "vote" appears in a message knowing the message is spam and
- **P(W|H)** expressing the probability for the word "vote" to be contained in messages we know are ham.
- We can also determine from the database **P(S)** and **P(H)** which denote the overall probability for a random message to be spam or ham.

Now derived from Bayes' theorem, we can determine **P(S|W)** which is the probability that a message is spam, knowing it contains the word "vote".

$$\mathbf{P(S|W)} = \frac{P(W \mid S) \times P(S)}{P(W \mid S) \times P(S) + P(W \mid H) \times P(H)}$$

Some filters use the hypothesis that P(S) = P(H) = 0.5 which is usually not the

case. This approach can be expanded easily to the complete email by adding up the spam probabilities of each word, relatively to the word count of the email or in most cases relatively to the number of the words taken into account. There are exceptions to these rules, for example if there are short words, of less than 3 characters like "the" or "Hi". These usually appear often in the natural language and are considered neutral. Also longer words can be neutral, like "some", so it makes sense to ignore such words since their spammicity is close to 0.5. How big the impact of a word has to be in order to be considered for the overall spammicity of an email is determined by its distance to the neutral value of 0.5. This minimum distance is set by the "minimum deviation" which is a local parameter used for example in Bogofilter. It is set in a configuration file but can be altered by the user for special purposes or local adjustments of the filter. Examples of Filters relying on the Bayesian algorithm are *Spambayes* [8], *Bogofilter* [9], *SpamAssassin*[10] and the native *Thunderbird* [11] filter.

3.5 Manipulation of Bayesian filters

The Bayesian filtering technique relying on the above mentioned structure is simple but also develops a large surface for numerous attacks. The most important one is Bayesian poisoning [12], used by spammers. It affects the effectiveness of the spam filter negatively until rendering it nearly useless. The idea is very simple and consists of sending a large amount of emails containing random words or legitimate words. These will confuse the database by flattening the probabilities making it more likely for future spam mails to end up in the inbox, misclassified as ham.

Other methods consist of altering spammy words like "viagra" into "viaagra" or "v!agra" and rendering an image containing the message. Such a picture can not be processed by the word-based filters. These attacks are easy to point out and have low impact. Bayesian poisoning, however, is harder to detect and not easily undone since it settles at the basis of the filtering.

Chapter 4

Manipulation Approach

Our intention is to get the official emails informing a specific user about elections to be filtered automatically as spam. This will achieve that the user ignores the email and will not submit his vote. We plan to achieve this by sending emails to the targeted account which are composed in a such way as to influence the Bayesian filter in our advantage.
We are performing a kind of guided Bayesian poisoning, meaning that we don't intend to globally damage the filter but only train it in a way that mails from Helios will be misclassified as spam.
For this reason we require a target email account which is actively using a filter that includes Bayesian filtering and weights it as the main spam filtering criteria like BogoFilter and SpamBayes, which are purely Bayesian filters. Such filters of real users have already been trained on thousands of legitimate spam and ham emails.

What we don't want is to affect any filtering behavior on a public or server-scale. Affecting public blacklists will affect the whole vote manipulation approach negatively. The attacks presented during the following experiments require the active participation of the victim who needs to manually point out misclassified emails as spam to his/her spam-filter.

Furthermore, we assume that the set of email accounts to be attacked is known. Determining which accounts to target is not part of this research. But the fact that we need to know which voter's voice to suppress is definitely a precondition of the following approach.

Chapter 5

Experiments

5.1 Setup

For our experimental setup, we want to be able to deliver accurate results while keeping the complexity of our experimental setup reasonable. The question is if there is a need to simulate every component of the environment which Helios is bound to. Some may suggest that such an approach would be more complete and result in more reliable results, but we will try to minimize the software interaction during our tests. In order to do so, we will evaluate the effectiveness of our manipulation approach mainly with one component at a time, which is the spam filter itself.

Generic

Looking at the real world environment, we identify three groups of actors. The servers running the Helios online voting system, a number of victims holding personal email accounts and an attacker, trying to influence the behavior of the victim's spam filter by sending harmful emails which we will refer to as "attack-emails" in this paper.
In the real world procedure there are components involved which are hard to configure on a local basis, such as email servers running for example Postfix and email forwarding tools such as Procmail. These would intercept and guide the email traffic as well as run the spam-filter on the incoming emails. In our case this spam-filter is Bogofilter.
Last but not least, we have an mail-client software involved which the victim(s) would use to manage their email-account(s). We can imagine Thunderbird for this task, since it allows to interpret the headers from various spam-filters i.e. SpamAssassin, Bogofilter, etc.

In theory, representing all these pieces of software in our experimental setup is not impossible but yet a complicated problem which requires a high amount of investigation and time-involvement. A well-known fact of engineering is that the more components involved, the higher the margin for errors. A very close to the real world setup may seem more evident but would not result in more accurate results while only complicating and slowing down the experiments themselves.

For example, a serious amount of interaction is involved on the victim side during the experiments, since this one will be sorting the incoming mails partly manually, supporting our attacks unknowingly. For a better time-investment we will go with a limited testing environment as described in the following paragraph.

Experimental

Our focus will be to limit the interaction between different components and actors. The running of tests should not consume too much time and resources. Configuration initialization and setup of different software is one of the first steps for every experiment. We need to start limiting ourselves here. In order to do so, we will setup only the spam filter and its dependencies - a database and different sets of email-inboxes representing the victims accounts. For the spam filter, our choice fell on Bogofilter since this one is well represented in the community and has already proved its effectiveness. Furthermore it's easy to reset, configure and provides many opportunities to inspect results in the form of very detailed shell outputs about what is contributing to the spammicity of messages, and the composition of the database. We also need resources to generate the attack-emails. These resources are certain dictionaries holding words from different sources and topics. The main ingredient for generating appropriate Spam-mails will be the email templates which we have from Helios. The dictionary of words extracted from these templates forms the base of the attack-emails.
Additionally we rely on dictionaries containing words, which are susceptible to occur in the "custom_message"-part of legitimate Helios-emails. These are associated with the topics of elections, referendums and other types of votings. Other dictionaries which are interesting for us in order to produce authentic spam, are the lists of words which are associated with spam in general on the Internet.

5.2 Definition

All our interactions will be implemented on the spam filter layer. As already discussed earlier this filter needs to be representative for a real email account. To achieve this, we orient ourselves on training sets. The training sets are mailbox files (.mbox) containing real spam and ham emails from different public sources such as the Trec05 [13] and the Enron [14] spam corpus.

The spamfilter will be trained firstly on these mailboxes, simulating different types of users by choosing training sets. These belong each to different people from companies that made their email traffic public like the "2005 TREC Public Spam Corpus" [13]. Just like for a real email account, where some users get more and some get fewer emails or train their spam filters more or less, the sizes of these training sets range from a few hundreds up to tens of thousands of mails.

The most important factor for the manipulations is the composition of the

attack-emails in terms of quantity and quality of the content. This affects how successful a single attack-email is and how long the overall manipulation takes until reaching satisfactory levels. There are numerous possibilities to mix the words from our dictionaries into the attacks. This results in many possible approaches to generate attack-emails.

Such a manipulation consists in the case of our experimental setup in training the filter incrementally on generated attack emails. After every attack, we evaluate the quality of the updated filtering capabilities.

To evaluate the performance of our spamfilter, we call the classifier on a set of legitimate emails. These are mocks for real world election/referendum texts which would be contained on the place of *custom_message* in official emails from Helios. For this purpose, we came up with 4 scenarios of different custom-messages. (Helios template can be found in section 2.4 2)

One more chance to influence the overall learning capacity of the filter itself would be separate or combined email attacks, devoted to Bayesian poisoning. This Bayesian poisoning can either involve all kinds of messages or focus on messages targeted to voting only. With combined we mean the merging of a wave of attack emails into one email, reducing the amount of theoretical clicks a user has to perform, while separate emails may yield better results.

5.3 Execution

Preparation

Getting the environment set up and initialized correctly before starting to manipulate it, is crucial in order to guarantee the comparability of different experiments. Our spam filter is using a database to store data about the ham and spam probabilities of certain words, encountered during previous tests. Since we want to start from a fresh spam-filter each time we perform a new test, we have to reset the local database, which is done by simply deleting the corresponding files. To perform the base training of our filter, we need to convert our real ham and spam messages, which are supposed to represent a real world situation, into a format which can be interpreted by the training algorithm. This comes short to the generation of the mailbox files (.mbox). Furthermore we need to prepare a set of legitimate texts, which will represent the case of a real worlds utilization of Helios. These are also used later on to evaluate the filter's effectiveness. Since there are three different types of Helios-templates for official emails, and we have four legitimate custom-messages that will be twelve emails to form the test set.

Looking for example at the template "vote", inserting one of the legitimate messages we imagined, we obtain the following email:

From Heliosvoting@dhdh.lx Fri Dec 25 00:06:42 2010
Subject: Vote for the future of California

An Election Message from Governor Jerry Brown

As a college student you can help decide the future of California by voting on November 6. But the first step is to register as a California voter before the deadline on Monday, October 22. The power is in your hands don't miss the opportunity to make your voice heard. Register before the deadline, Monday, October 22. Then vote on Tuesday, November 6.

With respect,
Governor Jerry Brown

Election URL: vote.grgdbtbrtrhtr.lu
Election Fingerprint: 534343453
Your voter ID: 254324534
Your password: 432544
Log in with your google facebook account.
We have recorded your vote with smart tracker: 61651hhdgfg681
You may re-vote if you wish: only your last vote counts. In order to protect your privacy, this election is configured to never display your voter login ID, name, or email address to the public. Instead, the ballot tracking center will only display your alias.
Your voter alias is logff12.
*IMPORTANTLY, when you are prompted to log in to vote, please use your *voter ID*, not your alias.*

– Helios

Experiment parameters

We identified some factors that we suspect to have an impact on our results. These will be considered as variables which can be altered over the run of the test series and have their impacts evaluated.
These experimental parameters apply to the preparation and execution phases of our tests:

- choice of training set for initial training (in our case, Trec05 and Enron).
- number and proportion of spam/ham emails initially trained on (between 3000 and 30000).
- ham and spam cutoff for the filter(s) (we currently use the default settings).
- min_dev of the spam filter, expressing at which impact a word is considered for the total spammicity of a message (from 0.1 to 0.375 which is the default).

- size of the attack emails.
- proportion of Helios/non-Helios-words in the generated attack-emails.
- which lists/dictionaries of non-Helios-words to consider for Bayesian poisoning.

Iterative process

The main part of the experiment is to generate attack emails, for every experiment with different parameters, and then sending them one by one to the spam filter, where they will be trained as spam. For the following example we made use of two word-lists also referred to as "dictionaries" later on. One containing all the words which are part of the Helios templates and another list holding words associated with spam-mails. We are alternating the words to make the email look like normal spam to the victims.

Example of an attack email:

From Luxury@experience.com Fri Dec 25 00:06:42 2009
Subject: Lower monthly payment passwords
Remuneration Election Subsidiary Link: payment Dear Usury – Reapportionment Helios Reply How Syndicate to Wholesale Vote Return ======== Computer Election roots URL: Coattail Your Challenger voter Believe ID: Decide Your Permit password: Advertisement Log Pamphlets in Broadcast with Downsize your ... (etc.)

Underlined are the original words from Helios. The other words have as goal to irritate the victim and persuade him/her that this message is spam and should be marked as such. This is supported also by the subject and sender line, which consists here of words from the spam-list only. After an attack email is sent and the filter got trained on it as spam, we determine the spamminess of our set of legitimate emails with Bogofilter. Bogofilter adds a header line to the email, from which we extract its spamminess, named originally "X-Bogosity". Repeating the attack a number of iterations, we can compare the evolution of the spamminess over the number of attack emails sent. These spamminess values reaching from 0.00 to 1.00 can be plotted later to better visualize the results.

5.4 Outcome

5.4.1 An Example

The raw data from the standard experiment has the following form:

email#	info	vote	simple	average
0	0.320146	0.438499	0.103746	0.287
1	0.506211	0.581081	0.503428	0.530
2	0.50652	0.673799	0.509639	0.563
3	0.506601	0.73371	0.516792	0.585
...	...	...	...	...

This example covers one of the four legitimate email-texts, as it is combined with the three different templates form Helios (info, vote and simple). In the first column, we see the number of attack emails which have been sent so far. The other columns express the spammicities of the test messages where each column represents one of the three templates we have for Helios-emails.we calculated the average over the results of the three templates, since we are interested in the overall effect. Figure 5.1 shows a graphical representation of the data in the example table.

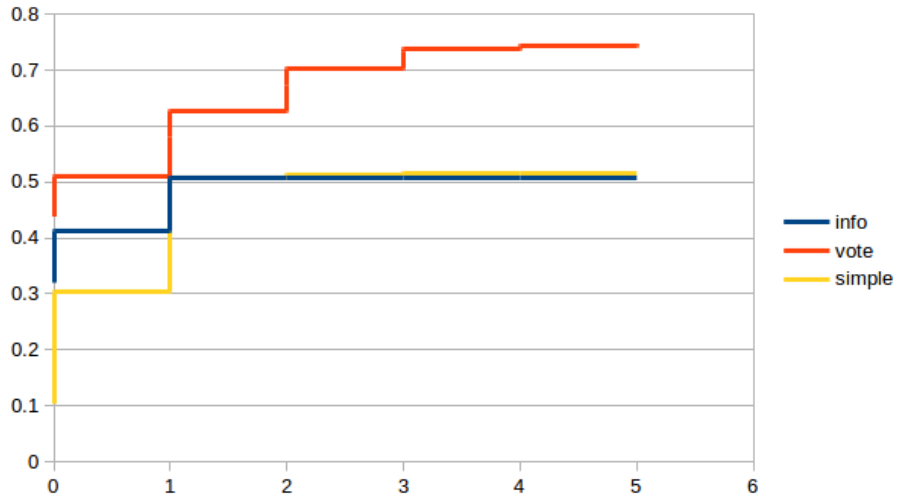

Figure 5.1: Representation of the data in the example table. On the X axis, we have the state of manipulation, expressing how many emails have been sent already. On the Y axis is the spamminess, from 0.00 (ham) to 1.00 (spam).

Looking at figure 5.1 we see at x=0 that the base spamminess of each of the templates is below 0.45. This is expected, since our legitimate messages should be considered ham by a not-manipulated spam-filter. Over the iterations/attacks, we observe a change in the filter's behavior. The misclassification we were trying to provoke takes place and the messages' spamminesses are rising. In this example we don't yet reach the point where a test-message is clearly classified as spam which would require a spamminess of 0.99 at least.

5.4.2 Experiments with variable base-training

This series of tests will focus on changing the size of the base training set, containing the ham and spam used to initially train our spam-filter. This simulates email accounts that receive more or less emails. We will use only the Helios-template "vote" for evaluation to avoid overloading the graphics.

The Samples 1-4 denote the 4 hand written legitimate email-texts which we prepared by hand. The set of emails used is the Enron training set, containing 33000 emails. The Enron email-dataset being subdivided into 6 folders of equal size, we considered 2, then 4 and 6 of the folders. This comes down to training

on $1/3^{rd}$ of the set (figure 5.2), then on $2/3^{rds}$ (figure 5.3) and finally on the complete set of emails (figure 5.4).

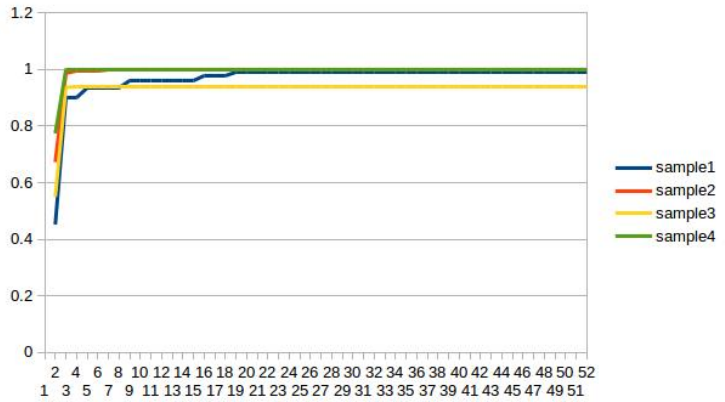

Figure 5.2: 1/3 of the Enron email-data used for base-training the spam-filter - about 11000 spam and ham emails

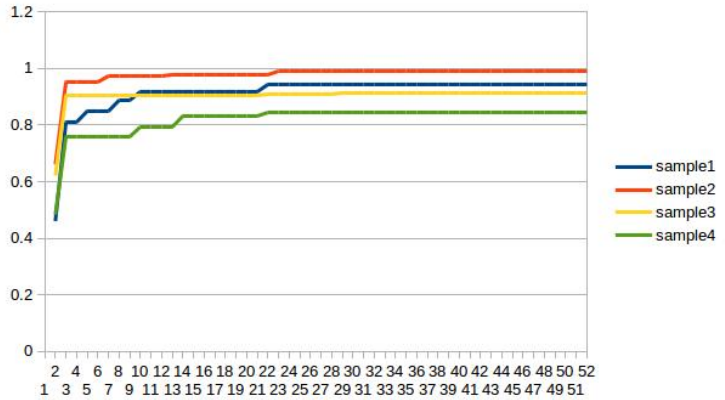

Figure 5.3: 2/3 of the Enron email-data used for base-training the spam-filter - about 22000 spam and ham emails

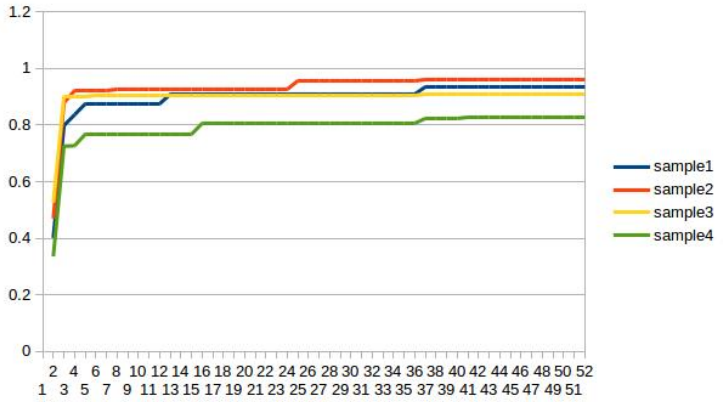

Figure 5.4: Complete Enron email-set used for base-training - about 33000 spam and ham emails

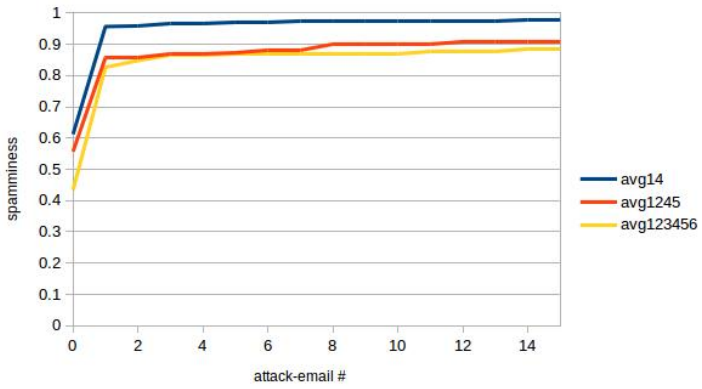

Figure 5.5: Average values of Figures 5.2 (trained on folders 1 and 4), 5.3 (trained on folders 1, 2, 4 and 5) and 5.4 (trained on folders 1, 2, 3, 4, 5 and 6) The "#"-symbol stands here for "number of" in this case: attack emails sent.

Finally figure 5.5 shows the combination of the 3 previous graphs, plotting only the average spamminess over the samples 1-4 every time.

5.4.3 Experiments with changed min_dev

Over the run of this experiment we will alter the min_dev parameter of Bogofilter and observe which consequences this has for our attacks. The min_dev variable, stands for minimal deviation and decides how big the impact of a word has to be in order to be taken into account for the spamminess of the whole email. The exact impact of minimal deviation is explained further in section 3.4. The default value being 0.375, we will perform experiments with other proposed values reaching from 0.1 to 0.5 always incrementing by 0.1. Already at 0.375, very few words are considered for the overall spamminess of the emails. A higher min_dev than 0.4 will most likely result in unreliable results since most

of the message contents are ignored.

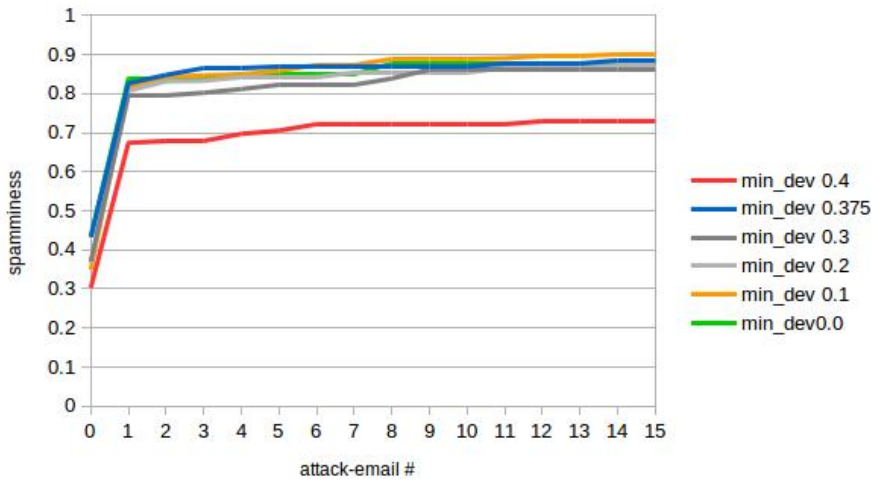

Figure 5.6: Comparing different values of min_dev

5.4.4 Changing the size of the attack emails

After having performed the basic changes for the environmental part of the learning process, we focus now on the first of the parameters concerning the attack-emails themselves. We will explore in what way the length, meaning the total number of words used per attack-email is influencing the overall manipulation. To generate the attack emails, we consider now only the list of words from the Helios templates. Making sure all the different words from the list are distributed equally over the entire set of attack emails is important to make optimal use of the given resources.

The length of the attack emails will be measured in relation to the average length of the legitimate emails, which is about 115 words.

The algorithm which generates the attack emails, takes as input:

- A: average number of words of legitimate election emails
- R: number of words per attack email / A.

A is fixed to 115 while we will vary R from 0.25 up to 4.00

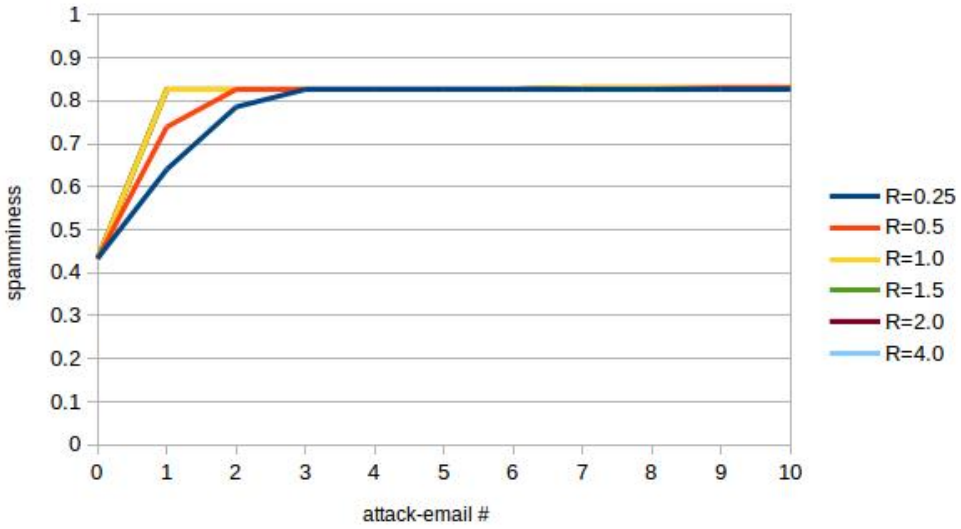

Figure 5.7: The influence of the length of the attack emails

5.4.5 Experiments including Bayesian poisoning

The attack-emails sent so far all contained only the words from Helios. Since these words are contained in the legitimate emails by default, we want to make sure to cover them all while sending the attack-emails. Now a legitimate Helios email also contains a message_body which contains information about the concerned voting. The words used for this part of the emails are what we consider natural language. Words from natural language are usually considered ham by a spam-filter.

There exist approaches that have as target to render the complete spam-filter useless, by sending attack-emails containing totally random words. This is called Bayesian poisoning. We could perform this kind of attack combined with the previous approaches we had, but we don't intend to damage the complete spam-filter. If we would poison the spam-filter of our victim completely, this would result in a victim checking his spam filter more often, which we don't want. We have to refine the general method of Bayesian poisoning in order to manipulate only the necessary parts of the spam-filter and leave the rest intact.

In general, we can not know exactly which words will be in the message_body because the host of the voting is free to put any message here. But knowing that the message will be about elections, referendums etc. we can come up with a refined list of words that have a chance to appear in the message_body. After doing some research, we gathered dictionaries containing words about the topics: business, economy, enterprise and voting.

We have the possibility to further influence the classification of legitimate emails by randomly adding words out of these dictionaries to our existing attack-emails. To generate the spam, we use the same algorithm as in section 5.4.4 with pa-

rameters A=115 and R=1.0 (which was the most effective) including now words from the above mentioned dictionaries in the attack-emails. To express how many words we include per word from Helios we will introduce a new parameter "rate". For example if we generate attack emails containing 114 words from Helios and have a "2X rate", the number of random words from the chosen dictionaries to be added to each attack email is 2*114 = 228 words
As for the subject and the header information like sender, we will be using another set of words, which is susceptible for being spam in general. This will make the victim(s) reject the message marking it as spam without even looking at it's content.

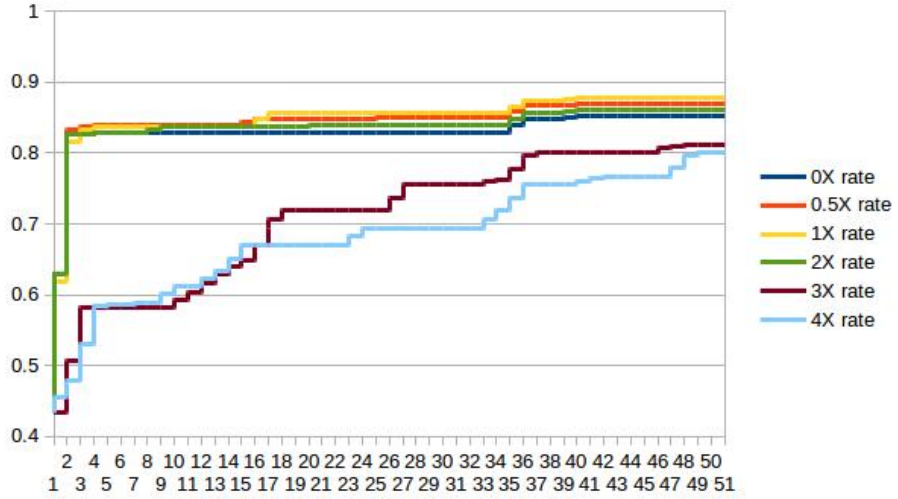

Figure 5.8: The influence of the length of the attack emails

5.4.6 Alternating dictionary and Helios attacks

Looking at the previous results from section 5.4.5 we notice that the higher the density of words from Helios is in our attack-emails, the higher the success rate. But we also prove that adding external words from the chosen dictionaries has a positive influence, even if it seems small.

The idea for this series of tests is to devote half of the attacks to the approach we had earlier under section 5.4.4 with A=115 and R=1.0 and the other half of the mails to our refined procedure of Bayesian poisoning. This means we alternately send one email with words selected randomly from our dictionaries followed by one email containing exclusively words from Helios.

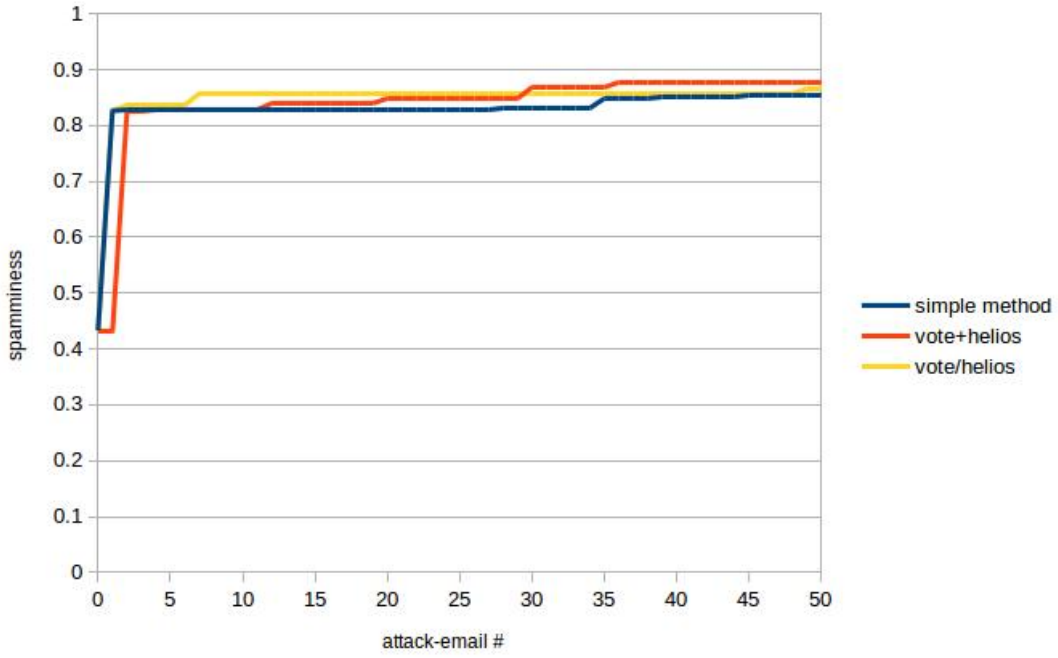

Figure 5.9: Alternating the attack-emails. In blue: the basic attack containing only Helios-words(as in section 5.4.4 with A=115 and R=1.0). In yellow: the approach combining Helios- and dictionary-words (as in section 5.4.5 with 1X rate). In red the current approach explained here in section 5.4.6)

5.4.7 Splitting up the attacks, minimization

First of all we want to find out how effective the Bayesian poisoning method is and how fast it shows results. We therefore perform the experiment, giving the blue line in figure 5.10, which shows the results from continuously attacking the spam-filter with emails that are purely poisoning and don't yet contain words from Helios but only words from the chosen dictionaries. This method takes advantage of the custom-message part of the legitimate Helios-emails. This custom-message part contains words from natural language but also very probably, some of these words are associated to voting, elections, economy and business. We try to cover the most words possible by choosing adequate dictionaries for the composition of the Bayesian attack-emails.

Secondly we perform a test, which is based on the first one (red curve in figure 5.10). We send 10 attacks with the same method of Bayesian poisoning as for the preceding test and then continue attacking like we did it in section 5.4.4. This means sending out 40 attack-emails containing words from Helios only.

Lastly we reduce the number of poisoning attack-emails to 3 and the Helios-based attack-emails to 2 which gives us the yellow curve in figure 5.10. Now looking at the stage after the 5^{th} attack-email, we trained the spam filter twice with emails covering all the words from the Helios templates which form the

corresponding dictionary. What we can observe at this point of the manipulation that the spam-filter already classifies our attack-emails as spam. This makes sense because they contain always contain the same words from Helios just in a different order. Therefore the emails that we send from now on will go directly into the spam filter.

This means, we don't require any more user interacting, in fact the victim won't even be able notice the attack anymore, unless he/she checks their spam-folder. This is a big advantage, since the spam filter will continue to learn from these emails (using an auto-update parameter, for the example of Bogofilter "-u"). This corresponds to a real world example where a user unknowingly lets the filter do its job including the automatic updates of the database, emerging from training on the already classified spam- and ham-emails.

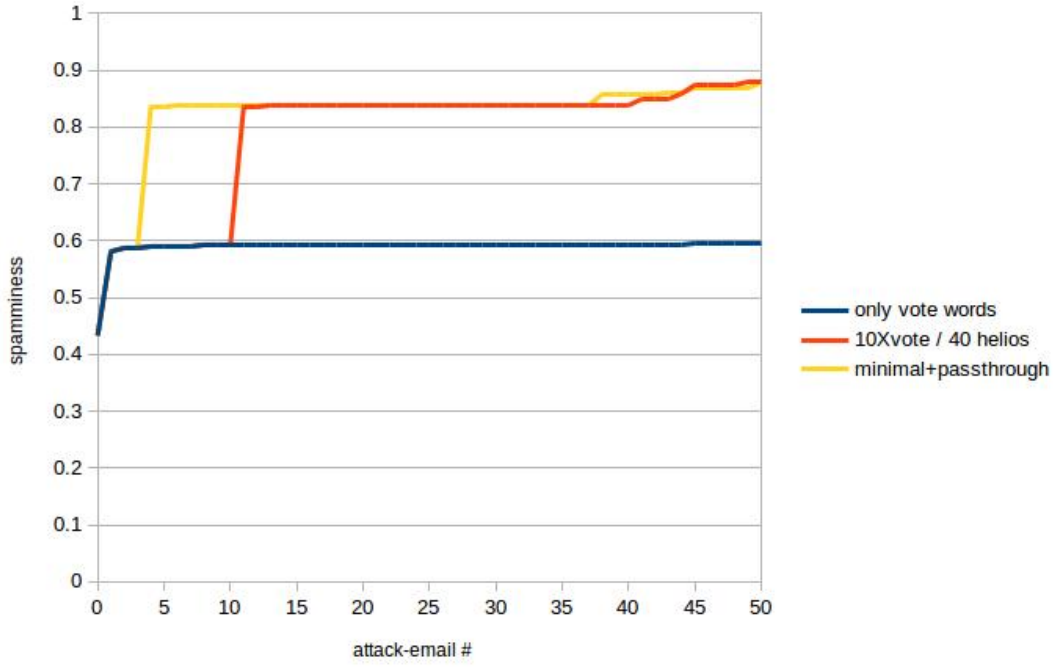

Figure 5.10: Splitting up and minimizing the user interaction

Figure 5.10 shows the average values over the complete test set of legitimate emails. This one included also the template "info" which consisted of about only 10 fixed words. This template showed up to harder to manipulate in a sense that the spamminess of those legitimate emails which follow the template "info" were barely affected by our attacks and did not reach a satisfactory spamminess level during our tests. Therefore we want to separate the templates to better visualize our success, looking at figure 5.11. We now took the average spamminess per Helios-template over the 4 different examples of legitimate emails. We also show the cutoffs for ham and spam.

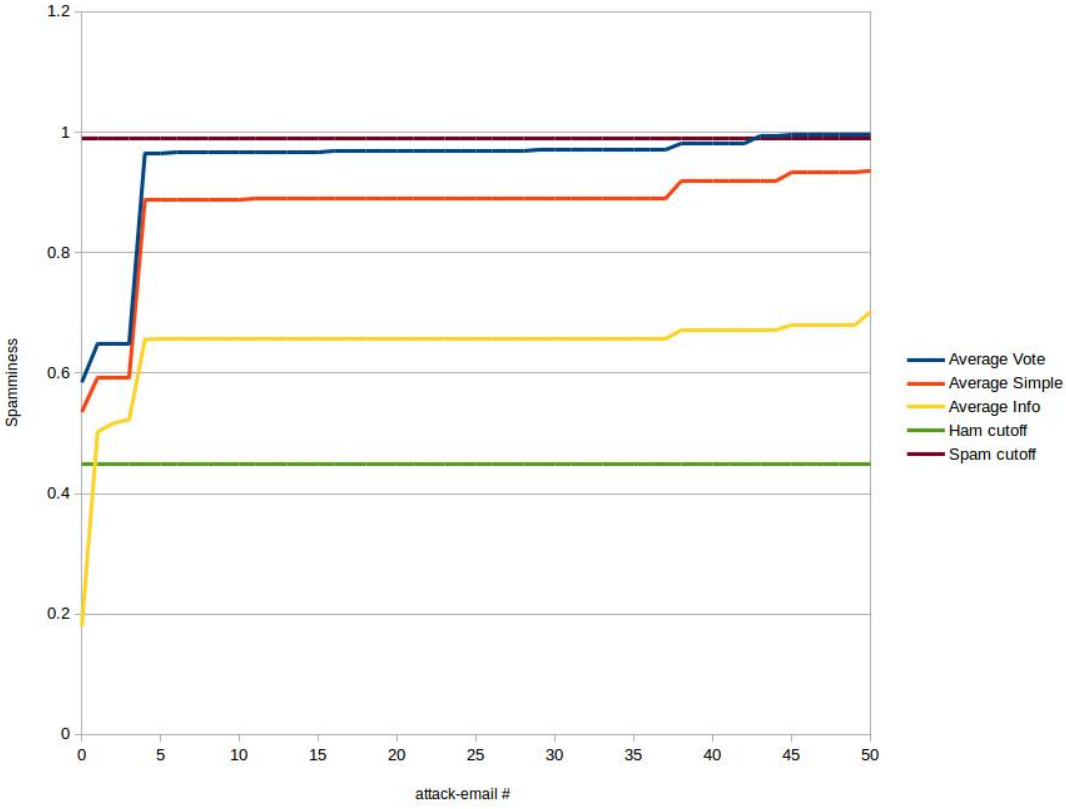

Figure 5.11: Splitting up and minimizing the user interaction

This now gives us the possibility to continue sending as many attack-emails as we want. Therefore we simulated an exhaustive experiment with 500 attacks. The resulting curve in figure 5.12 is showing the average over the spamminess of all legitimate emails which is tending to 1,00.

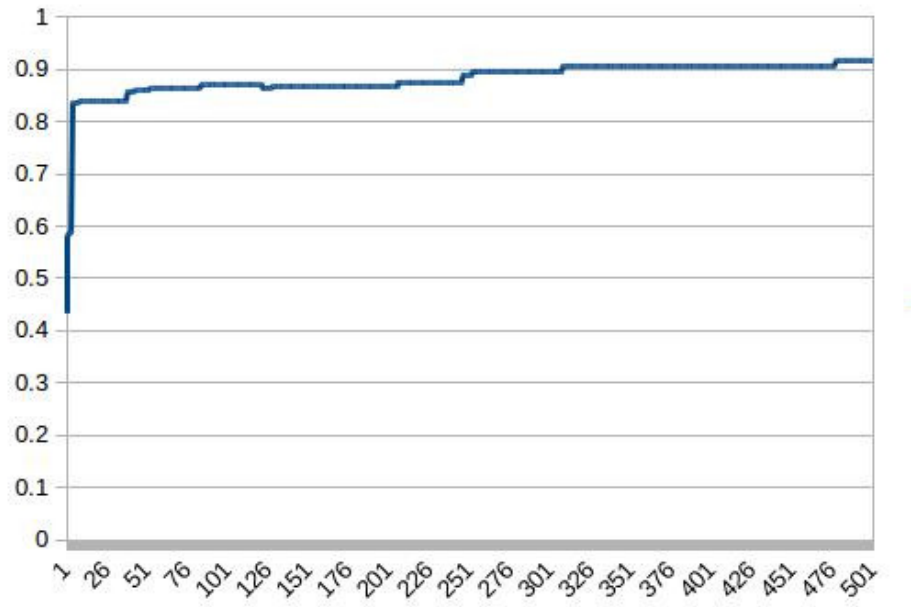

Figure 5.12: An exhaustive attack (500 emails), taking advantage of the self-learning strategy of filters.

5.5 Evaluation of Results

5.4.2 Training the spam filter with more or less big training-sets, influences the spam-filter dramatically, since its performance depends on its knowledge. The more legitimate ham- and spam-emails we use for the initial training, the more complete its database becomes which increases the accuracy.
This can be observed looking at the base classifications at attack-stage 0, before the first attack-email has been sent. The emails should be classified as ham (spammicity between 0.00 to 0.45), which is actually only the case when considering the complete Enron set of training-emails. Also we observe that the more extended the base-training is, the less impact our attack-emails have.
This makes sense, since the resulting database is growing, though also the probability that the words that we are using in our attacks have been encountered already quite often. Nevertheless we are able to convince even filters with large databases that the legitimate Helios-emails we use for testing are spam.

5.4.3 The minimal deviation (while below 0.375 which is the default value) should in general not influence the filter's outcome drastically but only the computation speed. This is positive for us because we can assure the reliability of our manipulations independently of the local user-variable min-dev.

In the experiments under section 5.4.4 we can observe that the filtering capabilities of our Bayesian filter depend on most of the words from the Helios-templates. The more words from the templates it knows the higher the overall spamminess of our legitimate Helios-mails. This can be seen when used R<1 , since then it took several attacks to cover the whole palette of Helios-words. For R>=1, we expected the emails to have more impact, once we start repeating the occurrence of Helios-words per email. What actually is the case is that the density of the words defines how much impact they have on the filter. Since we use the same set of words, just putting them twice in an email, will double as well the word count of the complete email. With this approach, the density of 1 can not be exceeded.

For section 5.4.5 the results are coherent with the experiments under section 5.4.4, proving that the density of Helios-words is important. We can see this with the 3X rate and 4X rate where where the importance of the words from Helios is definitely declining while increasing the overall length of the email.
But we can also see that adding words from our dictionaries shows success, comparing the blue and the yellow curves. The optimum is reached somewhere around 1X rate while somewhere between 2X and 3X rate both factors; the declining density of matching words, having negative influence, and the gain from adding words from natural language dictionaries equal themselves out.

The main idea of section 5.4.6 was to fight the decline in density of Helios-words in the attack-emails by evacuating the words that we originally just added into separate attack-emails. We can approve this behavior only partly, seeing that we can speed up the overall misclassification during the first 30 emails. So this procedure will be handy for the future, knowing that its impact is only significant over the first few attack emails.

In section 5.4.7 we mainly put our knowledge from previous experiments together. We conclude that the Bayesian poisoning should be performed separately from the attacks targeting Helios-words. We can also recover the observation that the influence of the Bayesian poisoning is only important for the first few emails. Therefore we limited this part of the manipulation to 10 attack-emails, and later on to 3.
Furthermore we take into account the natural behavior of a spam filter to consecutively learn from new emails and successfully take advantage of this. The implemented approach is based on only 5 interactions with the victim and can now be extended to as many attack-emails as we need to send to achieve our target. We provided an exhaustive experiment with 500 attack emails, which shows that we can constantly improve our results, with the average spamminess tending to 1.00 for the classification over all the 12 legitimate Helios-emails.

Chapter 6

Conclusion

6.1 Summary

While performing the tests, we discovered that it is fairly easy to manipulate Bayesian spam-filters as soon as one has understood the principle and how they operate. Our method of sending attack-emails based on word-lists is simple in terms of idea and execution. Getting the correct ingredients, meaning the necessary lists of words is crucial. The success rate depends on how many words are available and how big the template is in relation to the customizable text in the message_body of the legitimate emails. For strongly repetitive email-templates where only some strings are replaced, it is extremely easy to manipulate the spamfilter and attacks will reflect in almost instant success. For emails where we only have a small number of constantly used words, the attack may take more effort or could not show any success in the worst case. This is linked to the need to reach 0.99 spamminess for Bogofilter.

The experiments we did were with Bogofilter. If we switch the Spam-filter, the results can of course differ. In the beginning of the research we used another spam-filter, spamBayes, which actually responded in a similar way, meaning it was influenced highly by our attacks but we did not use it any further because of a lack in feedback and fragmentary insight to intermediate results. Bogofilter gives a lot of additional information which comes in very handy while trying to manipulate it. But in general, white-box systems are always easier to manipulate than black-box ones. Spambayes was more of a black-box with a rather incomplete documentation. Finally when looking at the outcome of the last experiment, we prove success after manipulation based on at most 5 emails which the victim had to classify by hand. After this period, our attack was autonomous and could have been continued even further without any more user interaction. But in general this was not needed since we got the important template "vote" blocked successfully after a bit more than 40 attack-emails. This can be seen in figure 5.11 visualized by the blue line which overcomes the spam-cutoff (0.99) in purple here. To be more precise, for the first legitimate email, it took 4 attacks to reach 0.99+ spamminess, for the second one 43 emails, third one again 4 emails and the last one 45 attempts. We can take some distance and pretend that in 50% of the cases we have immediate success while in some other

cases we need slightly more attempts. But generating and sending emails is a lightweight task creating no major effort on the performance side, which allows us to confidently consider even a higher number of attack-emails as acceptable.

6.2 Generalization

While looking at the bigger picture, we can assume that the methods implemented in this series of experiments can and will apply to a broader scene. We could easily replace the Helios-templates with other email-templates, determine new dictionaries and repeat the attack approach with similar success. Although the Bayesian poisoning will apply only to emails in the same language and concerning the same topic as the targeted emails. For other topics, other dictionaries would apply. This approach can be used to block any kind of email or show it's strength even in other messaging systems which make use of Bayesian filtering. It is astonishing how one can do unnoticed harm to a victim by simply sending emails containing certain words. This all relies on the simple dilemma that users are not experts. Once users better understand how to detect when they have become victim of targeted or random Bayesian poisoning, they will hopefully better respond to this attack and take the right actions which are to just delete the message and not mark it as spam. This can be achieved through prevention only. Although in theory Bayesian poisoning is reversible through an un-training or opposite training to the attack, this is not an option. The only effective remedy to a poisoned filter is a partial reset (if one can identify the time of occurrence of the attacks and has back-ups of the filter's database) or a complete reset of the database followed by a clean base training.

6.3 Future Research

As already stated, there is a need here to further test the methods presented in this paper on other spam filters and with real world email-accounts. The hardness of this problem is, that it involves interaction from users, meaning testers that receive emails without knowing and then at one point would be sent a control email. If they did respond to that one, the experiment having failed and if not, revealing success. One may argue that this could be done simulating all the components that are involved in a voting or other online procedures that rely on notification by email but there is always the factor of human psychology which is hard to determine. There are open questions like:

- How many people really check their spam folders and in what intervals?
- Would a victim recognize being attacked by such a method?
- May a victim rather delete a message instead of marking it as junk?

All these questions can only be answered with a rather large group (100-200) of independent and unaware testers. This approach would also offer a real chance

to evaluate spam filters that are not open source like the one implemented by Microsoft on their @live and @hotmail accounts.

Talking about the bigger filters these often linked to online databases, holding shared spam information. We could imagine a scenario where we get the sender of the official emails blacklisted, which would have negative consequences since it will not only block the emails to our victims but also to our allies who are supposed to be voting in our intend.

Bibliography

[1] Usability of Voter Verifiable, End-to-end Voting Systems: Baseline Data for Helios, Prt Voter, and Scantegrity II by *Claudia Z. Acemyan, Philip Kortum, Michael D. Byrne, Dan S. Wallach*

[2] The Prt Voter Verifiable Election System by *Peter Y. A. Ryan, David Bismark, James Heather, Steve Schneider, Zhe Xia*

[3] STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System by *Susan Bell, Josh Benaloh, Michael D. Byrne, ...*

[4] Online Voting Primer by *Jim Adler*. For more information: www.votehere.com/

[5] Helios: Web-based Open-Audit Voting by *Ben Adida*

[6] An Evaluation of Naive Bayesian Anti-Spam Filtering by *Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos, George Paliouras* and Constantine D. Spyropoulos

[7] Documentation of Bogofilter on the web. http://bogofilter.sourceforge.net/faq.shtml#unsure

[8] SpamBayes: Effective open-source, Bayesian based, email classification system. by *T.A Meyer and B Whateley*

[9] Scalable Centralized Bayesian Spam Mitigation with Bogofilter by *Jeremy Blosser and David Josephsen - VHA, Inc.*

[10] Filtering Spam With SpamAssassin at *HEANet Annual Conference 2002* by *Justin Mason*

[11] The Mozilla Thunderbird documentation can be found here: https://developer.mozilla.org/en-US/docs/Mozilla/Thunderbird

[12] Bachelor thesis: The effects of Different Bayesian Poison Methods on the Quality of the Bayesian Spam Filter SpamBayes by *Martijn Sprengers*

[13] A TREC along the Spam Trec with SpamBayes by *Tony Andrew Meyer*

[14] Papers: Introducing the Enron Corpus by *Bryan Klimt, Yiming Yang* Sources: From "Spam Filtering with Naive Bayes - Which Naive Bayes? " *http://www.aueb.gr/users/ion/docs/ceas2006_paper.pdf* - available dataset here : *http://www.aueb.gr/users/ion/publications.html*