Fuzzy Clustering for Intrusion Detection

Hiren Shah, Jeffrey Undercoffer and Anupam Joshi
University of Maryland Baltimore County
Department of Computer Science and Electrical Engineering
1000 Hilltop Circle, Baltimore, MD-21250
hiren1, undercoffer, joshi@umbc.edu

Abstract— **The newly formed Department of Homeland Security has been mandated to reduce America's vulnerability to terrorism. In addition to being charged with physical protection, this newly formed department is also responsible for protecting the nation's critical infrastructure. Protecting computer systems from intrusions is an important aspect of securing the nation's infrastructure. We are exploring how fuzzy data mining and concepts introduced by the semantic web can operate in synergy to perform *Distributed Intrusion Detection*. The underlying premise of our intrusion detection model is to describe attacks as instances of an ontology using a semantically rich language, reason over them and subsequently classify them as instances of an attack of a specific type. However, before an abnormality can be specified as an instance of the ontology, it first needs to be detected. Hence, our intrusion detection model is two phased, where the first phase uses data mining techniques to analyze low level data streams that capture process, system and network states and to detect anomalous behavior. The second phase reasons over instances of anomalous behavior specified according to our ontology. This paper focuses on the initial phase of our model: outlier detection within low level data streams. Accordingly, we present the preliminary results of the use of fuzzy clustering to detect anomalies within low level kernel data streams.**

I. Introduction

The Department of Homeland Security was formed after the tragic events of September 11. The mission of this department is to reduce America's vulnerability to terrorist attacks. In addition to being charged with physical protection, this newly formed department is also responsible for protecting the nation's critical infrastructure. According to Presidential Decision Directive 63 (PDD63), critical infrastructures are comprised of cyber-based systems to include those systems serving: telecommunications, energy, banking and finance, transportation, water systems and emergency services. There is little doubt that any attack that paralyzes the computer systems serving our critical infrastructures will cause enormous damage and can be as devastating in consequences as any physical attack. Moreover, the speed, virulence, and maliciousness of cyber attacks have increased dramatically in recent years. The recent *W32.Slammer* worm, which went as far as to block access to some ATM bank accounts in January 2003, is an ample proof of this fact. Reportedly, the Department of Homeland Security aims to place an especially high priority on protecting our cyber infrastructure from terrorist attack.

Intrusion detection is very important aspect of protecting the cyber infrastructure from terrorist attacks. The overarching philosophy driving intrusion detection is that, due to their static nature, intrusion prevention techniques such as firewalls and routing filter policies fail to stop many types of attacks. Therefore, no matter how secure you try to make your system, intrusions still happen and therefore must be detected. Consequently, Intrusion Detection Systems (IDS) form a second line of defense that identify and report attacks in near real time in order to mitigate their damage.

Research in the field of intrusion detection (ID) has been ongoing for approximately 20 years. One of the earliest papers in the field is James Anderson's 1980 paper *Computer Security, Threat Monitoring, and Surveillance* [1]. Seven years later, Dorothy Denning wrote *An Intrusion Detection Model* [2], providing a framework for IDSs. Denning held that evidence of malicious activity would be reflected in the audit records of the affected host.

IDSs are categorized according to model – signature or anomaly, and scope – network-based or host-based. Anomaly detectors attempt to detect usage outside of the bounds of preestablished statistical norms. To effectively do this, anomaly detectors must create and dynamically maintain usage profiles reflecting normative behavior for all users and processes of the system. In contrast, misuse detectors filter usage against a static set of attack signatures. Essentially, anomaly detectors compare events to what is deemed to be acceptable while misuse detectors compare events to what is deemed to be unacceptable. The key differentiator between host-based and network-based IDSs is that a network based IDS, although run on a single host, is responsible for an entire network, or some network segment, while a host based IDS is only responsible for the host on which it resides.

Similar to Denning's approach, we hold that evidence of malicious activity is reflected in system level data, however by analyzing data at much lower level than audit data, we can profile the system's normative state and perform anomaly detection instead of signature detection. The underlying premise of our intrusion detection model is to describe attacks as instances of an ontology using a semantically rich language like DAML [3]. This ontology captures information about attacks such as the system component it affects, the consequences of the attack, the means of the attack, the location of the attacker, etc. Such a target-centric ontology has been developed by Undercoffer et al., [4]. However, before anomalous system behavior can be specified as an instance of the ontology, it first needs to be detected. Hence our intrusion detection model consists of two phases. The initial phase uses data mining techniques to analyze data streams that capture process, system and network states and detect anomalous behavior. The second, or high level

phase, reasons over data that is representative of the anomaly defined as an instance of an ontology. Figure 1 illustrates our IDS model.

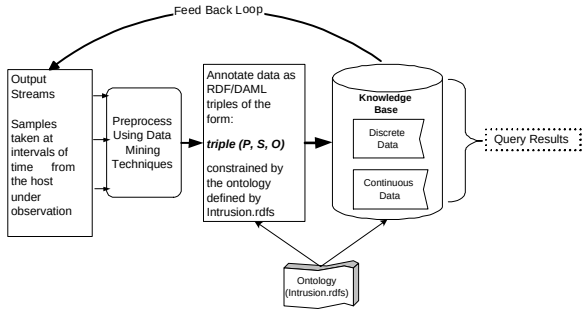

Fig. 1. Two Phase IDS Model

We model the quiescent state of a system based on these data streams. Some work has already been done on using system call data and TCP packet information for building anomaly detection models [5][6]. Our model not only includes streams of system call and network data but also includes low level kernel data such as memory usage, address offsets, MIB data, interrupts etc. These data streams are provided by the operating system for the processes present in the system. Thus one can collect data for any process and build models that represent the quiescent state of the process and the system. We are exploring the use of fuzzy data mining and concepts introduced by the semantic web to operate in synergy to perform *Distributed Intrusion Detection*.

One way to build models from these data streams is to use fuzzy clustering. Accordingly, we use the Fuzzy c-Medoids (FCMdd) algorithm developed by Krishnapuram et al., [7], which takes a dissimilarity matrix for the objects to be clustered as input. The objective functions are based on selecting n representative objects (medoids) from the feature set in such a way that the total fuzzy dissimilarity within each cluster is minimized. We use the Mahalanobis metric [8] as a dissimilarity measure because the differences within the variances of our feature set is large and the Mahalanobis metric mitigates against this by providing scale independence. Once we construct a model representing the quiescent state of the system and its processes, we use this model to detect outliers.

The work presented here is focused on the initial phase of our intrusion detection model: outlier detection within low level data streams. The remainder of this paper is structured as follows. Section II discusses related work on data mining and fuzzy approaches to intrusion detection. Section III details our approach to modeling normal system behavior using low level data streams. We present our results in Section IV and conclude in Section V.

II. Related work

Because anomaly detectors looks for an abnormalities, many of the data mining techniques that seek to identify outliers in data become readily applicable. Hence many researchers have explored applying data mining techniques to the problem of intrusion detection. Lee et al., [6] performed experiments on *sendmail* system call data and network *tcpdump* data. They used RIPPER [9] [10] to generate classifiers for these datasets. In another paper Lee et al., [11] describe how to use association rules and frequent episode algorithms to guide the process of audit data gathering and selection of useful features to build the classifiers.

Dokas et al., [12] have developed classification algorithms for intrusion detection. These algorithms are designed especially for learning from datasets in which the class of interest (i.e. the intrusion class) is significantly smaller than the class representing normal behavior. In this body of work, the authors discuss various outlier detection schemes for detecting network intrusions.

Dickerson et al., [13] developed the Fuzzy Intrusion Recognition Engine (FIRE) using fuzzy sets and fuzzy rules. FIRE uses the *Fuzzy C-Means Algorithm* developed by Bezdek [14] to generate fuzzy sets for every observed feature. The fuzzy sets are then used to define fuzzy rules to detect individual attacks. FIRE does not establish any sort of model representing the quiescent state of the system, but instead relies on attack specific rules for detection. The essential difference between FIRE and our work is that FIRE is a signature based detector while ours is an anomaly based detector.

III. Approach

During our experiments we employed a four step approach to construct and test our model of outlier detection. First, we exercise the target system in an attack free environment, collecting data at the process, system and network levels. We then use Principal Component Analysis (PCA) [15] [16] to reduce the dimensionality of the collected data. Fuzzy clustering is then performed on the data to obtain clusters that model the quiescent state of the system. Finally, the system is placed under attack and data containing anomalous behavior is collected and compared to the clusters in order to test the efficacy of the initial phase of our intrusion detection model. The following details these four steps.

A. Data Collection

We are using data generated by the LINUX kernel and the Apache web server [17], for our experiments. We use kernel and web server versions that are susceptible to the attacks that we launch against the targeted system. As previously stated, we are collecting low level kernel data at the process, system and network levels. Process data includes the number of child processes forked by the parent process, the amount of time spent by the process in user and system modes, attributes describing memory usage, and attributes describing signal information. Collected network data is comprised of statistical information regarding the IP, ICMP, UDP, and TCP layers of the network protocol stack. This data is inclusive of rates of change for the number of packets received, the number of packets re-

ceived in error, and the number of packets dropped. At the system level we collect information regarding memory usage, CPU load, etc. We obtain samples at a rate of once per second to obtain a time-series that is representative of the system state. In total, we have a feature set consisting 189 different values.

We have mirrored a web site on our target system and have obtained the server logs for the mirrored site. To exercise the system we employ three separate client machines to replay those logs against the server over a 5 hour period collecting data that is representative of the quiescent state of the system. To collect data that contains anomalous behavior, we attacked the Apache web server with the attacks described which will be described in section IV while concurrently replaying the same set of server logs from the three client machines. We simultaneously replay the server logs while attacking in order to create a realistic environment.

B. Principal Component Analysis

As is the case with all clustering techniques, FCMdd is affected by the *"curse of dimensionality"* suffering performance degradation when confronted with data sets of high dimensionality, which is the case of our feature set of 189 values. Consequently, we PCA to reduce the dimensionality of our feature set. Accordingly, PCA constructs a new representation of the feature set wherein the maximally variant dimensions of the data is captured. Once processed using PCA, the dimensionality of the feature set is reduced while the maximum amount of information and patterns in the data are preserved. For our feature set we found that the leading 12 eigenvalues accounted for 85% of the sum total of all of the eigenvalues. Hence, we selected the 12 most significant eigenvectors to project the data, reducing the dimensionality from an $m \times 189$ matrix to an $m \times 12$ matrix where m is the number of rows in our data set.

C. Fuzzy Clustering And Outlier Detection

The Mahalanobis metric associates the correlation between parameters and standardizes each parameter to a zero mean and unit variance by using the covariance matrix in the distance calculation.

Once the dimensionality of the data set is reduced, we use FCMdd to cluster the data points. In traditional clustering approaches membership of a data-point in a cluster is binary decision. Either the data-point is a member of the cluster or it is not. In fuzzy clustering the membership of a data-point in a cluster is a fuzzy decision. A data-point is considered to be a member of every cluster with a given possibilistic membership value that ranges from 0 to 1. The objective function of the FCMdd algorithm is based on selecting n representative objects (medoids) from the data set in such a way that the total fuzzy dissimilarity within each cluster is minimized. Here, the clusters of data-points represent the quiescent state of the system. The FCMdd algorithm takes pair wise distances between the data-points as input. We use the Mahalanobis metric to calculate the pair wise distances between the data-points. We use the Mahalanobis metric as opposed to the Euclidean distance because the difference in the variances of the feature set is large and the Mahalanobis metric gives better results than the standard Euclidean distance. Whereas the Euclidean distance measurement treats all values equally when calculating the distance from the mean point, the Mahalanobis metric mitigates the effects of this variability by providing scale independence. The Mahalanobis metric associates the correlation between parameters and standardizes each parameter to a zero mean and unit variance by using the covariance matrix in the distance calculation. The Mahalanobis metric is given in equation 1, where Σ^{-1} is the inverted covariance matrix, and z_i, z_j are the data points that the distance is calculated for.

$$\delta_{\Sigma}(z_i, z_j) = (z_i - z_j)\Sigma^{-1}(z_i - z_j)^T \tag{1}$$

Once the clusters are generated, we compute the *intra-cluster* and *inter-cluster* distances. *Intra-cluster* distances indicate how cohesive the clusters are. For a given cluster, it is calculated as the average of the distances between all pairs of data points within that cluster. The equation for calculating the intra-cluster distance is given in 2 where D denotes the distance matrix such that the entry $D(i, j)$ equals the distance between the i^{th} data-point and the j^{th} data-point.

$$\text{intra-cluster}_k = \frac{\Sigma_{i \in k} \Sigma_{j \in k, i \neq j} D(i, j)}{|k| \times (|k| - 1)} \tag{2}$$

Conversely, *Inter-cluster* distances indicate how well the clusters are spread out from each other. The *Inter-cluster* distance between two given clusters is calculated as the average of the distances between all pairs of data points such that one data point is selected from the first cluster and the other data point is selected from the second cluster. The equation for calculating the inter cluster distance is given in 3, where $k1$ and $k2$ denote two clusters with $|k1|$ and $|k2|$ data-points respectively.

$$\text{inter-cluster}_{k1,k2} = \frac{\Sigma_{i \in k1} \Sigma_{j \in k2} D(i, j)}{|k1| \times |k2|} \tag{3}$$

The *intra-cluster* distance provides the average diameter of the cluster, and is used for outlier detection. Hence the *intra-cluster* radius r is half of the *intra-cluster* distance. A data point is, therefore, a member of a given cluster if its distance from the medoid of that cluster is less than a specified threshold distance for that cluster. A data point is determined to be an outlier if it is not a member of any of the clusters that model the system's normative state. The threshold for a given cluster is based on the *intra-cluster* radius for that cluster. In our experiments we vary the threshold value from $1.5r$ to $0.5r$, where intuitively it would seem that the smaller the threshold size the greater the number of false positives and the greater the threshold size the larger the number of false negatives.

The FCMdd algorithm produces multiple medoids for a cluster. For a given test data-point, if the distance from any randomly selected medoid is the only consideration, it might wrongly classify the data-point as a member or non-member.

In order to resolve this issue, we employ two different outlier detection schemes, both of which take all the medoids of a cluster into account. In the **Absolute Distance** scheme, we calculate the distances of a test data-point from all the medoids of a cluster. If any of these distances is within the specified radius threshold the data-point is a member of that cluster. In the **Average Distance** scheme we calculate the average of the distances of the test data-point from all the medoids of the cluster. If this average distance is within the specified radius threshold of some cluster, the data-point is a member of that cluster and an outlier otherwise.

IV. Experiments and Results

We carried out two types of attacks to test the efficacy of our outlier detection mechanism. The first type of attack that was carried out was effected by making a series of HTTP requests that were ill formed and could potentially result in input validation errors. We refer to these series of attacks as *Input Validation Attack* and they consist of the following ill formed inputs:

- Cross Site Scripting [18]. Cross-site scripting is a vulnerability in the default error page of the Apache web server when *UseCanonicalName* is turned "Off" and support for wildcard DNS is enabled. It allows remote attackers to execute a script.
- Artificially Long Slash Path Directory Listing Vulnerability [19]. This vulnerability makes it possible for a malicious remote user to launch an information gathering attack, which could potentially result in compromise of the system.
- Long Strings [20]. A Buffer overflow in the *ApacheBench* benchmark support program causing a denial of service and possibly the execution of arbitrary code via a long response.
- Odd Characters. HTTP requests containing odd characters such as: #, $>$, $<$, $*$, !, etc.

The second type of attack that we carried out, is commonly referred to as an exploitation of the **Apache Web Server Chunk Handling Vulnerability** [21]. This attack exploits a vulnerability in the manner that the Apache web server handles data encoded in chunks [22]. Chunk encoding is a process that allows a client process to submit a variable-sized quantity of data to a web server, called a chunk. The chunks contain header information that allows the web server to assemble a message arriving as a series of chunks and ensure whether it has received the entire message. Some versions of the Apache web server contain a bug in the routines which deal with requests using chunked encoding. A carefully crafted invalid request can cause an Apache child process to call the memcpy() function in a way that will write past the end of its buffer, corrupting the stack. On some platforms this can be remotely exploited , allowing arbitrary code to be run on the server. On some other platforms such requests simply cause the child process handling the bad request to die resulting in a Denial of Service attack as the server's resources are wasted in setting up the killed child processes again. We refer to this attack as the *chunking* attack in the following discussion.

Radius	Input Validation Det. Rate	Chunking Det. Rate	False Positives
1.5r	10.20	16.34	4.55
1.25r	10.34	18.29	7.08
1.0r	12.1	25.93	12.22
0.75r	18.06	55.96	20.77
0.5r	80.36	83.02	34.84

TABLE I
Absolute Distance Scheme : All Clusters

Radius	Input Validation Det. Rate	Chunking Det. Rate	False Positives
1.5r	10.20	16.34	4.55
1.25r	10.34	18.29	7.08
1.0r	12.1	25.93	13.16
0.75r	18.06	55.96	25.73
0.5r	80.36	83.02	48.50

TABLE II
Average Distance Scheme : All Clusters

We collected data from the target system while it was under the above mentioned attacks. To test our outlier detection mechanism, we split the normal data into two sets: training dataset (80%) and test dataset (20%). The training dataset was used to generate the clusters. The test dataset and the attack datasets were then examined using the previously detailed outlier detection schemes. We measured the detection rate as the percentage of data-points detected as outliers. To determine the effect on detection rate, we varied the radius threshold from $1.5r$ to $0.5r$ in steps of $.25r$ where r is the *intra-cluster* radius. The 20% normal dataset was compared to the clusters along with the attack data in order to determine the false positive rate.

Using 80% of normal data, the FCMdd produced four clusters, two of which were dominant, containing 85% of the data points. Consequently, we conducted a series of four experiments. We applied both the *Absolute Distance* and *Average Distance* schemes to all four clusters during two experiments. In the other two experiments we discarded the two minority clusters and applied the *Absolute Distance* and *Average Distance* schemes to the two dominant clusters.

Table I illustrates the results of using the *Absolute Distance* scheme and all four baseline clusters. Table II illustrates the results of using the *Average Distance* scheme and all four baseline clusters. Table III illustrates the results of using the *Absolute Distance* scheme and only the two dominant clusters. Table IV illustrates the results of using the *Average Distance* scheme and only the two dominant clusters.

We use the performance measure presented by Dokas et al., [12] to define optimality. Equation 4 presents this measure.

Radius	Input Validation Det. Rate	Chunking Det. Rate	False Positives
1.5r	14.52	43.53	16.12
1.25r	30.17	62.20	21.02
1.0r	59.54	88.74	26.23
0.75r	100	98.54	40.20
0.5r	100	100	47.87

TABLE III
ABSOLUTE DISTANCE SCHEME : DOMINANT CLUSTERS ONLY

Radius	Input Validation Det. Rate	Chunking Det. Rate	False Positives
1.5r	16.03	46.05	17.76
1.25r	69.24	72.68	25.42
1.0r	99.80	92.30	36.39
0.75r	100	100	46.50
0.5r	100	100	63.16

TABLE IV
AVERAGE DISTANCE SCHEME : DOMINANT CLUSTERS ONLY

$$\frac{\text{true positive rate}}{\text{true positive rate} + \text{false positive rate}} \quad (4)$$

According to 4, when conducting outlier detection for the *Input Validation* attacks performance is optimized using *Average Distance* scheme using the majority clusters and setting the threshold radius to $1.0r$.

Similarly, when conducting outlier detection for the *Chunking Vulnerability* attack performance is optimized using the *Absolute Distance* scheme using all clusters and setting the threshold radius to $1.5r$. It is important to state, that although the performance is optimal, it is unacceptable because the true positive rate is only 16.34% when the maxium is 100% . Consequently, a trade-off needs to be made in order to increase the true positive rate.

It should be noted that our preliminary results indicate that our method of outlier detection has a high false positive rate while yielding very high true positive rates. This is not an unexpected trade-off in the area of intrusion detection. Recall that our IDS model calls for sampling low level kernel data at close intervals and using outlier detection to capture anomalous behavior within those data streams. This initial phase serves as a filter to the higher level reasoning phase which will mitigate the high false positive rate.

V. CONCLUSION

At this preliminary stage it appears that using low level kernel data streams to model the quiescent state of a system is a viable intrusion detection approach. Likewise, fuzzy clustering appears to be effective for outlier detection within these low level system, process and network data streams.

REFERENCES

[1] James P. Anderson. Computer Security, Threat Monitoring, and Surveillance. Technical report, James P. Anderson, Co., April 1980.

[2] Dorothy E. Denning. An Intrusion Detection Model. *IEEE Transactions on Software Engineering*, SE 13(2):222–232, February 1987.

[3] J. Hendler. *DARPA Agent Markup language. http://www.daml.org*. 2000.

[4] Jeffrey Undercoffer and Anupam Joshi. *On Web, Semantics and Data Mining: Intrusion Detection as a Case Study*. Next Generation Data Mining, MIT Press, 2003.

[5] S. Hofmeyr, S. Forrest, and A. Somayaji. *Intrusion detection using sequences of system calls.* Journal of Computer Security, Vol. 6, pp. 151-180, 1998.

[6] Wenke Lee and Salvator J. Stolfo. *Data Mining Approaches for Intrusion Detection*. Proc. 1998. 7th USENIX Security Symposium, 1998.

[7] Raghu Krishnapuram, Anupam Joshi, Olfa Nasraoui, and Liyu Yi. *Low-Complexity Fuzzy Relational Clustering Algorithms for Web Mining*. IEEE transactions on Fuzzy Systems, 2001.

[8] P.C.Mahalanobis. *On tests and Meassures of Groups Divergence*. International Journal of the Asiatic Society of Bengal, 1930.

[9] William W. Cohen. *Fast Effective rule induction*. In Proceedings of the 12th International Conference on Machine Learning, 1995.

[10] William W. Cohen. *Text categorization and relational learning*. "In Proceedings of the 12the International Conference on Machine Learning", 1995.

[11] Wenke Lee, Salvator J. Stolfo, and K. Mok. *Mining Audit Data to build Intrusion Detection Models*. Proc. KDD-98, 1998.

[12] Paul Dokas, Levent Ertoz, Vipin Kumar, Aleksandar Lazarevic, Jaideep Srivastava, and Pang-Nig Tan. *Data Mining for Network Intrusion Detection*. Next Generation Data Mining, 2002.

[13] John E. Dickerson, Jukka Juslin, Ourania Loulousoula, and Julie A. Dickerson. *Fuzzy Intrusion Detection*. IFSA World Congress and 20th North American Fuzzy Information Processing Society (NAFIPS) International Conference, 2001.

[14] J.C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York, 1981.

[15] E.Anderson, Z.Bai, C.Bischof, S.Blackford, J.Demmel, J.Dongarra, J.D.Croz, A.Greenbaum, S.Hammarling, A.McKenney, and D.Sorensen. *LAPACK User's Guide*. Society for Industrial and Applied Mathematics, 1999.

[16] G.H.Golub and C.F.V.Loan. *Matrix Computations*. The Johns Hopkins University Press, 1989.

[17] *http://httpd.apache.org/*.

[18] Mitre Common Vulnerabilities Index. Can-2002-0840. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0840, October 2002.

[19] Mitre Common Vulnerabilities Index. Can-2001-0925. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0925, November 2001.

[20] Mitre Common Vulnerabilities Index. Can-2002-0843. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0843, August 2002.

[21] Cert advisory ca-2002-17 apache web server chunk handling vulnerability. http://www.cert.org/advisories/CA-2002-17.html, June 2002.

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616: Hypertext transfer protocol – http/1.1. http://www.ietf.org/rfc/rfc2616.txt, June 1999.