Behavior Analysis-Based Learning Framework for Host Level Intrusion Detection

Haiyan Qiao, Jianfeng Peng, Chuan Feng, Jerzy W. Rozenblit
Electrical and Computer Engineering Department
University of Arizona
Tucson, Arizona, USA
{haiyanq, jpeng, fengc, jr}@ece.arizona.edu

Abstract

Machine learning has great utility within the context of network intrusion detection systems. In this paper, a behavior analysis-based learning framework for host level network intrusion detection is proposed, consisting of two parts, anomaly detection and alert verification. The anomaly detection module processes unlabeled data using a clustering algorithm to detect abnormal behaviors. The alert verification module adopts a novel rule learning based mechanism which analyzes the change of system behavior caused by an intrusion to determine whether an attack succeeded and therefore lower the number of false alarms. In this framework, the host behavior is not represented by a single user or program activity; instead, it is represented by a set of factors, called behavior set, so that the host behavior can be described more accurately and completely.

1. Introduction

With the growing number of network attacks, intrusion detection systems (IDSs) are becoming an integral part of any complete security package of a modern network system. The IDSs perform surveillance and security monitoring of the network infrastructure. A number of different intrusion detection systems have been developed for particular domains (e.g., hosts or networks), in specific environments (e.g., Windows NT or Solaris), and at different levels of abstractions (e.g., kernel-level tools or application level tools). However, computer systems and networks still suffer from an increased threat of intrusions. The existing IDSs are far from perfect and may generate false positive and non-relevant positive alerts [1].

The most widely deployed and commercially available methods for intrusion detection employ signature-based technique, where the signatures or patterns of well-known attacks are provided by human experts. The system or network traffic is scanned for attacks using well-known vulnerabilities and any instances that match the signatures are detected as intrusions. The advantage of this method is a low rate of false positives. The disadvantage is that the signature database has to be revised manually and the system is vulnerable to new types of attack until the revision is done. This limitation leads to active research on intrusion detection techniques based on data mining.

Data mining based network intrusion detection techniques are generally classified into two categories: misuse detection and anomaly detection [2]. In misuse detection, each instance in the training data set is labeled as either normal or intrusion. A machine learning algorithm is trained over the labeled data and then the trained model is applied to classify new data. Misuse intrusion detection is fast, requires little state information, and has a low false-positive rate. With different input data including new types of attacks, the intrusion detection modules are retrained automatically without the manual intervention. However, misuse detection cannot detect novel, previously unseen attacks. Anomaly detection, on the other hand, builds models of normal data to measure a "baseline" of such stats as CPU utilization, disk activity, user logins, file activity, and so on. When there is a deviation from this baseline, an alert is triggered. Currently, almost all commercial intrusion detection systems use misuse detection techniques. Yet, anomaly detection is getting more attention because of its capability to detect novel or unforeseen attacks. Essentially, anomaly detection is the machine learning problem of modeling a normal network or system behavior. Although anomaly detection is becoming an active research topic, widespread adoption of this method faces numerous obstacles, including complexity and high false positive rate.

In order to reduce false and irrelevant alerts, alert verification has to be a part of IDS. Alert verification is

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

a process to determine whether an attack has been successful or not. This information is passed to IDS to help differentiate the type of alerts [16]: 1) The sensor has correctly identified a successful attack; 2) The sensor has correctly identified an attack but the attack failed to meet its objectives; and 3) The sensor incorrectly identified an event as an attack. Alert verification effectively lowers the number of false alarms that an administrator or the decision support system has to deal with.

In this paper, a behavior analysis-based learning framework for host based intrusion detection is proposed, which includes anomaly detection and alert verification. The framework has two characteristics. First, it is learning-based. In the anomaly detection module, a cluster-based outlier detection algorithm detects anomalous data. In the alert verification module, a rule-learning algorithm is applied to learn the behavior changes of the targeted machine. The rules developed serve as an index of alert verification. Second, the framework is behavior analysis oriented. Instead of using a single activity as the indicator of the host behavior, a set of indicators, also called "behavior set", is defined and applied to describe the host behavior. The intrusions are detected and verified based on the analysis of the behavior set. Compared to other research work in host-based anomaly detection, this method does not need high-dimension data since the host baseline is represented by a refined behavior set and each element in the behavior set is of low dimensionality. Thus, state-of-the-art data normalization is not required when outlier detection is applied to detect anomalous data. In section 2, host based anomaly detection is reviewed. In section 3, the behavior analysis-based learning framework is presented and discussed. In section 4, the experimental results are given. Finally, conclusions and future work are summarized in Section 5.

2. Related Work

In host based anomaly detection research, various features are used to model the system behavior baseline, e.g., keystroke characteristics, user command data, system call sequences, file activities, etc. Generally these features fall into three categories: user profile, program profile, and system resource access.

Denning [3] first attempted to build anomaly detection by comparing previous user profiles to current user activity. Sequeira and Zaki [4] designed and implemented a user-profile dependent and temporal sequence clustering-based intrusion detection system by collecting and processing UNIX shell command data. Although analysis of user activity is a natural approach to detect intrusions, experience shows that it is far from accurate. This is because user behavior typically lacks strict patterns.

User dynamics allowed more reflection on the features that define host behavior. All actions carried out by users involve using programs. Programs obtain the required services by executing the specific system call that provides the needed function. Since the code of a given application should not change, the sequence of system calls executed by a program should be regular and predictive.

Most existing research on anomaly detection uses system calls to model system behavior. A number of approaches based on system calls are proposed. Forrest et al. [5] established an analogy between the human immune system and intrusion detection. Lee et al [6] applied a rule learning program to study a sample of system call data. Wagner and Dean [7] proposed to statically generate a non-deterministic finite automaton (NDFA) from the global control-flow graph of the program and simulated NDFA on observed system call trace. Ghosh et al. [10] utilized return address information extracted from the call stack to generate the execution path of a program for anomaly detection. Liao and Vemuri [9] used the k-Nearest Neighbor classifier to classify program behavior represented by frequencies of system calls instead of system call sequence. Eskin et al. [11] applied outlier detection algorithms to anomaly detection using system call data, where the system call data has to be mapped into feature space and the choice of feature space is application specific.

Because the system-call level data is fine grained, it increases overhead and decreases system performance. Thus, some researchers study anomaly detection by modeling files activities. Stolfo et al. [13] studied anomaly detection by learning file system access patterns. Apa et al. [12] noticed that in Windows OS almost all system activities interact with the registry, so they analyzed anomaly detection through modeling normal registry access.

In reality, an attack is usually unpredictable, and it is difficult to know which aspect of the system behavior is associated with the attack. For this reason, modeling system behavior based on only a single category is unreliable, no matter whether the category is user profile, program profile, or system resource access. In what follows, we try to model behavior from all three categories to increase the reliability of system behavior modeling, and thus to increase the accuracy of anomaly detection.

3. Behavior Analysist-Basd Learning Framework

A system consists of both the user and the host machine. It is appropriate to describe the system behaviors using a set of factors of user profile, program

profile, and system resource usage. We call the set of factors the "behavior set."

In this section, a learning framework of intrusion detection is proposed based on analysis of the behavior set, as shown in Fig. 1. This framework includes three modules: anomaly detection, alert fusion, and alert verification. The input of the framework is an event, modeled as a triple $\{subject \times verb \times object\}$ for user/program behavior, and the output is alerts with features of alert ID, alert type, timestamp, priority level, confidence factor, verification status. In the anomaly detection module, the normal behavior baseline is modeled adaptively and stored in the database. When a new event comes, it is classified as normal or intrusion by the module. If an alert is triggered, the alert is fused with other existing alerts to decrease the number of alerts with the same cause. Then the fused alerts are sent to the alert verification module to exclude false or unrelated alerts. In this paper, we discuss only the learning related modules: anomaly detection and alert verification.

To model normal system behavior, both supervised and unsupervised learning algorithms can be applied. We choose unsupervised learning over supervised learning. The main reason is that unsupervised learning does not need labeled (normal/abnormal) data, which is not readily available in reality. Labeled data is generally obtained by simulation or experiments. If labeled data is obtained by simulated intrusions, we are limited to the set of known attacks that we simulated. New types of attacks are not reflected in the training data set. If labeled data is obtained by experiments, then we must face the difficulties in manually classifying large volume of audit data. In addition, if the experimental data labeled normal have buried intrusions, then future instances of those intrusions will not be detected because they are assumed normal in the training set.

Unsupervised learning algorithms take as input a set of unlabeled data and attempt to find noise and intrusions buried within the data. If anomalies are rare, unsupervised learning can be treated as a variant of the outlier detection problem. Outlier based anomaly detections cluster the data based on certain metrics and the data located in sparse regions are claimed as intrusions. Not all intrusions can be detected using outlier based anomaly detection. For example, syn-flood DOS cannot be detected using outlier detection since they are not rare in data distribution. Only when system behavior deviates significantly from average behavior are outlier detections applicable.

3.1. *Outlier detection algorithm*

The outlier detection algorithm we propose is given in Table I. The algorithm is an improvement of fixed-width cluster estimation [11]. For each point, the algorithm approximates the density of points near the given point. The algorithm makes this approximation by counting the number of points that are within a sphere of radius w around the point. Points that are in a dense region of the feature space and contain many points within the circle or ball are considered normal. Points that are in a sparse region of the feature space and contain few points within the circle or ball are considered anomalies.

In the original fixed-width cluster estimation, data that does not belong to any existing clusters is assigned to a new cluster and works as the central point. So the central point of a cluster is sensitive to the sequence of data to be clustered and new data might not be assigned to the closest cluster. In the improved fixed width clustering algorithm, the central point of a cluster is adjusted as new data is added so data are always assigned to the closest cluster. The idea of updating the central point of a cluster is close to K-means clustering, one of the most popular statistical clustering algorithms. Unlike K-means clustering, the improved fixed-width cluster estimation specifies cluster width instead of fixing the number of clusters a priori. Thus the results are sensitive to the value of cluster width. However, this problem can be easily solved by interaction with the clustering results through GUI. Since the clustering is performed offline, it is easy to adjust the width adaptively based on the visual clustering results when the clustering data is of low dimension.

Using outlier detection, it is preferable that the training data be of low dimensionality for the following reasons. First, when distances are measured in all dimensions, it is more difficult to detect outliers effectively because of the average behavior of the noisy and irrelevant dimensions. Second, it is not easy to intuitively explain and understand the clustering results with high dimensions. Third, in high dimensional space, the data is sparse and the notion of proximity fails to retain its meaningfulness. Finally, with increasing dimensionality, it becomes increasingly difficult and inaccurate to estimate the multidimensional distribution of the data points.

Actually, in the learning framework we propose, we do not need to use high dimensionality to represent the feature space of the system behavior. Because the system behavior is described by a set of factors instead of a single complex indicator, all the elements in the behavior set are ensured to be represented by low dimensional data, e.g., the login time, access frequencies of the files, etc.

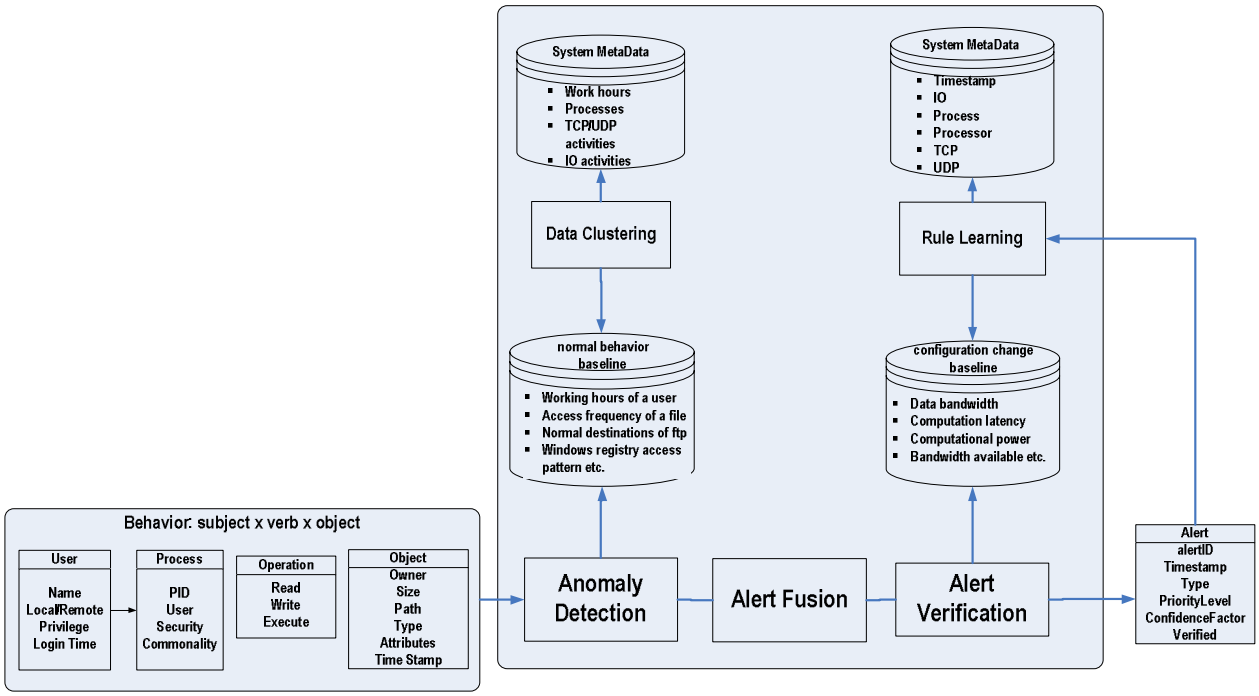

Figure 1. Behavior analysis-based learning framework for host-based intrusion detection

3.2. *Behavior analysis-based intrusion detection*

Anomaly detection can only detect intrusions that make the host behave differently from normal. Thus, before we attempt to model normal behavior, the key question is how to define host behavior. As analyzed at the beginning of the section, we use behavior set to describe system behavior in order to accurately and completely reflect the system characteristics. The behavior set is specified as a set {*user login time, frequency of applications launched, I/O activities, frequency of files access, network activities stats, CPU usage pattern over time, memory usage pattern, data bandwidth, network connection speed*}. In the set, user login time helps to locate normal login time intervals; frequency of application launched locates the most and the least used applications; frequency of files can locate the most and the least accessed files; I/O activities indicates the average bytes written to the files; network activities stats locate normal sequence, frequency, time interval, and other indexes of various network activities, e.g., web browsing, ftp etc.; CPU and memory usages etc. are related to system performance.

To describe the host behavior, we need metadata. A relational database is constructed to store system profiling metadata. The database constructed with WAMP is shown in Fig. 2. In the database, the table features are well designed to reveal the information defined in the behavior set. For example, the table *USER* has features of *userID, user name, login time, remote/local login, logoff time*. The table *PROCESS* has features of *process Id, process name, owner of the process, memory usage of the process, CPU usage of the process, time*.

To maintain data in real time, the data is updated periodically. The time interval to collect data can be modified by the user via a sliding bar in the GUI as shown in Fig. 2. In addition, the user can easily select a table and look at the data through the database interface. When the record size overflows, the new record overwrites the oldest record.

With the profiling database, the outlier detection algorithm is applied to each single factor in the behavior set if applicable. For instance, the user login time is trained to get the normal login time intervals, e.g., [8:00 9:00] and [13:00 14:00]. The CPU usage pattern is learned to model the normal distribution pattern, e.g. peak time at [9:00 11:00] and [14:00 16:00].

Typically, only a few indicators of the behavior set are detected as abnormal. So the significance of the host's deviation from normal behavior has to be measured in some way. In this framework, we adopt the idea of the weighted sum. First, the deviation of each behavior indicator is calculated using the outlier detection algorithm. Then we compute the overall deviation from the host baseline model:

$$(w_1 d_1 + w_2 d_2 + \cdots + w_m d_m)/(w_1 + w_2 + \cdots + w_m),$$

Where m is size of behavior set, w_i is predefined weight of indicator i in the behavior set, and dm is binary value 0 or 1, i.e., normal or abnormal. When the total deviation is beyond a threshold, an alert is generated. The values of w_i $(i = 1,2,\cdots m)$ can be customized based on the host function and the user type. For example, a software developer and an administrator will have very different file activities. If we know the user of the host is a software

developer and the machine is mainly used for software development, we can put more weight on deviations of frequency of applications launched, I/O activities, and frequency of file access.

TABLE I. ANOMALY DETECITON ALGORITHM

Inputs:

Cluster width w

The unlabeled data set D with n dimension

Initialization:

Set of clusters S is empty

Loop until D is empty

For each data d in data set D:

If S is empty

create a cluster C with one element d, add C the cluster to $SC \cup S \rightarrow S$, and set d as centroid (x_0, y_0) of C .

Otherwise

find the closest cluster C in S such that for any cluster C' in S, $dist(C, d) \leq dist(C', d)$, where $dist(C, d)$ is Euclidean distance from d to the centroid of cluster C.

If dist(C, d) $\leq w$

insert d into cluster C. adjust the old centroid (x_0, y_0) to $(x_0^{'}, y_0^{'})$, where $x_0^{'} = (m \cdot x_0 + m)/(m+1), y_0^{'} = (m \cdot y_0 + m)/(m+1), m = |C|$ -1.

Otherwise

a new cluster C_d with d as centroid is created, $S \cup \{C_d\} \rightarrow S$.

$D - \{d\} \rightarrow D$

Find outliers:

Sort the clusters based on sizes in ascending sequence, $|C_i| \leq |C_2| \cdots \leq |C_l|$, where l is the number of clusters.

Let $|C_i| + |C_2| + \cdots + |C_l| = p$.

For $i = 1$ to l

If $|C_i| < (p/l) * 20\%$, set the cluster $|C_i|$ as abnormal, i++

Otherwise, set any cluster $|C_j|, j \geq i$ as normal, break

When a new service is installed or the user's behavior changes suddenly, false alerts are triggered. In this case, the fuse module will fuse alerts generated for the same reason. When new data instances are treated as intrusion by the anomaly detection module at the beginning of a pattern change, the new data are not discarded. As the data forming new pattern increases, the host profiling database is updated, and the clustering algorithm is retrained over the new set of data for modeling the emerging new pattern

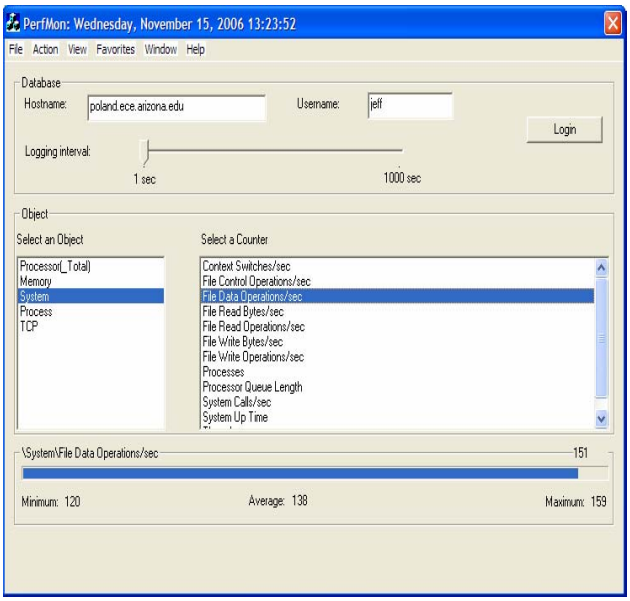

Figure 2. System performance database

When the behavior set is modified, the learning algorithm remains the same, but the database has to be modified or reconstructed.

3.3. *Behavior analysis-based alert verification*

Conventional alert verification approaches include active and passive mechanisms [1]. Passive alert verification mechanisms compare the configuration of the victim machine to the requirements of a successful attack, e.g., a vulnerable version of the Microsoft IIS server. It gathers configuration data before an attack occurs and determines whether the target machine is vulnerable to the attack. Active mechanisms model the expected "outcome" of attacks, checking the visible and checkable traces that a certain attack leaves at a host, e.g., a temporary file or an outgoing network connection. They gather configuration data or forensic traces after an alert occurs. Neither approach can detect unknown attacks. They assume that either the system vulnerability required by an attack or the outcome of an attack is known in advance. If this is true, the attack can be prevented by installing a new service pack. Then there is no need to detect the attack.

In the proposed learning framework, we focus on the verification of unknown and new types of attacks based on host behavior analysis. The after-attack performance measured in quantity might be different in different systems. However, the variance between prior-attack behavior and post-attack behavior is consistent on various machines. For example, an attack causes the network connection to be 50% slower no matter what type of CPU and bandwidth the machine has. Therefore, the basic idea of the alert

verification module is to learn the behavior features associated with an attack and check the change of the value of those features. It is applicable to the attacks that cause system behavior change.

In the training phase, once an alert is verified manually by system administration, the system behavior both before and after the attack occurs are sampled from the system performance database and stored in a separate system metadata database. Since each alert has a timestamp *t*, we just need to probe the system performance database and get the records at time *t* and the last sample time *t-1*. For each system performance-related feature in the records, we calculate the change of feature values:

$$\Delta feature_i = (feature_i(t) - feature_i(t-1) / feature_i(t-1).$$

The system performance features include CPU usage, memory usage, I/O usage, network stats etc. A record in the form of

$$\{alertID, alertType, \Delta feature1, \Delta feature2, \cdots, \Delta feature_l\}$$

is added to the configuration change baseline database, where l is the number of total features, $\Delta feature_i$ is the relative value change of feature i. When the configuration change baseline database is constructed, the rule learning algorithm RIPPER [14] will be applied to the database. When applying RIPPER, we get rules for different types of alerts separately. To extract rules for alert type i using RIPPER, all the records with that alert type are treated as positive data, all the other records are treated as negative data. RIPPER takes the positive and negative data sets as input, and outputs a rule set represented as a conjunction of conditions in the form of $A_n = v$, $A_c \le \theta$, or $A_c \ge \theta$, where A_n is a nominal attribute and v is a legal value for A_n, and A_c is a continuous variable and θ is some value for A_c.

4. Experimental Results

There are some benchmark data available for network intrusion detection research. For host-based intrusion detection, one data set is from the 1999 DARPA intrusion detection evaluation data which consists of BSM (Basic Security Module) data of all processes run on Solaris machines. Another set of data, obtained from Stephanie Forrest's group at the University of New Mexico [15], contains normal traces for certain programs as well as intrusion traces of system calls for several processes. In our study, these benchmark data are not applicable. First, because we use a novel concept, "behavior set," to describe system behavior instead of using system call traces, and second, because all behavior elements are of low dimension data for the accuracy of clustering. System call data cannot be used for clustering directly and the preprocessing of data is expensive. Eskin et al. [11] adopted spectrum kernels to map the system call data into feature space. However, the feature space corresponding to system calls is in large dimension, e.g., 26 possible system calls and sub-sequences of length 4 will give a dimension of the feature space 26^4, close to 500,000.

To implement the learning framework proposed, we started with system metadata collection. Based on the behavior set we defined, as described in section 3, we constructed the WAMP database server and wrote C++ code to collect the system performance data from a host and store the data into the database. The data will be used as a training set to model the normal behavior of the system. The data is collected in real time and the database is updated dynamically. The data sampling rate is predefined and can be modified through a sliding bar on the GUI as shown in Fig. 2. Fig. 3 shows one of the tables in the system profiling database -- CPU usage data collected from the host.

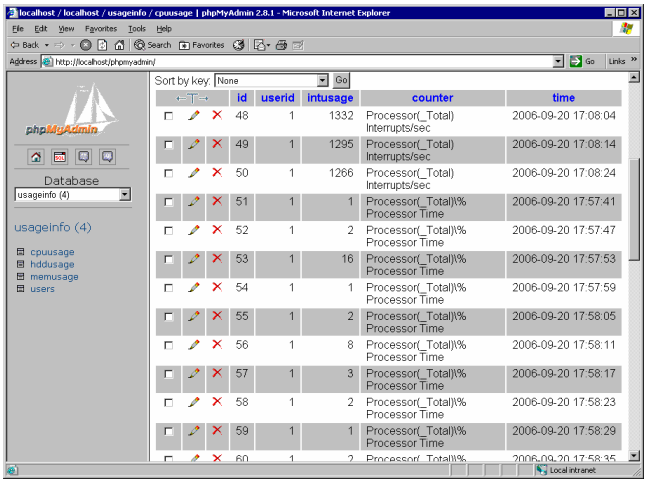

Figure 3. CPU usage data collected from host

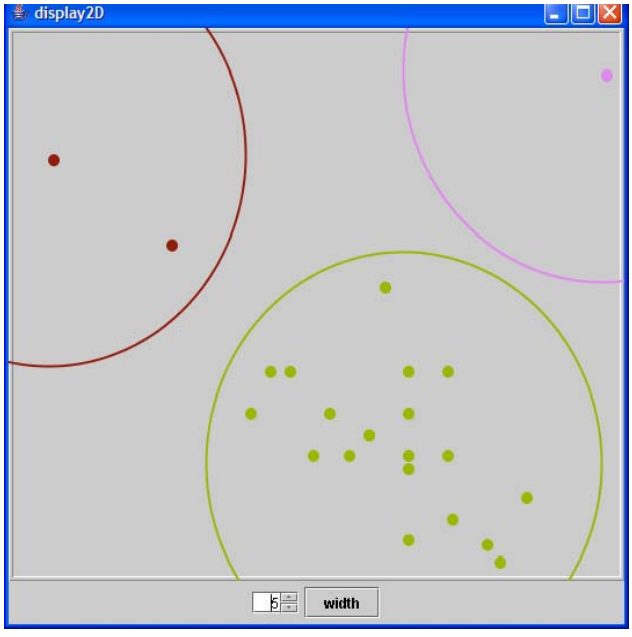

Figure 4. Experimental result of clustering

The unsupervised anomaly detection algorithm is implemented in Java. Given the training data set and cluster

width, the cluster-based outlier detection algorithm is applied and the simulation result is shown in Fig. 4. Using this algorithm, data instances in sparse regions are treated as noise and intrusions buried in the training data, and data instances in dense regions are treated as normal data. When a new data instance comes, whether it is labeled as normal or intrusion depends on which region it is located in. The algorithm needs to specify cluster width instead of the number of clusters. Because the clustering is performed offline, the cluster width can be adjusted easily through the GUI for accurate results.

The connection with the database server is implemented with Java JDBC and the anomaly detection algorithm is also applied to real data from the database. The initial experimental results on a few factors in the behavior set, e.g., the user login time and CPU usage distribution over time, are satisfactory. Further experiments need to be carried out.

To adapt to system behavior pattern changes, the database is updated periodically and the algorithm is applied over the new input data to update the clusters that represent normal behavior.

5. Conclusion

In this paper, behavior analysis-based intrusion detection at the host level has been discussed and a learning frame has been proposed. Two parts in the frame, anomaly detection and alert verification, have been designed using machine learning techniques. The behavior analysis-based framework has the following advantages: 1) the host behavior is described more accurately and comprehensively with a set of indicators, i.e., user profile, program profile, and system resource access; 2) the anomaly detection module does not need labeled data, which are difficult to obtain; 3) each indicator in the behavior set is represented by the data in low dimensionality since the host behavior is refined into a set of indicators; 4) with these low dimension data, the clustering algorithm can be applied over the data without normalization, which is data-dependent and application specific; 5) the clustering algorithm does not need to pre-define the number of clusters, and the width of clusters can be adjusted visually offline; and 6) a novel alert verification approach can check the changes in the host behavior caused by an attack and learn rules associated with the attack.

Currently, the anomaly detection module has been simulated and the host profiling database has been constructed. In the future, the anomaly detection module will be tested on data from a real-world database and the testing results will be carefully examined. In addition, we will implement the alert verification mechanism.

References

[1] F. Valeur, G. Vigna, C. Kruegel, R. A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. 2004

[2] S. Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Department of Computer Engineering, Chalmers University, March 2000.

[3] D. E. Denning, "An intrusion-detection model." IEEE Transactions on Software Engineering, Vol. SE-13(No. 2):222-232, Feb. 1987.

[4] K. Sequeira, M. Zaki, ADMIT: Anomaly-based Data Mining for Intrusions, Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, July 2002.

[5] S. Forrest, S.A. Hofmeyr, and A. Somayaji. Computer immunology. Communications of the ACM, 40(10):88-96, October 1997.

[6] W. Lee, S. Stolfo, and P.K. Chan. Learning patterns from unix process execution traces for intrusion detection. In Proceedings of AAAI97 Workshop on AI Methods in Fraud and Risk Management, 1997.

[7] D. Wagner and D. Dean, "Intrusion Detection via Static Analysis", *IEEE Symposium on Security and Privacy*, Oakland, CA, 2001

[8] Henry Hanping Feng, Oleg M. Kolesnikov, Prahlad Fogla, Wenke Lee, and Weibo Gong. Anomaly Detection Using Call Stack Information. 2003

[9] Yihua Liao, V. Rao Vemuri. Use of K-Nearest Neighbor classifier for intrusion detection. Computer and Security. Volume 21, Issue 5, 1 October 2002, pages 439-448

[10] A. K. Ghosh, A. Schwartzbard, and M. Schatz. Learning program behavior profiles for intrusion detection. In *Proceedings of the 1st* USENIX Workshop on Intrusion Detection and Network Monitoring. USENIX Association, April 11-12 1999.

[11] Eskin , E, A Arnold, M Prerau, L Portnoy, SJ Stolfo. A geometric framework for unsupervised anomaly detection: detecting intrusions in unlabeled data. In Data Mining for Security Applications, 2002.

[12] Apap, F., A. Honig, S. Hershkop, E. Eskin and S. Stolfo. Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses. *Fifth International Symposium on Recent Advances in Intrusion Detection*, RAID-2002. Zurich, Switzerland, 2002.

[13] Stolfo, S. J., L. Bui, Shlomo. Hershkop. Unsupervised Anomaly Detection in Computer Security and an Application to File System Access. *Proc. ISMIS*, pp. 14–28, 2005.

[14] Cohen, W. W. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*. Lake Tahoe, CA, 1995.

[15] Warrender , C., S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. In *1999 IEEE Symposium on Security and Privacy*, pages 133–145. IEEE Computer Society, 1999

[16] C. Kruegel and W. Robertson, "Alert Verification: Determining the Success of Intrusion Attempts," *Proc. First Workshop the Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2004),* July 2004.