Project Number: CS-CEW-0301

The Evolution of Spam and SpamAssassin

A Major Qualifying Project Report

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Jonathan Riedel
and

Zach Brown

Date: April 29, 2004

Professor Craig Wills, Adviser

Professor Mark Claypool, Adviser

Abstract

Spam is an annoyance familiar to most Internet users. As the amount of spam increases, many users receive more spam than legitimate email. Spammers change spam structure and spamming techniques because of filter deployment and spam filters change to detect newer spam. This project studies how spam and a popular filter called SpamAssassin have evolved over a two year period. This study reveals that SpamAssassin's effectiveness has not substantially changed over time as both it and spam have evolved.

Contents

1. Introduction

In the recent past, junk email, or spam as it is commonly referred to, has become an increasingly irritating problem for most people who receive email on a regular basis. In an effort to combat this problem, many users use spam filters. Soon after, people who send out spam, also known as spammers started to change the structure of the spam they were sending out in an effort to circumvent these filters. The filters also changed over time in an effort to keep up with the spammers. And thus began this seemingly never ending "race" between spammers and spam filters. In doing this project, we wanted to determine if there would eventually be a winner to this so-called race. Will spam ever be stopped? Will spam filters eventually become totally ineffectual?

There are a few main goals for this project. First, we want to identify how spam has evolved over time. We also want to look at the other side of the spam spectrum and analyze the evolution of spam filters. We realized that time would not allow us to look at how every type of spam filter has changed, so we decided to narrow the scope of this project to the evolution of SpamAssassin [8], a popular, open-source, rule-based spam filter. Along with these two main goals, we also tried to determine whether or not rule-based spam filtering alone was an effective method of spam filtering and how the addition of other types of filtering to rule-based filtering would affect the results.

In order to fulfill these goals, we collected spam that was received by a computer science student, two computer science professors and a system administrator at an engineering university. We ran this spam through several versions of SpamAssassin and analyzed the scores that were produced. We also analyzed the sets of rules for several versions of SpamAssassin and determined how they changed from one version to another.

By performing this analysis on how spam and spam filters have changed in the past, it will be possible to predict which direction spam and spam filters will go in the future. With this information, it may be possible for spam filters to stay one step ahead of spammers and possibly even put an end to spam eventually.

In the remainder of this report, we give an overview of spam, legitimate email (or ham as it is commonly referred to), SpamAssassin and other spam filters in Chapter 2. Chapter 3 discusses our initial explorations. Chapter 4 deals with the approach we took in order to accomplish the goals of the project. In Chapter 5, our results and the analysis performed on these results is discussed. Chapter 6 contains our conclusions and possible future work that could be conducted in this area.

2. Background

In order to better understand the work done in this project, it is necessary for the reader to have background knowledge in several areas. Namely, one needs to know about spam, ham and spam filters (SpamAssassin in particular). This chapter provides background knowledge in these areas that is needed to understand the rest of this report.

2.1 Spam

2.1.1 What Is Spam?

What is spam? According to Mail-Abuse.org [6], "An electronic message is "spam" IF: (1) the recipient's personal identity and context are irrelevant because the message is equally applicable to many other potential recipients; AND (2) the recipient has not verifiably granted deliberate, explicit, and still-revocable permission for it to be sent; AND (3) the transmission and reception of the message appears to the recipient to give a disproportionate benefit to the sender." Anybody who uses email on a regular basis most likely already knows what spam is. In the recent past, spam has become an annoyance that many people are forced to deal with on a daily basis, much like rush hour traffic or taking out the garbage.

2.1.2 History of Spam

While spam has only become a relatively big problem in recent years, it has been around for a long time. In fact, the first spam email was sent out over 25 years ago on May 3, 1978. It was on this date that a Digital Equipment Marketer named Gary Thuerk sent an email to everybody on the Arpanet on the West coast about an open house to showcase new models of the DEC-20 computer [7].

Spam is now commonly associated with email, but in the days before most people had email addresses, spam was prevalent in USENET groups. The first USENET spam was posted by Rob Noha on May 24, 1988 to every newsgroup he could find in an attempt to solicit money for college from other users [9].

While “spam“ was a common term among the MUD (Multi User Dungeon) community, the first known use of this term in USENET groups occurred on March 31, 1993 when a program that was supposed to aid in moderating forums malfunctioned and inadvertently posted approximately 200 messages to the news.admin.policy forum [9]. Chain letters and advertisements soon ran rampant within USENET and in the mid to late nineties, when email started to gain popularity, so did email spam [5]. Today, email spam is a huge problem. As of June, 2003, spam accounted for roughly 40% of all email traffic in the United States and results in an annual cost of over $10 billion to U.S. Organizations [4].

2.1.3 Different Types of Spam

According to Bright Mail [1], there are eleven categories of spam. Namely, these categories are products, financial, adult, Internet, health, scams, leisure, fraud, political, spiritual and other. These descriptions provided by Brightmail of these categories are contained in Table 2.1.

Table 2.1: Categories of Spam [1]

Product	Email attacks offering or advertising general goods and services.
Financial	Email attacks that contain references or offers related to money, the stock market or other financial opportunities.
Adult	Email attacks containing or referring to products or services intended for persons above the age of 18, often offensive or inappropriate.
Internet	Email attacks specifically offering or advertising Internet or computer related goods and services.
Health	Email attacks offering or advertising health-related products and Services.
Scams	Email attacks recognized as fraudulent, intentionally misguiding, or known to result in fraudulent activity on the part of the sender.
Leisure	Email attacks offering or advertising prizes, awards, or discounted leisure activities.
Fraud	Email attacks that appear to be from a well-known company, but are not. Also known as "brand spoofing" or "phishing", these messages are often used to trick users into revealing personal information such as email address, financial information and passwords.
Political	Messages advertising a political candidate's campaign, offers to donate money to a political party or political cause, offers for products related to a political figure/campaign, etc.
Spiritual	Email attacks with information pertaining to religious or spiritual evangelization and/or services.
Other	Emails attacks not pertaining to any other category.

2.1.4 Spamming Techniques

Over the years, spammers have become crafty in an effort to get their message (whatever it may be) out to as many people as possible. In the past, the main goal of spammers was to grab the recipient's attention and get them to read the message. To do this, methods such as putting the product name in all caps in the subject line or using lots of exclamation points or bright colored text were commonly used. With the advent of spam filters, the objective of spammers seemed to change from getting their message out to an audience to getting their message through the filters. This is logical, since a message that gets caught by a spam filter basically has no chance of being read by the recipient. In order to get by spam filters, spammers have tried techniques ranging from misspelling words to adding poetry to their emails to hijacking other people's email addresses and using them as their own.

2.2 Ham

2.2.1 What Is Ham?

Since junk email has become known as “spam“ and spam also happens to be a processed ham product produced by the Hormel Foods Corporation, all other email has come to be known as “ham“ [2]. It is difficult to classify the traits of ham, since one person's ham can vary so much from another person's. In some cases, it is actually feasible that one person's ham could resemble another person's spam. The structure of ham could even vary within one person's inbox. For example, one person could be receiving personal messages, emails from mailing lists that they have subscribed to and legitimate business messages. While these are all classified as ham, the structures of these different types of ham differ from each other. This is one of the reasons that spam filtering is so difficult. One can't just say “This is what ham looks like, don't let anything else by.“

2.3 Spam Filters

2.3.1 Vipul's Razor

Vipul's Razor is another type of spam filter. It is essentially a large, constantly updating catalog of spam that is referenced by email clients [7]. When an email comes in, the email client checks the database to see if the characteristics of that email match the characteristics of the spam in the catalog. It then determines whether or not the email is spam.

2.3.2 Bayesian Filters

Bayesian filters are a type of learning filter originally applied to spam by Paul Graham [3]. Bayesian filters take some time to “train“, but after an initial period of time, they can be successful at catching spam. Basically the way Bayesian filters work is whenever the user receives a piece of spam, they mark it as spam. The filter looks at all the ham and spam received by the user and uses a modified version of Bayes' Rule to figure out which words are likely to appear in ham and which words are likely to appear in spam [3].

An advantage to these types of filters is that they are tailored specifically to the email received by each individual user. As we mentioned previously, everybody's ham looks different. Well, different people also receive different looking spam to a certain degree also. With Bayesian filters, this distinction does not matter since the probabilities of an email being spam or ham are based on the email that user has received in the past.

Bayesian filters also have their disadvantages though. First of all, unless a user receives an enormous amount of spam, it takes the filter a while to learn which words are ham words and which words are spam words. Another downside is that the user still has to sift through all of the spam that they receive. With rule-based and other types of filters, the user has the option to automatically send their spam to a separate folder or even delete it right away. But in order to get a Bayesian filter working as well as possible, the user must look at each email (or at least the subject) and decide whether or not it is spam during that initial time period. Also, if the structure of the user's ham is not a great deal different from the structure of the spam they receive, it could result in lots of spam being flagged as ham (false negatives) and ham being flagged as spam (false positives).

2.3.3 SpamAssassin

SpamAssassin is a popular, open source, rule-based spam filter. It uses different types of tests including tests that analyze the text of the subject and body of emails, tests that analyze the headers and tests that check many common blacklists (such as mail-abuse.org and ordb.org). SpamAssassin also utilizes Vipul's Razor and a form of Bayesian filtering [8]. Vipul's Razor and Bayesian filtering are explained later in this chapter. Basically the way SpamAssassin works is it runs a series of tests on every received email and assigns a score to the email. For example, one test might look for the phrase “FREE!!!“ in the body of the email. If this phrase is found, a point may be added to the total score. There are also tests that assign negative points to the email. For example, if the sender is whitelisted, 10 points might be subtracted from the total score of the email. If the total score of the email is higher than a predetermined threshold (5 points by default), then the email is labeled as spam.

We chose to focus the scope of our project on SpamAssassin for several reasons. First, it is a popular filter and is available to us. Second, it is a comprehensive filter that incorporates a wide variety of tests. Third, it has been around for a few years, making it easier to see how it has changed and which direction it is going in. Finally, we chose to focus on SpamAssassin because it is open source and older versions are readily available to download and install. This feature was especially useful to us, since we were looking at how spam had evolved over time.

2.3.4 Blacklists and Whitelists

In addition to Bayesian filters and rule-based filters such as SpamAssassin, there are a wide variety of other types of filters. A common method of spam filtering that is often used in conjunction with other types of filters is blacklisting and whitelisting. These filters can be effective, especially in reducing false positives. Basically, blacklists are lists of known spammer email addresses and whitelists are lists of known ham email addresses. So for example, a user might add their mother to a whitelist to ensure that all emails sent from her get through (unless, of course, their mother is a spammer). On the other hand, if a user receives a piece of spam, they might add the address of the person who sent it to a blacklist. This is generally a better method of deterring false positives than it is for deterring false negatives for the sheer fact that spammers change email addresses frequently while most people sending ham do not.

2.4 Summary

While spam has existed for over a quarter of a century, it has grown to be an expensive and irritating problem in the recent past. In order to combat this problem, many people have begun to use spam filters. There are several categories of spam and each of these categories is structured differently. As a result of this, different types of spam filters have been developed in an effort to catch as much of each category of spam as possible. This project is mainly concerned with SpamAssassin, which is an open-source rule-based filter which incorporates other types of filters such as a Bayesian filter, Vipul's Razor, Blacklists and Whitelists.

3. Initial Exploration

In the early stages of this project, our goals were not yet clearly defined. During this time period, we performed several types of analysis which did not end up directly tying in with the overall purpose of our project, but could still serve as useful information. The results of this work are contained in this chapter.

3.1 Size of Email vs. SpamAssassin Score

We looked at spam at the word and character level rather than looking for specific phrases or rules. It is important to know if there is some pattern in the basic structure and length of spam. These attempts to find an overall structure pattern failed to provide useful results.

3.1.1 Word Counts

We attempted to find some relationship between word counts and spam messages. The computer science student and professor spam samples we ran provided no clear patterns for these counts, they appeared to be completely random. Figure 3.1 shows one of our result charts. The remainder of the charts are available in Appendix A.

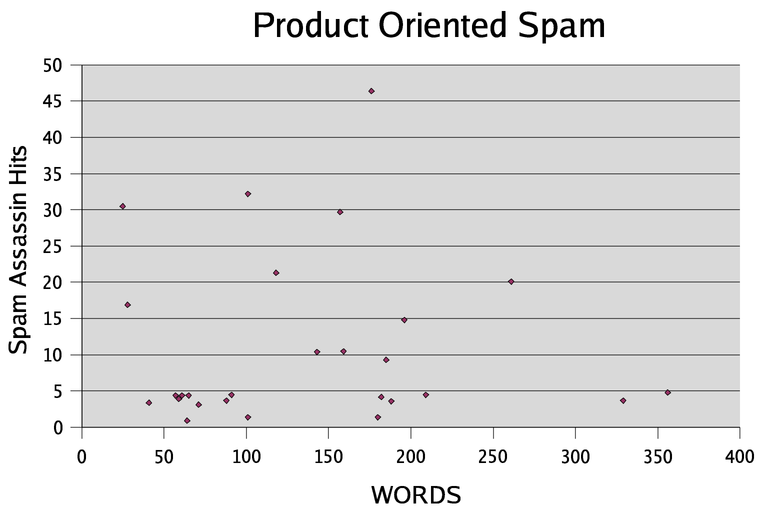

Figure 3.1. Relationship Between SpamAssassin Scores and Word Counts for Product Oriented Spam

We were able to get size estimates for each of the typical types of spam. Product oriented spam was typically around 200 words. If it relied on a keyword for that product, it was reduced to about 100 words. Message oriented spam was around 500 words long. The results were broad enough that they do not suggest any conclusions to draw, especially considering ham can range from extremely short to long.

3.1.2 Character Counts

Character counts are closely related to word counts; the more words the more characters. The result of this relationship is that using character counts did not provide any additional data to that provided by our word counts.

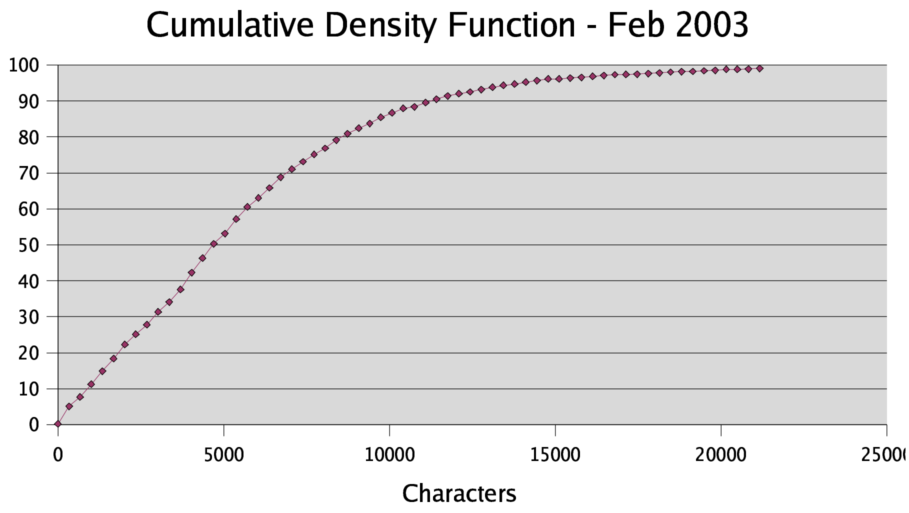

Figure 3.2. February 2003 Cumulative Density vs Spam Character Count

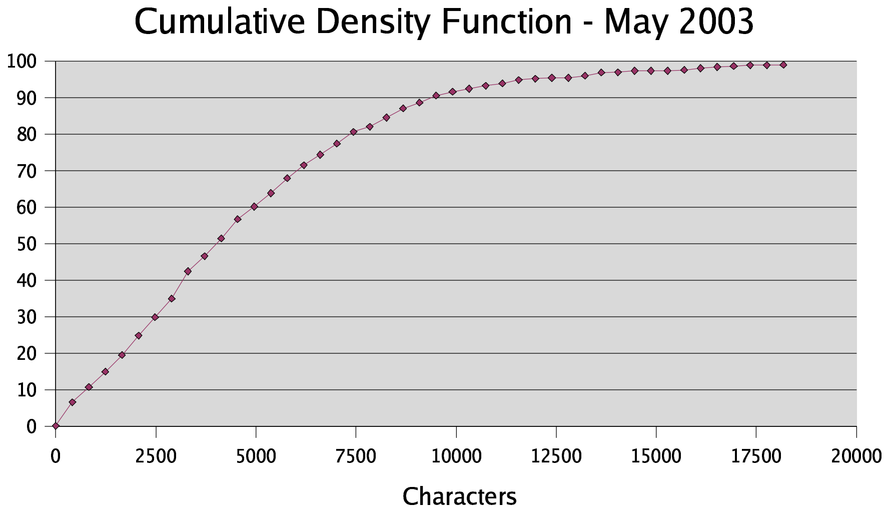

Figure 3.3. May 2003 Cumulative Density vs Spam Character Count

Figures 3.2 and 3.3 illustrate the average character counts for a computer science professor's spam samples from February and May of 2003. The average count for February was approximately 4700 characters. The count for May was 4100 characters. This change hints at a decreasing size of messages but is inconclusive. There were too many possible variables, and these messages were not sorted by type of spam.

3.1.3 Possible Sources of Error

The lack of a pattern in our counts may have been due to the fact that our tests did not differentiate between HTML tags and words, we treated them both the same. We reasoned this would not have a negative effect on the results because some spammers would attempt to add tags to their messages to confuse filters. Another potential problem with this method is the use of HTML compatible mailers. Ideally the results would need to be separated based on HTML use to see any potential pattern. Since there was no indication of a pattern though, we decided not to pursue this path further.

3.2 Reducing SpamAssassin Scores

We looked at different ways spammers could possibly reduce the scores of their emails. With the inclusion of Bayesian filtering in newer versions of SpamAssassin, we theorized that adding lots of "hammy words" or words that do not usually appear in spam could possibly lower spam scores. We also thought it would be easy for spammers to look at the rules which assigned negative points and simply adjust their spam until it followed these rules.

3.2.1 Hammy Words

In order to test this hypothesis that spammers could lower their scores by taking advantage of the Bayesian component of SpamAssassin and adding lots of words that do not usually appear in spam, we employed two different methods. First, we copied and pasted excerpts from online books into the body of several spam emails. After doing this with several different excerpts, we soon realized that none of them were lowering the score at all. This could be due to spammy words possibly being contained in the excerpts we chose. We then tried adding one hammy word multiple times in the same email. We chose to use the word "starfruit" since neither of us had ever seen that word in any spam we had ever received. This lowered the score slightly, but not a significant amount.

In carrying out these tests, there were several possibly sources of error. First off, we could not get the Bayesian component of SpamAssassin working properly. This is a significant source of error, since none of the results are reliable without this component working properly. In the single word test, we inserted the hammy word approximately 500 times. The results may have been more conclusive if the word was inserted more times than this, possibly several thousand times.

3.2.2 Tests with Negative Points

We assumed that it would be simple for spammers to lower their scores by taking advantage of tests that assign negative scores. Luckily the SpamAssassin developers had already realized this as evidenced by the lack of these types of rules in current versions of SpamAssassin. Older versions of SpamAssassin did contain more rules that were easy to take advantage of. For example, there were rules that subtracted points from emails that contained long lines of text or the phrase “Order Status.“ Most of these rules have been removed in recent versions. The remaining negative point tests are mainly related to whitelists.

3.3 Ham Analysis

We did not study ham due to a lack of available samples. Instead, we worked from the assumption that ham has not changed much over time. The main change came with the introduction of HTML into mail programs. As more programs, and free email accounts, have converted to HTML the amount of HTML tags have increased in the messages. Fortunately these were first introduced before our chosen test versions, so the effect should not be too drastic. There were HTML related tests as far back as version 1.0 of SpamAssassin, and we used 2.31 or later for our tests. We expect that the increased use of HTML increased at about the same rate for both spam and ham, and that SpamAssassin's rules would reflect it.

Table 3.1. Percent of Tests For HTML Related Aspects of Messages

Spam Assassin Version	Version 1.0	Version 2.0	Version 2.4	Version 2.5	Version 2.63
Percent of Tests	18	12	16	24	28

As Table 3.1 indicates, the relative amount of HTML related tests has been fairly consistent. There was an increase in both versions 2.5 and 2.63, which likely correlates with when HTML became more of an issue.

3.4 Summary

While the types of analysis described in this chapter did not prove to be beneficial in accomplishing the goals of this project, they are part of our work and are documented. This analysis may not be directly related to the goals of the project, but it did lead to the realization of these goals. This work could be useful in future work completed on this subject.

4. Approach

This project's initial goal was to determine if spam could be stopped and how close the Internet community was to achieving this goal. To study this, we had intended to approach the issue from the perspective of a spammer and try to get messages through existing filters. Unfortunately, it was impractical to attempt this without a thorough knowledge of how spam and spam filters had changed since their inception to the present. We decided we would enable others to achieve our goal in the future by doing the necessary study of spam and filter evolution.

To take this approach, we chose to exclusively study one specific filter. The obvious choice was the open source filter SpamAssassin [8]. According to a personal conversation with one of the developers, Theo Van Dinter, SpamAssassin is the number one spam filter and other filters often compare themselves to it. Its prominence in the field makes it an excellent filter to study. The best way to study how spam and SpamAssassin had changed was through obtaining spam from different time periods and processing the samples under controlled conditions. One of the primary conditions was to test the samples with different versions of SpamAssassin. This approach will tell us how spam and SpamAssassin have evolved.

SpamAssassin is currently used on some of the WPI mail servers and is available on the CS and CCC machines. Additionally, since it is open source, it is easy to install and access the various rules. This enabled us to control which versions we used and how we tested the messages. Additionally, many of our sample messages were already processed by SpamAssassin, providing a control set of data to verify our test results against.

Another reason we chose to use SpamAssassin were the tags it adds to email headers. These tags allow us to see what tests each message failed and how they impacted the score. Also, with the latest version we were easily able to look up explanations of the tests on the SpamAssassin website [8].

4.1 SpamAssassin Detail

Version 1.0 of SpamAssassin was released in September of 2001 and work has continued until the present day. The current release is version 2.63, and there is a “bleeding edge” version available. This version, 2.70, is referred to as “bleeding edge” because it is currently in development with frequent builds. Since this version was in development at the time of this project, it was potentially unstable. This could have skewed our results, so we decided not to use it.

In order to run a controlled series of tests on how SpamAssassin's performance has changed, we needed to obtain different sample versions. These versions would act as a snapshot of SpamAssassin during the time period in which they were released. In order to choose these versions, we had to look closely at the changes in each version. The only way to make an educated choice on test versions is to understand what changed. This understanding would better enable us to study the performance of SpamAssassin and understand how it evolved over time.

The most obvious difference between the versions is that each version has different tests and rules included. These rules fall into five major categories: sender related, subject content, body content, HTML/link related and header related. These are listed in Table 4.1.

Table 4.1 The Five Different SpamAssassin Test Categories

Sender Related	Tests that verify the sender related header information. These often verify the sender's address against whitelists and blacklists.
Subject Content	Tests that examine the email subject line. These look for key phrases or techniques that are used by spammers.
Body Content	Tests that examine the message body. These look at word usage, keywords, and message format.
HTML Related	Tests that handle HTML and links in the message body. These tests check the amount of HTML used, how it was used, and any links in the message body.
Header Related	Tests that verify the header information. This verifies the send date, routing information and other header fields.

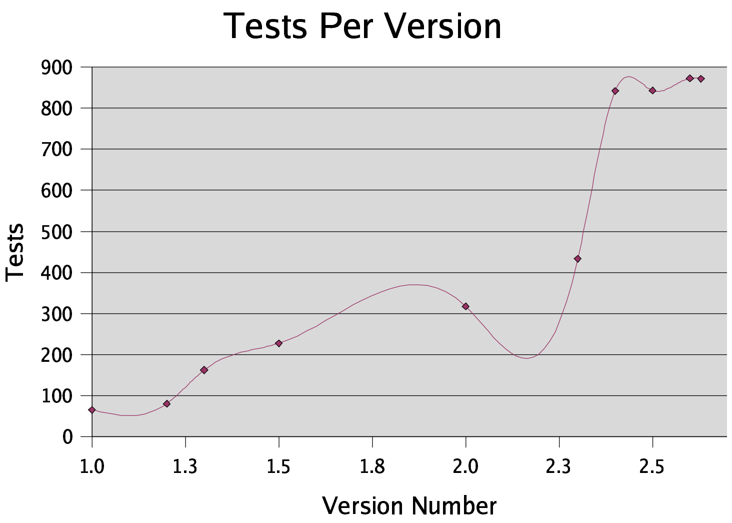

Figure 4.1. The Number of Tests Per SpamAssassin Version

Each version of SpamAssassin had tests that were added and removed. Figure 4.1 illustrates how many new tests were added in ten different SpamAssassin versions. Clearly, the number of tests steadily increased with each version. Version 1.0, the first major release, had 65 tests. Figure 4.2 shows what proportion of the tests fit into each category for five different versions. The majority of these dealt with content, although the effort was fairly evenly spread throughout all the different categories. The only section that was not widely represented was the subject content tests.

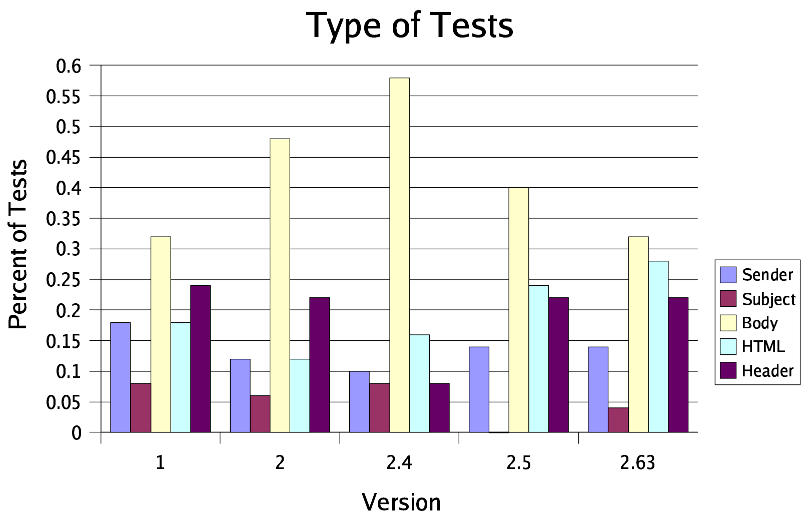

Figure 4.2. Distribution of Type of Tests in SpamAssassin

After version 1.0 only six rules were removed for version 2.0. Half of these rules were sender related and the other half content related. Version 2.0 primarily added rules, bringing the total rule count to 317. At this point roughly half the rules were content related. Roughly a fifth of the rules then dealt with the headers, which is relatively close to the previous version.

For version 2.4, 77 rules were removed. Despite this, the overall number increased to 842 rules. Unsurprisingly, most of the removed tests dealt with message content. A noticeable amount of HTML or link related rules and header related rules were removed as well. After considering the added tests, this version was still dominated by content tests, but the second most prevalent type was HTML and link related. The remaining types were close in number.

262 rules were removed for version 2.5, yet these were mostly replaced with the overall rule count increasing by one to 843. Unsurprisingly, most of the removed tests were content rules. Additionally, Link and HTML related rules, as well as header related rules were some of the more common ones removed for this version. That did not stop these rules from becoming more more prevalent overall though. The content based rules were still in the majority, but only 40 percent of the total rules. It is surprising to see that virtually all the subject content rules appeared to be removed, but this allowed the remaining three categories to carry about an equal amount of weight. Still, sender related rules were the lowest, which is logical since this would likely be taken care of by black and white listing.

The final version we looked at was 2.63, the latest stable release. Only 149 rules were removed, and the total count was increased to 872 tests. The majority of the removed tests were sender related, which is surprising considering it was less common already. The remainder of the removed rules came about equally from content, HTML and headers. After these tests were removed, and new ones added, the breakdown of the tests was rather surprising. While still in the majority, content rules no longer dominated the tests. Content rules were about equal with HTML and link rules in this version.

4.2 Chosen SpamAssassin Versions

Now that we had a basic understanding of what changed over time, we were able to decide on five specific versions to test. The versions we studied were:

Version 2.31, released 06/20/02
Version 2.43, released 10/15/02
Version 2.50, released 02/19/03
Version 2.55, released 05/19/03
Version 2.60, released 09/23/03

Version 2.31 was chosen because it was used on the WPI mail servers for a substantial amount of time. This meant our samples would allow us a glimpse of the actual time period, including the state of the whitelists and blacklists that SpamAssassin accessed. 2.43 was chosen for a similar reason, it was already installed on the WPI mail servers allowing us a version we could test before installing our own copies.

We chose version 2.50 because this is when SpamAssassin started making some major changes. The most notable change was the attempt to implement Bayesian filtering to allow SpamAssassin to evolve with time. Unfortunately Bayesian filtering was not fully implemented until version 3.0, which was outside the scope of this project. Version 2.60 was chosen because it was a recent stable version with a substantial amount of new rules introduced. It also correlated nicely with the end of our sample data. Lastly, 2.55 was chosen because it was the midpoint between the previous two choices. We were hoping the results from this test version would show how SpamAssassin changed from 2.50 to 2.60.

4.3 Installing SpamAssassin

In order to accurately test how SpamAssassin changed over time, we needed to install different versions which we could test. The first two versions, 2.31 and 2.43, were already available for our use. The remaining three were installed on our own server. These were installed locally for different user accounts. We did this so they would not interfere with each other and so we could ensure that they would function correctly.

We followed the published non-root install functions. Each version of SpamAssassin contained the following instructions in the INSTALL file:

```
To install as non-root, do something like this:

[unzip/untar the archive]
cd Mail-SpamAssassin-*
perl Makefile.PL PREFIX=~/sausr SYSCONFDIR=~/saetc
[option: add -DSPAMC_SSL to $CFLAGS to build an SSL-enabled spamc]
make
make install”
```

4.4 Data Collection

In order to test the different SpamAssassin versions we needed to obtain multiple data samples. These samples came from four sources: two WPI computer science professors (referred to as Prof1 and Prof2), a WPI system administrator (referred to as SysAdmin) and one student (referred to as Student). The professors and the system administrator were our primary samples. The student did not have much of a spam collection, and was used primarily because he was the only available source for ham messages. Each of these samples consisted of spam collected from the middle of a specified month.

Table 4.2 Number of Messages in Each Spam Sample

	08/15/02	***10/15/02***	***12/15/02***	***05/15/03***	***09/15/03***	***11/15/03***	***12/15/03***	***01/15/04***
Prof1	55	56	62	62	62	62		
Prof2			33	134	81	172		
SysAdmin			102	103	104	110		
Student						138	135	143

As indicated by Table 4.2, our samples ranged from August 2002 until January 2004. We attempted to have as much data overlap as possible, to verify our results, but only had four periods in which we had samples from our three primary sources. The result of that is that December 2002, May 2003, September 2003 and November 2003 were our primary time periods to examine.

Table 4.3. Number of Messages in Student's Ham Samples

Sample Date	***06/15/03***	***09/15/03***	***11/15/03***	***12/15/03***	***01/15/03***
Regular ham	23	30	61	77	55
Mailing Lists			90	75	52

We also had various ham samples provided by Student. We believe it was important to have ham samples to compare our spam results with. Studying the effectiveness of filters is less useful when the performance of the filter on ham is not considered. A filter which falsely marks ham as spam will typically result in the user having to look through their spam to ensure no ham was caught, therefore defeating the purpose of filtering by forcing the user to look at their spam anyways. Figure 4.4 lists the versions and size of the ham samples we had. These are all from the same source, Student, but we divided them among regular ham and mailing lists since they have so many differences. Unfortunately, the limited amount of samples we had, and the fact that they were all from one source, does not allow us to perform many tests or draw many conclusions.

4.4.1 Sample Size

Our sample sizes are much smaller than the typical SpamAssassin development run. This was done for two primary reasons, we were limited in our sources of spam and in processing power. Prof1 manually sorted his email, making it time consuming to gather each sample set. To counter this, we requested smaller sample sizes from him. There was the additional issue of processing ability as well. We ran the majority of our tests on our personal computer. This was an 800 mhz Linux computer with 256 MB of RAM. This meant that it took a substantial amount of time to run most tests.

4.4.2 Data Format

The data was in various formats, some of which was difficult to work with. Prof1's samples contained each message in an individual file and these files were unmodified. The messages were exactly as his mail program had received them. Student's samples were also saved in individual files, but these had been modified by SpamAssassin. SpamAssassin had added in spam reports which needed to be removed before any processing was done.

Alternatively, Prof2 and SysAdmin's files were more difficult to handle. These samples consisted of messages saved in one single, large file. These messages were modified, like the Student's, by SpamAssassin. In addition to having to remove the various spam reports these files had to be broken apart. Each individual message had to be extracted and saved to a new file. This procedure had to be done because SpamAssassin does not support a large file with multiple messages. It assumes each file is one message.

4.5 Sample Preparation

As mentioned, the messages needed to be converted into a specific format to be used. The first issue, having one large file, was solved with the use of a python script. This script, which is available in Appendix C, parsed the mail file for the string “Received: from mail1.WPI.EDU”. This was the line that every new message in Prof2 and SysAdmin's samples started with. It is important to note that this line would not work for dividing samples from sources other than the WPI mail server. It varies depending on the server, but it will typically be a similar format.

Once that string was found, every line up until the next occurrence of it would be saved into a numbered file. The file names we generated from a counter keeping track of how many messages were in the file. Each file name consisted of the current message count with ".eml" appended. Since it was a message counter that increased with each message it was guaranteed that each message from a specific sample would have a unique name. Once the entire file was read, the script would conclude.

The second issue which had to be dealt with involved the tags and spam reports inserted by SpamAssassin. These lines could modify the score received by the message as it was scanned by SpamAssassin because SpamAssassin was not aware that they were added to the message. This could have led to our results being skewed in favor of SpamAssassin's effectiveness. The previously computed scores would be read into SpamAssassin and that might result in the message being scored higher. Additionally, any messages that took size or word counts into consideration would be inaccurate because the word count would be larger. To prevent this from influencing our results, we wrote another script to remove these added lines. This script looked for specific phrases to replace with an empty string. This task was further simplified by the reports written by SpamAssassin to the files having clearly defined strings to label them.

After completion of those two scripts the samples were ready to be tested with the different versions of SpamAssassin. They were now in the format in which SpamAssassin would have seen the files when they were originally mailed. It is important to note though, that we maintained copies of the original message in case the data was corrupted during testing. It is important to protect against data corruption, and having a control sample can be useful in future work.

4.6 Testing Procedure

Testing the samples required a shell script to process the data.

```
for FILE in *;
do spamassassin -L < /FILE_PATH/$FILE > /FILE_PATH/VER_SAMPLE/$FILE;
done
```

All the script did was call SpamAssassin with each file in the directory as input. To simplify processing, the shell script was in the directory with all the files and we hard coded the file path and output path.

SpamAssassin has many different options available to the users, all of which are shown in Appendix B. The option we were primarily concerned with was -L. This option was used to disable non-local spam assassin tests. We made this decision since using the online tests substantially increased processing time and introduced extra variables. We also had no way of accessing old versions of the blacklists and whitelists that these tests referred to. Since we did not have access to the original lists, the weights for these tests would be inaccurate since there was no guarantee that the content was the same.

For arguments, SpamAssassin was given the complete path to each file we wanted to test. This was what each of spam assassin's tests were ran against. After running, the results were sent to an output file of the same name in a new directory named after the version we were testing and the sample we were running. We followed this procedure so it would be easy to process the output data.

In certain cases it was necessary to use the -P option to properly save the output. This option causes SpamAssassin to save the output to a specified output file. This was an important issue when we tested version 2.31, since this version sends the output to std::out rather than to the output file. This meant that when default conditions were used, none of the results were saved and that we could not examine the data. Fortunately this was remedied by the -P option.

4.7 Output Processing

Once each sample was run through SpamAssassin, the data needed to be extracted from the created files. This was done using a set of python scripts –

single_ave.py and single_detail.py. The core functionality of these scripts was the same in that they loaded each file and searched for the number of hits that particular message received. Following that they processed the data and saved the results to disk. Each script was called by "./script_name directory_path" and could easily have been combined. The combined version is included in Appendix D.

Single_ave.py calculated and recorded the average, median and number of messages in the sample. The average and median were not essential since we were looking at the percent of messages scoring more than X hits, where X was the SpamAssassin threshold. The values for average and median did help by presenting another perspective on the data. Still, Single_detail.py was much more useful in that it recorded how many messages scored each integer value. It rounded the hit score for each message down to the nearest integer and counted how many times each occurred. This script made calculating the percent above X hits simple.

Additionally, on the chance that we were given a single file output, we constructed versions of these scripts to process that format as well, mass_ave.py and mass_detail.py. These were not needed, but could possibly be useful in future expansion on the topic. These scripts were essentially the same as the previous script, only instead of processing file by file they processed the data line by line.

4.8 Summary

Our primary way of studying the evolution of SpamAssassin was to examine different versions of SpamAssassin. We wanted to look at how effectively this filter performed during different time periods. By seeing how SpamAssassin performed during each time period, we would be able to observe the changes in its performance over time. These changes would allow us to see if it made progress in reducing the amount of spam users were receiving.

In short, we chose five different versions of SpamAssassin covering a two year period of time. We tested several spam samples and extracted the score SpamAssassin gave it. These scores are what we use to evaluate how SpamAssassin and spam have changed over time.

5 Analysis

This chapter focuses on analyzing our results. Initially we examine the performance difference between using local and non-local tests. Once that is explained, we analyze how the effectiveness of SpamAssassin changed with each version. Following that we simulate the "real world" performance of SpamAssassin. Lastly we touch on how the various thresholds influence the amount of spam that bypasses the filter.

5.1 Local and Non-Local Tests

Ideally, SpamAssassin would catch 100 percent of all spam and mark no ham messages as spam. Currently SpamAssassin has not succeeded in those goals. This is especially true in the case of our results since we were not using SpamAssassin to its full potential. To achieve its full potential, we would need to use the "non-local" tests in addition to the "local" rules. The "non-local" tests primarily consist of centralized whitelists and blacklists. These are accessed via the Internet instead of being stored locally on the user's computer. This allows them access to much more data than any individual user is likely to get. "Local" tests are those which are stored on the user's computer and are always accessible regardless of whether an Internet connection is present or not. Without the non-local tests, the average of hit scores are three points lower than those that would be observed through a typical email provider. That amount of points could easily bring most of the message above the threshold of five.

To illustrate the difference that arises from using non-local tests, we looked at six samples. The first two samples were from Prof2. One of these samples was from May 2003 and the other was from November 2003. These were chosen because they had been processed by SpamAssassin version 2.31 when they were first received by the Professor. Additionally these were the two largest samples from Prof2. For consistency we chose the same months from SysAdmin. His samples had also already been processed when they were received. Lastly, to provide a different perspective, we chose the two student samples which were processed by SpamAssassin version 2.6. One of these happened to be the same time period as a sample from Prof2 and SysAdmin.

Table 5.1. Impact of Non-Local Tests on Average Scores

Sample	***Version***	***Average Scores For Local Tests Only***	***Average Scores Using All Tests***	***Score Change***
Prof2, May 2003	2.31	9.58	13.01	3.43
Prof2, Nov 2003	2.31	6.5	9.14	2.64
SysAdmin, May 2003	2.31	12.86	10.92	-1.94
SysAdmin, Nov 2003	2.31	12.12	12.64	0.52
Student, Nov 2003	2.6	14.37	22.01	7.64
Student, Dec 2003	2.6	14.69	20.57	5.88
			Average	3.03

The fact that our messages are receiving a reduced score is important to keep in mind while looking at the results. SpamAssassin is much less effective without the aid of whitelists and blacklists. Table 5.1 shows the effect whitelists and blacklists have on SpamAssassin scores. As shown by the difference between Prof2's and

SysAdmin's samples the amount a score is modified can vary drastically. Prof2's samples scores increased by 2.64 to 3.43 points while SysAdmin's samples could have lost up to 1.94 points or increased by 0.52 points. The non-local tests do not always increase the score, they can reduce it as well. The typical effect is to increase the score.

5.2 Effectiveness of SpamAssassin Over Time

This section deals with how SpamAssassin's performance has changed over time. We analyze five different versions of SpamAssassin and compare how they scored each of our sample sets. Each version's effectiveness is illustrated by the percent of spam messages caught using the default threshold of five points. Once we have graphs of the performance verse time, we look for trends that develop over time and compare the different versions performance.

The general trend that we expected was for each version to catch a high percent of spam that was from before it was released and then to slowly decrease in effectiveness as spammers realized what each message was being checked for. Once the spammers learned what was being looked for, we expected the performance to decline.

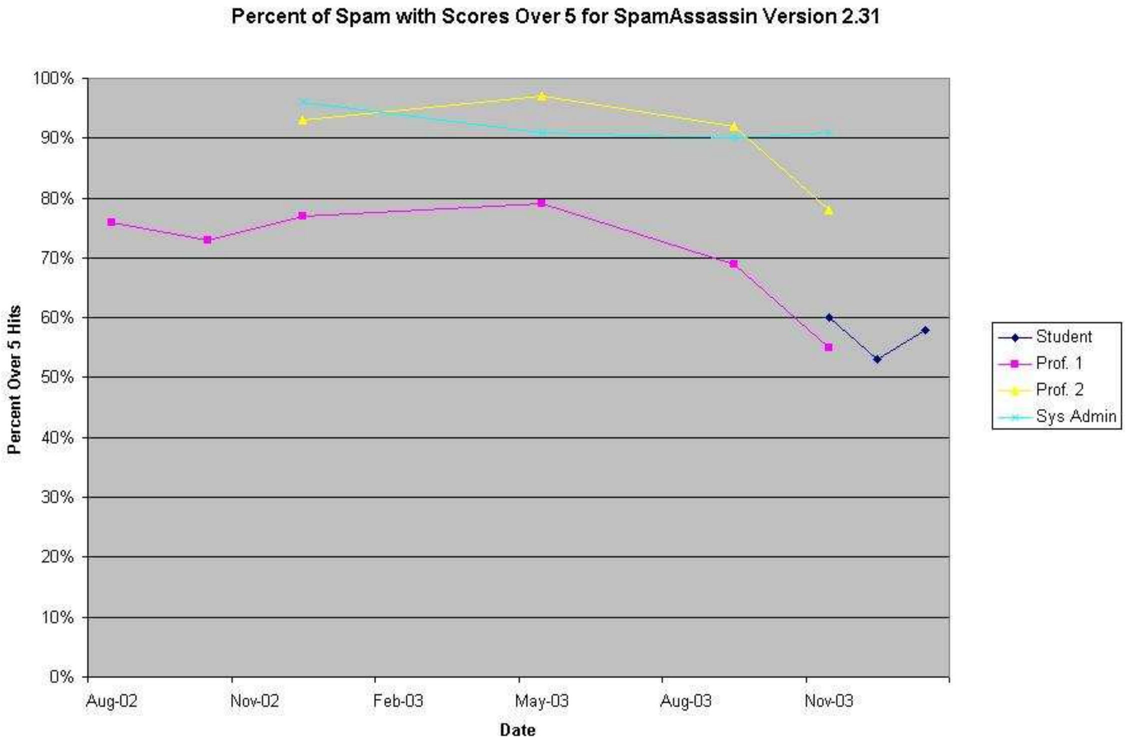

Figure 5.1. Performance of SpamAssassin Version 2.31

Figure 5.1 shows how version 2.31 was initially doing well on the data and then started to rapidly decline in May 2003. Prof2's sample was doing excellent for most of his spam. Without nonlocal tests it was still above 90 percent. The decline was still apparent though. SysAdmin's sample started the decline earlier, around December 2002, but maintained the highest success rating at the end, never dropping below 90 percent. This was an excellent version for catching spam.

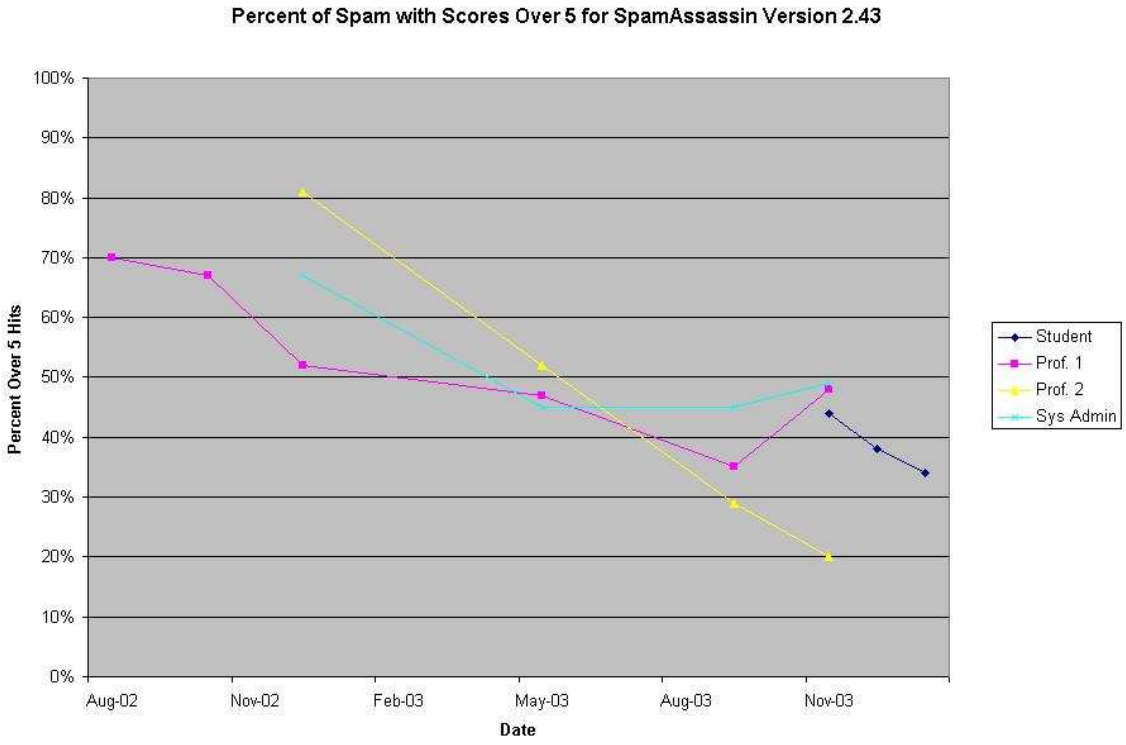

Figure 5.2. Performance of SpamAssassin Version 2.43

Version 2.43 was less effective than the previous version. Less than 60 percent of each sample was caught using this version. This was even true during this version's release date of October 2002. As Figure 5.2 shows, this version rapidly declined with SysAdmin's samples being the only one whose scores stabilized within the period. SysAdmin's percent leveled off at approximately 45 percent. This sudden drop in performance shows the SpamAssassin was concerned with scoring messages too highly. One possible explanation is the developers were concerned with falsely marking ham as spam.

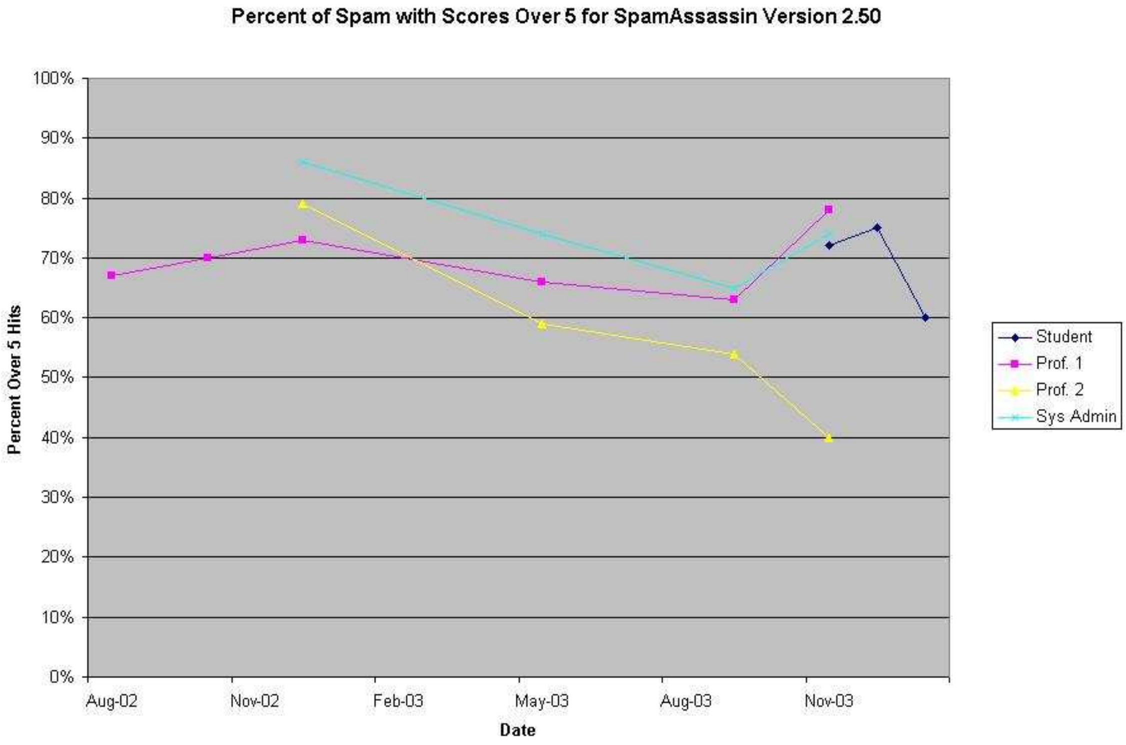

Figure 5.3. Performance of SpamAssassin Version 2.50

The next version, 2.50, compromised in performance between versions 2.31 and 2.43. Figure 5.3 reveals that almost all the samples remain between 60 and 80 percent success. Surprisingly, they do not reach as high as they did in 2.31. This version was released in February 2003, so we do not have much data before it, but from Prof1's sample the effectiveness increased and then started to drop off sometime around the release date.

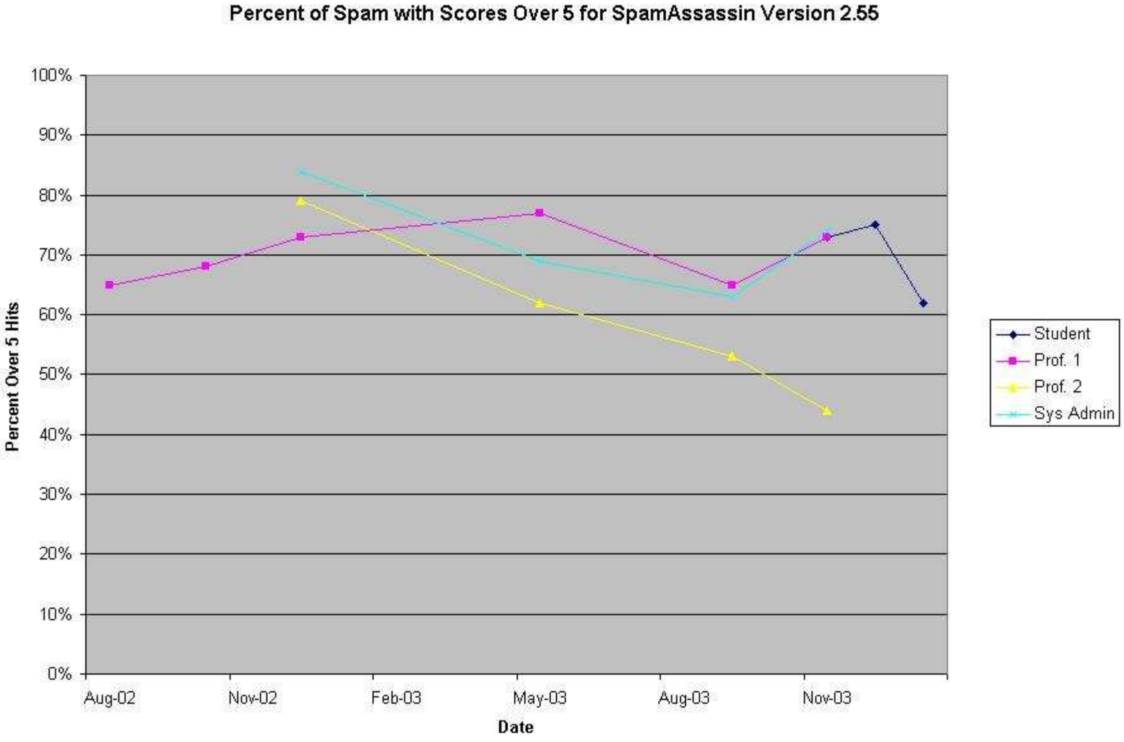

Figure 5.4. Performance of SpamAssassin Version 2.55

Version 2.55 performed similarly to version 2.50. As illustrated in Figure 5.4, the scores appear to be increasing up until about the release date of May 2003. Also, the success rates vary slightly less than the previous version. Overall there was not much change, but that is to be expected since this is essentially an update of version 2.50. There were no major changes.

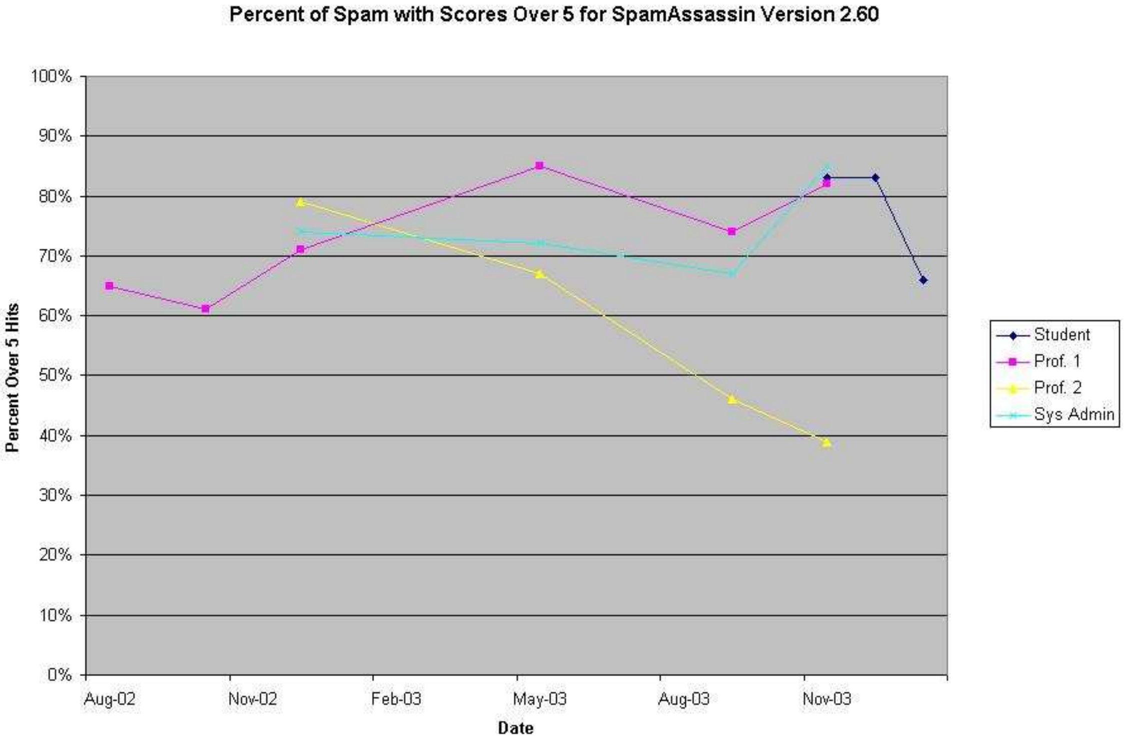

Figure 5.5. Performance of SpamAssassin Version 2.60

The performance of version 2.60 is shown in Figure 5.5. This version had a noticeable increase in caught spam. Prof1's scores consistently increase up until May 2003, although they surprisingly drop off in September. This is surprising because it is expected that the scores would be higher when the new version came out. After the release date in September though, Prof1 and SysAdmin's scores increased as expected. Oddly enough Prof2's scores continued to drop, His final score was almost as low as during the lenient, 2.43, version.

5.3 Real-World Effectiveness of SpamAssassin

Looking at the changes in SpamAssassin performance over time is useful for gaining a general impression of how it works. It is not the best means of examining it though. In a real world context, the software would typically be updated with each new release. That leads to the issue of combining all the previous graphs into one simple "sawtooth" graph. This is so called because the expected performance of SpamAssassin is to start out doing well and then decline until the next version is released. The decline is expected because the performance will likely decrease as spammers adapt to the current version. Following that release, the performance is expected to improve again before again declining. This pattern is expected to repeat for each version. It is also expected that as the versions get newer, the overall effectiveness will get higher, indicating the filters are out performing the spammers. The actual performance in this situation is shown in Figure 5.6, with each vertical line being a date close to when an update was released. The date is not exact, because we did not always have a spam sample from the release month.

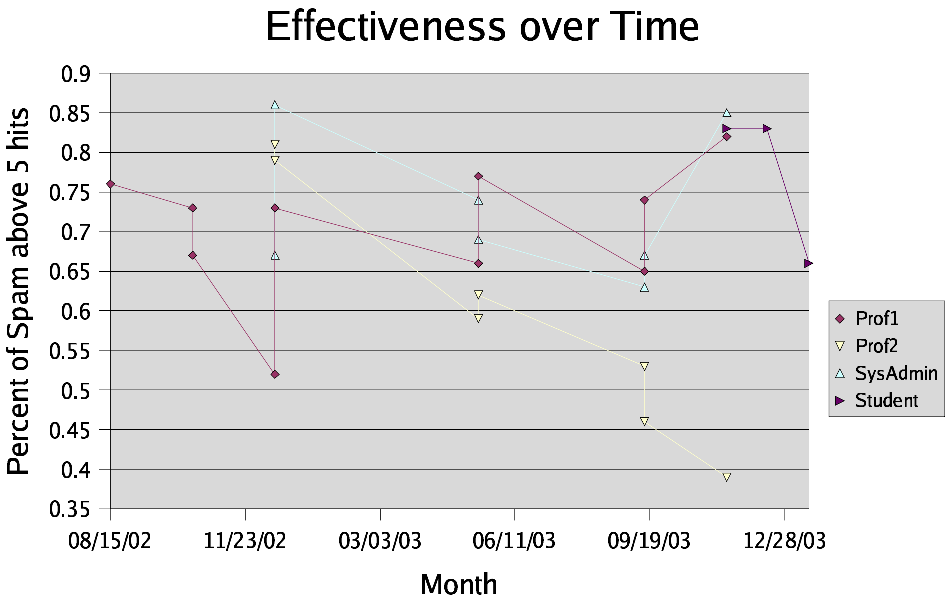

Figure 5.6. Real World Representation of SpamAssassin's Performance

Figure 5.6 shows the data for our four datasets, with Prof1's sample being the most complete. In the beginning, version 2.31, SpamAssassin was performing acceptably, catching about 75 percent of spam without using whitelists or blacklists. Following that is started to slowly decline until version 2.43 was released in October 2002. With the release of version 2.43, the effectiveness dropped substantially and unexpectedly. Version 2.50 corrected this decline in performance, bringing it back above version 2.31's performance on Prof1's spam.

Following the release of version 2.50, the data starts to more closely represent the predicted sawtooth path. Prof1's results are close to what we expected would happen, that the efficiency of SpamAssassin would decline and then spike up to close to the original efficiency with the newly released version. This pattern repeats with each future version for Prof1's results. Unfortunately, the other the samples for Prof2 and SysAdmin do not follow this pattern. While they do jump up in efficiency with each released version, they do not increase enough to offset the decline in efficiency. Prof2's percent of caught mail dramatically declines, falling below 40%. SysAdmin's efficiency declines as well, but not as drastically. Surprisingly there is a sudden increase at the end.

There are several possible explanations for these variances. One explanation as to why Prof2's results are so low is that his spam may rely largely on the non-local tests, such as the centralized whitelists or blacklists. Another possibility is that the mail client modified the email on receiving it, removing some of the offending data or modifying the headers. This is a possibility since Prof2 and SysAdmin used the same mail client while Prof1 used a different one.

5.4 Version Effectiveness on Ham

As the amount of spam that was caught by SpamAssassin varied, the amount of ham that was caught changed as well. Due to our limited amount of ham messages, we were not able to study this in depth, but we had enough to see one possible pattern.

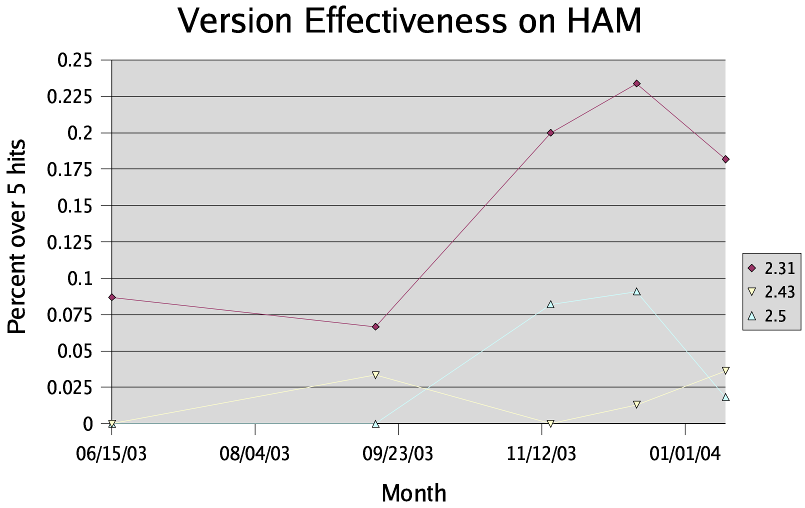

Figure 5.7. Percent of False Positives.

As can be seen clearly from Figure 5.7, version 2.31 had the largest number of false positives on ham. The number of false positives decreased substantially with the release of version 2.43. Unfortunately our ham samples were not from the period when version 2.43 was initially released, but the pattern appears to be consistent enough that we can conclude it dramatically reduced the number of false positives.

As with most attempts to gauge measurements, the makers of SpamAssassin went from being overly strict and catching noticeable amounts of ham to being overly lenient. They started allowing much more Spam to get through so as to not accidentally block ham. This led to the necessity of version 2.50 compromising between the two extremes and being much more effective in both areas. The first three versions we looked at were an attempt to find the proper sensitivity to ham so as to reduce the amount of lost ham messages. The versions we examined following 2.50 performed similarly to 2.50 for ham.

5.5 SpamAssassin Threshold Sensitivities

The dramatic variance in the percent of spam caught combined with the fact that not all providers use the non-local tests makes SpamAssassin's ability to modify the spam threshold useful. Additionally, the fact that some users may have more spam bypass the filters make this a tempting option for those users.

Analyzing the effect of changing the spam threshold essentially requires reexamining all the results to see how they are effected by each individual version. Figure 5.8 shows the average percent of spam caught for thresholds of four, five and six hits. Five is the default threshold for SpamAssassin, four is more sensitive and six is more lenient. The average we used is the collective average for the sample sets of all our participants.

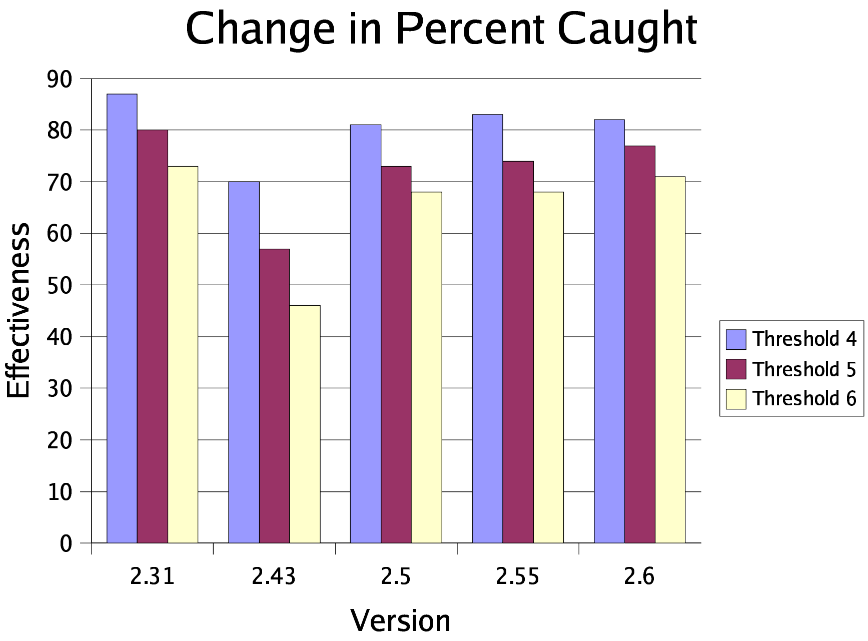

Figure 5.8. Percentage of Spam Caught with Different Thresholds

This figure clearly illustrates how the different thresholds performed in relation to each other. The largest change in effectiveness was typically when the threshold was changed from four to five. The version with the largest drop was version 2.43. Changing the threshold in this version resulted in the effectiveness dropping from 70 percent with a threshold of four to 57 percent. The version the changed the least was 2.60. In this version the effectiveness only dropped from 82 percent caught to 77 percent. The remaining versions all had about the same drop in effectiveness. Version 2.31 went from catching 87 percent to 80 percent, version 2.50 went from 81 percent to 73 percent and version 2.55 changing from 82 percent to 77 percent.

The change brought by increasing the threshold from five to six was about the same magnitude as lowering the threshold. The primary difference observed was that while similar in magnitude, it was slightly lower. The highest change was again see in version 2.43, changing from 57 percent to 46 percent. The least about of change this time was in version 2.50, which changed from 73 percent to 68. Version 2.31 changed the same amount for this difference as it did for the previous change, dropping from 80 percent to 73. The remaining two versions both changed the same amount, with version 2.55 dropping from 74 percent to 68 percent and 2.60 dropping from 77 percent to 71 percent.

5.6 Summary

SpamAssassin has clearly evolved with time. A substantial amount of this evolution is directly related to completing with the evolution of spam. The results vary depending on the user. Surprisingly we had one dataset that reflected each possible case, one improving with time, one staying about the same and one decreasing with time. This variation suggests that neither spam nor SpamAssassin are evolving faster than the other. There is a stalemate between the two sides.

6. Conclusions

Spam and the filters designed to counter it are evolving at a fast pace. Spam is getting harder to identify and filters are getting more sophisticated, developing better and better methods of fighting the influx of spam. The number of rules in SpamAssassin are increasing, enabling it to handle more tricks that the spammers devise. Additionally newer methods of detecting spam, like Bayesian filtering, are being implemented. The field is changing so rapidly that is hard to chose a clear winner, it appears that both sides are evenly matched. One can only wonder what the future will bring, and suggest various methods to further expand on this topic.

6.1 Changes

Spam is getting more and more sneaky and cryptic. What was once a clear message to buy Viagra appears as a friendly message from a long lost friend. Plain text messages are turning into messages which only include images. Spam now has hidden tags, random characters and unrelated words all designed to defeat filters. It seems like spammers find a counter strategy to new filtering methods shortly after they are implemented.

This drives filters, like SpamAssassin to attempt to evolve equally fast. Small fixes are frequently released and major releases at least twice a year. Unfortunately the changes are typically only reactionary, countering the previous advances in spamming. Additionally, since SpamAssassin is a large scale open source project, the develop time limits how quickly it can be released. Additionally, each Internet Service Provider has to get the latest version and install it on their mail server.

There is an apparent pattern in the evolution of spam and SpamAssassin, but regrettably it is still open to interpretation. Our limited sample size prevented up from seeing any clear trends. It appears that the filters start off effective, blocking much of the spam. Then their effectiveness starts to decline as spammers adapt. They upgrade the core rules, add some new ones, and remove some old ones. They change the values of certain tests as well. Then the next version is released, bringing the performance up. The problem is this is not consistent, sometimes the effectiveness improves above the starting level and others it does not. It remains lower, catching less spam.

As indicated by the performance of our sample, rules based systems, such as SpamAssassin, are not stopping the growing influx of spam. Our results were inconsistent, SpamAssassin performed well on some samples and rather poorly on others. One possible reason for this is that they rely on active maintenance to keep up with changes. Whitelists and blacklists help to a certain extent. Whitelists ensure a user's friends and family can reach them. This also requires active maintenance on the user end. Blacklists are excellent for blocking known spammers. Servers which do not have clear anti-spam policies can be blocked as well as servers chosen by the user. Again, this relies on actively updating the lists.

We believe Bayesian filtering to be the next logical step. Bayesian filtering allows SpamAssassin to learn from each message and may be the first step towards learning filters. Using Bayesian filtering in addition to rules and blacklists or whitelists may allow filters to stay ahead of spam. Until then, there appears to be a stalemate in the fight to prevent spam.

6.2 Improvements and Future Work

This report is suggestive of one possible outcome, but there are many variables that can still be considered. More work can be done on studying blacklists and whitelists. There has been research into ways of automatically listing messages. This area could easily be expanded on. Bayesian filtering is a relatively new field in which more options can be considered. There are many aspects to consider, such as how to counter the common strategy of inserting paragraphs of safe words. Another approach that could yield positive results would be to work from the perspective of a spammer. Knowing one's enemy make it possible to adapt or anticipate their next moves. A final different approach would be to study how ham has changed. Spam gets much more attention than ham does, perhaps looking at ham would allow new insight.

It is also possible to improve on our work. For instance, using more samples could reveal useful results. Perhaps Prof2's data was an exception to the rule. Maybe only a small fraction of all users suffer declining performance. Also, most of our samples were

Computer Science oriented mail, coming from two computer science professors and a system administrator. We cannot draw clean statements about any type of user except for those who are highly involved with computers. Approaching business people, students, and other groups would be helpful.

Another area which could be improved would be to increase our sample set. Perhaps if we had a sample from every month we would see a more clear pattern emerge. Using samples from more time periods would allow us to clearly pinpoint how quickly the efficiency of SpamAssassin started to decline. Also, having more overlap for our early samples would further validate our conclusions on versions 2.31 and 2.43. In fact, one option is to specifically study these two versions, and those in between, to see what drove the makers of SpamAssassin to change their scoring so drastically.

In a field as diverse as spam and filtering there are many possible approaches on how to continue work. This is an area that will continue to change and no one knows how it will end. Spam may eventually be stopped, completely disappearing from public knowledge. Another possibility is that is will overrun all filters, making email much less effective to use. From what we have learned, we believe the most likely resolution is a compromise between the two – spam will remain relatively controlled. Filters will keep reacting to different techniques, and the stalemate will remain in effect.

A. Word Count Charts

We looked at the word counts for various types of spam. We looked at the word counts for product, keyword, and message related spam. Additionally, we looked at spam that consisted solely of foreign characters.

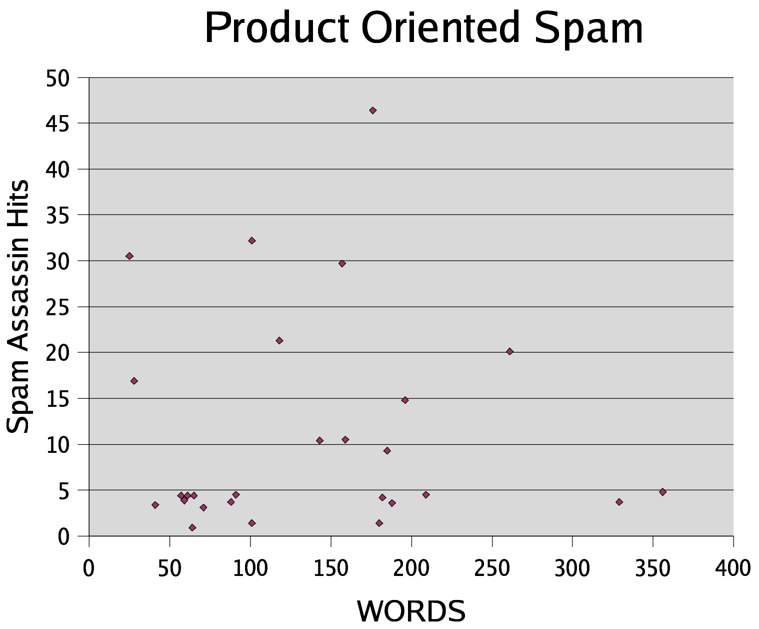

Chart illustrating the relationship between SpamAssassin scores and word count for product oriented spam.

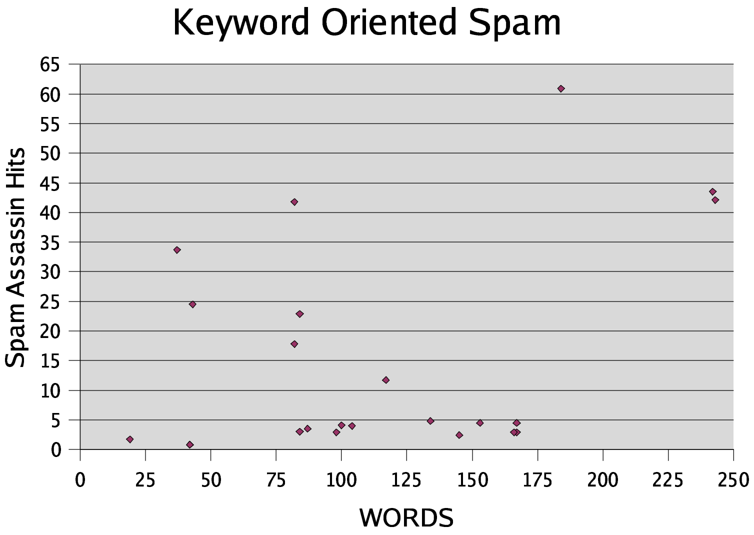

Chart illustrating the relationship between SpamAssassin scores and word count for keyword oriented spam.

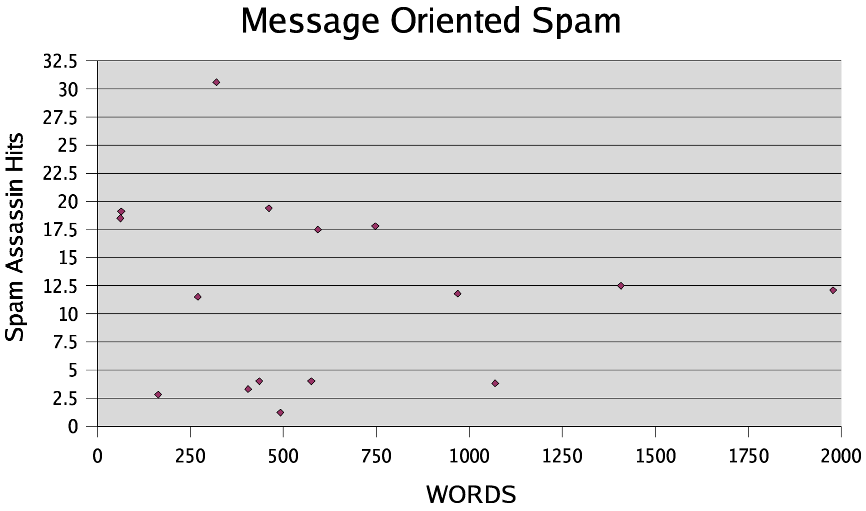

Chart illustrating the relationship between SpamAssassin scores and word count for message oriented spam.

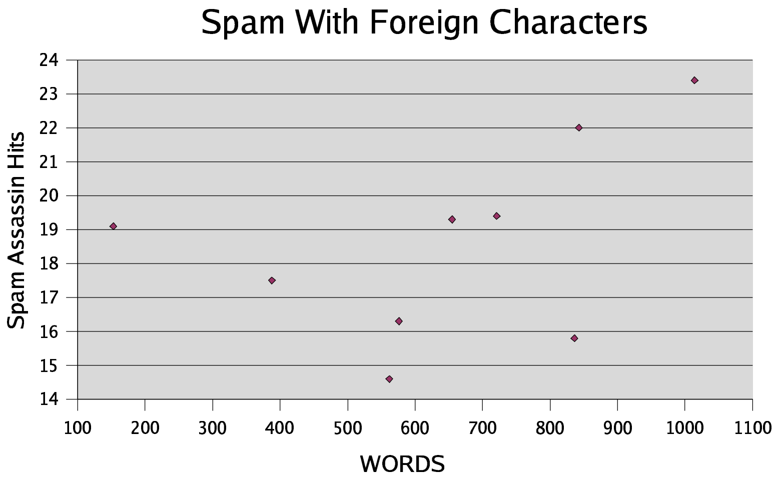

Chart illustrating the relationship between SpamAssassin scores and word count for spam with foreign characters.

B. SpamAssassin Help Output

This section displays the output from the SpamAssassin help command for version 2.43. The format was slightly modified to improve readability, this consisted of adding tabs instead of using spaces for the different options.

```
[xxxxx@xxxxxx Mail-SpamAssassin-2.43]$ ./spamassassin -h
SpamAssassin version 2.43
For more information read the spamassassin man page
Usage:
   spamassassin [options] < *mailmessage* > *output*
   spamassassin -d < *mailmessage* > <output>
   spamassassin -r [-w *addr*] < *mailmessage*
   spamassassin -W|-R < *mailmessage*
   Options:
    -P, --pipe                          Deliver to STDOUT (now default)
    -L, --local                         Local tests only (no online tests)
    -r, --report                        Report message as spam
    -w addr, --warning-from=addr        Send a warning mail to sender from addr
    -d, --remove-markup                 Remove spam reports from a message
    -C file, --config-file=file         Set configuration file
    -p prefs, --prefs-file=file         Set user preferences file
    -x, --nouser-config                 Disable user config files
    -e, --exit-code                     Exit with a non-zero exit code if the
                                        tested message was spam
    -l filename, --log-to-mbox=file     Log messages to a mbox file
    -t, --test-mode                     Pipe message through and add extra
                                        report to the bottom
    --lint                              Lint the rule set: report syntax errors
    -a, --auto-whitelist                Use auto-whitelists
    -W, --add-to-whitelist              Add addresses in mail to whitelist
    --add-to-blacklist                  Add addresses in mail to blacklist
    -R, --remove-from-whitelist         Remove all addresses found in mail
                                        from whitelist
    --add-addr-to-whitelist=addr        Add addr to whitelist
    --add-addr-to-blacklist=addr        Add addr to blacklist
    --remove-addr-from-whitelist=addr Remove addr from whitelist
    -M, --whitelist-factory             Select whitelist factory
    -D, --debug [area=n,...]            Print debugging messages
    -V, --version                       Print version
    -h, --help                          Print usage message
```

C. Email_breaker Script

```
#!/usr/bin/python2.2

"""
Author: J.R. Riedel

DESC: This is the initial version of email_breaker script we used.
It was written with the purpose converting a large file with multiple messages into a
separate file for each message.
This was done so that SpamAssassin could process the data accurately.

Improvements: Since this wasn't designed for continued use, we would hard code the
value for each input file. This could easily be improved to allow a more flexible script.
Additionally we hard coded the "break phrase" to decide when a new file began.
One some samples, this script would create an extra data file, typically 0.eml.
This file was completely empty, and should be deleted so as to not skew results.

Running: The run command: ./email_breaker.py.
Before running, modify the indicated lines.

Environments: This script has only been testing in a Red Hat 8.0 Linux environment.
Python2.2 must be installed.
"""

import os, sys, re

def email_breaker():
		"""
		This is the base directory.
		This is where files will obtained from.
		This is also where files will be saved.
		"""
		base = "/home/MQP/Scripts/email_breaker/"

		"""
		Change the string to reflect the file you want broken.
		This is where the input data is from.
		"""
		f = base + "prof-dec02-15-17-SMALL.out.out"

		"""
```

```
Change the string to reflect the destination directory.
This is where the files will be saved.
"""
out_dir = base + "12/"

#Log file
log_file = open("breaker.log", "a")
log_file.write("Starting new session... \n\n\n")
in_file = open(f, "r")
log_file.write("Processing " + f + "\n")

#Readline
data = in_file.readline()
count = 0
total_hits = 0
temp = str(count)
out_file = open(out_dir + temp + ".eml", "w")
flag = 1

"""
The primary processing portion of the script
This cycles through the entire file and decides where to divide it.
Change the key = line to reflect the marker for a new message beginning.
In our case, since we used the WPI servers, each message began with
"Received: from mail1.WPI.EDU"
"""
while data:
        key = data.find("Received: from mail1.WPI.EDU")
        #Flag off, key found
        if flag == 0 and key > -1:
                print "New"
                flag = 1
                temp = str(count)
                out_file = open(out_dir + temp + ".eml", "w")
                out_file.write(data)
                data = in_file.readline()
        #Flag on, key found
        elif flag == 1 and key > -1:
                print "End of msg"
                flag = 0
                count = count + 1
                out_file.close()
        #No key
        else:
            out_file.write(data)
            data = in_file.readline()
    out_file.close()
    print "Done i think..."

#Command Line Call
if __name__ == '__main__':
    email_breaker()
    sys.exit(0)
```

D. Data Collection Script

There are two kinds of data collection scripts we used. The primary type of script was for handling individual files for each email. The second kind is designed to handle a single file that contains multiple emails. Due to the length of the scripts, only one version is included here.

```
#!/usr/bin/python2.2
"""
Author: J.R. Riedel
* DESC: This is the second version of single_ave script we used.
        It was written with the purpose of extracting various data from our test
        results. This data includes message count, average hits and the median
        number of hits. The main change between this version and the previous was
        the inclusion of a more detailed report.
* Running: The run command: ./process.py <directory>
        "directory" is the name of the directory in which the processed files are
        stored. This should be written without the file slash.

* Environments: This script has only been testing in a Red Hat 8.0 Linux environment.
        Python2.2 must be installed.
"""

import os, sys, re

"""
A helper function designed to convert a dictionary into readable data.
This is called with a dictionary and returns a string.
"""
def format_dict(dict):
        #dict.sort()
        temp = str(dict)
        temp = temp[1:len(temp)-1]
        temp = temp.replace(", ", "\n")
        temp = temp.replace("'", "")
        return temp

"""
The primary processing is done here.
First change the directory variable to reflect the correct path.
```

```
"""
def single_ave(s):
        directory = os.listdir("/home/MQP/Test_Results/" + s + "/")

        l = []
        fo = open("detail_log.out", "a")
        fo.write(s)
        fo.write("\n---------\n")
        count = 0
        total = 0
        """
        A while loop to process all the files in the specified directory.
        Be aware that this will process all files, so remove any that haven't been
        processed by SpamAssassin before running
        This loop will open each file, extract the hits it scored and store the data in a
        dictionary.
        """
        while directory:
                name = directory.pop()
                print name + "\n"
                f = open("/home/MQP/Test_Results/" + s + "/" + name, "r")
                data = f.read()
                i = data.find("hits=")
                j = data.find("required")
                if  i > -1 and j > i:
                        count = count + 1
                        temp = data[i+5:j-1]
                        temp = temp.strip()
                        print temp
                        temp = float(temp)
                        total = total + temp
                        l.append(temp)
                f.close()
                l.sort()
                mean = total / count
                half = count / 2
                med = l[half]
        f.close()
        fo.write("\nAverage: " + str(mean))
        fo.write("\nMedian: " + str(med))
        fo.write("\nCount: " + str(count))
        fo.write("\n\n\n")
```

```
def single_email(s):
        directory = os.listdir("/home/MQP/Test_Results/" + s + "/")
        l = {}
        fo = open("detail_log.out", "a")
        fo.write(s)
        fo.write("\n---------\n")
        while directory:
                name = directory.pop()
                print name + "\n"
                f = open("/home/MQP/Test_Results/" + s + "/" + name, "r")
                data = f.read()
                i = data.find("hits=")
                j = data.find("required")
                if  i > -1 and j > i:
                        temp = data[i+5:j-1]
                        temp = temp.strip()
                        print temp
                        temp = float(temp)
                        temp = int(temp)
                        if l.has_key(temp):
                                l[temp] = l[temp] + 1
                        else:
                                l[temp] = 1
                f.close()
        data = format_dict(l)
        f.close()
        fo.write(data)
        fo.write("\n\n\n")

if __name__ == '__main__':
        if len(sys.argv) < 2:
                print "single_ave.py <dir> (no slashes)"
                sys.exit(0)
        else:
                single_ave(sys.argv[1])

                single_email(sys.argv[1])
```

References

[1] Brightmail. Brightmail – Spam Percentages and Spam Categories. http://www.brightmail.com/spamstats.html

[2] Email Validation System. Email Validation System. http://www.evsmail.com/hamspam.html

[3] Graham, Paul. A Plan for Spam. http://www.paulgraham.com/spam.html

[4] Krim, Jonathan. Spam's Cost to Business Escalates. http://www.washingtonpost.com/ac2/wp-dyn/A17754-2003Mar12

[5] Lynch, Keith. Keith Lynch's timeline of spam related terms and concepts. http://keithlynch.net/spamline.html

[6] Mail Abuse Prevention System. What is “spam“?. http://www.mail-abuse.org/standard.html

[7] Prakash, Vipul Ved. Vipul's Razor: home. Forge. http://razor.sourceforge.net/

[8] SpamAssassin. Welcome to SpamAssassin. http://www.spamassassin.org

[9] Templeton, Brad. Origin of the term “spam“ to mean net abuse. http://www.templetons.com/brad/spamterm.html

[10] Templeton, Brad. Reflections on the 25th Anniversary of Spam. http://www.templetons.com/brad/spam/spam25.html