Journal of Information Security, 2011, 2, 158-168
doi:10.4236/jis.2011.24016 Published Online October 2011 (http://www.SciRP.org/journal/jis)

Anomalous Network Packet Detection Using Data Stream Mining

Zachary Miller, William Deitrick, Wei Hu*
Department of Computer Science, *Houghton College*, *Houghton*, *USA*
E-mail: *wei.hu@houghton.edu
Received July 4, 2011; *revised July* 29, 2011; *accepted August* 8, 2011

Abstract

In recent years, significant research has been devoted to the development of Intrusion Detection Systems (IDS) able to detect anomalous computer network traffic indicative of malicious activity. While signature-based IDS have proven effective in discovering known attacks, anomaly-based IDS hold the even greater promise of being able to automatically detect previously undocumented threats. Traditional IDS are generally trained in batch mode, and therefore cannot adapt to evolving network data streams in real time. To resolve this limitation, data stream mining techniques can be utilized to create a new type of IDS able to dynamically model a stream of network traffic. In this paper, we present two methods for anomalous network packet detection based on the data stream mining paradigm. The first of these is an adapted version of the DenStream algorithm for stream clustering specifically tailored to evaluate network traffic. In this algorithm, individual packets are treated as points and are flagged as normal or abnormal based on their belonging to either normal or outlier clusters. The second algorithm utilizes a histogram to create a model of the evolving network traffic to which incoming traffic can be compared using Pearson correlation. Both of these algorithms were tested using the first week of data from the DARPA'99 dataset with Generic HTTP, Shell-code and Polymorphic attacks inserted. We were able to achieve reasonably high detection rates with moderately low false positive percentages for different types of attacks, though detection rates varied between the two algorithms. Overall, the histogram-based detection algorithm achieved slightly superior results, but required more parameters than the clustering-based algorithm. As a result of its fewer parameter requirements, the clustering approach can be more easily generalized to different types of network traffic streams.

Keywords: Anomaly Detection, Clustering, Data Stream Mining, Intrusion Detection System, Histogram, Payload

1. Introduction

Since the 1990's, internet usage has become an integral part of our daily lives. As a result, computer networks have experienced an increased number of sophisticated malware attacks. Whereas attackers previously attempted to gain access to restricted resources to demonstrate their skill, a new wave of internet-based attacks has shifted the focus primarily towards criminal motives. Due to the availability of software tools designed to exploit vulnerabilities, attackers can create viruses with greater structural complexity and damaging capability using less sophisticated skills. The security challenges resulting from an increasing number of devices connected to the internet has prompted a significant amount of research devoted to network security.

1.1. Intrusion Detection Systems

One notable topic of network security research is the development of Intrusion Detection Systems (IDS), which attempt to detect threats to a network or host through signature-based or anomaly-based methods. To detect intrusions, signature-based IDS generate "signatures" based on characteristics of previous known attacks. This allows the systems to focus on detecting attacks regardless of ordinary network traffic. Signature-based detection is the most common form of intrusion detection because it is simple to implement once a set of signatures has been created. Although this approach is effective in

finding known threats to a network, it is unable to identify new threats until a new signature is made. To generate an accurate signature, a human expert is generally needed because this cannot easily be done automatically. Since the detection of new threats in a signature-based system is impossible without the aid of a new signature, an alternative method has been proposed.

In contrast to the signature-based approach, anomaly-based IDS adaptively detect new attacks by first generating a "normal" pattern of network traffic. These systems then find anomalies by comparing incoming packets with the "normal" model. Anything that is considered statistically deviant is classified as anomalous. This allows for the systems to automatically detect new attacks though risking possible misclassification of normal behavior(false positive).In addition to the potential for false positives, anomaly-based systems also fall prey to "mimicry attacks", which attempt to evade the IDS by imitating normal network traffic. One such attack is known as a Polymorphic Blending Attack (PBA), in which the attacker uses byte padding and substitution to avoid detection [1]. Recent research has focused on increasing the efficiency, robustness, and detection rates of these systems while lowering their often high false-positive rates.

One of the first well-developed anomaly-based systems is NIDES [2], which builds a model of normal behavior by monitoring the four-tuple header of packets. The four-tuple contains the source and destination IP addresses and port numbers of packet headers [3]. Another system proposed by Mahoney *et al.* [4] was comprised of two different programs, PHAD and ALAD. Whereas PHAD monitors the data contained in the header fields of individual packets, ALAD looks at distinct TCP connections consisting of multiple packets [3, 4]. To detect anomalies, PHAD and ALAD use port numbers, TCP flags, and keywords found in the payload. Yet another approach, known as NETAD [5], monitors the first 48 bytes of each IP packet header and creates different models based on each individual network protocol. Then, using the information recovered from the packet's header, NETAD creates different models each corresponding to a particular network protocol [6].

Two recently developed network anomaly-based intrusion detection systems are PAYL and McPAD [6,7]. Both used n-grams, sequences of n consecutive bytes in a packet's payload, as features to represent packets

To perform anomaly detection, PAYL utilizes 1-grams. This system first generates a histogram for normal traffic, and then a new histogram for each packet's payload. The two histograms are compared using the simplified Mahalanobis distance. If the distance is above a certain threshold, the new packet is flagged as anomalous [7]. Despite this approach's effectiveness, it suffers from a high false positive rate. To combat this, an extension of PAYL was proposed to use n-grams, creating a more precise detection model [8].

McPAD further develops the effectiveness of the n-gram version of PAYL by using 2-nu-grams, sequences of two bytes separated by a gap of size nu. The 2-gram contains the correlation between two bytes, a feature that 1-gram lacks. By combining the 2-gram with nu, McPAD is able to analyze structural information from higher n-grams while keeping the number of features the same as a 2-gram.By varying the value of nu, McPAD builds multiple one-class support vector machine (SVM) classifiers to detect anomalies as an ensemble. These classifiers are first trained on anomaly free data then tested with mixed normal and abnormal packets [6]. Using this approach, McPAD has successfully detected multiple virus types while maintaining a low false positive rate.

1.2. Stream Data Mining

Although PAYL and McPAD have been able to achieve desirable results, they are not designed to deal with the gradual or abrupt change in the data flows. Also because of the Internet's high speed, systems such as PAYL and McPAD can not efficiently store and evaluate the large amount of traffic generated in real-time. To counter these issues, we propose the application of data stream mining techniques to anomaly detection.

Stream mining differs from the traditional batch setting in a number of ways. First and foremost, because data streams are of extremely large or even infinite size, individual objects within the stream may only beanalyzed once—not repeatedly as is possible in batch mode [9]. The continuous nature of data streams also places significant time constraints on stream mining solutions. For a stream mining approach to be practical and effective, it must be able to process incoming information as quickly as it arrives [10].

One of the salient features of any data stream mining algorithm is the ability to detect fluctuations within a continuous stream of data over an unknown length of time. This dynamic tendency of streaming data is called "concept drift" when a change occurs gradually, or "concept shift" when it occurs more quickly [10]. To deal with this characteristic of streaming data, many stream mining algorithms employ a window of time intervals to temporarily hold the most recent data points in a stream [10,11]. The three types of windows typically implemented are landmark window, sliding window and damped window [11].

2. Materials and Methods

2.1. Data

Two publicly available datasets were used to evaluate the anomaly detection algorithms proposed in this study. These were the DARPA'99 intrusion detection evaluation dataset (http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.html), and the attack dataset provided by [6].

The DARPA'99 dataset was used to provide a sampling of normal network traffic. This dataset simulates network communication from a fictitious United States Air Force base [12], and provides both attack-free and attack-containing network traces. Data samples for this study were obtained from HTTP requests found in outside tcp dump data for each day from week one of the DARPA'99 dataset. This first week of data is provided for training purposes and contains no anomalous network traffic. Using Jpcap, a free Java-based library for network packet manipulation, (http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/), the numeric character values for all HTTP packet payloads with lengths at least 1400 characters were extracted for each day. The resulting dataset provided a total of 5594 packets representing normal network traffic divided by days as is shown in **Table 1**.

The anomalous data used in this study were compiled by [6], and are freely available online (http://roberto.perdisci.com/projects/mcpad). We chose to analyze the algorithms' performance in the detection of three out of the four attack types provided: Generic HTTP Attacks, Shell-code Attacks, and CLET Shell-code attacks. [6] obtained 63 of the attacks included in their Generic HTTP dataset from [13]. These attacks include a variety of HTTP attacks collected in a live environment from test web servers, as well as various archives and databases. The attacks fall into several categories, including buffer overflow, URL decoding error, and input validation error, and were directed against numerous web servers such as Microsoft IIS, Apache, Active Perl ISAPI, CERN 3.0A, etc. [6] further supplements these attacks, bolstering the dataset to include a total of 66 HTTP threats. The Shell-code attack dataset includes 11 shell-code attacks (attacks with packets containing executable code in their payload), which are also included in the Generic HTTP attack dataset. Finally, the CLET attacks were generated by [6] using the CLET polymorphic shell-code engine [14]. This created 96 polymorphic shell-code attacks containing ciphered data meant to evade pattern-matching based detection techniques.

Following the same procedure used to process the DARPA'99 week one data, the numeric character values contained in all HTTP packet payloads from each of the three attack datasets with lengths of at least 1400 characters were extracted using Jpcap. This provided a total of 843 attack packets, with varying numbers of packets from each attack type as is detailed in **Table 2**.

The payload information extracted from the DARPA and attack datasets was used to create training and testing datasets for our anomaly detection systems. For each of the five days in the DARPA dataset, 20% of the day's packets were extracted to be used for training, and the remaining 80% of the day's data were set aside to be used for testing. To simulate the network traffic in real time, anomalous packets were then sporadically inserted into both the training and testing data after an initial interval consisting of only normal traffic (50 packets for training data and 200 packets for testing data).In this way, different datasets were created with each attack type for all five days of DARPA'99 week one (See **Figure 1**). The total number of abnormal packets inserted into both the training and testing data was no more than 10% of all normal data for the given day with payload length 1400 characters or more. In some cases, as shown in **Table 3** and **Table 4**, the number of abnormal packets inserted into data was less than 10% because there was not enough attack data available for that day. For the training data, 20% of the abnormal data was mixed with 20% of the normal data for each day. Likewise, 80% of the abnormal data was mixed with 80% of the normal data selected from each day for testing. For each packet, 256 1-gram and 65,536 2-gram features were extracted to produce separate representations of the training and testing datasets detailed in **Table 3** and **Table 4**. The datasets were stored in the ARFF file format used the by the open source machine learning software WEKA [15].

The payload information extracted from the DARPA and attack datasets was used to create training and testing

Table 1. Packets Extracted from DARPA'99 Week 1.

Day	Number of Packets
Monday	688
Tuesday	968
Wednesday	860
Thursday	2308
Friday	770
Total	5,594

Table 2. Packets extracted from McPAD attack datasets.

Attack Type	Number of Packets
Generic HTTP	122
Shell-code	73
CLET Shell-code	648
Total	843

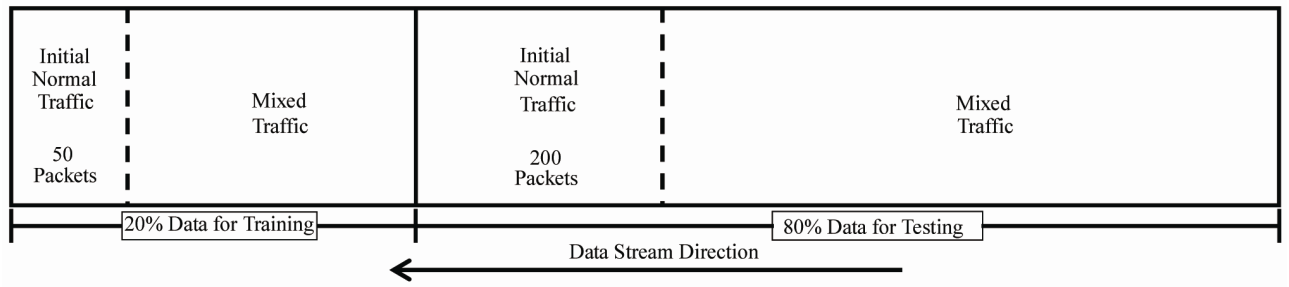

Figure 1. Testing and training data stream diagram.

Table 3. Training dataset.

		CLET	Generic	Shell-code
Mon	Norm.	138	138	138
	Abnorm.	14	14	14
	Total	152	152	152
Tue	Norm.	194	194	194
	Abnorm.	19	19	15
	Total	213	213	209
Wed	Norm.	172	172	172
	Abnorm.	17	17	15
	Total	189	189	187
Thu	Norm.	462	462	462
	Abnorm.	46	24	11
	Total	508	486	473
Fri	Norm.	154	154	154
	Abnorm.	15	15	15
	Total	169	169	169

Table 4. Testing dataset.

		CLET	Generic	Shell-code
Mon	Norm.	550	550	550
	Abnorm.	55	55	55
	Total	605	605	605
Tue	Norm.	774	774	774
	Abnorm.	78	78	58
	Total	852	852	832
Wed	Norm.	688	688	688
	Abnorm.	68	68	54
	Total	756	756	742
Thu	Norm.	1846	1846	1846
	Abnorm.	185	98	58
	Total	2031	1944	1904
Fri	Norm.	616	616	616
	Abnorm.	62	62	50
	Total	678	678	666

datasets for our anomaly detection systems. For each of the five days in the DARPA dataset, 20% of the day's packets were extracted to be used for training, and the remaining 80% of the day's data were set aside to be used for testing. To simulate the network traffic in real time, anomalous packets were then sporadically inserted into both the training and testing data after an initial interval consisting of only normal traffic (50 packets for training data and 200 packets for testing data). In this way, different datasets were created with each attack type for all five days of DARPA'99 week one (See **Figure 1**). The total number of abnormal packets inserted into both the training and testing data was no more than 10% of all normal data for the given day with payload length 1400 characters or more. In some cases, as shown in **Tables 3** and **4**, the number of abnormal packets inserted into data was less than 10% because there was not enough attack data available for that day. For the training data, 20% of the abnormal data was mixed with 20% of the normal data for each day. Likewise, 80% of the abnormal data was mixed with 80% of the normal data selected from each day for testing. For each packet, 256 1-gram and 65,536 2-gram features were extracted to produce separate representations of the training and testing datasets detailed in **Tables 3** and **4**. The datasets were stored in the ARFF file format used the by the open source machine learning software WEKA [15].

2.2. Clustering-Based Anomaly Detection

Clustering algorithms are commonly used for anomaly detection, and are generally created for the batch environment [16]. However, some batch clustering algorithms, such as DBSCAN, can be modified to process stream data.

2.2.1. DBSCAN

DBSCAN is a density-based clustering algorithm devel-

oped for the batch setting. The algorithm takes two user-defined parameters, epsilon (ε) and minimum points, and relies on the concepts of ε-neighborhood and core-objects. An ε-neighborhood is defined by DBSCAN as being a set of points that have a distance to another point less than the user-defined parameter ε. More specifically, given point p and dataset D, the ε-neighborhood of $p\left(N_\varepsilon(p)\right)$ is equal to:

$$N_\varepsilon(p) = \{q \in D |\ dist(p,q) \le \varepsilon\}, \quad (1)$$

where $dist(p,q)$ is the Euclidean distance between points p and q [17].

A core-object is defined as a set of points within an ε-neighborhood that contain more points than the minimum points parameter. If p is part of a core-object, DBSCAN will expand the cluster around p.

The basic structure of the algorithm is as follows:

1) DBSCAN takes the ε and minimum points parameters and then chooses a point p that has not been visited.

2) DBSCAN calculates $N_\varepsilon(p)$. If the size of $N_\varepsilon(p)$ is greater than minimum points, DBSCAN expands a cluster around p. Otherwise, the point is considered noise.

3) DBSCAN iterates to a new un-visited point and repeats the process [18].

Although DBSCAN was originally developed for a batch environment, it has provided an inspiration for stream clustering algorithms.

2.2.2. DenStream

DenStream is a stream clustering algorithm based on DBSCAN with a damped window model. It expands the concept of an ε-neighborhood in DBSCAN with a fading function to maintain up-to-date information about the data stream. The fading function is defined as:

$$f(t) = 2^{-\lambda t}, \quad (2)$$

where $\lambda > 0$ represents the decay factor and t represents the time.

DenStream also modifies the core-object concept of DBSCAN, creating a core-micro-cluster with three additional attributes: radius, center and weight. The radius must be less than or equal to ε, and the weight of a cluster must be greater than the user-defined parameter μ [11]. The weight w, center c and radius r of a core-micro-cluster are more formally defined at time t, for a set of close points, $p_1, p_2, \cdots, p_n$ with time-stamps $T_1, T_2, \cdots, T_n$ as:

$$w = \sum_{i=1}^{n} f(t - T_i), \quad (3)$$

$$c = \frac{\sum_{i=1}^{n} f(t - T_i)\, p_i}{w}, \quad (4)$$

$$r = \frac{\sum_{i=1}^{n} f(t - T_i)\, dist(p_i, c)}{w}, \quad (5)$$

where $dist(p_i, c)$ is the Euclidean distance between the point p_i and the center c.

Because DenStream operates in a stream environment, the core-micro-clusters need to change dynamically as time passes. To facilitate this, a potential core-micro-cluster or p-micro-cluster is introduced. P-micro-clusters are similar to core-micro-clusters, except they differ in that the center and radius values are based on the weighted sum and squared sum of the points ($\overline{CF^1}$ and $\overline{CF^2}$). Also, the weight must be greater than or equal to $\beta\mu$ where β defines the threshold between p-micro-clusters and outliers(described in the next paragraph) such that $0 < \beta \le 1$. $\overline{CF^1}$ and $\overline{CF^2}$ are calculated using the formulas:

$$\overline{CF^1} = \sum_{i=1}^{n} f(t - T_i)\, p_i, \quad (6)$$

$$\overline{CF^2} = \sum_{i=1}^{n} f(t - T_i)\, p_i^2. \quad (7)$$

This changes the center and radius values to be [11, 19]:

$$c = \frac{\overline{CF^1}}{w}, \quad (8)$$

$$\text{and } r = \sqrt{\frac{\overline{CF^2}}{w} - \left(\frac{\overline{CF^1}}{w}\right)^2}. \quad (9)$$

Although the p-micro-cluster permits the model to be updated dynamically, it generally will not provide a representative view of a data stream as new points appear. To handle this concept drift, DenStream also introduces the outlier-micro-cluster (or o-micro-cluster) and an outlier-buffer that temporarily stores o-micro-clusters and allows them to become p-micro-clusters. The operation of DenStream is as follows:

Initial Step: run DBSCAN on a set of initial points to generate starting p-micro-clusters.

Online Steps, when a new point p arrives in the stream:

1) The algorithm attempts to merge p with the closest p-micro-cluster. If the radius of the potential micro-cluster is less than or equal to the value of ε, the point is merged.

2) If the point is not merged to a p-micro-cluster, it tries to merge p with an existing o-micro-cluster. If the radius is less than ε, it is merged with the o-micro-cluster. Then if the o-micro-cluster now has a weight large enough to become its own p-micro-cluster, it is removed

from the outlier-buffer and added to the model as a p-micro-cluster.

3) If the point cannot be merged to an existing o-micro-cluster, it creates a new o-micro-cluster and gets placed in the outlier-buffer.

After the merging phase of the DenStream algorithm, the lower weight limit is calculated for all o-micro-clusters in the outlier buffer. This is done using the formula:

$$\xi\left(t_c, t_0\right) = \frac{2^{-\lambda\left(t_c - t_0 + T_p\right)} - 1}{2^{-\lambda T_p} - 1}, \qquad (10)$$

where $\lambda > 0$ represents the decay factor, t_c and t_0 represent the current and starting time for the o-micro-cluster, and T_p is the predetermined time-period.

4) If the weight of a particular cluster is less than the lower weight limit, the o-micro-cluster can be removed from the outlier buffer.

2.2.3. Our DenStream-Based Detection System

To detect anomalous packets, DenStream was modified to create the DenStreamDetection algorithm, which treats incoming packets as points to be clustered. When a packet is merged with a p-micro-cluster, it is classified as normal. Otherwise, it is sent to the outlier-buffer and classified as anomalous. The ability for o-micro-clusters to be promoted to p-micro-clusters was removed because the majority of the packets clustered to the outlier-buffer are abnormal packets. If one of these o-micro-clusters became a p-micro-cluster, the model would be tainted and therefore unable to differentiate between abnormal and normal packets.

The basic structure of DenStreamDetectionis shown in **Algorithm 1**.

2.2.4. Creation of the Detection Model

The anomaly detection model was created in two steps. The first step used the training data to find a range for the parameters in DenStreamDetection such as ε and minimum points. Using 50 initial points, multiple DenStreamDetection models for each day were created to find a range of optimal parameters that could be used in the testing step. We found that ε had a larger impact on the predictions than the minimum points. During the first step, different parameter ranges were identified based on day and abnormal packet type.

The second step used the testing data to make a prediction model, which was evaluated with the sensitivity and false positive rates defined in Section 2.4. A false positive is a normal packet classified as abnormal. The parameters used in this step were 200 initial packets, 10 minimum points and a range of ε values specific to each day and attack type determined from the first step. Using these parameters, models were generated with a range of false positive and sensitivity rates to demonstrate overall performance.

```
Algorithm: DenStreamDetection (iniP, minP, ε)
Parameters: iniP: number of initial packets
            minP: number of minimum packets in initial p-micro-clusters
            ε: distance threshold
Input: File containing normal and abnormal packets in n-gram format.
Output: Predictions of normal and abnormal packets.

1.  Initialize DenStream using iniP and minP to build p-micro-clusters.
2.  As each packet p comes in:
3.  Try to merge p into its nearest p-micro-cluster c_p
4.  if radius(c_p) ≤ ε then
5.      Merge p into c_p;
6.      Classify as normal packet;
7.  else
8.      Try to merge p into nearest o-micro-cluster c_0;
9.      if radius(c_0) ≤ ε then
10.         Merge p into c_0;
11.         Classify as abnormal packet;
12.     else
13.         Create a new o-micro-cluster and insert into outlier-buffer;
14.         Classify as abnormal packet;
15.     end if
16. end if
```

Algorithm 1. DenStream Detection algorithm.

2.3. Histogram-Based Anomaly Detection

Another approach to the detection of anomalous network packets has involved the use of histograms to maintain statistical information about network packet payloads. PAYL is an example of such a system [7], in which a model is created for known normal packet payloads and then compared with incoming packet payloads to determine whether or not the newly arriving packets are anomalous. Due to the evolutionary nature of streaming data, it is important that any abnormal packet detection method is able to update its normal model as concept drift occurs in the incoming data stream. With this in mind, we present a histogram-based classification method capable of modeling dynamic network traffic in real time.

2.3.1. Algorithm Description

The histogram-based detection algorithm provides a simple method for classification of network traffic. The algorithm, summarized in **Algorithm 2**, creates a histogramen compassing a "normal" model of the network packets expected to be encountered. This histogram is generated by counting the frequency of n-gram features


```
Algorithm: HistogramDetection (x, w, q, t, r, h, λ)
Parameters: x: number of initial normal packets
            w: size of Pearson correlation queue
            q: size of rebuild queue
            t: classification threshold
            r: rebuild count
            h: rebuild threshold
            λ: decay factor
Input: File containing normal and abnormal packets in n-gram
       format
Output: Predictions of normal and abnormal packets

1.  Initialize histogram g using x initial packets
2.  Initialize rebuild queue b to queue of size w
3.  Initialize Pearson correlation log l to queue of size w
4.  As each new packet p arrives
5.  decay (g)
6.  Create histogram c from packet p
7.  if pearsonCorrelation(g,c) >t then
8.      Classify p as normal
9.      b.addFirst(p)
10.     if (b.size() >q) then
11.         b.removeLast()
12.     end if
13.     l.addFirst(pearsonCorrelation(g,c))
14.     if (l.size() > w) then
15.         l.removeLast()
16.     end if
17.     Calculate number of packetsn in l with values ≤ h
18.     if n ≥ r then
19.         rebuild(g)
20.         l.clear()
21.     else
22.         Classify p as abnormal
23.     end if
24. end if
```

Algorithm 2. Histogram-based detection algorithm.

found within packet payloads. To begin classification of a stream of packets, the algorithm first requires x initial normal packets to construct the normal model histogram. This histogram contains frequency counts from all initial packets for each possible n-gram attribute. Since we are attempting to model normal traffic, it is imperative that no abnormal packets are included when this model is created or else the model will be contaminated and detection rates will decrease. To effectively reflect the evolutionary nature of network traffic, the same fading function with decay factor λ used in DenStream is applied to the histogram after each new packet is processed. This helps to reduce the impact of outdated stream data. After the initial histogram has been built, the algorithm can begin to classify the subsequent packets.

In order to classify an incoming network packet, the algorithm builds a histogram from the newly arrived packet's payload. The histogram generated from the new packet is then compared with the normal model histogram (to which the fading function has been applied as each new packet comes in) by computing the Pearson correlation value between the two histograms. If the computed Pearson correlation value is above auser-defined threshold t, the packet is classified as normal; otherwise, the packet is classified as abnormal.

To account for the possibility of concept drift and shift occurring in data flows, the normal histogram model may need to be rebuilt using packets that have arrived since the initialization of the normal histogram model. This allows the normal model to stay current, modeling packets most recently classified as normal. In order to facilitate this rebuilding process, the algorithm maintains two queues of user-defined size containing information from previously processed normal packets. One of these queues, of size q, stores the histogram data for the previous packets, while the other, with size w, stores Pearson correlation values computed between the packets and the normal histogram model. Note that only data for packets classified as normal are included in these two queues; any packets classified as abnormal are not taken into account. If the normal histogram is to be rebuilt, a set of user-specified conditions must be met, giving the user control of the rebuilding process. When the model is rebuilt too often, the algorithm's efficiency will decrease significantly; however, if it is not rebuilt enough, accuracy will diminish. To determine when rebuilding the normal model is necessary, the algorithm calculates the number of Pearson correlation values in the stored queue that are below the user-defined threshold h. If this count is found to be of a certain value r, the normal model is rebuilt using packets stored in the histogram data queue and the queue containing previous Pearson correlation values is emptied.

2.3.2. Critical Parameters

Though the histogram-based algorithm requires several parameters, it is important to note that these are not equally important. Rather, two parameters in particular have the greatest effect on the algorithm's ability to detect anomalous packets.

The first of these most critical parameters is q, the size of the queue of previously processed instances used to rebuild the normal histogram model. If this value is too small, the normal histogram model generated when the model is rebuilt will not take into account a sufficient number of previously processed packets. This results in an insufficiently robust model, causing both undesirable sensitivity values and false positive rates.

While q has a noticeable influence on the effectiveness of the histogram detection algorithm, t, which defines the Pearson correlation threshold between instances classified as normal and abnormal, is undoubtedly the most important parameter. This is understandable, as t directly controls the classification of each individual instance as it is processed by the algorithm. Furthermore, t also plays

a role in controlling the rebuilding of the normal histogram model. Because parameter h specifies an interval above t, t is directly related the frequency at which the normal model histogram is rebuilt. Thus, the value of t is closely connected to the core functionality of the histogram-based detection algorithm.

2.4. Performance Metrics

To evaluate the performance of the anomaly detection models, the sensitivity and false positive rates were calculated using the following formulas:

$$\text{sensitivity} = \frac{TP}{TP + FN}, \tag{11}$$

$$\text{false positive rate} = \frac{FP}{TN + FP}, \tag{12}$$

where TP/FN stand for the number of correctly/incurrectly classified abnormal packets, and TN/FP are the number of correctly/incorrectly classified normal packets. Sensitivity measured how well the model detected abnormal packets, and false positive rate indicated the percentage of false alarms generated.

3. Results and Discussion

3.1. Density-Based Detection Results

After tuning the DenStreamDetection-based system on packets using 2-gram features, we discovered a range of ε values for each day that could be used to evaluate the model. For every day except for Tuesday, the false positive rate was kept between 0% and 10% so that an appropriate detection rate could be found. Tuesday, however, needed the false positive limit to be heavily relaxed in order to achieve a moderate sensitivity. When testing the detection system, the false positive and sensitivity rates for the highest, middle and lowest ε values were generated for both1-gram and 2-gram feature representations. These are displayed in **Table 5**. The results with highest sensitivity for each virus type were then averaged to find best overall performance.

In general, the DenStreamDetection-based system was able to correctly detect most Shell-code attacks, achieving on average 91% sensitivity with a 14% false positive rate. Similarly, Generic HTTP attacks produced 78% average sensitivity and a 13% false positive rate. CLET attacks, however, had a similar false positive rate of 14%, but a substantially lower average sensitivity of 65%. This disparity was likely due to the polymorphic nature of CLET attacks, which are designed to mimic normal network traffic.

Table 5. DenStreamDetection system results.

DenStreamDetection System Results 2-gram(1-gram)

Day	ε	CLET		Generic Http		Shell-code	
		FP	Sens	FP	Sens	FP	Sens
Mon	30	7(20)	75(78)	7(20)	78(95)	7(20)	98(100)
	45	7(9)	67(67)	7(10)	76(76)	7(10)	96(95)
	60	6(6)	49(49)	6(7)	73(70)	6(7)	93(93)
Tue	65	35(35)	62(62)	33(35)	74(74)	35(35)	86(86)
	80	34(33)	56(56)	33(34)	72(72)	34(34)	81(81)
	100	33(33)	51(50)	32(33)	69(69)	33(33)	76(79)
Wed	95	11(11)	38(37)	11(11)	63(62)	11(11)	81(81)
	110	9(10)	29(29)	10(10)	60(60)	10(10)	78(78)
	125	8(8)	25(25)	8(8)	54(56)	8(8)	70(72)
Thu	5	6(4)	96(98)	6(3)	99(1)	6(3)	98(1)
	30	3(2)	84(84)	4(1)	90(90)	4(1)	93(93)
	55	0(2)	76(78)	0(1)	81(81)	0(1)	81(82)
Fri	55	9(9)	56(58)	9(9)	76(77)	9(9)	92(94)
	65	8(9)	53(53)	8(9)	74(74)	8(9)	92(92)
	75	8(8)	50(49)	8(8)	71(71)	8(8)	88(88)
Best. Avg		14(16)	65(67)	13(16)	78(82)	14(16)	91(92)

Thursday exhibited the highest detection rates (up to 99%) whilst keeping the false positive rates below 6%. Also, Thursday experienced both the lowest ε value sand the largest ε range to achieve its results.

The models utilizing both 1-gram and 2-gram featuresproduced similar results. Using the 1-gram representation for the same ε values, the system experienced slightly better detection rates at the expense of higher false positive rates. Also, because the 1-gram representation has a much smaller feature space than 2-gram, the total run-time of 1-gram was significantly less.

3.2. Histogram-Based Detection Results

3.2.1. Optimal Parameters

The histogram-based algorithm was applied to anomalous packet detection in two steps: training and testing. In the training step, favorable parameters for the algorithm were approximated by performing several experiments on the training data. Since the training data included 50 initial normal packets, this value was used for x during the training step. Most critically, appropriate values of t were ascertained for the different attack types on each day, as this parameter has the greatest effect on the performance of the algorithm. Suitable values were also obtained for all other parameters during the training phase. The optimum value of q was found to be 200, as this allowed the algorithm to maintain a fairly accurate model of normal traffic while minimizing the time needed

to for this model to be rebuilt. Also, 30 was generally used for w, with r valued at 10 and h at 0.2. These parameters effectively limited the frequency of rebuilding the normal histogram model while still allowing the algorithm to handle concept drift in the data. A λ value of 0.01 was found to work sufficiently well, as this decay factor helped to better maintain an up-to-date normal model for normal traffic.

Once appropriate parameters were identified, the testing phase began. In this step, the value 200 was assigned to x since the testing data contained 200 initial normal packets. With the rest of the algorithm's parameters remaining static, the algorithm was tested using varying values of t in order to gauge sensitivity values at different false positive rates. This produced the results summarized in **Table 6**, which displays a sampling of t values used for different days and attack types with the resulting sensitivities and false positive rates.

3.2.2. Results Achieved

As can be seen from **Table 6**, we were able to attain fairly consistent results across all days' testing data for each attack type using optimal parameters. The algorithm performed best at detecting shell-code attacks, achieving an average of 97% sensitivity and 1% false positive rate across all shell-code attack testing datasets. Performance was slightly less desirable in detection of Generic HTTP attacks, but was nevertheless acceptable, with an average detection rate of 84% and 3% false positive rate. CLET attacks proved most difficult for the histogram-based algorithm to detect. In order for reasonable detection rates to be obtained, false positive rates generally had to be pushed much higher than necessary for the other two attack types, to an average of 7% with optimal parameters. Despite this fact, sensitivity remained comparatively low, at an average of 61%. This difficulty detecting CLET attacks was likely due to their polymorphic nature, which also proved troublesome to the DenStream Detection system.

Table 6. Histogram-based detection system results.

Histogram Detection System Results 2-gram									
Day	CLET			Generic HTTP			Shell-code		
	t	FP	Sens	t	FP	Sens	t	FP	Sens
Mon	0.0025	0	25	0.0025	0	76	0.0005	0	58
	0.0060	7	31	0.0070	5	76	0.0010	0	78
	0.0080	8	60	0.0090	9	76	0.0035	0	96
Tue	0.00005	1	27	0.0002	1	27	0.00001	1	31
	0.0025	7	37	0.0010	2	71	0.0010	2	79
	0.0035	9	40	0.0015	2	83	0.0015	2	97
Wed	0.0020	1	29	0.0004	0	38	0.0005	0	43
	0.0040	2	35	0.0010	0	66	0.0010	0	78
	0.0045	3	44	0.0030	1	81	0.0020	1	96
Thu	0.0050	0	69	0.0005	0	78	0.0005	0	97
	0.1505	3	98	0.0010	0	88	0.0755	2	98
	0.4000	3	100	0.0855	2	100	0.1000	3	100
Fri	0.0005	0	26	0.0005	0	39	0.0010	0	60
	0.0020	2	27	0.0010	0	61	0.0015	1	82
	0.0065	13	63	0.0015	1	79	0.0020	1	96
Best Avg.		7	61		3	84		1	97

While results were relatively consistent across each days' worth of training data, those achieved using Thursday's testing data were markedly superior. Using optimal parameters, the histogram-based detection algorithm was able to achieve perfect detection on all three types of attacks, each with a false positive rate of 3% or less. There are several possible reasons for this exceptional performance related to the nature of Thursday's testing data as discussed in Section 3.3. Overall, the relatively high t values used indicate greater consistency in Thursday's network traffic. Due to these elevated t values, the algorithm was able to more effectively identify abnormal network packets.

The number of parameters required by the histogram-based detection algorithm necessitated fairly specific tuning for our different datasets in order to perform optimally. This was demonstrated by the algorithm's performance on 1-gram features when the same parameters used for 2-gram were applied. While the clustering-based algorithm was able to achieve comparable results from both 1-gram and 2-gram with the same parameters, the histogram-based algorithm performed poorly on 1-gram when applied with the same parameters as 2-gram. As a result, 1-gram results have not been reported for the histogram-based algorithm.

3.3. Concept Shift

The performance of our anomaly detection systems is heavily influenced by the evolutionary nature of network traffic. To demonstrate this challenge, we calculated the Pearson correlation between segments of ten packets for each day. By measuring the correlation between two consecutive segments, changes in the stream can be visualized (**Figure 2**), which offers a possible explanation to the results observed in **Tables 5** and **6**.

Monday, Wednesday, and Friday exhibit continuous concept shift, particularly during the training phase, as the calculated Pearson correlation values oscillate regularly. Therefore, a robust model for normal traffic was built in each case that responded accurately to the evolving data stream. As a result, greater sensitivity was achieved on each of these three days.

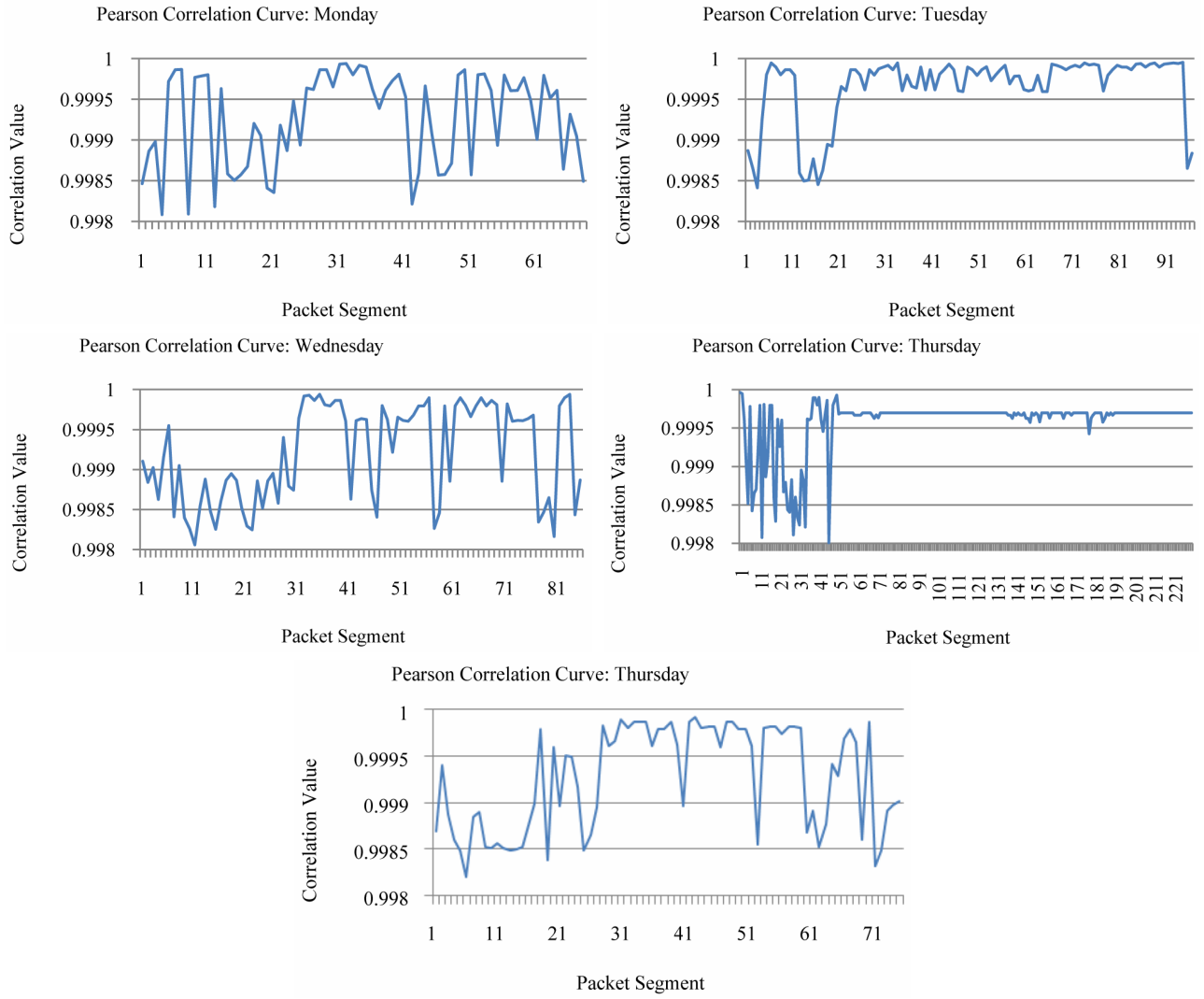

Figure 2. Pearson correlation curves for DARPA'99 week 1.

Tuesday demonstrates a very distinct pattern, as it starts with slight shift, but remains relatively stable throughout the stream. In contrast to Monday, Tuesday experiences only two major shifts, both of which occur during the training phase. Therefore, Tuesday's models may have a less accurate view of the incoming stream, causing decreased performance. Since Tuesday only experienced shift during the training phase, it had a less accurate view of the changing stream environment. This may have led to the performance issues stated previously.

The high sensitivity values for Thursday can also be explained through concept shift. During the training phase, consistent concept shift occurred, allowing Thursday's model to effectively capture the changing pattern of normal network traffic. Following the training step, the remainder of Thursday's data was relatively stable. Therefore, Thursday likely experienced consistent concept drift, which led to superior detection results.

4. Conclusions

In this paper, two data stream mining techniques were applied to the problem of anomaly detection. First, a stream clustering algorithm was used to detect abnormal packets. Using 1-gram and 2-gram features, this approach achieved moderate success with Generic HTTP and Shell-code attacks but had a higher average false positive rate. Second, a stream adaptation of the relative frequency histogram approach found in [7] was created using Pearson correlation to detect anomalies.

Though the histogram-based approach achieved moderately better results, it required more fine-tuning because of the number of parameters used. In contrast,

generalization of the clustering algorithm was easier to achieve since it uses fewer parameters. This was evidenced by the ability of the clustering algorithm to perform effectively on both 1-gram and 2-gram features with the same parameters, while the histogram algorithm required specific parameter tuning for each feature type.

Lastly, to better explain the performance differences between certain days, we analyzed the Pearson correlation between consecutive segments of 10 packets. By plotting these values on a graph, concept drift and shift were visualized, and clear variations were observed between days. The location and frequency of concept shift and drift in the data streams, especially within the training phase, provided an account for the observed changes in performance.

5. Acknowledgements

We would like to thank the Summer Research Institute at Houghton College for providing funding for our research.

6. References

[1] R. Perdisci, G. Gu and W. Lee, "Using an Ensemble of One-Class svm Classifiers to Harden Payload-Based Anomaly Detection Systems," *ICDM*'06: *Proceedings of the Sixth Integnation Conference on Data Mining*, Hong Kong, 18-22 December 2006, pp. 488-498. doi:10.1109/ICDM.2006.165

[2] D. Anderson, T. Lunt, H. Javits and A. Tamaru, "Nides: Detecting Unusual Program Behavior Using the Statistical Component of the Next Generation Intrusion Detection Expert System," *Technical Report SRI-CSL*-95-06, Computer Science Laboratory, SRI International, Menlo Park, May 1995.

[3] R. Perdisci, "Statistical Pattern Recognition Techniques for Intrusion Detection in Computer Networks, Challenges and Solutions," University of Cagliari, Italy, 2006.

[4] M. Mahoney and P. Chan, "Learning Non Stationary Models of Normal Network Traffic for Detecting Novel Attacks," *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, July 2002, pp. 376-385.

[5] M. Mahoney, "Network Trafic Anomaly Detection Based on Packet Bytes," ACM-SAC, Melbourne, 2003 pp. 346-350.

[6] R. Perdisci, D. Ariu, P. Fogla, G. Giacinto and W. Lee, "McPAD: A Multiple Classifier System for Accurate Payload-based Anomaly Detection," *Computer Networks*, *Special Issue on Traffic Classification and Its Applications to Modern Networks*, Vol. 5 No. 6, 2009, pp. 864-881.

[7] K. Wang and S. Stolfo, "Anomalous Payload-Based Network Intrusion Detection," *Recent Advances in Intrusion Detection* (*RAID*), Vol. 3224, 2004, pp. 203-222. doi:10.1007/978-3-540-30143-1_11

[8] K. Wang, "Network Payload-Based Anomaly Detection and Content-Based Alert Correlation," Columbia University, New York, 2006.

[9] J. Tang, "An algorithm for Streaming Clustering," MSc. Thesis, Uppsala University, Uppsala, 2011.

[10] A. Bifet, G. Holmes, R. Kirkby and B. Pfahringer, "MOA: Massive Online Analysis," *Journal of Machine Learning Research*, Vol. 11, 2010, pp. 1601-1604.

[11] F. Cao, M. Ester, W. Qian and A. Zhou, "Density-Based Clustering over an Evolving Data Stream with Noise," *SIAM Conference Data Mining*, Bethesda, 2006.

[12] R. Lippmann, J. Haines, D. Fried, J. Korba and K. Das, "The 1999 DARPA Off-Line Intrusion Detection Evaluation," *Computer Networks*, Vol. 34, No. 4, 2000, pp. 579-595. doi:10.1016/S1389-1286(00)00139-0

[13] K. L. Ingham and H. Inoue, "Comparing Anomaly Detection Techniques for HTTP," *Recent Advances in Intrusion Detection* (*RAID*), 2007.

[14] T. Detristan, T. Ulenspiegel, Y. Malcom and M. Underduk, "Polymorphic Shellcode Engine Using Spectrum Analysis," *Phrack*, Vol. 11, No. 61, 2003.

[15] I.H. Witten and E. Frank, "Data Mining: Practical Machine Learning Tools and Techniques," 2nd Edition, Morgan Kaufmann Publishers, Waltham, 2005.

[16] L. Portnoy, E. Eskin and S. Stolfo, "Intrusion Detection with Unlabeled Data Using Clustering," *Proceedings of ACM CSS Workshop on Data Mining Applied to Security* (*DMSA*-2001), Philadelphia, 2001, pp. 333-342.

[17] M. Ester, H. Kriegel, J. Sander and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *International Conference on Knowledge Discovery in Databases and Data Mining* (*KDD*-96), Portland, August 1996, pp. 226-231.

[18] K. Mumtaz and K. Duraiswamy, "An Analysis on Density Based Clustering of Multi Dimensional Spatial Data," *Indian Journal of Computer Science and Engineering*, Vol. 1, No. 1, 2010, pp. 8-12.

[19] A. Forestiero, C. Pizzuti and G. Spezzano, "FlockStream: a Bio-Inspired Algorithm for Clustering Evolving Data Streams," *ICTAI*'09 *Proceedings of the* 2009 21*st IEEE International Conference on Tools with Artificial Intelligence*, Washington DC, 2009, pp. 1-8.

