

APE: Adaptive Prevention Environments

A Framework for Adaptive Virus Detection and Containment

Steve Martin, Blaine Nelson, Anil Sewani
{steve0, nelsonb, anil}@cs.berkeley.edu
University of California, Berkeley

Abstract

Recent outbreaks of computer viruses and worms have caused immense damage. Many widely deployed virus detection and containment technologies use techniques that work well on clearly identified threats, but are not flexible enough to react to new virus outbreaks. To make matters worse, contemporary worms spread at extremely fast rates and are capable of exploiting several software vulnerabilities.

In this paper, we present Adaptive Prevention Environments (APE), an intelligent defense mechanism against novel virus threats. We build a model of network behavior using machine learning in the form of a Gaussian Mixture Model fit to data by the Expectation Maximization algorithm. The model detects behavior anomalies indicative of virus infection on local area networks. Once APE has identified an infected computer, it can be quarantined from the network, preventing virus propagation and further damage.

To evaluate the effectiveness of our approach, we implemented a prototype of APE that detects anomalies specific to e-mail traffic. Our initial results based on simulated traces of normal and infected e-mail traffic are promising. We anticipate a high ceiling on future research.

1 Introduction

It is a well known fact that malicious, self replicating computer programs known as viruses continue to be a major security problem that has already caused the technology industry to lose billions of dollars in damage and business. As an example, the 2001 outbreak of the Nimda virus cost the tech sector \$635 million. Earlier the same year, the widespread Code Red virus hit even harder, weighing in at \$2.62 billion. Finally, in 2000, the spread of the email virus Love Bug accumulated a staggering total of \$8.75 billion in damages. [2] As industry, governments, and militaries grow more reliant on computer systems, the potential for a truly crippling virus attack is immense.

While virus detection and protection continues to be an area of intense research, the traditional defenses

against computer viruses deployed in the field have not changed much in recent years. At the network level, firewalls, which accept or deny network traffic based on a set of rules, can block ports used by viruses to propagate or issue remote commands. At the server or individual machine level, virus scanners are typically installed, which scan all incoming file traffic to detect known virus signatures. [3]

The inherent problem with these methods of protection is that they require human intervention to be effective against new threats. In the case of a firewall, a network administrator must manually reconfigure the firewall to block incoming traffic from a novel virus. At the virus scanner level, antivirus software companies must first analyze the new virus to generate a signature for it. Then, the user must be depended upon to connect, download, and install the update along with any necessary patches. Unfortunately, recent events have demonstrated that by the time humans can actively respond to a new virus threat and accomplish these tasks, it is highly possible that an infection is already well underway. A practical example is the recent MSBlaster worm, which infected machines on local area networks at an incredible rate. [19] In fact, previous research has shown that it is not only possible, but feasible to create an internet worm that can infect up to ten million hosts in roughly fifteen minutes. [18]

A potential solution to this problem is to design and implement a system that is able to prevent widespread infection by quarantining infected machines until human intervention is possible. Such a system must guarantee a fast reaction time, as well as the ability to be effective against novel viruses.

Our goal is to implement such a solution for local area networks. To this end, we designed and implemented Adaptive Prevention Environments (APE). APE is a standalone program that monitors a network for abnormal activity. Using unsupervised machine learning, APE builds a model of network behavior that it can use to detect anomalies such as virus infections. The current prototype of APE is limited to viruses propagating through email. However, there is no reason why it cannot be easily extended to monitor any level of the network communication stack.

This paper presents the results of our work to date. The following section gives a brief overview of previous research related to this paper. Section 3 describes the overall structure of the APE prototype as well as planned future versions of the software. Section 4 discusses the theory behind the unsupervised machine learning engine that is the heart of APE. Section 5 evaluates our program, shows how we tested APE, and discusses the results. Section 6 talks of future work we have in mind for this project, while Section 7 concludes this paper with some remarks on APE and our technique in general.

2 Previous Research

Recent work on worm analysis started after the Code-Red epidemic in 2001. Code-Red infected 359,000 hosts during the first 24 hours of activation. An enhanced version of Code-Red that used better random probing of IP addresses reached its peak infection rate of over 20,000 hosts per hour that was maintained for about 4 hours. [11] Since the Code-Red worm, there have been several other worm outbreaks causing immense damage. The most noteworthy is the SQL Slammer/Sapphire worm that was activated in January 2003. It was the fastest computer worm in history, doubling its size every 8.5 seconds. It infected over 90% of all vulnerable hosts within the first 10 minutes. [22, 13] An even more virulent design of a hypothetical 'Warhol' worm has been proposed by Nicholas Weaver at UC Berkeley. The Warhol Worm uses topologically aware partitioned scanning of vulnerable hosts. A variant of this class of worms called 'Flash Worms' use priori information in the form of a hit list of potential targets [18].

Code-Red and following worms have inspired research into several countermeasure technologies to contain worm propagation. [26, 12] Signature-based countermeasures have mostly been deemed ineffective due to the extremely fast infection rate of recent worms. Current research on containing these worms is focused on automated detection of anomalous network behavior. The La Brea project, for example, attempts to slow the growth of TCP based worms by slowing probes to unallocated addresses. [7] This is accomplished by blocking the thread making the probe. This technique however can be easily circumvented by running the virus asynchronously. Another approach described by Twycross et al uses per-host TCP 'throttling' by restricting the rate of new connections a machine makes in a given time. [21] Apart from requiring universal deployment to be effective, this technique

involves the difficult task of determining the threshold used to consider a TCP connection as 'new'. This also makes it inflexible to changing network behavior.

Email viruses are different from the various worms discussed so far. Although they propagate autonomously like traditional worms, they still require human action to activate (e.g., opening an email that results in an execution of a script, or opening an email attachment). The SoBig email virus, for example, activates when a user clicks on a .pif email attachment on the Windows operating system. Once launched, it installs itself in the user's start-up folder and propagates by sending emails to valid email addresses stored on the infected machine via its own SMTP engine. It is also capable of receiving updates from a 'master' host though the network. Other email viruses such as Klez and Yaha also propagate using their own SMTP engines.

One method of non-signature based email virus protection is further work by Williamson on email throttling as an extension to the TCP throttling approach described earlier. [24] Williamson's technique seeks to limit the rate at which email can be sent to new users as a way of slowing email virus propagation. Apart from suffering drawbacks similar to TCP throttling, the system works best if implemented inside an actual SMTP server.

The use of machine learning in actual virus detection is an area of research that is surprisingly thin. Early attempts at learning models for the detection of viruses use neural networks to find boot-sector viruses. [20] However, the problem of detecting virus propagation on a network is similar to that of detecting malicious intruders. There has been much recent work in this area using machine learning that we believe is relevant.

Several previously published methods of network intrusion detection involve variants of signature detection. Signature detection uses libraries of past attacks and rule-learning techniques to identify attacks on the network. Paxon's work on Bro provides a strongly-typed language for implementing policies to deal with abnormal events detected on network streams at a TCP/UDP level. [15] However, we believe this technique is inadequate because modern virus attacks can overwhelm network resources before supervised learned systems can be manually updated to provide protection against new threats.

More recent work applies dynamic learning techniques to intrusion detection. Chan, Mahoney, and Arshad's work on Learned Rules for Anomaly Detection (LERAD) and Clustering for Anomaly Detection (CLAD) constructs rules for valid traffic from known clean data that is filtered by CLAD

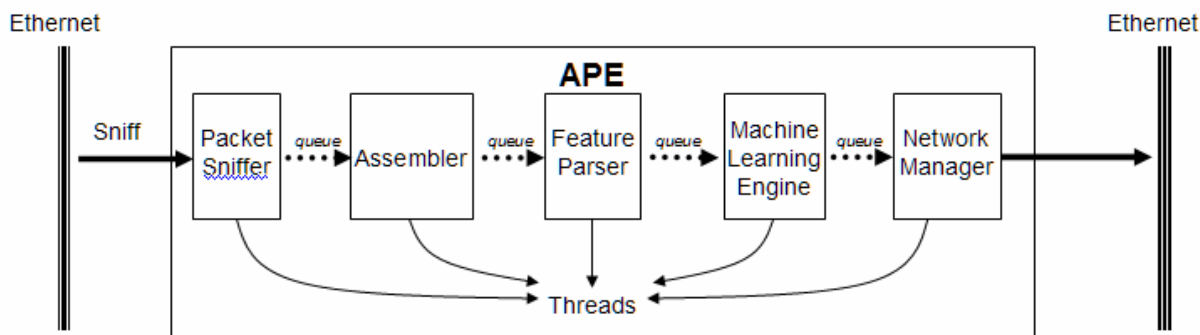


Figure 1. The planned architecture for Adaptive Protection environments.

using a k-Nearest Neighbors approach. These rules are used to score violators. [1]

While there are a wide-variety of available models for providing online novelty detection [9], APE is based on Gaussian Mixture Models (GMMs) and is trained using the EM algorithm for parameter updates. [6, 10] This algorithm has been further extended to incremental and sparse variants [Neal and Hinton] as well as to partial updates. [10]

There is an entire statistics community devoted to the problem of outlier detection in unsupervised learning situations. An overview of this field is provided in Markou et al's work on parametric approaches and non-parametric approaches, while more recent research has revealed the utility of Support Vector Machines (SVMs) in this area. [8] Outlier detection with GMMs has been implemented using several approaches including classical extreme-value theory and more recently using data-mining techniques such as were employed by the SmartSifter system. [4, 25].

3 Architecture of APE

APE is designed with the overall goal of providing consistently high throughput in the analysis of network traffic. To meet this goal, we chose a modular architecture not unlike that of the Click router as the basis for APE. [5]

Each module in APE encapsulates a separate stage in the process of network traffic analysis. These stages are joined together using queues to form an event-driven pipeline. On startup, each module is assigned a thread in which to run. This creates a modular architecture reminiscent of the work on Click, but using threads as the unit of concurrency in a manner similar to research done on Scalable Event-Driven Architectures, or SEDA. [12] We expect this design will provide good performance even under high load.

APE is implemented under Linux version 2.4.18 using gcc version 3.2. The Perl components of the current prototype are interpreted with Perl version 5.8. Threading is handled by the Linux thread library pthreads, and packet sniffing is handled by the pcap library.

Figure 1 gives a visual overview of the modular architecture we have planned for APE, while figure 2 shows the form of the first prototype of our system that is evaluated for this paper. In both figures, captured email goes into the pipeline and is incorporated into the model of network behavior. The model classifies the email as infected or clean using what it knows about what is considered 'normal' network behavior. This classification is fed to the network manager, which must decide if the user who sent the email is infected by a virus or not, and then take the proper action.

The following sections will give an overview of the functionality of each module, including comparisons between the prototype version currently implemented and our planned architecture. Section 3.1 will discuss the packet sniffer and assembler modules. Section 3.2 will describe the feature generator. Finally Section 3.3 will outline the machine learning engine and network manager modules.

3.1 The Packet Sniffer and Assembler

As is shown in figure 1, the first stage of APE handles copying traffic from the network, which is then pushed onto a queue for the next stage to process. Although our current prototype focuses on email, this module has also been implemented at the packet level using the Linux pcap library to capture and filter packets.

The next module in the pipeline is the assembly stage, which takes traffic off the queue and assembles

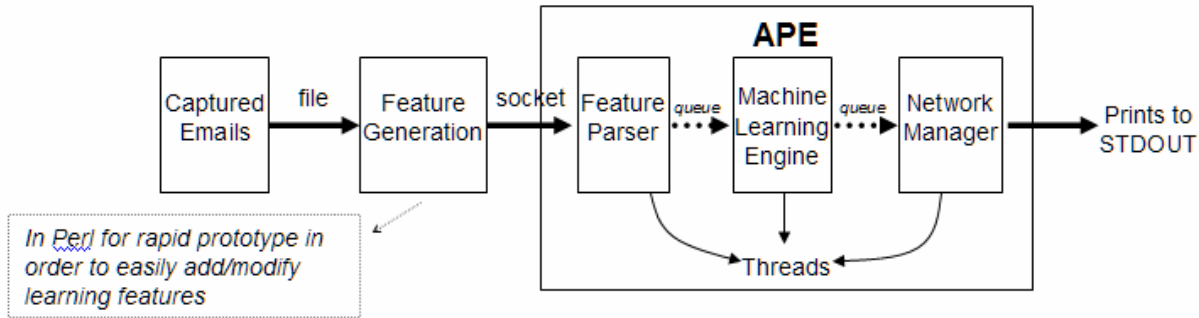


Figure 2. The current architecture of the prototype of APE evaluated in this paper.

it into useful data. As shown in figure 2, the current version of APE reads from pre-captured email, which replaces the network capture and assembly stages. For this reason, the assembler module has not yet been written.

3.2 The Feature Parser

The next stage in the APE pipeline takes the assembled emails and generates vectors of feature values describing each email. Both per-email and behavioral metrics are used when generating these features. The actual methods used to generate the features will be discussed in Section 4.

In the current prototype of APE, this module has been implemented in two parts. The feature vector generator itself is currently written using the Perl programming language, as is shown by figure 2. The reason for this is that Perl allows for rapid development of parsing algorithms. Once the vector is generated, it is then sent to the APE program using Linux socket communication. The first stage of the prototype pipeline then parses the incoming feature vector and puts it on a queue for the machine learning engine to use.

A machine learning model is only as good as the features it learns on; to this end, metrics to generate feature vectors were constantly experimented with throughout the development phase of this project. Once the feature generation is finalized, we will eliminate the Perl part of this stage and implement the module in C++ for speed, adding it into the pipeline as is shown in figure 1.

3.3 The Machine Learning Engine and Network Manager

Once an email has been successfully captured and processed, the next stage is to classify and learn on the new data using unsupervised machine learning.

Our learning engine consists of a Gaussian Mixture Model (GMM) that is trained using the

Expectation-Maximization (EM) algorithm. Details on the implementation of the model will be covered in Section 4 of this paper.

In both the current prototype and the future version, the learning engine pulls feature vectors out of the incoming queue and then attempts to classify the points that have been passed to it using Extreme-Value Theory (EVT). If the vector is not classified as indicating viral infection, the engine learns on it, adding it to the overall model of network behavior.

If the vector is classified as a virus, it is then placed in the queue for the next stage in pipeline. This module is the network manager. In the prototype described in figure 2, the network manager stage just prints out the offending vectors and which machine generated the emails they describe. However, in our future version, we will need to write heuristics for this module to take these classifications and decide if a given machine is actually infected or not. If it is determined to be infected, the module will take some action to quarantine that machine.

4 The Machine Learning Engine

4.1 An Overview Gaussian Mixture Models and the EM algorithm

A classic approach to machine learning is the development of a model capable of distinguishing classes of objects based on a set of features that describe such objects. The model we chose, a Gaussian Mixture Model (GMM), is one example of a model for approximating probability distributions in high dimensional feature spaces. A GMM is a constrained linear superposition of Gaussian (Normal) Distributions whose sum approximates the more complex distribution the model is representing; that is, the model consists of several Gaussian distributions that when added together with a weight on each Gaussian produces the probability of each point in the distribution's space. Such a "mixture" of

Gaussian distributions can be made to approximate any probability density function. The constraints placed on the model are the classic probability axioms. These are 1) the weight on each Gaussian is greater than zero 2) the sum of the weights is one and 3) the probability of a point in the feature space is given by the weighted sum of each distribution's probability at that point. The last of these constraints can be expressed as follows. If the i -th Gaussian distribution of the model has a weight π_i , a mean $\boldsymbol{\mu}_i$, and a covariance $\boldsymbol{\Sigma}_i$, then the model represents an m -dimensional distribution given by:

$$(1) \quad p_{GMM}(\mathbf{x}) = \sum_{i=1}^k \pi_i * N(\mathbf{x} | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$$

$$(2) \quad N(\mathbf{x} | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = \frac{1}{\sqrt{(2\pi)^m \det(\boldsymbol{\Sigma}_i)}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)}$$

where the mean $\boldsymbol{\mu}$ and the covariance $\boldsymbol{\Sigma}$ of each Gaussian are defined as classical statistics describing that Gaussian.

Now that the model has been defined, we show how the parameters of the model, namely the weight π_i , mean $\boldsymbol{\mu}_i$, and covariance $\boldsymbol{\Sigma}_i$ of the i -th Gaussian, can be made to optimally represent the training data set $X_{train} = \{\mathbf{x}_n : n \in 1, 2, K, N\}$. A typical goal of learning is to produce a set of parameters, $\theta_{ML} = \{(\pi_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) : i \in 1, 2, K, k\}$, such that the likelihood, or probability, of the model given the data is maximized. One such technique that can achieve this maximum likelihood is the widely used algorithm expectation-maximization (EM).

The first step of EM, expectation, calculates the expected value of each data point given each Gaussian in the model. In our case, this expected value corresponds to the probability of the data point given the model. These expected values are then used in the second step, maximization, to update the parameters of each Gaussian distribution. This two-step cycle continues until the parameters have sufficiently converged, a condition guaranteed to occur within a finite number of iterations. Moreover, as can be shown, the EM algorithm will converge towards the best fit of the Gaussians to the data. However, EM is prone to local extrema, or globally sub-optimal solutions that appear optimal at a local level. For a more complete description of GMMs and the EM algorithm and its application to fitting a Gaussian Mixture Model to observed data, refer to Jordan's work. [6]

We first discuss the maximization step of our algorithm. Because we chose to use a Gaussian Mixture Model the maximization step is dictated

entirely by the statistical interpretation of the mean and covariance of each of the GMM's Gaussians. In order for the maximization step to calculate the parameters for the next, or $(t+1)$ -th, update of the model, the expectation of the current, or (t) -th, model gives $\tau_n^{i(t)}$, the contribution of the n -th data point to the i -th Gaussian. This is the percentage of ownership the i -th Gaussian has over the n -th point. Thus, the new parameter estimates for the i -th Gaussian are simply the average weight allocated to the Gaussian, the weighted mean of the points "owned" by the Gaussian, and the weighted covariance of the points "owned" by the Gaussian. These weighted parameters will maximize the likelihood of the model with respect to the expected values given in the expectation step. The parameter updates can be expressed as follows: [6, 10]

$$(3) \quad \pi_i^{(t+1)} = \frac{1}{N} \sum_{n=1}^N \tau_n^{i(t)}$$

$$(4) \quad \boldsymbol{\mu}_i^{(t+1)} = \frac{\sum_n \tau_n^{i(t)} \mathbf{x}_n}{\sum_n \tau_n^{i(t)}}$$

$$(5) \quad \boldsymbol{\Sigma}_i^{(t+1)} = \frac{\sum_{n=1}^N \tau_n^{i(t)} (\mathbf{x}_n - \boldsymbol{\mu}_i^{(t+1)}) (\mathbf{x}_n - \boldsymbol{\mu}_i^{(t+1)})^T}{\sum_{n=1}^N \tau_n^{i(t)}}$$

While these equations are sufficient to perform EM, they can be rewritten in terms of their data moments M_i of the GMM. The following reformulated equations allow for greater numerical stability and simplifies the task of implementing other variants of EM: [10]

$$(6) \quad M_i(a, b) = \sum_{n=1}^N (\tau_n^{i(t)})^a (\mathbf{x}_n)^b$$

$$(7) \quad \pi_i^{(t+1)} = \frac{M_i(1, 0)}{M_i(0, 0)}$$

$$(8) \quad \boldsymbol{\mu}_i^{(t+1)} = \frac{M_i(1, 1)}{M_i(1, 0)}$$

$$(9) \quad \boldsymbol{\Sigma}_i^{(t+1)} = \frac{M_i(1, 2)}{M_i(1, 0)} - (\boldsymbol{\mu}_i^{(t+1)}) (\boldsymbol{\mu}_i^{(t+1)})^T$$

We now describe the expectation step. The generalized expectation step is defined by a partition function τ that divides the ownership of point among the Gaussian components of the GMM, commonly by minimizing a distance or error measure. In this framework, $\tau_n^{i(t)}$ is a measure of the degree of ownership of the data point \mathbf{x}_n attained by

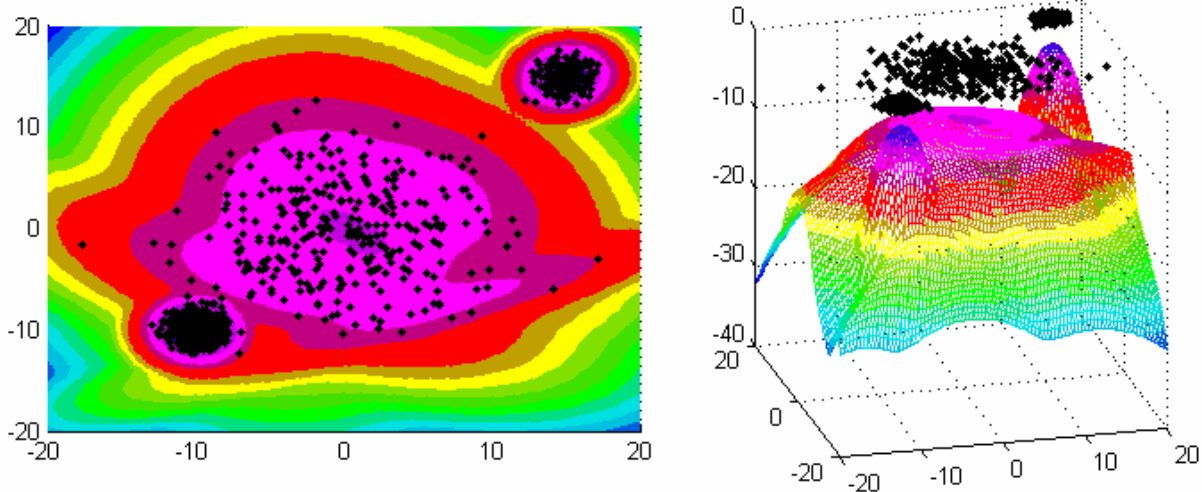


Figure 3. A plot of the log-likelihood of the GMM trained to approximate the distribution that generated a sample two-dimensional training set. On the left is a contour plot of the log-likelihood overlaid by the training data. On the right is the log-likelihood surface plot overlaid by the training data.

the i -th Gaussian given the model's parameters at the current (t -th) update step. These measures are constrained such that the sum of the weights for a particular data point, \mathbf{x}_n , is one, i.e.

$$(10) \quad \sum_{i=1}^k \tau_n^{i(t)} = 1$$

However, this constraint can be relaxed when certain data points are more relevant than others.

There are several ways of defining these measures $\tau_n^{i(t)}$. As an example, during the initialization of our model we use the K-Means algorithm. Under K-Means, the expectation step is a discrete assignment of each point to one of the Gaussians of the mixture model. This is done according to the minimization of Euclidean distance as given by the following equation:

$$(11) \quad \tau_n^{i(t)} = \begin{cases} 1 & \text{if } i = \underset{j}{\operatorname{argmin}} \|\mathbf{x}_n - \boldsymbol{\mu}_j^{(t)}\| \\ 0 & \text{otherwise} \end{cases}$$

Once the K-means initialization is complete, our algorithm becomes classical EM by simply switching the partition function described above. The classic expectation partition function is the probability of the i -th Gaussian component of the GMM given of the n -th training point \mathbf{x}_n : [6]

$$(12) \quad \begin{aligned} \tau_n^{i(t)} &= p(Z^{(t)} = 1 | \mathbf{x}_n, \boldsymbol{\theta}^{(t)}) \\ &= \frac{\pi_i^{(t)} \cdot \mathbf{N}(\mathbf{x}_n | \boldsymbol{\mu}_i^{(t)}, \boldsymbol{\Sigma}_i^{(t)})}{\sum_j \left[\pi_j^{(t)} \cdot \mathbf{N}(\mathbf{x}_n | \boldsymbol{\mu}_j^{(t)}, \boldsymbol{\Sigma}_j^{(t)}) \right]} \end{aligned}$$

where $\mathbf{N}(\ast)$ defines the normal probability density function as given in equation (2).

To show how well a Gaussian Mixture Model is able to approximate a probability density function, two-dimensional data was generated from three normal distributions. The resulting fit of a GMM to this data is shown in figure 3 as a plot of the log-likelihood of the model overlaid by the actual training data.

4.2 Online EM

To make EM into an on-line algorithm able to quickly adapt to changes in network behavior, other variants must be employed. In classical EM, the expectation and maximization steps are prohibitively large as they require a full scan over the entirety of the training data. Moreover, classical EM approaches evolves too slowly since most of the data being trained on is old and possibly outmoded. Hence, modifications of EM known as GEM and ECM, have been proposed to make up for these deficiencies.

Variants of EM known as Generalized Expectation-Maximization (GEM) algorithms modify EM by only updating based on a subset of the data. [14] These GEM algorithms include the partial E-

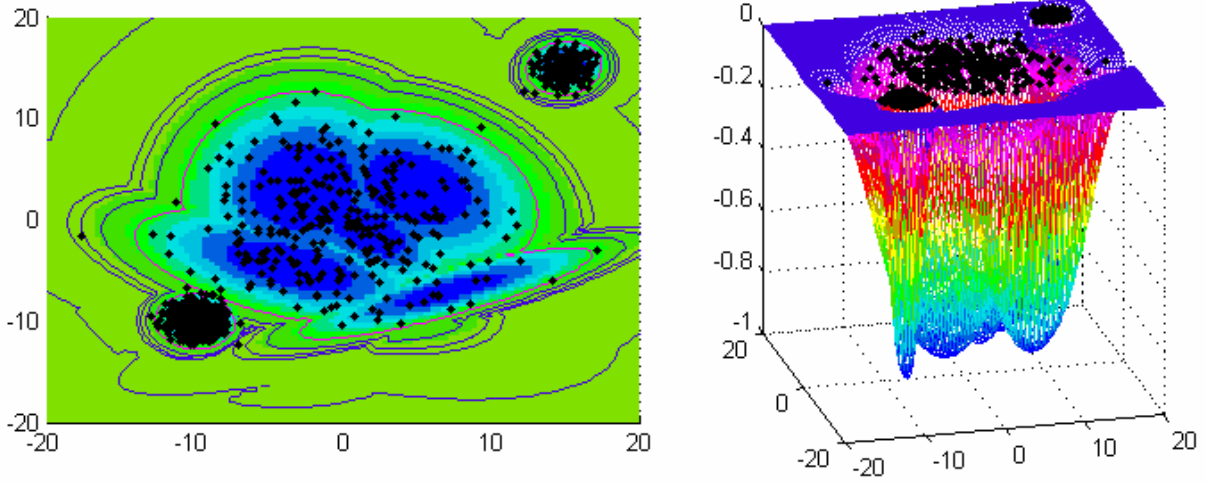


Figure 4. A plot of the EVT probability of the GMM used in Figure 3. On the left is a contour plot of the EVT probability overlaid by the training data. On the right is a surface of the EVT probability overlaid by the training data. Note that the EVT surface was shifted down by 1 so the data would be above the surface plot.

steps that we plan to implement in future versions of APE and are discussed in more detail. [10]

Other variants of EM algorithm modify the expectation step and are collectively known as Expectation Conditional Maximization (ECM) algorithms. [14] By modifying the partition function used in the expectation step, ECM algorithms can streamline EM convergence or implement additional constraints. Using ECM, an on-line version of EM can be constructed by adding aging into the expectation step of our algorithm. This on-line aging can be implemented by multiplying our classic expectation's partition function by an aging function constructed to reduce the value of the partition function for older data points, such that:

$$(13) \quad \hat{\tau}_n^{(t)} = A(\dagger_n) \tau_n^{(t)}$$

$$(14) \quad A(\dagger_n) \in [0.0 \dots 1.0]$$

where preferably $A(*)$ is a monotonically decreasing function from 1 to 0 and \dagger_n is some measure of the elapsed time for the n -th data point. Thus as the data grows older, its relevance to the model decreases, and the model is able to evolve to modern behavior rather than being hindered by old irrelevant data. It should be noted that this function $\hat{\tau}_n^{(t)}$, violates the constraints placed on partition functions in equation (10). However, this violation is valid since, in aging, different points are considered more relevant than others.

4.3 Extreme-Value Theory

One of the most important functions of APE is its ability to classify incoming network traffic based on a model developed from training on a set of unlabeled data. The traditional approach to novelty classification via a Gaussian Mixture Model falls under the area of statistics know as Extreme-Value Theory (EVT). The central idea of EVT is to calculate the probability that a new data point x was not generated by the underlying distribution that generated the training data; that is, the probability that x is an outlier.

To calculate extreme-value statistics for Gaussian Mixture Models the Gumbel distribution is used. The Gumbel distribution is a specific instance of the generalized extreme-value (GEV) distribution with the λ parameter of this distribution, below, approaching zero. The GEV and Gumbel distributions are defined as follows: [17]

$$(15) \quad y_m = \sigma_m^{-1}(x - \mu_m)$$

$$(16) \quad P_{GEV}(X \leq x | \lambda, \mu_m, \sigma_m) = \exp\left\{-\left(1 + \lambda y_m\right)^{-1/\lambda}\right\}$$

$$(17) \quad P_{Gumbel}(X \leq x | \mu_m, \sigma_m) = \exp\left(-\exp(-y_m)\right)$$

where X is a random variable drawn from the underlying distribution, x is point being considered as an extrema, and μ_m and σ_m are “norming parameters” from the underlying distribution. These “norming parameters” are defined asymptotically for the

distribution $D = |N(0,1)|$ (a one-sided normal distribution) as: [17]

$$(18) \quad \mu_m = \sqrt{2 \ln m} - \frac{\ln \ln m + \ln 2\pi}{\sqrt{2 \ln m}}$$

$$(19) \quad \sigma_m = \frac{1}{\sqrt{2 \ln m}}$$

where m is the number of training points drawn from the underlying distribution. For more on extreme-value theory see work by Roberts and Gumbel.

Since extreme-value distributions are univariate and feature spaces are multivariate, we must extract a statistic from the model that describes the distance between points in the feature space and the distribution described by model. In terms of Gaussian Mixture Models, this statistic is the Mahalanobis distance between a data point \mathbf{x} and the Gaussians of the GMM. This distance measure is defined for the i -th Gaussian (mean $\boldsymbol{\mu}_i$ and a covariance $\boldsymbol{\Sigma}_i$) of the GMM as:

$$(20) \quad h_i(\mathbf{x}) = \sqrt{(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)}$$

We can define our extreme value probability measure strictly in terms of i^* , the Gaussian component of the GMM that minimizes the Mahalanobis distance to the data point, since the closest Gaussian dominates the extreme-value analysis. It can be shown that the EVT probability measure of a point relative to a GMM is given by the following equation: [16, 17]

$$(21) \quad P(\mathbf{x} | m) = P_{Gumbel}(X_{i^*} \leq h_{i^*}(\mathbf{x}) | m_{i^*})$$

where m is the number of training points supporting the GMM and $m_{i^*} = m \cdot \pi_i$.

Using the Gumbel distribution for the component-wise Mahalanobis distance of the GMM produces a probability distribution that describes the probability that a point is an outlier to the underlying GMM. Unlike the smoothly varying surface of log-likelihood seen in figure 3, the EVT distribution for the sample GMM changes rapidly to a probability of one outside of the sample data. An example EVT plot using the same data as in figure 3 is shown in figure 4. Not only is the border of GMM is clearly defined, but also the threshold values used to define outliers of this distribution correspond directly to the probability that a point is an outlier rather than some ad-hoc threshold.

4.4 Feature Generation

In between the model and the network, APE must extract features from network traffic to provide a representation of network activity to the model. The choice of these features is a critical decision: features that fail to distinguish between viral and non-viral activity cannot produce a model capable of making that distinction irregardless of the model. Hence, the goal of our feature research is to produce a feature set with a maximal degree of interclass variation.

APE attempts to distinguish between two classes of network activity, normal and viral, in which viral classes are considered outliers to the normal class. To this end, we have defined two types of features: those derived on a per e-mail basis, and those describing network behavior on a per machine basis.

Feature values calculated on a per e-mail basis are:

- Number of attachments
- Total size of attachments
- Number of characters in the subject text
- Number of words in the body

Features describing network behavior over a sliding window are as follows:

- Frequency of emails sent
- Ratio of emails with attachments to those without
- Number of different destination addresses

These features were chosen as a first approximation of potentially relevant features. Our choices were based on observations and reading on viral activity. Feature research is an ongoing part of this project that is further discussed in future work.

5 Evaluation

The ideal evaluation of our prototype would be to perform real time processing of email data from a live network. However, due to time constraints and privacy issues involved in manipulating email data, we explored the next most feasible option. To evaluate APE, we created a Network Email Traffic Simulator based on real email data from several email accounts. The simulator is written using Java 1.4.2, uses Java Mail API to parse emails, and is described in detail in the following section.

Despite the fact that our data is simulated, we believe that using it will still provide insight into the performance of APE. Previous research on email traffic patterns [24] shows that email activity to be spontaneous, with long periods of inactivity (e.g., during the night). In a typical local area network, the percentage of emails sent at a high rate (more than 1

Table 1. Summary of preliminary results obtained by testing the APE prototype with simulated traces of normal and infected email traffic. The numbers shown are calculated as percentages of total emails in the trace.

Virus Name	% Email Infected	% Classified Correctly	% False Positives	% False Negatives
<i>No Virus</i>	0	91	9	N/A
<i>Anna Kournikova</i>	19	90	5	5
<i>SoBig.F</i>	32	79	2	19
<i>Test Virus</i>	20	91	4	5

email per minute) is usually low. Normal email users follow some sending address locality over time; a behavior quite different from an email virus that attempts to send emails to several different addresses in a relatively short time. The traces generated by our simulator follow these patterns. In addition, since our simulator generates a trace based on real email data, we believe that it models actual email content accurately.

5.1 The Network Email Traffic Simulator

To generate a repository of real email, we gathered outgoing email from several accounts on the same network. The accounts chosen had a varying degree of activity, with the most active user sending a little more than 75 emails per week whereas the most passive user sending about 5 in the same time period. We then built a simulator that reads user email files and generates a trace file sorted by email sending time. A similar simulation technique was used by Williamson in evaluating the Email Virus Throttle. [24]

To simulate traces with infected machines, the simulator can insert a stream of virus emails in the trace file. The infected user(s), infection time(s) and average number of virus emails sent per minute can be configured in the simulator. The simulator inserts virus emails with slightly randomized sending times in an attempt to acknowledge network congestion delays while sending email. The simulator generates virus emails by randomly selecting one from a per-virus-pool of infected emails. These infected emails were carefully constructed to be replicas of actual virus emails. The destination address of the virus message is selected at random from a fixed set of email addresses for the infected user. This simulates the fact that several email viruses propagate by retrieving emails from email address books (or other similar sources on the infected machine).

5.2 Experiments and Results

We tested our prototype on normal email traffic and three different email viruses, namely 'Anna Kournikova', 'SoBig.F', and a virus we created to exhibit different behavior from known email viruses. Each experiment was performed with a two week

long trace from several email accounts. For experiments with simulated virus infections, the trace reflected an arbitrary user getting infected after approximately one week. A summary of our results is given in table 1. Details on specific experiments and their results are discussed in the following sections.

5.2.1 Normal Email Traffic

Using the simulated email trace described earlier, our prototype classified over 90% of emails correctly when no viruses were inserted into the trace. As table 1 shows, approximately 9% of the total emails were classified as false positives. We believe that this is because they differed significantly from other emails, and the features currently used for our model are too sensitive.

5.2.2 The AnnaKournikova Virus

The AnnaKournikova virus activates when a Visual Basic script attachment in the virus email titled 'AnnaKournikova.jpg.vbs' is opened. Once activated, the virus propagates by sending replicas of itself to everyone in the infected user's address book with the same format. [19]

We model the Anna Kournikova virus in our trace by inserting 50 infected emails with the same content, attachment size and name as in the actual virus email. After searching for potential email targets, the Anna Kournikova virus sends out infected emails as fast as it can. Hence, we simulate the average number of infected emails sent per minute at a rate of 30 emails per minute.

As table 1 shows, our prototype successfully classified around 90% of AnnaKournikova-infected emails. Among the emails detected inaccurately, we got approximately the same number of false positives as false negatives.

5.2.3 The SoBig.F Virus

The SoBig virus is relatively difficult to detect via statistical methods in general. This is because it propagates by sending nine different email messages and attachments, causing high variation in infected email traffic. The virus propagates via its own SMTP

engine, and sends itself to email addresses found in files with eight particular extensions on the infected host. [19]

Our trace for the SoBig virus contains 100 infected emails. The number of infected emails is more than that used in the AnnaKournikova virus because SoBig performs a more extensive search for email addresses to propagate itself. Each virus email added to the trace was obtained from a set of nine emails that exactly represent the actual SoBig virus emails in terms of message content, attachment size and attachment name. Like the Anna Kournikova virus, SoBig sends emails to the retrieved email addresses as fast as possible. Hence, we simulate the spread rate in the trace at a rate of 30 emails per minute.

APE's classification results for SoBig are not as accurate as the Anna Kournikova virus. As given in table 1, approximately 79% of the total emails were classified correctly. Unlike the Anna Kournikova virus however, false negatives were 12 times the number of false positives. This can likely be attributed to the fact that approximately 30% of all the emails in the trace were infected. We surmise that the reason for lower accuracy with SoBig is the high variability email virus content. While such variability will most likely lead to lower accuracy for any statistical model, we believe that our results can be further improved in the future by more experimentation with the feature set being used by the model.

5.2.4 The Test Virus

To test different virus behavior, we created a test virus that behaves differently from the SoBig and Anna Kournikova viruses. In particular, we simulated the spread of the virus at a much slower rate--approximately 30 times slower than the rate of the previously discussed worms.

As table 1 shows, our simulation results show a high percentage of correctly classified emails (over 90%). We believe this to be a major contribution towards mitigating the grave threat of the more subtle and hard-to-detect 'Stealth Worms', described by Vern Paxson, et al. Such worms spread surreptitiously, masquerading themselves as normal network traffic, and have a lesser chance of being detected via Intrusion Detection Systems that monitor network traffic for anomalies at lower granularity levels. [18]

6 Future Work

APE is an ongoing research effort with several arenas for potential improvements. We believe that the most important of these are the completion of the planned APE architecture and improving the underlying learning techniques used by APE to make them more robust. The following sections will give more detail on what direction we think future work should take in both of these areas.

6.1 Structural Work

The initial version of APE implements a reduced subset of our planned architecture. Some of the modules are still under development. In particular, APE currently lacks the ability to directly obtain and reconstruct packets from the network, for now relying on other packages to do this. In addition, the module for feature generation, currently implemented in Perl, has yet to be directly incorporated into APE.

The APE project to date has only been applied to email traffic. However, future work could extend APE to handle more general types of traffic by including network monitoring on the UDP/TCP level. Such an extension would require new feature research, as it is not as easy to extract features from these more generalized streams of information.

6.2 Learning Work

The current implementation of APE uses a GMM for unsupervised learning of network data. While this model might prove to be best suited to the task of novelty detection on real time network data, other models must be compared in future versions of APE in order to maximize the utility of the software. In particular, the recent successes in one-class SVMs in the domain of unlabeled novelty detection should be considered in future study. [8]

Future GMM development for APE should include additional functionality to make the model more robust. In particular, data aging for online GMMs has not yet been implemented. This or a similar technique will be necessary for a fast-evolving network model. Similarly, other techniques such as using partial E-Steps for faster updates and techniques for dynamically changing the number of mixture components such as Gaussian splitting should be considered for future versions of APE. However, in implementing such features, one must be careful to avoid classic learning pitfalls such as overfitting, which seriously hinders the model's ability to adapt to changing network behaviors. [10]

There is also a great deal of work that remains to be done in the area of feature research. This project shows that GMMs built on primitive features are capable classifying E-mail traffic. However, our current feature set had too high of an error rate and failed to adequately identify the SoBig virus. Stringent feature research is still necessary. Techniques such as principle components analysis (PCA) could potentially be used as an analysis tool in order to reduce the number of dimensions, thus helping to avoid the curse of dimensionality that often plagues learning models. [6]

The final area of future work in learning is how APE classifies machines as infected. While novelty detection is able to detect abnormal network behavior with moderate success, the significant number of false positives might unnecessarily hinder uninfected users. Future versions of APE will need a fully functional module for classifying users as infected based on the classification of their network activities.

6.3 Future Testing

Robust stress testing using real live network data will be necessary to verify that the completed APE software is capable of handling large amounts of network traffic. Such testing should include a wide variety of viral attacks, including attacks specifically designed against the APE's detection mechanisms. Finally, APE should be deployed on real networks. This will verify that it performs as expected on real network traffic, providing users with real-time virus protection without interfering with user productivity.

7 Conclusion

APE, or Adaptive Protection Environments, demonstrates that it is possible to build a program that detects computer virus infections via network behavior. We believe that our approach is more effective at reacting to new viral threats than current commercial rule-based virus protection software. This is because our technique involves unsupervised learning on network transmissions, making APE adaptive to anomalies in the network activity of any given machine.

Our performance analysis on a preliminary prototype of APE backs up this claim. Initially limiting ourselves to email activity, APE was able to detect upwards of 90% of emails from a simulated infection of a novel virus we created expressly to have different behavior than known email viruses. In a real network, APE would be able to detect an infection from this virus and quarantine the infected machine almost immediately, whereas traditional

virus scanners would require database updates before being effective.

While there is much work yet to be done on APE, we sincerely believe that this project could eventually provide a viable first line of defense against the spread of viruses and other malignant traffic. Our prototype demonstrates APE's feasibility and potential; with further work, we anticipate being able to release a deployable tool that could greatly contribute to the struggle against virus attacks.

Acknowledgements

The authors would like to thank Dr. Eric Brewer, Dr. Peter Bartlett, Dr. Michael Jordan, Dr. David Wagner, Dr. Anthony Joseph, and Dr. Tom Anderson for their contributions to and advice on this project. Thanks are also due to the distributors of the mailsnarf software, the TNT and JAMA C++ matrix libraries, the GNU JavaMail API providers Chris Budress et. al., and CERN for their math libraries.

References

- [1] P. K. Chan, M. Mahoney, M. H. Arshad: A Machine Learning Approach to Anomaly Detection, 2003.
- [2] Michelle Delio: Finding the cost of (Virus) Freedom. <http://www.wired.com>, Jan 2002.
- [3] C. Erlanger: How does a Computer Virus Scan Work? www.scientificamerican.com, Jan 2002.
- [4] E. J. Gumbel: Statistics of Extremes. *Columbia University Press, NY, 1958.*
- [5] E. Kohler, R. Morris, B. Chen, J. Jannotti and M. Kaashoek: The Click Modular Router. *ACM Transactions on Computer Systems 18(3), Aug 2000, pg 263-297.*
- [6] M. I. Jordan: *An Introduction to Probabilistic Graphical Models. (unpublished), 2003.*
- [7] T. Liston: Welcome to My Tarpit: The Tactical and Strategic Use of LaBrea. *Dshield.org WhitePaper, 2001.*
- [8] J. Ma and S. Perkins: Online Novelty Detection on Temporal Sequences. *Proceedings of the International Conference on Knowledge Discovery and Data Mining, 2003.*

- [9] M. Markou, S. Sameer: Novelty Detection: A Review. Part 1: Statistical Approaches. *Signal Processing*, 83(12), 2481-2497, 2003.
- [10] S. Martin, et al: Style-Based Inverse Kinematics. (to be submitted for publication SIGGRAPH 2004).
- [11] D. Moore, C. Shannon and K. Claffy: Code-Red: A Case Study on the Spread and Victims of an Internet Worm. *Proc. Of the 2nd Internet Measurement Workshop*, pages 273-284, Nov 2002.
- [12] D. Moore, C. Shannon, G. Voelker and S. Savage: Internet Quarantine: Requirements for Containing self-propagating code. *Proc of 22nd Annual Joint Conference of IEEE Computer and Communication Societies (INFOCOM 2003)*.
- [13] D. Moore, V. Paxson, et al.: Inside the Slammer Worm. *IEEE Security and Privacy* 1(4) pg. 33-39, 2003.
- [14] R. M. Neal, G. E. Hinton: A View of the EM Algorithm that Justifies Incremental, Sparse and other Variants: *Learning in Graphical Models*, M.I. Jordan, Ed. *Kluwer Academic Publishers*. 355—368, 1998.
- [15] V. Paxson: Bro: A System for Detecting Network Intruders in Real-Time. *Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, January 1998*.
- [16] S. J. Roberts: Extreme Value Statistics for Novelty Detection in Biomedical Signal Processing. *Proc. 1st International Conference on Advances in Medical Signal and Information Processing*, 166-172. 2002
- [17] S. J. Roberts: Novelty Detection using Extreme Value Statistics. *IEEE Proceedings on Vision, Image & Signal Processing*, 146(3): 124-129, 1999
- [18] S. Staniford, V. Paxson and N. Weaver: How to Own the Internet in your Spare Time. *Proc. Of the 11th USENIX Security Symposium*, 2002.
- [19] Symantec Corporation: <http://www.symantec.com>. 2003.
- [20] G. Tesauro, J. Kephart, G. Sorkin; Neural Networks for Computer Virus Detection. *IEEE Expert*, 11(4) pgs 5-6, 1996.
- [21] J. Twycross, M. Williamson: Implementing and Testing a Virus Throttle. *Proc of 12th USENIX Security Symposium, Aug 2003*.
- [22] N. Weaver, V. Paxson, et al: The Spread of the Sapphire/Slammer Worm. <http://www.cs.berkeley.edu/~nweaver/sapphire>. 2003.
- [23] M. Welsh, D. Culler, E. Brewer: SEDA: An Architecture for Well-Conditioned, Scalable Internet Services. *Proc of the 18th Symposium on Operating Systems Principles (SOSP-18)*, 2001.
- [24] M. Williamson: Design, Implementation and Test of an Email Virus Throttle. *HP Labs Tech Report HPL-2003-118 20030626*, 2003.
- [25] K. Yamanishi, J. Tekuchi, G. Williams. On-line Unsupervised Outlier Detection Using Finite Mixtures with Discounting Learning Algorithms. *Proc of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.
- [26] C. Zou, W. Gong and W. Towsley: Worm Propagation Modeling and Analysis under Dynamic Quarantine Defense. *WORM 2003*.