MEF: Malicious Email Filter
A UNIX Mail Filter that Detects Malicious Windows Executables

Matthew G. Schultz and Eleazar Eskin
Department of Computer Science
Columbia University
{mgs,eeskin}@cs.columbia.edu

Erez Zadok
Department of Computer Science
State University of New York at Stony Brook
ezk@cs.sunysb.edu

Manasi Bhattacharyya, and Salvatore J. Stolfo
Department of Computer Science
Columbia University
{mb551,sal}@cs.columbia.edu

Abstract

We present *Malicious Email Filter*, MEF, a freely distributed malicious binary filter incorporated into Procmail that can detect malicious Windows attachments by integrating with a UNIX mail server. The system has three capabilities: detection of known and unknown malicious attachments, tracking the propagation of malicious attachments and efficient model update algorithms.

The system filters multiple malicious attachments in an email by using detection models obtained from data mining over known malicious attachments. It leverages preliminary research in data mining applied to malicious executables which allows the detection of previously unseen, malicious attachments. In addition, the system provides a method for monitoring and measurement of the spread of malicious attachments. Finally, the system also allows for the efficient propagation of detection models from a central server. These updated models can be downloaded by a system administrator and easily incorporated into the current model. The system will be released under GPL in June 2001.

1 Introduction

A serious security risk today is the propagation of malicious executables through email attachments. A malicious executable is defined to be a program that performs a malicious act, such as compromising a system's security, damaging a system or obtaining sensitive information without the user's permission. Recently there have been some high profile incidents with malicious email attachments such as the ILOVEYOU virus and its clones. These malicious attachments caused significant damage in a short time. The *Malicious Email Filter* (MEF) project provides a tool for the protection of systems against malicious email attachments.

An email filter that operates within a mail to detect malicious Windows binaries has many advantages. Operating from a mail server, the email filter could automatically filter the email each host receives. The mail server could either wrap the malicious email with a warning addressed to the user, or it could block the email depending upon the server's settings. All of this could be done without the server's users having to scan attachments themselves or having to download updates for their virus scanners. This way the system administrator can be responsible for updating the filter instead of relying on end users. We present such a system on a UNIX implementation of sendmail using procmail.

The standard approach to protecting against malicious emails is to use a virus scanner. Commercial virus scanners can effectively detect known malicious executables, but unfortunately they can not detect unknown malicious executables reliably. The reason for this is that most of these virus scanners are *signature based*. For each known malicious binary, the scanner contains a byte sequence that identifies the malicious binary. However, an unknown malicious binary, one without a pre-existing signature, will most likely go undetected.

We built upon preliminary research at Columbia University on data-mining methods to detect malicious binaries [2]. The idea is that by using data-mining, knowledge of known malicious executables can be generalized to detect unknown malicious executables. Data mining methods are ideal for this purpose because they detect patterns in large amounts of data, such as byte code, and use these patterns to detect future instances in similar data along with detecting known instances. Our framework used *classifiers* to detect malicious executables. A classifier is a rule set, or detection model, generated by the data mining algorithm that was trained over a given set of training data.

The goal of this paper is to describe a data mining based filter which integrates with Procmail's pre-existent security filter [3] to detect malicious executables. The MEF system is an application of more theoretical research into this problem [10]. The data mining-based detection system within MEF is a preliminary system that will become more accurate and efficient as our research progresses, and new data sets are analyzed. It uses a scoring system based on a data mining classifier to determine whether or not an attachment may be malicious. If an attachment's score is above a certain threshold it is considered malicious.

This work expanded upon Procmail's pre-existant filter which already defangs active-content HTML tags to protect users who read their mail from a web brower or HTML-enabled mail client. Also, if the attachment is labeled as malicious, the system "mangles" the attachment name to prevent the mail client from automatically executing the attachment. It also has built in security filters such as long filenames in attachments, and long MIME headers, which may crash or allow exploits of some clients. This filter lacks the ability to automatically update its list of known malicious executables leaving the system vulnerable to attacks by new and unknown viruses. Furthermore, its evaluation of an attachment is based solely on the name of the executable and not the contents of the attachment itself. We replaced this signature based detection algorithm with our data mining classifier that added the ability to detect both the set of known malicious binaries and a set of previously unseen, but similar malicious binaries. Although the MEF implementation was designed for the data mining-based detection system, any method to evaluate binaries including a standard signature based scanner can be used.

Since the methods and classifier models we describe are probabilistic, we provide a means of determining whether a binary was *borderline*. A borderline binary is a program that has similar probabilities for both classes (i.e. could be either a malicious executable or a benign program). As a parameter of the filter, the system administrator may specify what a borderline case is. Guidelines on how to set this parameter are described in detail later. If it is a borderline case then along with the option to wrap it as a malicious program there is an option in the network filter to send a copy of the malicious executable to a central repository such as CERT. There, it can be examined by human experts. After analysis by virus experts, the model can be updated to be more accurate by including these borderline cases.

The detection model generation works as follows. The binaries are first statically analyzed to extract byte-sequences, and then the classifiers are generated by analyzing a subset of the data. Then the classifier (or detection model) is tested on a set of previously unseen data. We implemented a traditional, signature-based algorithm to compare its performance with the data mining algorithms. Using standard statistical cross-validation techniques, the data mining-based framework for malicious binary detection had a detection rate of 97.76%, over double the detection rate of a signature-based scanner.

The organization of the paper is as follows. In Section 2, we present the system features and their integration with Procmail. In Section 3, we detail the methods that are employed to track the propagation of malicious attachments. Section 4 describes how the detection algorithms work, and their results. Section 5 discusses the system's performance, and Sections 6 and 7 conclude the paper and discuss future work.

2 Incorporation into Procmail

MEF filters malicious attachments by replacing the signature based virus filter found in Procmail with a data mining generated detection model. Procmail is a program that processes email messages looking for particular information in the headers or body of each message, and takes actions based on what it finds [11]. Currently the mail server supported is sendmail. MEF uses a procmail script to extract attachments from emails and save them temporarily based on their name. The script then runs the filter on each attachment.

The filter first decodes each binary and then examines the binary using a data mining classifier. It evaluates the attachment by comparing it to all the byte strings found with it to the byte-sequences contained in the detection model. The system calculates the probability of the binary being malicious, and if it is greater that its likelihood of being benign then the executable is labeled malicious. Otherwise, the binary is labeled benign. This is reported as a score back to Procmail, and then is used to either send the mail along untouched, or the entry is logged as the attack and email is wrapped with a warning. The log is a collection of information about the attachment. Exactly what this information is depends upon the configuration of the system.

2.1 Borderline Cases

Borderline binaries are binaries that have similar probabilities of being benign and malicious (e.g. 50% chance it is malicious, and 50% chance it is benign). The binaries are important to keep track of because they are likely to be mislabeled, so they should be included in the training set. To facilitate this, the system archives the borderline cases, and at periodic intervals the collection of borderline binaries is sent back to a central server by the system administrator.

Once at the central repository, these binaries can then be analyzed by experts to determine whether they are malicious or not, and subsequently included in the future versions of the detection models. Any binary that is determined to be a borderline case will be forwarded to the repository and wrapped with a warning as though it were a malicious attachment.

A simple metric to detect borderline cases and redirect them to an evaluation party is to define a borderline case to be a case where the difference between the probability it is malicious and the probability it is benign is above a threshold. This threshold is set based on the policies of the host.

For example in a secure setting, the threshold could be set at 20%. In this case all binaries that have a 60/40 split are labeled as borderline. In other words, binaries with a 60% chance (according to the model) of being malicious and 40% chance of being benign would be labeled borderline, and vice versa. This setting can be determined by the system administrator or left on the default setting of 51.25/48.75, a threshold of 2.5%.

Receiving borderline cases and updating the detection model is an important aspect of the data mining approach. The larger the data set that is used to generate models then the more accurate the detection models will be. This is because borderline cases are executables that could potentially lower the detection and accuracy rates by being misclassified, so they should be trained over.

2.2 Update Algorithms

This system will require updates periodically, and in the following section we detail the update algorithm. After a number of borderline cases have been received, it is necessary to generate a new detection model, and subsequently distribute updated models.

A new model is first generated by running the data mining algorithm on the new data set that contained the borderline cases along with their correct classification, and the previous data set. This model will then be distributed.

Updating the models is accomplished by distributing portions of the models that changed, and not the entire model. This is important because the detection models are large. In order to avoid constantly sending a large model to the filters, the administrator has the option of receiving this smaller file. Using the update algorithm, the older model can then be updated. The full model will also be available to provide additional options for the system administrator.

Efficient update of the model is possible because the underlying representation of the models is probabilistic. As is explained later, the model is a count of the number of times that each byte string appears in a malicious program versus the number of times that it appears in a benign program. An update model can then be easily summed with the older model to create a new model.

In future versions of MEF, the model will be made available for the system administrator on a public ftp site. If a system administrator subscribes to the mailing list then when a new model is made available, the system administrator will receive an email. The email will detail where the model is located, what version it is, and include a form of authentication. At the ftp site the model will be available to download as either an upgrade from a previous version, or as a full model. An archive of old models will also be kept on the ftp site.

There are also a host of options for automatically receiving the updates. One way to distribute the email is just to attach the update to the notification email. Then the administrator could update the model later without having to ftp it. In the future, a program included in the email filter could automatically poll the central server to see if a new model is available and then download it and update the current model. These last methods have not yet been implemented.

3 Monitoring the Propagation of Email Attachments

Tracking the propagation of email attachments is beneficial in identifying the origin of malicious executables, and in estimating a malicious attachment's prevalence.

The monitoring works by having each host that is using the system log all malicious attachments, and the borderline attachments that are sent to and from the host. This logging may or may not contain information about the sender or receiver depending on the privacy policy of the host.

In order to log the attachments, we needed a way to obtain a unique identifier for each attachment. We did this by using the MD5 algorithm [9] to compute a unique number for each binary attachment. The input to MD5 was the hexadecimal representation of the binary. These identifiers were then kept in a log along with other information such as whether the attachment was malicious, or benign and with what certainty the system made those predictions.

The logs of malicious attachments are then sent back to the central server according to the policy of each host. Some hosts may wish to never send these logs, and can turn the feature off, while other hosts could configure the system to only send logs of borderline cases, etc.

After receiving the logs, the system measures the propagation of the malicious binaries across hosts. From these logs it can be estimated how many copies of each malicious binary were circulating the Internet, and these reports will be forwarded back to the community, and used for further research.

The current method for detailing the propagation of malicious executables is for an administrator to report an attack to an agency such as *WildList* [8]. The wild list is a list of the propagation of viruses in the wild and a list of the most prevalent viruses. This is not done automatically, but instead is based upon a report issued by an attacked host. Our method would reliably, and automatically detail a malicious executable's spread over the Internet.

4 Methodology for Building Data Mining Detection Models

We gathered a large set of programs from public sources and defined a learning problem with two classes: *malicious* and *benign* executables. Each example in the data set is a Windows or MS-DOS format executable, although the framework we present is applicable to other formats. To standardize our data-set, we used MacAfee's [6] virus scanner and labeled our programs as either malicious or benign executables.

We split the dataset into two subsets: the *training set* and the *test set*. The data mining algorithms used the training set while generating the rule sets, and after training we used a test set to test the accuracy of the classifiers on a set of unseen examples.

4.1 Data Set

The data set consisted of a total of 4,301 programs split into 3,301 malicious binaries and 1,000 benign programs. The malicious binaries consisted of viruses, Trojans, and cracker/network tools. There were no duplicate programs in the data set and every example in the set is labeled either malicious or benign by the commercial virus scanner. All labels are assumed to be correct.

All programs were gathered either from FTP sites, or personal computers in the Data Mining Lab at Columbia University. To obtain the dataset please contact us through our website *http://www.cs.columbia.edu/ids/mef/*.

4.2 Detection Algorithms

We statically extracted byte sequence *features* from each binary example for the learning algorithms. Features in a data mining framework are properties extracted from each example in the data set, such as byte sequences, that a classifier uses to generate detection models. These features are then used by the algorithms to generate detection models. We used *hexdump* [7], an open source tool that transforms binary files into hexadecimal files. After we generated the hexdumps we produced features in the form displayed in Figure 1 where each line represents a short sequence of machine code instructions.

```
646e 776f 2e73 0a0d 0024 0000 0000 0000
454e 3c05 026c 0009 0000 0000 0302 0004
0400 2800 3924 0001 0000 0004 0004 0006
000c 0040 0060 021e 0238 0244 02f5 0000
0001 0004 0000 0802 0032 1304 0000 030a
```

Figure 1: Example Set of Byte Sequence Features

4.3 Data Mining Approach

The classifier we incorporated into Procmail was a Naive Bayes classifier [1]. A naive Bayes classifier computes the likelihood that a program is malicious given the features that are contained in the program. We assumed that there were similar byte sequences in malicious executables that differentiated them from benign programs, and the class of benign programs had similar sequences that differentiated them from the malicious executables.

The model output by the naive Bayes algorithm labels examples based on the byte strings that they contain. For instance, if a program contained a significant number of malicious byte sequences and a few or no benign sequences, then it labels that binary as malicious. Likewise, a binary that was composed of many benign features and a smaller number of malicious features is labeled benign by the system.

The naive Bayes algorithm computed the probability that a given feature was malicious, and the probability that a feature was benign by computing statistics on the set of training data. Then to predict whether a binary, or collection of hex strings was malicious or benign, those probabilities were computed for each hex string in the binary, and then the Naive Bayes independence assumption was used. The independence assumption was applied in order to efficiently compute the probability that a binary was malicious and the probability that the binary was benign.

One draw back of the naive Bayes method was that it requires more than 1 GB of RAM to generate its detection model. To make the algorithm more efficient we divided the problem into smaller pieces that would fit in memory and generated a classifier for each of the subproblems. The subproblem was to classify based on every 16 subsets of the data organized according to the first letter of the hex string.

For this we trained sixteen Naive Bayes classifiers so that all hex strings were trained on. For example, one classifier trained on all hex strings starting with an "A", and another on all hex strings starting with a "0". This was done 16 times and then a voting algorithm was then used to combine their outputs.

A more thorough description along with an example, can be found in a companion paper [10].

4.4 Signature-Based Approach

To compare our results with traditional methods we implemented a signature based method. First, we calculated the byte-sequences that were only found in the malicious executable class. These byte-sequences were then concatenated together to make a unique signature for each malicious executable example. Thus each malicious executable signature contained only byte-sequences found in the malicious executable class. To make the signature unique, the byte-sequences found in each example were concatenated together to form one signature. This was done because a byte-sequence that was only found in one class during training could possibly be found in the other class during testing [4], and lead to false positives when deployed.

The virus scanner that we used to label the data set contained signatures for every malicious example in our data set, so it was necessary to implement a similar signature-based method. This was done to compare the two algorithms' accuracy in detecting new malicious executables. In our tests the signature-based algorithm was only allowed to generate signatures for the same set of training data that the data mining method used. This allowed the two methods to be fairly compared. The comparison was made by testing the two methods on a set of binaries not contained in the training set.

4.5 Preliminary Results

To quantify the performance of our method we computed statistics on the preformance of the data mining-based method versus the signature-based method. We are interested in four quantities in the experiments. They are the counts for *true positives*, *true negatives*, *false positives*, and *false negatives*. A true positive, TP, is an malicious example that is correctly tagged as malicious, and a true negative, TN, is a benign example that is correctly classified as benign. A false positive, FP, is a benign program that has been mislabeled by an algorithm as malicious, while a false negative, FN, is a malicious executable that has been mis-classified as a benign program.

The *overall accuracy* of the algorithm is calculated as the number of programs the system classified correctly divided by the total number of binaries tested. The *detection rate* is the number of malicious binaries correctly classified divided by the total number of malicious programs tested.

We estimated our results for detecting new executables by using 5-fold cross validation [5]. Cross validation is the standard method to estimate the performance of predictions over unseen data. For each set of binary profiles we partitioned the data into five equal size partitions. We used four of the partitions for training a model and then evaluated that model on the remaining partition. Then we repeated the process five times leaving out a different partition for testing each time. This gave us a reliable measure of our method's accuracy on unseen data. We averaged the results of these five tests to obtain a measure of how the algorithm performs in detecting new malicious executables.

4.6 Performance on New Executables

Table 1 displays the results. The data mining algorithm had the highest detection rate, 97.76% compared with the signature based method's detection rate of 33.96%. Along with the higher detection rate the data mining method had a higher overall accuracy, 96.88% vs. 49.31%. The false positive rate at 6.01% though was higher than the signature based method, 0%.

Figure 2 displays the plot of the detection rate vs. false positive rate using *Receiver Operation Characteristic* curves [13]. Receiver Operating Characteristic (ROC) curves are a way of visualizing the trade-offs between detection and false positive rates. In this instance, the ROC curve show how the data mining method can be configured for different environments. For a false positive rate less than or equal to 1% the detection rate would be greater than 70%, and for a false positive rate greater than 8% the detection rate would be greater than 99%.

Profile Type	**Detection Rate**	**False Positive Rate**	**Overall Accuracy**
Signature Meth.	33.96%	0%	49.31%
Data Mining Meth.	97.76%	6.01%	96.88%

Table 1: The results of testing the algorithms over new examples. Note the Data Mining Method had a higher detection rate and accuracy while the Signature based method had the lowest false positive rate.

4.7 Performance on Known Executables

We also evaluated the performance of the models in detecting known executables. For this task, the algorithms generated detection models for the entire set of data. Their performance was then evaluated by testing the models on the same training set.

As shown in Table 2, both methods detected over 99% of known executables. The data mining algorithm detected 99.87% of the malicious examples and misclassified 2% of the benign binaries as malicious. However, we have the signatures for the binaries that the data mining algorithm misclassified, and the algorithm can include those signatures in the detection model without lowering accuracy of the algorithm in detecting unknown binaries. After the signatures for the executables that were

Profile Type	Detection Rate	False Positive Rate	Overall Accuracy
Signature Meth.	100%	0%	100%
Data Mining Meth.	99.87%	2%	99.44%

Table 2: Results of the algorithms after testing on known programs. In this task both algorithms detected over 99% of known malicious programs. We explain later the data mining algorithm can be boosted to have 100% accuracy by using some signatures.

misclassified during training had been generated and included in the detection model, the data mining model had a 100% accuracy rate when tested on known executables.

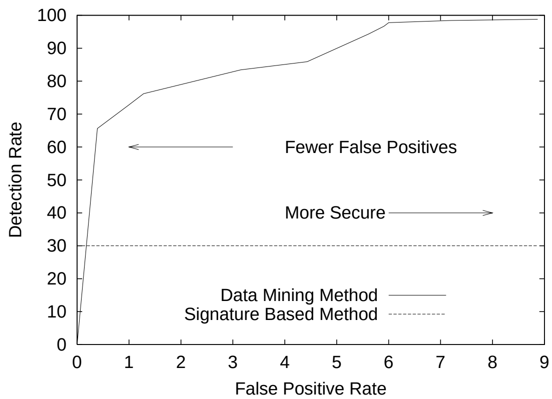

Figure 2: ROC: This figure shows how the data mining method can be configured to have different detection and false positive rates by adjusting the threshold parameter. More secure settings should choose a point on the data mining line towards the right, and applications needing fewer false alarms should choose a point towards the left.

5 System Performance

The system requires different time and space complexities for model generation and deployment.

5.1 Training

In order for the data mining algorithm to quickly generate the models, it requires all calculations to be done in memory. The algorithm consumed space in excess of a gigabyte of RAM. By splitting the data into smaller pieces, the algorithm was done in memory with no loss in accuracy.

The training of a classifier took 2 hours 59 minutes and 49 seconds running on a Pentium III 600 Linux machine with 1GB of RAM. The classifier took on average 2 minutes and 28 seconds for each of the 4,301 binaries in the data set.

5.2 During Deployment

Current work is being done to make the system accurate on a system with smaller memory. At this point in development, only systems that have a 1GB or more of memory can use our models. The amount of system resources taken for using a model are equivalent to the requirements for training a model. So on a Pentium III 600 Linux box with 1GB of RAM it would take on average 2 minutes 28 seconds per attachment.

The ongoing work we are doing is to make the model small enough to be loaded into a computer with 128MB of RAM without losing more than 5% in accuracy. If this is accomplished then the resources required in CPU time and memory will be notably reduced.

There are other options in making the system perform its analysis faster such as sharing the load over several computers. These options are not currently being explored, but they are open problems that the community should examine. We are primarily concerned with improving the space complexities of the algorithm without sacrificing a significant amount accuracy.

6 Conclusions

The first contribution that we presented in this paper was a freely distributed filter for Procmail that detected known malicious Windows executables and previously unknown malicious Windows binaries from UNIX. The detection rate of new executables was over twice that of the traditional signature based methods, 97.76% compared with 33.96%.

One problem with traditional, signature-based methods is that in order to detect a new malicious executable, the program needs to be examined and a signature extracted from it and included in the anti-virus database. The difficulty with this method is that during the time required for a malicious program to be identified, analyzed and signatures to be distributed, systems are at risk from that program. Our methods may provide a defense during that time. With a low false positive rate the inconvenience to the end user would be minimal while providing ample defense during the time before an update of models is available.

Virus scanners are updated about every month, and 240–300 new malicious executables are created in that time (8–10 a day [12]). Our method may catch roughly 216–270 of those new malicious executables without the need for an update whereas traditional methods would catch only 87–109. Our method tested on a particular data set more than doubles the detection rate of signature based methods.

Secondly, we presented a system that improves it's accuracy by regenerating models after receiving borderline cases. This feature is of interest because as more servers and clients use this system the system will receive additional borderline cases. Training on these borderline cases will increase the accuracy of the filter.

Finally, the system has the optional ability to monitor the propagation of malicious attachments. Depending upon the user specified setting, email tracking can be turned on or off. If tracking is turned on then statistics can be generated detailing how a malicious binary attacked a system and propagated. If tracking is turned off then the system loses no accuracy in detecting malicious attachments.

The system that we presented detected malicious Windows binaries from UNIX, and detected new examples of similar malicious binaries because of the data mining algorithms. It tracked the propagation of email attachments, and with the inclusion of borderline cases it will become more accurate with time. Also with a larger, more realistic data set work can be done to show the algorithm is practical.

7 Future Work

One of the most important areas of future work for this application is the development of more efficient algorithms. The current probabilistic method requires a machine with a significant amount of memory to generate, and employ the classifiers. This memory requirement makes the computation of the models expensive.

To make the algorithms use less space will require theoretical bounds on how to prune features from the data without losing accuracy. The details of how the pruning may work is beyond the scope of this paper.

After developing more efficient algorithms, the next most important work to be done is generating a more complete data set. The current, malicious data set, 3,301 examples, is smaller than the known number of malicious programs, 50,000+ examples. Work needs to be done with industry or security sources to develop a standard data set consisting of infected programs, macro and visual basic viruses, and many different sets of benign data. [1]

On more obvious future work would be to incorporate the system into Windows and Macintosh mail servers and clients. This would require work with the individual vendors because their systems are not open-sourced. As a result of our system being freely available, these vendors could work with us to incorporate it or they could do it themselves.

Another potential future work of this filter is to make it into a stand alone virus scanner. Once the system has been fully completed, and thoroughly tested in the real world it would be possible to port the algorithms to different operating systems, such as Windows, or Macintosh.

This scanner could be run in a similar manner to traditional virus scanners. A user could run the system at bootup, or when required and analyze all the files on a personal computer. This requires though that the system be efficient enough to run on older computers with slower processors, and less memory.

8 Software Availability

The software described in this paper can be downloaded from *http://www.cs.columbia.edu/ids/mef* in June 2001.

References

[1] D.Michie, D.J.Spiegelhalter, and C.C.Taylor. *Machine learning of rules and trees*. Ellis Horwood, 1994.

[2] MEF Group. Malicious Email Filter Group. Published as *Online publication*, 2001. http://www.cs.columbia.edu/ids/mef.

[3] John Hardin. Enhancing E-Mail Security With Procmail. *Online publication*, 2001. http://www.impsec.org/email-tools/procmail-security.html.

[4] Jeffrey O. Kephart and William C. Arnold. Automatic Extraction of Computer Virus Signatures. *4th Virus Bulletin International Conference*, pages pages 178-184, 1994.

[5] R Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. *IJCAI*, 1995.

[6] MacAfee. Homepage - MacAfee.com. Published as *Online publication*, 2001. http://www.mcafee.com.

[7] Peter Miller. Hexdump. Published as *Online publication*, 2001. http://www.pcug.org.au/ millerp/hexdump.html.

[8] Wildlist Organization. Virus description of viruses in the wild. Published as *Online Publication*, 2001. http://www.f-secure.com/virus-info/wild.html.

[1] We are currently working on establishing such a data set with Cigital, *http://www.cigital.com*, but more resources are needed. If you would like to work with us please contact us at our website *http://www.cs.columbia.edu/ids/mef.*

[9] Ronald L. Rivest. The MD5 Message Digest Algorithm. Published as *Internet RFC 1321*, April 1992.

[10] Matthew G. Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. Data Mining Methods for Detection of New Malicious Executables. *To appear in IEEE Symposium on Security and Privacy*, May 2001.

[11] Stephen R. van den Berg and Philip Guenther. Procmail. *Online publication*, 2001. http://www.procmail.org.

[12] Steve R. White, Morton Swimmer, Edward J. Pring, William C. Arnold, David M. Chess, and John F. Morar. Anatomy of a Commercial-Grade Immune System. *IBM Research White Paper*, 1999. http://www.av.ibm.com/ScientificPapers/White/Anatomy/anatomy.html.

[13] KH Zou, WJ Hall, and D Shapiro. Smooth non-parametric ROC curves for continuous diagnostic tests. *Statistics in Medicine*, 1997.