Anomalous Payload-based Network Intrusion Detection

Ke Wang, Salvatore J. Stolfo

Computer Science Department, Columbia University
500 West 120th Street, New York, NY, 10027
{kewang, sal}@cs.columbia.edu

Abstract. We present a payload-based anomaly detector, we call PAYL, for intrusion detection. PAYL models the normal application payload of network traffic in a fully automatic, unsupervised and very effecient fashion. We first compute during a training phase a profile byte frequency distribution and their standard deviation of the application payload flowing to a single host and port. We then use Mahalanobis distance during the detection phase to calculate the similarity of new data against the pre-computed profile. The detector compares this measure against a threshold and generates an alert when the distance of the new input exceeds this threshold. We demonstrate the surprising effectiveness of the method on the 1999 DARPA IDS dataset and a live dataset we collected on the Columbia CS department network. In once case nearly 100% accuracy is achieved with 0.1% false positive rate for port 80 traffic.

1 Introduction

There are many IDS systems available that are primarily signature-based detectors. Although these are effective at detecting known intrusion attempts and exploits, they fail to recognize new attacks and carefully crafted variants of old exploits. A new generation of systems is now appearing based upon anomaly detection. Anomaly Detection systems model normal or expected behavior in a system, and detect deviations of interest that may indicate a security breach or an attempted attack.

Some attacks exploit the vulnerabilities of a protocol, other attacks seek to survey a site by scanning and probing. These attacks can often be detected by analyzing the network packet headers, or monitoring the network traffic connection attempts and session behavior. Other attacks, such as worms, involve the delivery of bad payload (in an otherwise normal connection) to a vulnerable service or application. These may be detected by inspecting the packet payload (or the ill-effects of the worm payload execution on the server when it is too late after successful penetration). State of the art systems designed to detect and defend systems from these malicious and intrusive events depend upon "signatures" or "thumbprints" that are developed by human experts or by semi-automated means from known prior bad worms or viruses. They do not solve the "zero-day" worm problem, however; the first occurrence of a new unleashed worm or exploit.

Systems are protected after a worm has been detected, and a signature has been developed and distributed to signature-based detectors, such as a virus scanner or a firewall rule. Many well known examples of worms have been described that propagate at very high speeds on the internet. These are easy to notice by analyzing the rate of scanning and probing from external sources which would indicate a worm propagation is underway. Unfortunately, this approach detects the early onset of a propagation, but the worm has already successfully penetrated a number of victims, infected it and started its damage and its propagation. (It should be evident that slow and stealthy worm propagations may go unnoticed if one depends entirely on the detection of rapid or bursty changes in flows or probes.)

Our work aims to detect the first occurrences of a worm either at a network system gateway or within an internal network from a rogue device and to prevent its propagation. Although we cast the payload anomaly detection problem in terms of worms, the method is useful for a wide range of exploit attempts against many if not all services and ports.

In this paper, the method we propose is based upon analyzing and modeling normal payloads that are expected to be delivered to the network service or application. These normal payloads are specific to the site in which the detector is placed. The system first learns a model or profile of the expected payload delivered to a service during normal operation of a system. Each payload is analyzed to produce a *byte frequency distribution* of those payloads, which serves as a model for normal payloads. After this *centroid* model is computed during the learning phase, an anomaly detection phase begins. The anomaly detector captures incoming payloads and tests the payload for its consistency (or distance) from the centroid model. This is accomplished by comparing two statistical distributions. The distance metric used is the Mahalanobis distance metric, here applied to a finite discrete histogram of byte value (or character) frequencies computed in the training phase. Any new test payload found to be too distant from the normal expected payload is deemed anomalous and an alert is generated. The alert may then be *correlated* with other sensor data and a decision process may respond with several possible actions. Depending upon the security policy of the protected site, one may filter, reroute or otherwise trap the network connection from being allowed to send the poison payload to the service/application avoiding a worm infestation.

There are numerous engineering choices possible to implement the technique in a system and to integrate the detector with standard firewall technology to prevent the first occurrence of a worm from entering a secured network system. We do not address the correlation function and the mitigation strategies in this paper; rather we focus on the method of detection for anomalous payload.

This approach can be applied to any network system, service or port for that site to compute its own "site-specific" payload anomaly detector, rather than being dependent upon others deploying a specific signature for a newly detected worm or exploit that has already damaged other sites. As an added benefit of the approach described in this paper, the method may also be used to detect encrypted channels which may indicate an unofficial secure tunnel is operating against policy.

The rest of the paper is organized as follows. Section 2 discusses related work in network intrusion detection. In Section 3 we describe the model and the anomaly detection technique. Section 4 presents the results and evaluations of the method applied to different sets of data and it's run time performance. One of the datasets is publicly available for other researchers to verify our results. Section 5 concludes the paper.

2 Related Work

There are two types of systems that are called anomaly detectors: those based upon a specification (or a set of rules) of what is regarded as "good/normal" behavior, and others that learn the behavior of a system under normal operation. The first type relies upon human expertise and may be regarded as a straightforward extension of typical misuse detection IDS systems. In this paper we regard the latter type, where the behavior of a system is automatically learned, as a true anomaly detection system.

Rule-based network intrusion detection systems such as Snort and Bro use hand-crafted rules to identify known attacks, for example, virus signatures in the application payload, and requests to nonexistent services or hosts. Anomaly detection systems such as SPADE [5], NIDES [6], PHAD [13], ALAD [12] compute (statistical) models for normal network traffic and generate alarms when there is a large deviation from the normal model. These systems differ in the features extracted from available audit data and the particular algorithms they use to compute the normal models. Most use features extracted from the packet headers. SPADE, ALAD and NIDES model the distribution of the source and destination IP and port addresses and the TCP connection state. PHAD uses many more attributes, a total of 34, which are extracted from the packet header fields of Ethernet, IP, TCP, UDP and ICMP packets.

Some systems use some payload features but in a very limited way. NATE is similar to PHAD; it treats each of the first 48 bytes as a statistical feature starting from the IP header, which means it can include at most the first 8 bytes of the payload of each network packet. ALAD models the incoming TCP request and includes as a feature the first word or token of each input line out of the first 1000 application payloads, restricted only to the header part for some protocols like HTTP and SMTP.

The work of Kruegel et al [8] describes a service-specific intrusion detection system that is most similar to our work. They combine the type, length and payload distribution of the request as features in a statistical model to compute an anomaly score of a service request. However, they treat the payload in a very coarse way. They first sorted the 256 ASCII characters by frequency and aggregate them into 6 groups: 0, 1-3, 4-6, 7-11, 12-15, and 16-255, and compute one single uniform distribution model of these 6 segments for all requests to one service over all possible length payloads. They use a chi-square test against this model to calculate the anomaly score of new requests. In contrast, we model the full byte distribution conditioned on the length of payloads and use Mahalanobis distance as fully described in the following discussion. Furthermore, the modeling we introduce includes automatic clustering of centroids that is shown to increase accuracy and dramatically reduce resource consumption. The method is fully general and does not require any parsing, discretization, aggregation or tokenizing of the input stream (eg, [14]).

Network intrusion detection systems can also be classified according to the semantic level of the data that is analyzed and modeled. Some of the systems reconstruct the network packets and extract features that describe the higher level interactions between end hosts like MADAMID [9], Bro [15], EMERALD [18], STAT [24], ALAD [13], etc. For example, session duration time, service type, bytes transferred, and so forth are regarded as higher level, temporally ordered features not discernible by inspecting only the packet content. Other systems are purely packet-based like PHAD [14], NATED [12], NATE [23]. They detect anomalies in network packets directly without reconstruction. This approach has the important advantage of being simple and fast to compute, and they are generally quite good at detecting those attacks that do not result in valid connections or sessions, for example, scanning and probing attacks.

3 Payload Modeling and Anomaly Detection

There are many design choices in modeling payload in network flows. The primary design criteria and operating objectives of any anomaly detection system entails:

- automatic "hands-free" deployment requiring little or no human intervention,
- generality for broad application to any service or system,
- incremental update to accommodate changing or drifting environments,
- accuracy in detecting truly anomalous events, here anomalous payload, with low (or controllable) false positive rates,
- resistance to mimicry attack and
- efficiency to operate in high bandwidth environments with little or no impact on throughput or latency.

These are difficult objectives to meet concurrently, yet they do suggest an approach that may balance these competing criteria for payload anomaly detection.

We chose to consider "language-independent" statistical modeling of sampled data streams best exemplified by well known n-gram analysis. Many have explored the use of n-grams in a variety of tasks. The method is well understood, efficient and effective. The simplest model one can compose is the 1-gram model. A 1-gram model is certainly efficient (requiring a linear time scan of the data stream and an update of a small 256-element histogram) but whether it is accurate requires analysis and experimentation. To our surprise, this technique has worked surprisingly well in our experiments as we shall describe in Section 4. Furthermore, the method is indeed resistant to mimicry attack. Mimicry attacks are possible if the attacker has access to the same information as the victim to replicate normal behavior. In the case of application payload, attackers (including worms) would not know the distribution of the normal flow to their intended victim. The attacker would need to sniff for a long period of time and analyze the traffic in the same fashion as the detector described herein, and would also then need to figure out how to pad their poison payload to mimic the normal model.

3.1 Length Conditioned n-gram Payload Model

Network payload is just a stream of bytes. Unlike the network packet headers, payload doesn't have a fixed format, small set of keywords or expected tokens, or a limited range of values. Any character or byte value may appear at any position of the datagram stream. To model the payload, we need to divide the stream into smaller clusters or groups according to some criteria to associate similar streams for modeling. The port number and the length are two obvious choices. We may also condition the models on the direction of the stream, thus producing separate models for the inbound traffic and outbound responses.

Usually the standard network services have a fixed pre-assigned port number: 20 for FTP data transmission, 21 for FTP commands, 22 for SSH, 23 for Telnet, 25 for SMTP, 80 for Web, etc. Each such application has its own special protocol and thus has its own payload type. Each site running these services would have its own "typical payload" flowing over these services. Payload to port 22 should be encrypted and appear as uniform distribution of byte values, while the payload to port 21 should be primarily printable characters entered by a user and a keyboard.

Within one port, the payload length also varies over a large range. The most common TCP packets have payload lengths from 0 to 1460. Different length ranges have different types of payload. The larger payloads are more likely to have non-printable characters indicative of media formats and binary representations (pictures, video clips or executable files etc.). Thus, we compute a payload model for each different length range for each port and service and for each direction of payload flow. This produces a far more accurate characterization of the normal payload than would otherwise be possible by computing a single model for all traffic going to the host. However, many centroids might be computed for each possible length payload creating a detector with a large resource consumption..

To keep our model simple and quick to compute, we model the payload using n-gram analysis, and in particular the byte value distribution, exactly when n=1. An n-gram is the sequence of n adjacent bytes in a payload unit. A sliding window with width n is passed over the whole payload and the occurrence of each n-gram is counted. N-gram analysis was first introduced by [2] and exploited in many language analysis tasks, as well as security tasks. The seminal work of Forrest [3] on system call traces uses a form of n-gram analysis (without the frequency distribution and allowing for "wildcards" in the gram) to detect malware execution as uncharacteristic sequences of system calls.

For a payload, the feature vector is the relative frequency count of each n-gram which is calculated by dividing the number of occurrences of each n-gram by the total number of n-grams. The simplest case of a 1-gram computes the average frequency of each ASCII character 0-255. Some stable character frequencies and some very variant character frequencies can result in the same average frequency, but they should be characterized very differently in the model. Thus, we compute in addition to the mean value, the variance and standard deviation of each frequency as another characterizing feature.. So for the payload of a fixed length of some port, we treat each character's relative frequency as a variable and compute its mean and standard deviation as the payload model.

Figure 1 provides an example showing how the payload byte distributions vary from port to port, and from source and destination flows. Each plot represents the characteristic profile for that port and flow direction (inbound/outbound). Notice also that the distributions for ports 22 (inbound and outbound) show no discernible pattern, and hence the statistical distribution for such encrypted channels would entail a more uniform frequency distribution across all of the 256 byte values, each with low variance. Hence, encrypted channels are fairly easy to spot. Notice that this figure is actually generated from a dataset with only the first 96 bytes of payload in each packet, and there is already a very clear pattern with the truncated payload. Figure 2 displays the variability of the frequency distributions among different length payloads. The two plots characterize two different distributions from the incoming traffic to the same web server, port 80 for two different lengths, here payloads of 200 bytes, the other 1,460 bytes. Clearly, a single monolithic model for both length categories will not represent the distributions accurately.

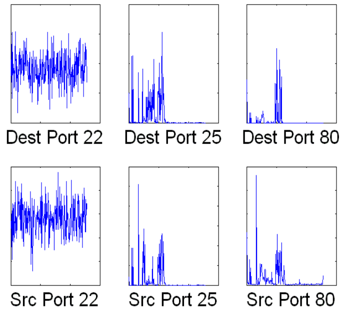

Fig. 1. Example byte distributions for different ports. For each plot, the X-axis is the ASCII byte 0-255, and the Y-axis is the average byte frequency

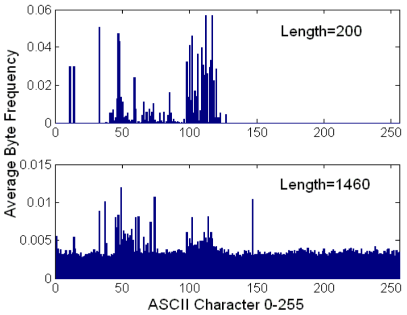

Fig. 2. Example byte distribution for different payload lengths for port 80 on the same host server

Given a training data set, we compute a set of models M_{ij}. For each specific observed length i of each port j, M_{ij} stores the average byte frequency and the standard deviation of each byte's frequency. The combination of the mean and variance of each byte's frequency can characterize the payload within some range of payload lengths. So if there are 5 ports, and each port's payload has 10 different lengths, there will be in total 50 centroid models computed after training. As an example, we show the model computed for the payload of length 185 for port 80 in figure 3, which is derived from a dataset described in Section 4. (We also provide an automated means of reducing the number of centroids via clustering as described in section 3.4.)

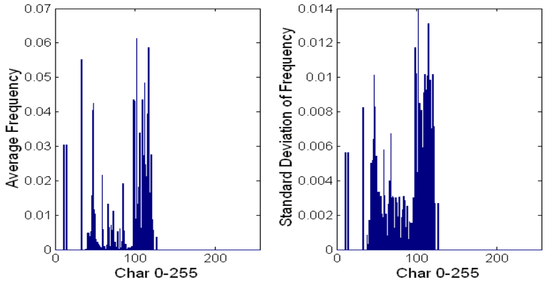

Fig. 3. The average relative frequency of each byte, and the standard deviation of the frequency of each byte, for payload length 185 of port 80

PAYL operates as follows. We first observe many exemplar payloads during a training phase and compute the mean and variance of the byte value distribution producing model M_{ij}. During detection, each incoming payload is scanned and its byte value distribution is computed. This new payload distribution is then compared against model M_{ij}; if the distribution of the new payload is significantly different from the norm, the detector flags the packet as anomalous and generates an alert.

The means to compare the two distributions, the model and the new payload, is described next.

3.2 Simplified Mahalanobis Distance

Mahalanobis distance is a standard distance metric to compare two statistical distributions. It is a very useful way to measure the similarity between the (unknown) new payload sample and the previously computed model. Here we compute the distance between the byte distributions of the newly observed payload against the profile from the model computed for the corresponding length range. The higher the distance score, the more likely this payload is abnormal.

The formula for the Mahalanobis distance is:

$$d^2(x, \overline{y}) = (x - \overline{y})^T C^{-1} (x - \overline{y})$$

where x and $\overline{y}$ are two feature vectors, and each element of the vector is a variable. x is the feature vector of the new observation, and $\overline{y}$ is the averaged feature vector computed from the training examples, each of which is a vector. And C^{-1} is the inverse covariance matrix as $C_{ij} = Cov(y_i, y_j)$. y_i, y_j are the *i*th and *j*th elements of the training vector.

The advantage of Mahalanobis distance is that it takes into account not only the average value but also its variance and the covariance of the variables measured. Instead of simply computing the distance from the mean values, it weights each variable by its standard deviation and covariance, so the computed value gives a statistical measure of how well the new example matches (or is consistent with) the training samples.

In our problem, we use the "naïve" assumption that the bytes are statistically independent. Thus, the covariance matrix C becomes diagonal and the elements along the diagonal are just the variance of each byte.

Notice, when computing the Mahalanobis distance, we pay the price of having to compute multiplications and square roots after summing the differences across the byte value frequencies. To further speed up the computation, we derive the *simplified Mahalanobis distance*:

$$d(x, \overline{y}) = \sum_{i=0}^{n-1} (|\, x_i - \overline{y_i} \,| / \overline{\sigma_i})$$

where the variance is replaced by the *standard deviation*. Here *n* is fixed to 256 under the 1-gram model (since there are only 256 possible byte values). Thus, we avoid the time-consuming square and square-root computations (in favor of a single division operation) and now the whole computation time is linear in the length of the payload with a small constant to compute the measure. This produces an exceptionally fast detector (recall our objective to operate in high-bandwidth environments).

For the simplified Mahalanobis distance, there is the possibility that the standard deviation $\overline{\sigma_i}$ equals zero and the distance will become infinite. This will happen when a character or byte value never appears in the training samples or, oddly enough, it appears with exactly the same frequency in each sample. To avoid this situation, we give a smoothing factor α to the standard deviation similar to the prior observation:

$$d(x, \overline{y}) = \sum_{i=0}^{n-1} (|\, x_i - \overline{y_i} \,| / (\overline{\sigma_i} + \alpha))$$

The *smoothing factor* α reflects the statistical confidence of the sampled training data. The larger the value of α, the less the confidence the samples are truly representative of the actual distribution, and thus the byte distribution can be more variable. Over time, as more samples are observed in training, α may be decremented automatically.

The formula for the simplified Mahalanobis distance also suggests how to set the threshold to detect anomalies. If we set the threshold to 256, this means we allow each character to have a fluctuation range of one standard deviation from its mean. Thus, logically we may adjust the threshold to a value in increments of 128 or 256, which may be implemented as an automatic self-calibration process.

3.3 Incremental Learning

The 1-gram model with Mahalanobis distance is very easy to implement as an incremental version with only slightly more information stored in each model. An incremental version of this method is particularly useful for several reasons. A model may be computed on the fly in a "hands-free" automatic fashion. That model will improve in accuracy as time moves forward and more data is sampled. Furthermore, an incremental online version may also "age out" old data from the model keeping a more accurate view of the most recent payloads flowing to or from a service. This "drift in environment" can be solved via incremental or online learning [25].

To age out older examples used in training the model, we can specify a decay parameter of the older model and emphasize the frequency distributions appearing in the new samples. This provides the means of automatically updating the model to maintain an accurate view of normal payloads seen most recently.

To compute the incremental version of the Mahalanobis distance, we need to compute the mean and the standard deviation of each ASCII character seen for each new sample observed. For the mean frequency of a character, we compute $\overline{x} = \sum_{i=1}^{N} x_i \Big/ N$ from the training examples. If we also store the number of samples processed, *N*, we can update the mean as $\overline{x} = \frac{\overline{x} \times N + x_{N+1}}{N+1} = \overline{x} + \frac{x_{N+1} - \overline{x}}{N+1}$ when we see a new example x_{N+1}, a clever update technique described by Knuth [7].

Since the standard deviation is the square root of the variance, the variance computation can be rewritten using the expected value *E* as:

$$Var(X) = E(X - EX)^2 = E(X^2) - (EX)^2$$

We can update the standard deviation in a similar way if we also store the average of the x_i^2 in the model.

This requires maintaining only one more 256-element array in each model that stores the average of the x_i^2 and the total number of observations *N*. Thus, the n-gram byte distribution model can be implemented as an incremental learning system easily and very efficiently. Maintaining this extra information can also be used in clustering samples as described in the next section.

3.4 Reduced Model Size by Clustering

When we described our model, we said we compute one model M_{ij} for each observed length bin *i* of payloads sent to port *j*. Such fine-grained modeling might introduce several problems. First, the total size of the model can become very large. (The payload lengths are associated with media files that may be measured in gigabytes and many length bins may be defined causing a large number of centroids to be computed.) Further, the byte distribution for payloads of length bin *i* can be very similar to that of payloads of length bins *i-1* and *i+1;* after all they vary by one byte. Storing a model for each length may therefore be obviously redundant and wasteful.

Another problem is that for some length bins, there may not be enough training samples. Sparseness implies the data will generate an empirical distribution that will be an inaccurate estimate of the true distribution leading to a faulty detector.

There are two possible solutions to these problems. One solution for the sparseness problem is relaxing the models by assigning a higher smoothing factor to the standard deviations which allows higher variability of the payloads. The other solution is to "borrow" data from neighboring bins to increase the number of samples; i.e. we use data from neighboring bins used to compute other "similar" models.

We compare two neighboring models using the simple Manhattan distance to measure the similarity of their average byte frequency distributions. If their distance is smaller than some threshold *t*, we merge those two models. This clustering technique is repeated it until no more neighboring models can be merged. This merging is easily computed using the incremental algorithm described in Section 3.3; we update the means and variances of the two models to produce a new updated distribution.

Now for a new observed test data with length *i* sent to port *j*, we use the model M_{ij}, or the model it was merged with. But there is still the possibility that the length of the test data is outside the range of all the computed models. For such test data, we use the model whose length range is nearest to that of the test data. In these cases, the mere fact that the payload has such an unusual length unobserved during training may itself be cause to generate an alert.

The reader should note that the modeling algorithm and the model merging process are each linear time computations, and hence the modeling technique is very fast and can be performed in real time. The online learning algorithm also assures us that models will improve over time, and their accuracy will be maintained even when services are changed and new payloads are observed.

3.5 Unsupervised Learning

Our model together with Mahalanobis distance can also be applied as an unsupervised learning algorithm. Thus, training the models is possible even if noise is present in the training data (for example, if training samples include payloads from past worm propagations still propagating on the internet.) This is based on the assumption that the anomalous payload is a minority of the training data and their payload distribution is different from the normal payload. These abnormal payloads can be identified in the training set and their distributions removed from the model. This is accomplished by applying the learned models to the training dataset to detect outliers. Those anomalous payloads will have a much larger distance to the profile than the "average" normal samples and thus will likely appear as statistical outliers. After identifying these anomalous training samples, we can either remove the outliers and retrain the models, or update the frequency distributions of the computed models by removing the counts of the byte frequencies appearing in the anomalous training data. We demonstrate the effectiveness of these techniques in the evaluation section.

3.6 Z-String

Consider the string of bytes corresponding to the sorted, rank ordered byte frequency of a model. Figure 4 displays a view of this process. The frequency distribution of payloads of length 185 is plotted in the top graph. The lower graph represents the same information by the plot is reordered to the rank ordering of the distribution. Here, the first bar in the lower plot is the frequency of the most frequently appearing ASCII character. The second bar is likewise the second most frequent, and so on. This rank ordered distribution surprisingly follows a Zipf-like distribution (an exponential function or a power law where there are few values appearing many times, and a large number of values appearing very infrequently.)

The rank order distribution also defines what we call a “Z-string”. The byte values ordered from most frequent to least frequent serves as a representative of the entire distribution. Figure 5 displays the Z-String for the plot in Figure 4. Notice that for this distribution there are only 83 distinct byte values appearing in the distribution. Thus, the Z-string has length 83.

Furthermore, as we shall see later, this rank ordered byte value distribution of the new payload deemed anomalous also may serve as a simple representation of a “new worm signature” that may be rapidly deployed to other sites to better detect the appearance of a new worm at those sites; if an anomalous payload appears at those sites and its rank ordered byte distribution matches a Z-string provided from another site, the evidence is very good that a worm has appeared. This distribution mechanism is part of an ongoing project called “Worminator” [11, 22] that implements a “collaborative security” system on the internet. A full treatment of this work is beyond the scope of this paper, but the interested reader is encouraged to visit http://worminator.cs.columbia.edu/ for details.

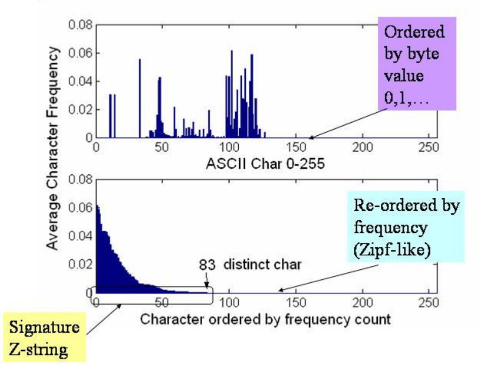

Fig. 4. Payload distribution appears in the top plot, re-ordered to the rank-ordered count frequency distribution in the bottom plot. Notice there are only 83 distinct characters used in the average payload for this service (port 80, http) for this length distribution of payloads (all payloads with length 185 bytes)

eto.c/a $\alpha\beta$ lsrw:imnTupgbhH|-
0AdxEPUCG3*vF@_fyR,~24RzMk9=();SDWIjL6B7
Z8%?Vq[]ONK+JX&

α : LF – Line feed β : CR – Carriage return

Fig. 5. The signature “Z-string” for the average payload displayed in Figure 4. “e” is the most frequent byte value, followed by “t” and so on. Notice how balanced characters appear adjacent to each other, for example “()” and “[]” since these tend to appear with equal frequency

4 Evaluation of the 1-gram models

We conducted two sets of experiments to test the effectiveness of the 1-gram models. The first experiment was applied to the 1999 DARPA IDS Data Set which is the most complete dataset with full payload publicly available for experimental use. The experiment here can be repeated by anyone using this data set to verify the results we report. The second experiment used the CUCS dataset which is the inbound network traffic to the web server of the computer science department of Columbia University. Unfortunately, this dataset cannot be shared with other researchers due to the privacy policies of the university. (In fact, the dataset has been erased to avoid a breach of anyone’s privacy.)

4.1 Experiments with 1999 DARPA IDS Data Set

The 1999 DARPA IDS data set was collected at MIT Lincoln Labs to evaluate intrusion detection systems. All the network traffic including the entire payload of each packet was recorded in tcpdump format and provided for evaluation. In addition, there are also audit logs, daily file system dumps, and BSM (Solaris system call) logs. The data consists of three weeks of training data and two weeks of test data. In the training data there are two weeks of attack-free data and one week of data with labeled attacks.

This dataset has been used in many research efforts and results of tests against this data have been reported in many publications. Although there are problems due to the nature of the simulation environment that created the data, it still remains a useful set of data to compare techniques. The top results were reported by [10].

In our experiment on payload anomaly detection we only used the inside network traffic data which was captured between the router and the victims. Because most public applications on the Internet use TCP (web, email, telnet, and ftp), and to reduce the complexity of the experiment, we only examined the inbound TCP traffic to the ports 0-1023 of the hosts 172.016.xxx.xxx which contains most of the victims, and ports 0-1023 which covers the majority of the network services. For the DARPA 99 data, we conducted experiments using each packet as the data unit and each connection as the data unit. We used tcptrace to reconstruct the TCP connections from the network packets in the tcpdump files. We also experimented the idea of “truncated payload", both for each packet and each connection. For truncated packets, we tried the first N bytes and the tail N bytes separately, where N is a parameter. Using truncated payload saves considerable computation time and space. We report the results for each of these models.

We trained the payload distribution model on the DARPA dataset using week 1 (5 days, attack free) and week 3 (7 days, attack free), then evaluate the detector on weeks 4 and 5, which contain 201 instances of 58 different attacks, 177 of which are visible in the inside tcpdump data. Because we restrict the victims' IP and port range, there are 14 others we ignore in this test.

In this experiment, we focus on TCP traffic only, so the attacks using UDP, ICMP, ARP (address resolution protocol) and IP only cannot be detected. They include: smurf (ICMP echo-reply flood), ping-of-death (over-sized ping packets), UDPstorm, arppoison (corrupts ARP cache entries of the victim), selfping, ipsweep, teardrop (mis-fragmented UDP packets). Also because our payload model is computed from only the payload part of the network packet, those attacks that do not contain any payload are impossible to detect with the proposed anomaly detector. Thus, there are in total 97 attacks to be detected by our payload model in weeks 4 and 5 evaluation data.

After filtering there are in total 2,444,591 packets, and 49556 connections, with non-zero length payloads to evaluate. We build a model for each payload length observed in the training data for each port between 0-1023 and for every host machine. The smoothing factor is set to 0.001 which gives the best result for this dataset (see the discussion in Section 3.2). This helps avoid over-fitting and reduces the false positive rate. Also due to having an inadequate number of training examples in the DARPA99 data, we apply clustering to the models as described previously. Clustering the models of neighboring length bins means that similar models can provide more training data for a model whose training data is too sparse thus making it less sensitive and more accurate. But there is also the risk that the detection rate will be lower when the model allows more variance in the frequency distributions. Based on the models for each payload length, we did clustering with a threshold of 0.5, which means if the two neighboring model's byte frequency distribution has less than 0.5 Manhattan distance we merge their models. We experimented with both unclustered and clustered models. The results indicate that the clustered model is always better than the unclustered model. So in this paper, we will only show the results of the clustered models.

Different port traffic has different byte variability. For example, the payload to port 80 (HTTP requests) are usually less variable than that of port 25 (email). Hence, we set different thresholds for each port and check the detector's performance for each port. The attacks used in the evaluation may target one or more ports. Hence, we calibrate a distinct threshold for each port and generate the ROC curves including all appropriate attacks as ground truth. The packets with distance scores higher than the threshold are detected as anomalies.

Figure 6 shows the ROC curves for the four most commonly attacked ports: 21, 23, 25, and 80. For the other ports, eg. 53, 143, 513 etc., the DARPA99 data doesn't provide a large enough training and testing sample, so the results for those ports are not very meaningful.

For each port, we used five different data units, for both training and testing. The legend in the plots and their meaning are:

1) Per Packet Model, which uses the whole payload of each network packet;
2) First 100 Packet Model, which uses the first 100 bytes of each network packet;
3) Tail 100 Packet Model, which uses the last 100 bytes of each network packet;
4) Per Conn Model, which uses the whole payload of each connection;
5) Truncated Conn Model, which uses the first 1000 bytes of each connection.

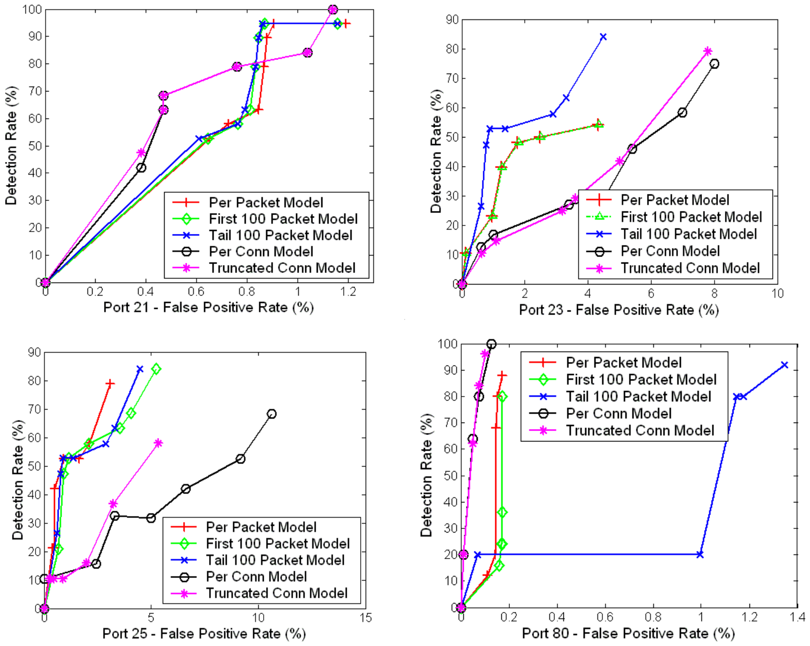

Fig. 6. ROC curves for ports 21, 23, 25, 80 for the five different models. Notice the x-axis scale is different for each plot and does not span to 100%, but limited to the worst false positive rate for each plot

From Figure 6 we can see that the payload-based model is very good at detecting the attacks to port 21 and port 80. For port 21, the attackers often first upload some malicious code onto the victim machine and then login to crash the machine or get root access, like casesen and sechole. The test data also includes attacks that upload/download illegal copies of software, like warezmaster and warezclient. These attacks were detected easily because of their content which were rarely seen executable code and quite different from the common files going through FTP. For port 80, the attacks are often malformed HTTP requests and are very different from normal requests. For instance, crashiis sends request "GET ../.."; apache2 sends request with a lot of repeated "User-Agent:sioux\r\n", etc. Using payload to detect these attacks is a more reliable means than detecting anomalous headers simply because their packet headers are all normal to establish a good connection to deliver their poison payload. Connection based detection has a better result than the packet based models for port 21 and 80. It's also important to notice that the truncated payload models achieve results nearly as good as the full payload models, but are much more efficient in time and space.

For port 23 and port 25 the result is not as good as the models for port 21 and 80. That's because their content are quite free style and some of the attacks are well hidden. For example, the framespoofer attack is a fake email from the attacker that misdirects the victim to a malicious web site. The website URL looks entirely normal. Malformed email and telnet sessions are successfully detected, like the perl attack which runs some bad perl commands in telnet, and the sendmail attack which is a carefully crafted email message with an inappropriately large MIME header that exploits a buffer overflow error in some versions of the sendmail program. For these two ports, the packet-based models are better than the connection-based models. This is likely due to the fact that the actual exploit is buried within the larger context of the entire connection data, and its particular anomalous character distribution is swamped by the statistics of the other data portions of the connection. The per packet model detects this anomalous payload more easily.

There are many attacks that involve multiple steps aimed at multiple ports. If we can detect one of the steps at any one port, then the attack can be detected successfully. Thus we correlate the detector alerts from all the ports and plot the overall performance. When we restrict the false positive rate of each port (during calibration of the threshold) to be lower than 1%, we achieve about a 60% detection rate, which is pretty high for the DARPA99 dataset. The results for each model are displayed in the Table 1:

Table 1. Overall detection rate of each model when false positive rate lower than 1%

Per Packet Model	57/97 (58.8%)
First 100 Packet Model	55/97 (56.7%)
Tail 100 Packet Model	46/97 (47.4%)
Per Conn Model	55/97 (56.7%)
Truncated Conn Model	51/97 (52.6%)

Modeling the payload to detect anomalies is useful to protect servers against new attacks. Furthermore, careful inspection of the detected attacks in the tables and from other sources reveals that correlating this payload detector with other detectors increases the coverage of the attack space. There is large non-overlap between the attacks detected via payload and other systems that have reported results for this same dataset, for example PHAD [13]. This is obvious because the data sources and modeling used are totally different. PHAD models packet header data, whereas payload content is modeled here.

Our payload-based model has small memory consumption and is very efficient to compute. Table 2 displays the measurements of the speed and the resulting number of centroids for each of the models for both cases of unclustered and clustered. The results were derived from measuring PAYL on a 3GHz P4 Linux machine with 2G memory using non-optimized Java code. These results do not indicate how well a professionally engineered system may behave (re-engineering in C probably would gain a factor of 6 or more in speed). Rather, these results are provided to show the relative efficiency among the alternative modeling methods. The training and test time reported in the table is seconds per 100Mof data, which includes the I/O time. The number of centroids computed after training represents an approximation of the total amount of memory consumed by each model. Notice that each centroid has fixed size: two 256-element double arrays, one for storing averages and the other for storing the standard deviation of the 256 ASCII bytes. A re-engineered version of PAYL would not consume as much space as does a Java byte stream object. From the table we can see that clustering reduces the number of centroids, and total consumed memory by about a factor from 2 to 16 with little or no hit in computational performance. Combining Figure 6, Table 1 and Table 2, users can choose the proper model for their application according to their environment and performance requirements.

Table 2. Speed and Memory measurements of each model. The training and testing time is in units of seconds per 100M data, including the I/O time. The memory comsuption is measured in the number of centroids that were kept after clustering or learning.

Unclustered /Clustered	Per Packet	First 100	Tail 100	Per Conn.	Trunc Conn.
Train time(uncl)	26.1	21.8	21.8	8.6	4.4
Test time(uncl)	16.1	9.4	9.4	9.6	1.6
No. centroid(uncl)	11583	11583	11583	16326	16326
Train tme(clust)	26.2	22.0	26.2	8.8	4.6
Test time(clust)	16.1	9.4	9.4	9.6	1.6
No. centroid(clust)	4065	7218	6126	2219	1065

This result is surprisingly good for such a simple modeling technique. Most importantly, this anomaly detector can easily augment existing detection systems. It is not intended as a stand alone detection system but a component in a larger system aiming for defense in depth. Hence, the detector would provide additional and useful alert information to correlate with other detectors that in combination may generate an alarm and initiate a mitigation process. The DARPA 99 dataset was used here so that others can verify our results. However, we also performed experiments on a live stream that we describe next.

4.2 Experiments with CUCS Dataset

The CUCS dataset denotes Columbia University CS web server dataset, which are two traces of incoming traffic with full payload to the CS department web server (www.cs.columbia.edu). The two traces were collected separately, one in August 2003 for 45 hours with a size of about 2GB, and one in September 2003 for 24 hours with size 1GB. We denote the first one as A, the second one as B, and their union as AB. Because we did not know whether this dataset is attack-free or not, this experiment represents an unlabeled dataset that provides the means of testing and evaluating the unsupervised training of the models.

First we display the result of unsupervised learning in Table 3. We used an unclustered single-length model since the number of training examples is sufficient to adequately model normal traffic. Also the smoothing factor is set to 0.001 and 256 as the anomaly threshold. Dataset A has 331,236 non-zero payload packets, and B has 199,881. The third column shows the number and percentage of packets that are deemed anomalous packets. Surprisingly, when we manually checked the anomalous packets we found Code Red II attacks and extremely long query string buffer overflow attacks in dataset B. (“yes” means the attack is successfully detected.)

Table 3. Unsupervised learning result on CUCS dataset

Train	Test	Anomaly #	CR-II	Buffer
A	A	28(0.0084%)	----	----
A	B	2601(1.3%)	Yes	Yes
B	A	686(0.21%)	----	----
B	B	184(0.092%)	Yes	Yes
AB	AB	211(0.039%)	Yes	Yes

There is a high anomaly rate when we train on A and test on B; this is because there are many pdf file-uploads in B that did not occur in A. (Notice the dates. A was captured during the summer; B was captured later during student application time.) Because pdf files are encoded with many nonprintable characters, these packets are very different from other normal HTTP request packets. For the rest of those detected packets, more than 95% are truly anomalous. They include malformed HTTP headers like “~~~~~~:~~~~~~~~~”, a string with all capital letter’s, “Weferer” replacing the standard Referer tag (apparently a privacy feature of a COTS product), extremely long and weird parameters for “range”, javascripts embedded html files sent to the CS server, etc. These requests might do no harm to the server, but they are truly unusual and should be filtered or redirected to avoid a possible attack. They do provide important information as well to other detectors that may deem their connections anomalous for other reasons.

Figure 7 displays a plot of the distance values of the normal packets against the attacks. For illustrative purposes, we selected some packets of the Code Red II attack and the buffer overflow attack, which has length 1460 and were detected to be anomalous, and compare these with the distances of the normal packets of the same length. The training and test data both use data set A for these plots.

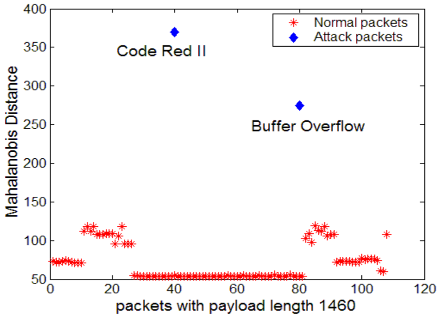

Fig. 7. The computed Mahalanobis distance of the normal and attack packets

We also tested some other packets collected from online sources of virus's as they appeared in the wild and within the DARPA 99 data set. These were tested against the CUCS dataset. They include Code Red I & II, Nimbda, crashiis, back, apache2 etc. All of these tcpdump packets containing virus's were successfully caught by our model.

For illustrative purposes, we also display in Table 4 the Z-strings of the Code Red II and, the buffer overflow attacks and the centroid Z-string to demonstrate how different each appears from the norm. Notice these are the Z-strings for one of the single malicious packets we captured at packet payload length 1460. Because the full Z-string is too long (more than 200 characters) and contains many nonprintable characters, we only display the first 20 characters' ASCII value in decimal for illustration. The buffer overflow packet only has 4 different characters, so its Z-string has length 4 and are all displayed in the table.

Table 4. Illustrations of the partial Z-strings. The characters are shown in ASCII

Code Red II (first 20 characters)									
88	0	255	117	48	85	116	37	232	100
100	106	69	133	137	80	254	1	56	51
Buffer Overflow (all)									
65	37	48	68						
Centroid (first 20 characters)									
48	73	146	36	32	46	61	113	44	110
59	70	45	56	50	97	110	115	51	53

5 Conclusion

The experimental results indicate that the method is effective at detecting attacks. In the 1999 DARPA IDS dataset, the best trained model for TCP traffic detected 57 attacks out of 97 with every port's false positive rate lower than 1%. For port 80, it achieves almost 100% detection rate with around 0.1% false positive rate. It also successfully detected the Code Red II and a buffer overflow attack from the unlabeled CUCS dataset. The payload model is very simple, state-free, and quick to compute in time that is linear in the payload length. It also has the advantage of being implemented as an incremental, unsupervised learning method. The payload anomaly detector is intended to be correlated with other detectors to mitigate against false alarms, and to increase the coverage of attacks that may be detected. The experiment also demonstrated that clustering of centroids from neighboring length bins dramatically reduce memory consumption up to a factor of 16.

The Z-string derived from the byte distributions can be used as a "signature" to characterize payloads. Each such string is at most 256 characters, and can be readily stored and communicated rapidly among sites in a real-time distributed detection system as "confirmatory" evidence of a zero-day attack. This can be accomplished faster than is otherwise possible by observing large bursts in probing activity among a large segment of the internet. This approach may also have great value in detecting slow and stealthy worm propagations that may avoid activities of a bursty nature!

In our future work, we plan to evaluate the technique in live environments, implement and measure the costs and speed of the Z-string distribution mechanism and most interestingly whether higher order n-grams provide added value or not in modeling payload. Furthermore, we plan to evaluate the opportunity or difficulty for mimicry attack by comparing the payload distributions across different sites. If, as we suspect, each site's payload distributions are consistently different (in a statistical sense), then the anomaly detection approach proposed here, based upon site-specific payload models, will provide protection for all sites.

Acknowledgments

We'd like to thank Nick Edwards, Phil Gross, Janak J. Parekh, Shlomo Hershkop, Morris Pearl, Wei-Jen Li for help on collecting data and the experimental set up, and for useful discussions and helpful comments on this paper.

References

1. D. Armstrong, S. Carter, G. Frazier, T. Frazier. A Controller-Based Autonomic Defense System. Proc. of DISCEX, 2003.
2. M. Damashek. Gauging similarity with n-grams: language independent categorization of text. *Science*, 267(5199):843--848, 1995.
3. S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, A Sense of self for Unix Processes. *Proc. of IEEE Symposium on Computer Security and Privacy*, 1996.
4. A. K. Ghosh, A. Schwartzbard, A study in Using Neural Networks for Anomaly and Misuse Detection, *Proc. 8th USENIX Security Symposium* 1999.
5. J. Hoagland, SPADE, Silican Defense, http://www.silicondefense.com/software/spice, 2000.
6. H. S. Javits and A. Valdes. The NIDES statistical component: Description and justification. *Technical report, SRI International, Computer Science Laboratory*, 1993.

7. D. E. Knuth, the Art of Computer Programming, Vol 1 Fundamental Algorithms. *Addison Wesley*, 2nd edition, 1973.
8. C. Kruegel, T. Toth and E. Kirda, Service Specific Anomaly Detection for Network Intrusion Detection. In *Symposium on Applied Computing (SAC)*, Spain, March 2002.
9. W. Lee and S. Stolfo, A Framework for Constructing Features and Models for Intrusion Detection Systems. *ACM Transactions on Information and System Security*, 3(4), November 2000.
10. R. Lippmann, et al. The 1999 DARPA Off-Line Intrusion Detection Evaluation, *Computer Networks* 34(4) 579-595, 2000.
11. M. Locasto, J. Parekh, S. Stolfo, A. Keromytis, T. Malkin and V. Misra, Collaborative Distributed Intrusion Detection, *Columbia University Tech Report, CUCS-012-04*, 2004.
12. M. Mahoney. Network Traffic Anomaly Detection Based on Packet Bytes. *Proc. ACM-SAC* 2003.
13. M. Mahoney, P. K. Chan, Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks, *Proc. SIGKDD* 2002, 376-385.
14. M. Mahoney, P. K. Chan, Learning Models of Network Traffic for Detecting Novel Attacks, *Florida Tech, Technical report 2002-08*, http://cs.fit.edu/~tr.
15. M. Mahoney, P. K. Chan: An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. *RAID 2003*: 220-237.
16. D. Moore, C. Shannon, G. Voelker and S. Savage, Internet Quarantine: Requirements for Containing Selp-Propagating Code, *Proc. Infocom* 2003.
17. V. Paxson, Bro: A system for detecting network intruders in real-time, *USENIX Security Symposium*, 1998.
18. P. Porras and P. Neumann, EMERALD: Event Monitoring Enabled Responses to Anomalous Live Disturbances, *National Information Systems Security Conference*, 1997.
19. S. Robertson, E. Siegel, M. Miller, and S. Stolfo, Surveillance Detection in High Bandwidth Environments, *In Proceedings of the 2003 DARPA DISCEX III Conference*, 2003.
20. M. Roesch, Snort: Lightweight intrusion detection for networks, *USENIX LISA Conference*, 1999.
21. S. Staniford, V. Paxson, and N. Weaver, How to Own the Internet in Your Spare Time, *Proceedings of the 11th USENIX Security Symposium,* 2002.
22. S. Stolfo, Worm and Attack Early Warning: Piercing Stealthy Reconnaissance, *IEEE Privacy and Security,* May/June, 2004 (to appear).
23. C. Taylor and J. Alves-Foss. NATE – Network Analysis of Anomalous Traffic Events, A Low-Cost approach, *New Security Paradigms Workshop*, 2001.
24. G. Vigna and R. Kemmerer, NetSTAT: A Network-based intrusion detection approach, *Computer Security Application Conference*, 1998.
25. T. Lane and C. E. Broadley, Approaches to online learning and concept drift for user identification in computer security. 4th International Conference on Knowledge Discovery and Data Mining, 1998.