An Unsupervised Anomaly Detection Framework for Multiple-connection Based Network Intrusions

by

Wei Lu
M.Sc, Huazhong University of Science and Technology, 2001
B.Sc, Huazhong University of Science and Technology, 1999

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

Supervisor: Dr. Issa Traore

ABSTRACT

In this dissertation, we propose an effective and efficient online unsupervised anomaly detection framework. The framework consists of new anomalousness metrics, named IP Weight, and a new hybrid clustering algorithm, named I-means. IP Weight metrics provide measures of anomalousness of IP packet flows on networks. A simple classification of network intrusions consists of distinguishing between single-connection based attacks and multiple-connection based attacks. The IP weight metrics proposed in this work characterize specifically multiple-connection based attacks. The definition of specific metrics for single-connection based attacks is left for future work. The I-means algorithm combines mixture resolving, a genetic algorithm automatically estimating the optimal number of clusters for a set of data, and the k-means algorithm for clustering.

Three sets of experiments are conducted to evaluate our new unsupervised anomaly detection framework. The first experiment empirically validates that IP Weight metrics reduce dimensions of feature space characterizing IP packets at a level comparable with the principal component analysis technique. The second experiment is an offline evaluation based on 1998 DARPA intrusion detection dataset. In the offline evaluation, we compare our framework with three other unsupervised anomaly detection approaches, namely, plain k-means clustering, univariate outlier detection and multivariate outlier detection. Evaluation results show that the detection framework based on I-means yields the highest detection rate with a low false alarm rate. Specifically, it detects 18 types of attacks out of a total of 19 multiple-connection based attack types. The third experiment is an online evaluation in a live networking environment. The evaluation result not only confirms the detection effectiveness observed with the DARPA dataset, but also shows a good runtime efficiency, with response times falling within few seconds ranges.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

CERT	Computer Emergency Response Team
DARPA	Defense Advanced Research Projects Agency
DDOS	Distributed Denial Of Service
DR	Detection Rate
EM	Expectation Maximization
FNR	False Negative Rate
FPR	False Positive Rate
GMM	Gaussian Mixture Model
GP	Genetic Programming
IDS	Intrusion Detection System
KDD	Knowledge Discovery and Data mining
MAFPR	Maximum Acceptable False Positive Rate
MBF	Mean Best Fitness
MOD	Multivariate Outlier Detection
PCA	Principal Component Analysis
R2L	Remote to Local
ROC	Receiving Operator Characteristic
SR	Success Rate
SSO	System Security Officer
U2R	User to Root
UADR	Unknown Attack Detection Rate
UOD	Univariate Outlier Detection

Acknowledgement

First of all, I would like to express my deepest gratitude to my thesis advisor, Dr. Issa Traore, for his help, encouragement and financial support. He has profoundly influenced me during my Ph.D. studies. It is a pleasure to acknowledge his guidance and support.

I would like to thank Dr. Nikitas J. Dimopoulos, Dr. Kin F. Li, Dr. Bruce Kapron and Dr. Benjamin Yu for being on my thesis committee. Their suggestions and comments are very valuable for me to improve the quality of this thesis.

My association with ISOT (Information Security and Object Technology) research group has been a source of invaluable experience and friendship for me. I thank my colleagues Ahmed Awad E. A., Amany Mohamed, Ahmad Al Mulhem, Yanguo Liu, Suraiya Khan, Bassam Sayed and Hong Ye for their enlightening discussion, generous friendship, friendly assistance, and productive cooperation.

Outside the ISOT research group, I would like thank to Dr. Nanyan Wang, Dr. Wei Li, Dr. Yajun Kou, Dr. Zhiwei Mao, Yali Wang, Yihai Zhang, Wei Yu, Xingming Wang, Haoran Zhang, Xianming Wang, Sabbir Ahmad, Jian Wang, Yongsheng Shi, Mingjie Cai, Fei Huang, Qi Li and many others with names not list here for their kindly help during my stay in Victoria.

I also wish to thank our staff Ms. Vicky Smith, Ms. Moneca Bracken and Ms. Mary-Anne Teo for their enormous support and continuous cooperation.

Finally, my special thanks go to my parents for their love, deep understanding, and strong support on the pursuit of my Ph.D. degree.

Chapter 1

Introduction

Current Internet activities have shown a dramatic increase in the number of computer and network attacks. The Computer Emergency Response Team (CERT) reported, in 2003, 137,529 security incidents, which represent approximately a 40% in the total number of incidents since 1988. Early security mechanisms dedicated exclusively to prevention are highly insufficient. Under current circumstances, password-based authentication and access control mechanisms, which represent the cornerstone of traditional protection systems, can easily be circumvented using widely available exploits. As a result, the concept of intrusion detection was proposed by Anderson to supplement traditional prevention-based security mechanisms [1].

Intrusion detection plays the role of a watchdog for a computing system. When an attack occurs, instead of taking preventive measures, intrusion detection mechanisms usually will only log or report the incident. The implementation of early intrusion detection mechanisms was primarily based on the audit records generated by the host operating system. Audit data were manually inspected by system administrators or security experts in order to detect intrusions. This was expensive, time-consuming, and inaccurate due to the extremely large amount of audit data generated. Moreover, the fact that manual processing and expert knowledge were required ruled out real-time detection. In order to automatically find attacks from a large amount of audit data, the "misuse detection scheme" was developed. In misuse detection, previous attack signatures are

stored and attacks are detected by matching audit events with the stored signatures. Although misuse detection methods can find most known attacks if the signatures are well defined, they are useless for detecting unknown intrusions. Moreover, defining attack signatures is not an easy task at all.

To address the weaknesses of misuse detection, the concept of anomaly detection was formalized in the seminal report of Denning [2]. Denning assumed that security violations could be detected by inspecting abnormal system usage patterns from the audit data. Deviations from normal behavior patterns are flagged systematically as intrusions. The implementations of early anomaly detection techniques were based on self-learning. Knowledge about normal behaviors of subjects was automatically formed through training. Thus, according to whether the learning process is supervised or unsupervised, the anomaly detection schemes are naturally classified into two categories: unsupervised and supervised.

Supervised anomaly detection schemes depend on labeled training datasets, making the intrusion detection process error-prone, costly and time consuming. Any mistake in labeling the training data may lead to decreased performance of the detector. In this dissertation, we study a rule-based supervised anomaly detection approach using genetic programming techniques in order to discuss the limitations of supervised anomaly detection.

Unsupervised anomaly detection schemes allow training based on unlabelled datasets, facilitating online learning and improving detection accuracy. By facilitating online learning, unsupervised approaches provide higher potential to find novel attacks, which are not always included in the training data. By removing the need to label the dataset, unsupervised approaches carry greater potential for detection accuracy. Network intrusions are generally divided into two categories: attacks based on single connection and those based on multiple connections. Multiple-connection based attacks represent a large and important range of network intrusions. A novel online unsupervised anomaly

detection framework for detecting multiple-connection based intrusions is proposed in this dissertation. In the remainder of this section, we briefly state the current research challenges, propose our approaches for solving the corresponding problems and summarize the contributions of the dissertation.

1.1 Problem Statement

Supervised anomaly detection establishes normal profiles of systems or networks by training using a labeled dataset. Unsupervised anomaly detection uses unlabelled or noisy data to identify intrusions. As indicated earlier, the main drawback of supervised anomaly detection is the need to label the training data. Unsupervised anomaly detection allows training based on unlabelled datasets, facilitating online learning and improving detection accuracy.

Clustering analysis is the most widely used learning technique in unsupervised anomaly detection schemes [4]. When applying clustering techniques for intrusion detection, determining the number of clusters is a difficult issue since the occurrence of intrusions is unknown. The general approach and current practice assume that data instances always belong to two categories: normal clusters and intrusive clusters, and that the number of normal data instances largely outnumbers the number of intrusions [3, 4, 5]. However, if all data instances are normal, these assumptions unavoidably lead to a high false alert rate. To improve the performance of unsupervised intrusion detection, we need to accurately determine the optimal number of clusters involved in the data instances.

Clustering is the organization of data patterns into groups or clusters based on some measure of similarity [6]. Several clustering algorithms have been proposed so far, and new ones are appearing in the literature. As indicated by Jain et al., the main reason for the increasing number of clustering algorithms is that "there is no clustering technique that is universally applicable in uncovering the variety of structures present in

multidimensional data sets…each new clustering algorithm performs slightly better than existing ones on a specific distribution of patterns" [7]. Furthermore, they indicate that thorough knowledge of the domain and the data gathering process is essential in achieving higher quality feature extraction, similarity computation, grouping, and data abstraction. As such, precisely characterizing the domain has been one of the biggest challenges faced by unsupervised anomaly detection. The blurred line between the notions of anomaly and intrusion often causes difficulty when accurately discriminating between non-intrusive anomalous activities and intrusive ones. This has a considerable impact on detection effectiveness, despite the quality of the selected clustering algorithm.

Therefore, to achieve an effective and efficient unsupervised anomaly detection, the clustering framework should be based on a feature space that provides a good characterization of anomalous activities, as well as an algorithm that not only clusters data accurately, but also handles large data sets in a timely manner.

1.2 Proposed Approach

In order to address the challenges mentioned above, we developed an effective and efficient clustering framework for online unsupervised anomaly detection of multiple-connection based intrusions. The proposed detection framework consists of a feature extraction technique based on new anomalousness metrics named IP Weights and a new hybrid clustering algorithm named I-means. The I-means algorithm combines mixture resolving, genetic algorithm and k-means algorithm, integrating the capability of determining automatically the optimal number of clusters for a set of data.

Figure 1.1 depicts the general architecture of our framework, which consists of three main stages as follows:

1. Feature analysis: during this phase, IP weights are generated from standard packet information, allowing extraction of salient and useful domain knowledge, as well

as significant reduction of the dimensionality in the feature space.

2. Clustering: the IP weights computed during the previous step are clustered using the I-means algorithm.
3. Cluster Analysis and Intrusion Decision: analyzing the clustering outcomes using heuristics leads to a final classification of corresponding data as either intrusive or non-intrusive data.

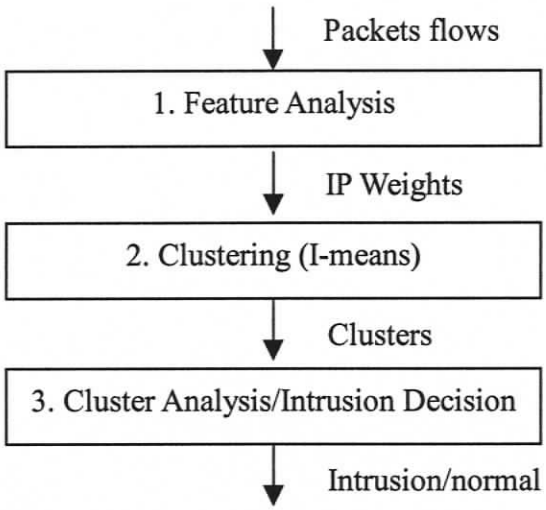


Figure 1.1. *General Architecture of the Detection Framework*

1.3 Contributions

Four important contributions are made in this dissertation.

First, we propose novel anomalousness metrics named IP weights in order to measure the degree of anomalousness of IP packet flows. The IP weights metrics are defined as a set of utility functions. Specifically, four ordinal utility functions are defined to characterize the degree of anomalousness of network activities, and each of them maps several packet features into a single numerical feature, thereby reducing the dimension of the packet feature space.

Secondly, we propose a new clustering algorithm, named I-means. I-means is a

hybrid clustering algorithm that is built around the k-means algorithm. Specifically, I-means extends k-means by addressing inherent limitations which otherwise would represent serious impediments to achieving efficient and effective intrusion detection, the main purpose of the dissertation. Moreover, I-means is an intelligent algorithm, which determines automatically the optimal number of clusters for a set of data.

Thirdly, we apply outlier detection techniques on unsupervised anomaly detection. Specifically, the proposed outlier detection algorithm is based on Gaussian mixture model and it includes two versions: multivariate outlier detection and univariate outlier detection, according to the dimensions of the Gaussian distributions. During an offline evaluation using 1998 DARPA intrusion detection dataset, we compare the I-means framework with approaches based on plain k-means, univariate outlier detection, and multivariate outlier detection.

Fourthly, we propose a novel approach for detecting unknown network intrusions using genetic programming. The detection is based on a rule base generated by the genetic programming technique. Although this approach belongs to the category of supervised anomaly detection, it shows a strong potential to find new attacks on networks.

1.4 Dissertation Organization

The rest of this dissertation is organized as follows.

Chapter 2 summarizes background information on intrusion detection. Specifically, we revisit several classical approaches in the two main categories of intrusion detection techniques, namely misuse detection and anomaly detection.

Chapter 3 introduces a supervised anomaly detection technique based on genetic programming. Genetic programming has a capability to extend search spaces for best solutions by randomly using genetic selection and operation. Intrusive or normal rules are

coded as genetic individuals. New rules are used to find unknown attacks. We evaluate the new rule base with 1999 KDDCUP intrusion detection dataset [8]. Evaluation results show that the approach has a strong potential to find novel network attacks. However, there are several drawbacks underlying the approach due to the supervision required. As such, in order to address limitations related to this aspect, we investigate further unsupervised learning techniques, such as clustering and outlier detection, for computer intrusion detection.

Chapter 4 proposes a new clustering algorithm, named I-means, and an outlier detection algorithm based on Gaussian mixture model for unsupervised anomaly detection. I-means is a hybrid algorithm that combines k-means, mixture resolving, and a new genetic algorithm, estimating automatically the optimal number of clusters for a dataset. We highlight the limitations of k-means and discuss how we address these limitations using an evolutionary approach to mixture resolving. The idea of the proposed outlier detection algorithm is based on a popular statement in pattern recognition, that is Gaussian mixture distribution could approximate any distribution up to arbitrary accuracy, as long as a sufficient number of components are used [9]. Thus the unknown probability density function for any given variables can be expressed as a weighted finite sum of Gaussian with different parameters and mixing proportions [10]. A data pattern not belonging to any existing Gaussian component shall be flagged as an outlier.

As indicated earlier proper domain characterization is an essential prerequisite for performing quality data analysis. Chapter 5 outlines number of empirical observations that serve as basis for characterizing the concept of anomalous network activities. We define an initial set of features and then derive from empirical observations a collection of utility functions, to which we give the name of IP Weight metrics. IP Weight metrics assess the degree of anomalousness of IP packet flows. Specifically, we define in this chapter four IP weight components according to the following dimensions: structure, frequency, randomness, and load. This allows us to convert the defined original packet

feature set into a reduced feature set along four dimensions. Validation results obtained by comparing with principal component analysis (PCA) technique show that IP weight metrics provide the same capability as PCA using a simpler model.

Chapter 6 presents the experimental evaluation of our unsupervised anomaly detection framework. In this chapter, we evaluate the detection effectiveness offline with 1998 DARPA intrusion detection dataset and online in a live networking environment. During the offline evaluation, we analyze 1998 DARPA dataset and extract a multiple-connection based intrusion detection subset. Using the extracted multiple-connection intrusion dataset, we evaluate four different detection frameworks based on I-means clustering, plain k-means clustering, multivariate outlier detection and univariate outlier detection. The comparison of the evaluation results shows that I-means based detection framework has the best performance of all. During the online evaluation, we implemented the I-means based detection framework and deployed it in a live networking environment. The online evaluation result not only confirms the detection effectiveness observed with DARPA dataset, but also shows a good runtime efficiency, with response times falling within few seconds.

Chapter 7 concludes this dissertation and discusses future work.

Chapter 2

Intrusion Detection

2.1 Computer Intrusions

Computer security aims at protecting three fundamental aspects of information systems: namely, confidentiality, integrity and availability. Confidentiality prevents unauthorized disclosure of protected information; integrity prevents unauthorized modification of sensitive information; availability ensures that authorized users can always access system resources. On this ground, computer intrusion can be defined as a set of malicious activities aimed at compromising the integrity, confidentiality or availability of computer systems.

Many efforts have been made to categorize computer intrusions. The early classification proposed by Anderson divides intrusions into three categories: external penetrations, internal penetrations and misfeasances [1]. This classification method is based on the intruder's perspective. Specifically, external penetrators are users not allowed to use the computer; internal penetrators can use the computer but are not authorized to use certain resources or services; misfeasors are authorized to use both resources and services but take advantage of that to abuse their privileges. Another early classification, proposed by Neumann, organizes intrusions into nine classes. These include external misuse, hardware misuse, masquerading, setting up subsequent misuse, bypassing intended controls, active misuse of resources, passive misuse of resources,

misuse resulting from inaction, and use as an indirect aid in committing other misuse [11]. Although Neumann's classification seems to be comprehensive, it is actually complex in practice.

In this regard, more recently, Kendall presented a simpler classification based on the source of misuse. According to Kendall's classification, computer intrusions fall into the following categories: social engineering, implementation bug, abuse of feature, system misconfiguration, and masquerading [12]. In particular, Kendall defined a subset of possible types of computer intrusions and grouped them into four major categories, illustrated as follows,

1. Denial of Service (DoS) – attacks which attempt to crash the victim host by exhausting its computing or memory resources, so it cannot handle legitimate requests.
2. User to Root (U2R) – attacks in which a normal user with login access can gain the privileges of root users by bypassing normal authentications.
3. Remote to Local (R2L) – attacks in which an unauthorized user can gain local access through bypassing normal authentication and executing commands on the victim machine.
4. Probes – attacks scanning computer networks to gather information or find known vulnerabilities, which are exploited for further or future attacks.

Further level of simplification can be achieved by abstracting Kendall's classification into two simple categories: namely, single-connection based intrusions and multiple-connection based intrusions.

A single-connection intrusion, as the name implies, involves only a single of few network connections in compromising a computer system and corresponding services. Many intrusions using social engineering, implementation bugs, or system misconfiguration belong to this group. Using social engineering, an unauthorized user gets authentication information from authorized users and gain access to the system.

Implementation bugs, such as buffer overflow, can be exploited by the attacker to gain unauthorized access. System misconfiguration caused by administrators provides a chance for the attacker to compromise the information system. An example of single-connection based intrusions is the *syslogd* attack characterized in 1998 DARPA intrusion detection dataset [12]. The attack *syslogd* can remotely kill the *syslogd* service on a Solaris 2.5 server by exploiting the server's vulnerabilities. When the *syslogd* service port of Solaris 2.5 server receives an external message, the corresponding DNS will resolve its source IP address. If the source IP address cannot be resolved, the *syslogd* service will be crashed with a segmentation fault [13], and as a result the Solaris 2.5 server will need to be manually restarted by the administrator in order to restore the logging service.

Multiple-connection based intrusions are defined as a set of malicious activities involving a sequence of multiple connections. Many instances of intrusions related to abuse of feature or masquerading fall into this category. By abusing specific features, an attacker may perform legitimate actions taken to the extreme, which may eventually lead to the system failure. For example, an attacker can abuse *telnet* service by generating thousands of normal *telnet* connections to use up resources of process table provided by the computer system.

A popular example of multiple-connection based attacks is the SYN Flood attack, illustrated by Figure 2.1. The idea behind SYN Flood attack consists of abusing the TCP connection establishment process. Normal TCP connection establishment involves a three-way handshake process. Firstly, the client sends request packets with SYN bit to the server. Secondly, the server allocates a TCP control block to prepare to receive the next packet and sends a reply packet with SYN/ACK bits to the client. At this point, the server remains in half-open state. Thirdly, the client sends packets with ACK bit to the server. After the server receives packets from the client, a normal TCP connection is established. SYN Flood attack generates a large number of half-open TCP connections to fill up the

TCP control block and then exhausts the available resources. As such, the target host becomes unable to accept any more incoming TCP connections. This creates a denial of service for legitimate requests.

In Figure 2.1, A is the attack host, which sends large number of SYN packets to host B, the victim, by using the spoofed source IP address of host C. SYN/ACK packets replied by B are directed to C, which naturally will drop them since it has not initiated the connection request. As a result, B will not receive ACK packets from A and will still remain in a half-open state. This will quickly run out the TCP control blocks of host B, making it unable to accept any more incoming TCP connections.

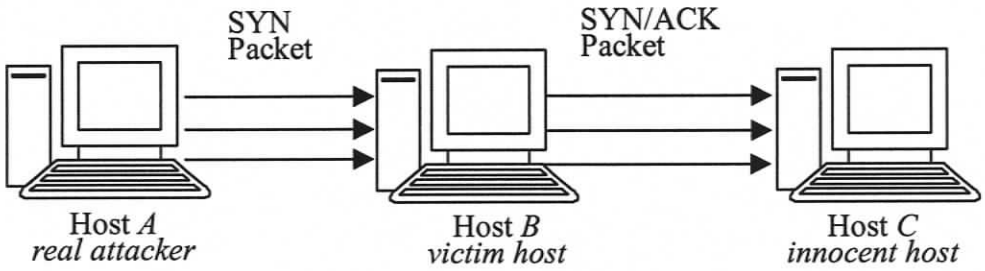


Figure 2.1. *SYN Flood Attack Scenario*

2.2 A Brief Survey of Intrusion Detection Techniques

The basic principle of intrusion detection is based on the assumption that intrusive activities are noticeably different from normal ones and thus are detectable [2]. Many intrusion detection approaches have been suggested in the literature since the seminal report of Anderson. Traditionally, these approaches are classified into three categories: misuse detection, anomaly detection and specification-based detection. In the remainder of this section, we discuss these different approaches in detail and summarize some representative examples in each category.

2.2.1 Misuse Detection

The study of misuse detection began with Anderson's report in 1980. Intrusions are detected by matching actual behavior recorded in audit trails with known suspicious patterns. While misuse detection is fully effective in uncovering known attacks, it is useless when faced with unknown or novel forms of attacks for which the signatures are not yet available. Moreover, for known attacks, defining a signature that encompasses all possible variations of the attack is difficult. Any mistakes in the definition of these signatures will increase the false alarm rate and decrease the effectiveness of the detection technique.

Figure 2.2 illustrates a typical misuse detection model. The model consists of four components: namely, data collection, system profile, misuse detection and response. Data are collected from one or many data sources including audit trails, network traffic, system call trace, etc. Collected data are transferred into a format that is understandable by the other components of the system. The system profile is used to characterize normal and abnormal behaviors. The profiles characterize what a normal subject behavior should be and what operations the subjects typically would perform or not on the objects. The profiles are matched with actual system activities, and reported as intrusions in case of deviations.

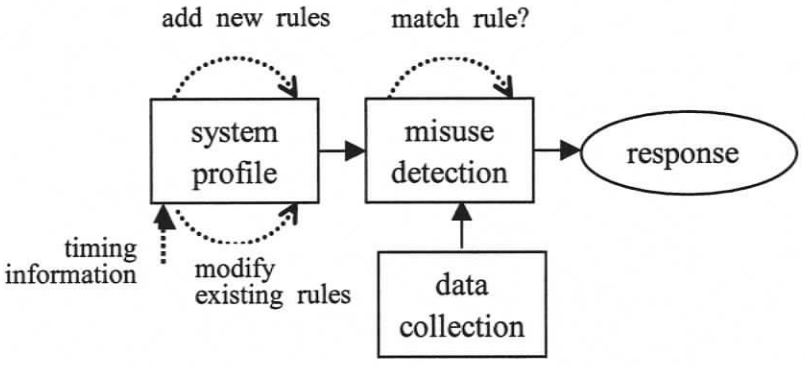


Figure 2.2. *A Typical Misuse Detection System*

Three classes of techniques are commonly used to implement misuse detection, namely rule-based techniques, state-based techniques, and data mining. We discuss in detail these techniques and sample systems in the rest of this section.

2.2.1.1 Rule-based Techniques

Rule-based expert system is one of the earliest techniques used for misuse detection. Expert systems encode intrusive scenarios as a set of rules, which are matched against audit or network traffic data. Any deviation in the rule matching process is reported as an intrusion. Examples of rule-based systems include MIDAS (Multics Intrusion Detection and Alerting System) [14], IDES (Intrusion Detection Expert System) [15], and NIDES (Next-generation Intrusion Detection Expert System) [16, 17].

2.2.1.1.1 MIDAS

MIDAS was designed and developed by the National Computer Security Center (NCSC) to monitor intrusions for NCSC's networked mainframe, Dockmaster. It uses and analyzes audit log data by combining the expert system technology with statistical analysis. MIDAS uses the Production Based Expert System Toolset (P-BEST) [18] in discriminating and implementing the rule base, which is written in LISP language.

The structure of the rules in the P-BEST rule base includes two layers. The first (lower) layer is used to match certain types of events such as number of user logins, and then fire new events by setting up a particular threshold of suspicion. Rules in the second (higher) layer process these suspicions and decide whether the system should raise an alert or not.

Figure 2.3 illustrates an example of MIDAS rule. The rule defines an intrusion scenario involving some unusual login time. It determines whether the time when the user logins is outside normal hours or not. The rule also illustrates that an unusual behavior does not necessarily stand for an intrusion.

```
defrule unusual_login_time states
    if there exists a login_entry
        such that user is userid and
        time_stamp is login_time and
        (unusual_login_time userid login_time)
    then
        remember a user_login_anomaly
            such that user is userid and
            time_stamp is login_time
```

Figure 2.3. *Unusual Login Time Rule*

2.2.1.1.2 IDES

IDES is one of the early implementation of the basic ideas and notions underlying intrusion detection. The IDES model results from Denning's seminal paper [2], which provides a mathematical codification of intrusion detection mechanisms. The model is based on the assumption that normal interactions between subjects (e.g. users) and objects (e.g. files, programs, or devices) can be characterized, and also that users always behave in a consistent manner when they perform operations on the computer system. These usages can be characterized by computing various statistics, and correlated with established profiles of normal behaviors. New audit records are verified by matching known profiles for both subjects and their corresponding groups. Deviations are then flagged as intrusions. To improve the detection rate, IDES monitors the subject depending on whether the activity happens on an *on* or *off* day since user activities on different day types are usually different. For example, activities for normal users on a working day may be abnormal on an off-working day.

IDES uses P-BEST to describe its rule base, consisting of two types of rules: generic rules and specific rules. Generic rules can be used for different target systems and specific rules are strictly dependent on the operating system and the corresponding

implementation. IDES architecture consists of three main components, namely audit database, profiles database and the system security officer (SSO) user interface.

2.2.1.1.3 NIDES

NIDES, the successor of IDES, is the acronym for the Next-generation Intrusion Detection Expert System. NIDES is a hybrid intrusion detection system consisting of a signature-based expert system component as well as a detection component based on statistical approaches. The expert system improves the old IDES version by encoding more known intrusion scenarios and updating the P-BEST version used. The detection component based on statistical approaches is based on anomaly detection. In these approaches, over 30 criteria are used to establish normal user profiles, including CPU or I/O usage, command used, local network activity, system errors, etc.

The NIDES system is highly modularized, with well-defined interfaces between components. Compared with the IDES, NIDES has higher detection rate since it includes two complementary detection components: intrusions missed by one component maybe caught by the other one.

2.2.1.1.4 Limitations of Rule-based Techniques

Using rule-based techniques for misuse detection, quite often the burden of extending the rule-base as new intrusion scenarios are discovered falls on the shoulder of the security officer. Moreover, developing intrusion scenarios is not an easy task and requires a certain level of expertise and security insight and awareness. In addition, determining the relations between rules is difficult. When many related rules are included in an expert system, correctness of rules is difficult to verify due to the interactions among these rules. Consequently, in practical settings, most of the rule bases are outdated and quickly become obsolete.

2.2.1.2 State-based Techniques

State-based techniques detect known intrusions by using expressions of the system state and state transitions. State models simplify the specification of patterns for known attacks and can be used to describe attack scenarios easier than rule-based languages such as P-BEST. In state-based techniques, activities contributing to intrusion scenarios are defined as transitions between system states, and thus intrusion scenarios are defined in the form of state transition diagrams. Figure 2.4 depicts a generic state diagram; a node represents a system state, and an arc stands for an action.

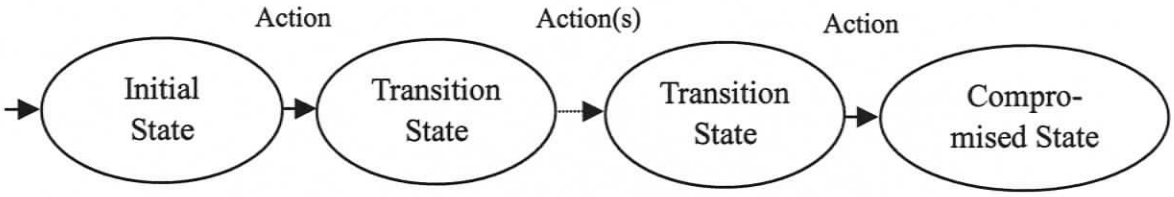


Figure 2.4. *Generic State Transition Diagram*

The state of the system is a function of users or processes. Intrusion scenarios defined by the state transition diagram include three types of states, namely initial state, transition state and compromised state. An initial state refers to the beginning of the attack, while a compromised state stands for the successful completion of the attack. Transition states correspond to the successive states occurring between an initial state and a compromised state. An intrusion occurs if and only if a compromised state is finally reached. In the following section, we present two examples of state-based techniques, namely the state transition analysis tool proposed by Ilgun and colleagues [19, 20], and the IDIOT system based on colored petri-nets proposed by Kumar et al. [21, 22, 23, 24].

2.2.1.2.1 UNIX State Transition Analysis Tool (USTAT)

The UNIX State Transition Analysis Tool (USTAT) is based on the assumption that all attackers start from an initial state where they possess limited authorization to access a target system, and then after completing some operations on the target system, they acquire some previously unauthorized capabilities. USTAT is a mature prototype implementation of the state transition analysis technique for intrusion detection. It monitors the system state transition from safe to unsafe by representing all known vulnerabilities or intrusion scenarios in the form of a state transition diagram.

In USTAT, over 200 audit events are represented by ten USTAT actions, such as read(file_var), modify_owner(file_var) where the parameter file_var stands for the name of certain files. Known attacks are modeled as a sequence of state transitions that lead from an initial limited authorization state to a final compromised state. An inference engine in USTAT maintains a state transition table and determines whether the current action will cause a state transition to its successor state by matching the next state with the state transition table. Once the next new state is matched to the final state of the transition table, the intrusion alarm is raised.

Two additional assumptions underly the state transition analysis technique. First, intrusive behavior must have a visible effect on the system state, and second, this visible effect must be recognizable without using any external knowledge outside the system itself. Not all possible intrusions, however, satisfy these assumptions. For instance, passive listening of broadcast network traffic violates these assumptions. As a result, this technique carries the potential of missing a lot of intrusions that are either not recorded by the audit trail or not represented by state transition diagrams.

2.2.1.2.2 Colored Petri-nets

IDIOT, the acronym of Intrusion Detection In Our Time, is a state-based misuse detection system, which uses pattern matching techniques based on the colored petri-nets (CPN) model. Intrusion scenarios are encoded into patterns in IDIOT, and incoming events are

verified by matching them against these patterns. In the CPN model used in the implementation of IDIOT, a *guard* represents an intrusive signature context and the *vertices* represent system states. The selected CPN model is referred to as colored petri automata (CPA). The CPA defines a strict declarative specification of intrusions and specifies what patterns need to be matched, instead of how to match them.

Figure 2.5 illustrates a simple example of CPA describing the following intrusion scenario: *if the number of unsuccessful login attempts exceeds four within one minute, report an intrusion.* The combination of arrows and vertical bar stands for a transition between system states. For example, the transition from states S1 to S2 occurs when there is a token in S1; this stands for an unsuccessful login attempt. The time of first unsuccessful login attempt is saved in the token variable T1. The transition from S4 to S5 happens if there is a token in S4. The time difference between this and the first unsuccessful login attempt should be more than one minute, otherwise the system state is transferred to the final state S5, in which an alarm will be generated.

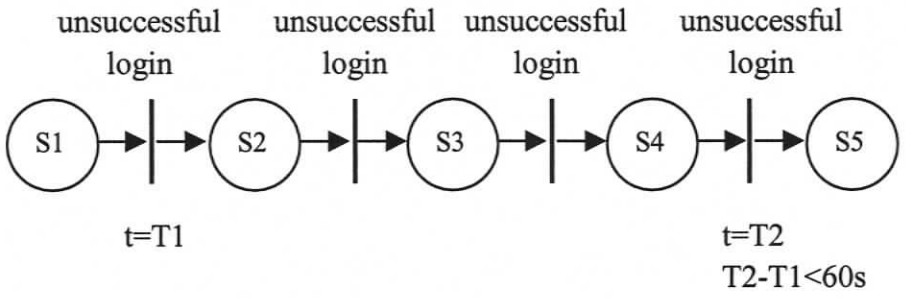


Figure 2.5. *Example of CPA Illustrating Four Failed Login Attempts within One Minute*

It is suggested that this technique has several advantages. First, since the intrusion signatures are written in a system independent script, they can be exchanged across different operating systems and different audit logs. Second, the IDIOT system achieves an excellent real-time performance; only 5-6% CPU overhead was reported when it scanned for 100 different patterns. Third, multiple event sessions can be processed

independently and then corresponding detection results are analyzed together to make the final decision.

However, the main limitation of this technique is that it can only detect known vulnerabilities. Also, translating known intrusions into patterns is not always easy. In addition, strict declarative expression about the intrusive patterns leads to a potential problem, that is, sophisticated attackers can easily bypass the detection system by changing their intrusion strategies.

2.2.1.3 Techniques based on Data Mining

In recent years, data mining techniques have been applied to network and host audit data for building misuse detection models [25, 26, 27]. In this case, intrusion detection is considered as a data analysis process, in which data mining techniques are used to automatically discover and model features of user's normal or intrusive behaviors. It is reported that three types of algorithms are particularly useful for mining audit data, namely classification, link analysis and sequence analysis.

Classification algorithms, such as decision tree, generate classifiers by learning based on a sufficient amount of normal or abnormal audit data. New audit data are labeled as either normal or abnormal according to the classifier. Link analysis determines the relation between fields in the audit database records and normal profiles are usually derived from these relations. Sequence analysis is used to find sequential patterns in audit data, and embed these patterns into intrusion detection models.

Data mining based techniques performed very well in detecting known attacks during the 1998 DARPA intrusion detection evaluation. However, just like other misuse detection techniques, they operated fairly poorly on detecting new attacks. In addition, applying data mining techniques requires labeling the training dataset, which makes the detection process error-prone, costly and time consuming.

2.2.2 Anomaly Detection

Different from misuse detection, anomaly detection is dedicated to establishing normal activity profiles for the system. It is based on the assumption that all intrusive activities are necessarily anomalous. Anomaly detection studies start by forming an opinion on what the normal attributes for the observed objects are, and then decide what kinds of activities should be flagged as intrusions and how to make such particular decisions.

A typical anomaly detection model is illustrated in Figure 2.6. It consists of four components, namely data collection, normal system profile, anomaly detection and response. Normal user activities or traffic data are obtained and saved by the data collection component. Specific modeling techniques are used to create normal system profiles. The anomaly detection component decides how far the current activities deviate from the normal system profiles and what percentage of these activities should be flagged as abnormal. Finally, the response component reports the intrusion and possibly corresponding timing information.

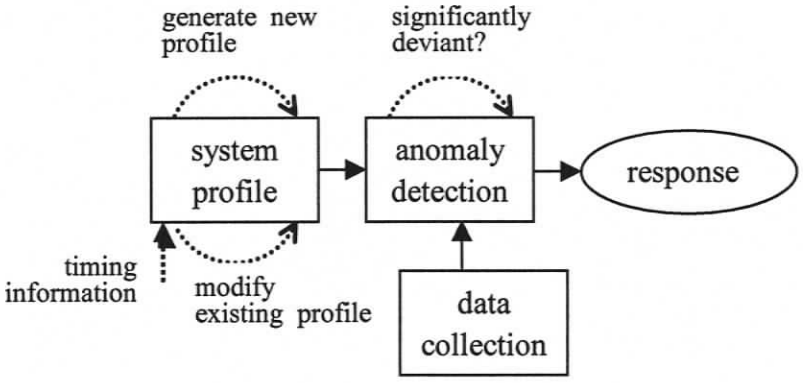


Figure 2.6. *A Typical Anomaly Detection System*

The primary advantage of anomaly detection is its capability to find novel attacks; as such it addresses the biggest limitation of misuse detection. However, due to the

assumptions underlying anomaly detection mechanisms, their false alarm rates are in general very high. Specifically, the main reasons for this limitation include the following:

1. The user's normal behavior model is based on data collected over a period of normal operations; intrusive activities missed during this period are likely to be considered as normal behaviors.
2. Anomaly detection techniques can hardly detect stealthy attacks because these kinds of attacks are usually hidden in large number of instances of normal behaviors. Moreover, the types of parameters used as inputs of normal models are usually decided by security experts. Any mistake occurring during the process of defining these parameters will increase the false alarm rate and decrease the effectiveness of the anomaly detection system.

As a result, the design of the detection method and the selection of the system or network features to be monitored are two of the main open issues in anomaly detection.

Many anomaly detection techniques have been proposed in the literature. These range from basic statistical models to artificial intelligence, and biological models based on human immune systems. Although it is difficult to classify these techniques, we divide them in this dissertation into four categories based on previous surveys on anomaly detection systems [28, 29, 30, 31, 32, 33, 34]. These include basic statistical models, rule-based models, learning models and biological models.

2.2.2.1 Basic Statistical Models

Denning proposed in a seminal paper the earliest theoretical characterization of anomaly detection. Her detection framework, which is based on basic statistical analysis, consists of four models, namely operational model, mean and standard deviation model, multivariate model and markov process model [2]. Examples of intrusion detection systems using statistical models are Haystack [35] and NIDES. We give an overview of

these systems in the rest of this section.

2.2.2.1.1 Haystack

The haystack system was designed and implemented for the detection of intrusions in a multi-user Air Force computer system [35]. Statistical techniques are used to detect anomalous activities. A set of features such as amount of I/O, CPU utilization, number of file accesses are observed and then the normal range of values for these features are defined. Activities falling outside these ranges are reported as intrusions.

Haystack is one of the earliest anomaly detection systems based on statistical models. It uses a very simple statistical model, in which each feature is assigned a weight by the SSO. The main weakness of Haystack is that no test is conducted to verify if the weight value is sensitive to intrusion patterns and furthermore no rationale explanation is given as to how the weight value is assigned. Moreover, the underlying assumption that features are statistically independent is usually not satisfied in practice.

2.2.2.1.2 NIDES

As indicated in the earlier section, NIDES includes a statistical anomaly detector as well. The audit information collected consist of user names, names of files accessed, elapsed user CPU time, total number of files opened, number of pages read from secondary storage, identities of machines onto which user has logged, etc. Statistics are computed from the collected audit information. NIDES stores only statistics related to frequencies, means, variances or covariance of measures instead of the total audit data. Given m measures in NIDES, if the point in the m-space of measures is far away from the expected value, the corresponding event represented by the point is considered anomalous.

2.2.2.2 Rule based Techniques

Several anomaly detection models proposed in the literature were implemented using rule-based techniques. We present in this section four examples of such models, namely Wisdom & Sense, NSM, TIM and NADIR.

2.2.2.2.1 Wisdom & Sense (W&S)

W&S is a unique approach to anomaly detection, which consists, as the name indicates, of two components: wisdom and sense [36]. The wisdom component consists of a set of rules describing normal behaviors of the system based on historical audit data. About 10,000 records per user are read from the file in order to create the rules. The sense component is an expert system based on the previous rules, which verifies whether subsequent audit data violate the rule base or not. The SSO is then alerted when the sense detects some anomalous behavior. In the implementation of W&S, the Figure of Merit (FOM) metric is used to compute an anomalousness score for each thread. The FOMs values for several events are summed and compared against a threshold. When the compared value is above the threshold, corresponding events are flagged as anomalies by the system.

2.2.2.2.2 Network Security Monitor (NSM)

Network Security Monitor (NSM) is a rule-based intrusion detection system targeting anomalous network activities. It is the first intrusion detection system using network traffic directly as the primary source of data [37]. All network traffic passing through a broadcast LAN are observed by NSM. Since network traffic is based on standard protocols such as TCP/IP, telnet, SMTP, etc., the traffic information on heterogeneous hosts is based on a consistent format.

Network traffic profiles are created by generating connection vectors derived from the network data. Host vectors and connection vectors are the main input to the expert system in NSM. The expert system illustrates the data-path with which systems are expected to

communicate and the specifications of higher-level application layer protocols. Intrusions are decided according to analysis results of the expert system. The final result reported to the SSO consists of a connection vector and corresponding suspicion level. The suspicion level measures the likelihood that a particular connection represents some intrusive behaviors.

2.2.2.2.3 Time-based Inductive Machine (TIM)

TIM models user's normal behavior patterns by dynamically generating activity rules using inductive generalization [38]. TIM discovers normal behavior patterns in a sequence of events instead of a single event. Each rule represents a model that can predict the next event from a given sequence of events. If user's behavior matches with previous event sequences and the actual next event is not included in the predicted event sets, the user's behavior is considered as intrusive. The major limitation of TIM is that the rules only model adjacent events. When many applications are executed at the same time, events occurring in one application may be interleaved with events occurring in other applications, in which case the rules generated by TIM may not accurately identify user's normal behaviors. Moreover, events that cannot match any previous event sequences in the training dataset always fire an intrusion alarm, which most likely may correspond to some false alert.

2.2.2.2.4 NADIR

NADIR (Network Anomaly Detection and Intrusion Reporter) was developed at Los Alamos National Laboratory to monitor their internal computer networks [39, 40]. It collects audit information from three different kinds of service nodes, namely network security controller, common file system and security assurance machine. Network security controller provides user authentication and access control; common file system stores the data, and security assurance machine records attempts to degrade the security level of the data. The audit information collected for each event include a unique user ID associated with the subject, the date and time of the event, an accounting parameter, the

error code, a flag indicating whether the corresponding action is successful or not, and the description of the event.

Individual user profile is computed on a weekly basis from the audit database. The user profiles are compared with a set of expert system rules in order to detect possible deviations in users' behaviors. The expert rules are mainly derived from the security policies and the statistical analysis of previous audit logs. Each rule is assigned a level-of-interest. If the corresponding sum of level-of-interest for a user is too big, the intrusion alarm is raised.

2.2.2.3 Biological Models

Several anomaly detection models inspired from biological principles have been proposed in the literature. One of the earliest works in this area was authored by Forrest et al. Forrest et al. suggested and studied the analogy between human immune system's capability of distinguishing *self* from *nonself* and intrusion detection system [41, 42, 43, 44,45]. In the human systems, T cells are created in the thymus and then a censoring process happens. If T cells bind with proteins or peptides (sub-units of proteins), then they will be transferred because they bind with themselves (*self*). Some T cells that do not bind with proteins are released to monitor the foreign material (*nonself*) of the body. A hypothesis is that those T cells will bind with foreign materials and then remove them from human systems.

When applying the *self-nonself* technique to intrusion detection, behaviors of the system are viewed as a string and then this string is divided into sub-strings of equal length k. *Self* contains some of the possible 2^k strings. The rest of the sub-strings $\{2^k - Self\}$ or the complement of *self* is called *nonself*. *Nonself* contains some detector strings, which are of length k and does not exist in the collection of *self*. In the practical implementation, current behaviors of the system are first divided into sub-strings of

length k and then are periodically compared with detector strings in *nonself*. A match indicates a possible intrusion or an unexpected change.

Nonself can be generated through various ways. A typical implementation for this is to randomly generate strings of length k, and then remove any string that is part of the collection of *self*. Consequently, a sub-string that is not in *self* is used as a detector and is added to *nonself*. The size of *nonself* determines the effectiveness of detecting anomalous behaviors: the larger the size of *nonself*, the more likely intrusion will be detected.

Forrest et al. point out that perfect matching between strings is rare. Thus, they define a partial matching rule: given any two strings p and q, expression $match(p, q)$ is true if p and q match in at least r contiguous locations. For example, considering two strings x and y, where x = CEFGOPRTSY and y = BABSOPRTTZ, with $r > 4$, $match(x, y)$ is false, but with $r < 5$, $match(x, y)$ is true. Since perfect matching is rare, the partial matching rule increases the likelihood of detecting anomalous behaviors.

Another interesting biological method for intrusion detection proposed by Yu et al. is based on the DNA sequence [46]. Yu et al. define DNA sequences for a computer system based on the knowledge that the DNA characterizes the make up of human body and that any anomaly in tissues can be reflected in a particular DNA sequence. Any change in the behavior patterns of the computer system may be traced to the change of DNA sequences that can be either normal or abnormal. A standard back-propagation neural network was used to train normal DNA sequences for network traffic, and then a UDP Flood attack was successfully detected based on the DNA sequence for normal network traffic. Although encouraging, this preliminary result needs to be extended in order to define a more complete DNA scheme for computing systems.

More recently, Ahmed and Traore proposed the use of behavioral biometric for detecting masqueraders [47]. Biometrics have so far been used only for authentication not for intrusion detection. The framework proposed by Ahmed and Traore, uses mouse and keystroke dynamics, both of which are behavioral biometrics that can be collected

passively and checked dynamically. The results reported in [47] seem encouraging.

2.2.2.4 Learning Models

Learning models incorporate learning capabilities in intrusion detection process, using artificial leaning techniques. In recent years, learning techniques have been widely used in anomaly detection since the self-learning techniques can automatically form an opinion of what the subject's normal behavior is. According to whether they are based on supervised or unsupervised learning techniques, we divide the anomaly detection schemes into two categories: unsupervised and supervised. Supervised anomaly detection establishes the normal profiles of systems or networks through training based on labeled datasets. In contrast, unsupervised anomaly detection attempts to detect intrusions without using any prior knowledge of attacks or normal instances. The main drawback of supervised anomaly detection is the need of labeling the training data, which makes the process error-prone, costly and time consuming, and difficult to find new attacks. Unsupervised anomaly detection addresses these issues by allowing training based on unlabelled datasets and thus facilitating online learning and improving detection accuracy. Unsupervised anomaly detection is relatively new compared with supervised anomaly detection schemes. We discuss both approaches in the rest of this section.

2.2.2.4.1 Supervised Anomaly Detection

In order to illustrate supervised anomaly detection techniques, we summarize in the following section two of the proposed approaches, namely neural network and evolutionary approaches based on genetic algorithms.

2.2.2.4.1.1 Using Neural Network

Hyperview is an early attempt for intrusion detection using neural network, which consists of two components [48]. The first component is an expert system, which monitors audit trails for known intrusion signatures. The second component uses neural

network to learn user behaviors; it then fires an alarm when there is a deviation between current behaviors and learnt behaviors. The use of neural network in Hyperview assumes that the audit data consist of multivariate time series, since user behavior exhibits a dynamic process executing an ordered series of events. In the implementation of Hyperview, a recurrent network is used for training and the time series are mapped with the neural network. The output is then selected as the next input to the system until the training process is terminated.

Another example of intrusion detection system using neural network is proposed by Ghosh [49]. A back-propagation network is used for training. In the practical implementation, Ghosh uses the program internal state and the input program as the input of back-propagation networks. The experimental results show that this approach increases the performance of anomaly detection by randomly generating data as anomalous input. However, choosing input parameters is not easy. Any mistake in input data will increase the false alarm rate. In addition, how to initialize the weights remains unclear.

2.2.2.4.1.2 Using Genetic Algorithm

The genetic algorithm is an iterative search technique based on the theory of Darwinian evolution applied to mathematical models. It operates on a population of individuals, in which each individual is a potential solution to a given problem. After the initial population is randomly generated, the algorithm circularly evolves the population through three basic operators: selection operator, crossover operator and mutation operator until the best individual is obtained.

Genetic Algorithm as an Alternative Tool for Security Audit Trails Analysis (GASSATA) is one of the earliest attempts for using genetic algorithm (GA) for intrusion detection [50]. In GASSATA, Me defines a *n*-dimensional hypothesis vector H, where $H_i = 1$ if attack i is taking place according to the hypothesis, otherwise $H_i = 0$. As a result, the intrusion detection becomes the problem of finding the H vector that maximizes the product $W \times H$, subject to the constraint $(AE \times H)_i \leq O_i$; where W is a

n-dimensional weight vector, AE is an attacks-events matrix, and O is the observed n-dimensional audit trail vector. Each individual in the population corresponds to a particular H vector. The fitness function is defined as follows,

$$Fitness = \sum_{i=1}^{n} W_i \times I_i$$

Where I_i stands for an individual. The experimental evaluation showed that GA for intrusion detection had a low false alarm rate and the likelihood of detecting misuse behaviors was 0.996. However, the major limitation of GASSATA is that it cannot locate attacks precisely.

Another attempt using GA for intrusion detection is made by Chittur [51]. Chittur uses a certainty formula C_i to classify whether a record x is an intrusive behavior or a normal behavior. C_i is defined as follows,

$$C_i(x) = \sum_{j=1}^{n} (\Re_{ij} \times x_j)$$

Where $\Re_{ij}$ is the Ephemeral Random Constant-based coefficient for attribute x_j and n is the number of attributes. A threshold value for C_i is established and any certainty value exceeding this threshold value is classified as a malicious attack.

The following fitness function is used:

$$F(\delta_i) = \frac{\alpha}{A} - \frac{\beta}{B}$$

Where δ_i refers to the individual; α means the number of correctly detected attacks and A stands for the number of total attacks; β refers to the number of false positives and B is the total number of normal connections. The range of the fitness value is from –1 to 1. A high detection rate $\frac{\alpha}{A}$ and a low false positive rate $\frac{\beta}{B}$ will yield a high fitness value for an individual.

The experimental results show that GA successfully generates an accurate empirical behavior model from the training data. However, the major limitation of this approach is that an improper threshold value might easily lead to a high false alarm rate in detecting new attacks.

More efforts using GA for intrusion detection are made in [52, 53, 54]. Gómez et al. propose a linear representation scheme for evolving fuzzy rules using the concept of complete binary tree structures. GA is used to generate genetic operators for producing useful and minimal structure modifications to the fuzzy expression tree represented by chromosomes. However, the training process in this approach is computationally very expensive and time consuming. Bridges and Vaughn employ GA to tune the fuzzy membership functions and select an appropriate set of features in their intelligent intrusion detection system. Balajinath and Raghavan use GA to learn individual user behaviors. Active user behaviors are predicted based on past observed user behaviors, which can be used for intrusion detection. The training process for both approaches is, however, time consuming.

Crosbie and Spafford propose a framework combining genetic programming (GP) and agent technology for detecting anomalous behaviors in a system [55]. Autonomous agents are used to detect intrusions using log data of network connections. Each autonomous agent is used to monitor a particular network parameter. Autonomous agents whose predication is correct are assigned a higher weight value to decide whether a session is intrusive or not. There are a number of advantages to having many small agents, instead of a single large one. However, communications among so many agents lower the detection efficiency. Moreover, an improper primitive for each agent will make the training time long.

2.2.2.4.2 Unsupervised Anomaly Detection

Several unsupervised learning techniques for anomaly detection have been proposed in recent years. Clustering and outlier detection based techniques are among the most

popular approaches suggested so far.

Lankewicz et al. author one of the earliest works in the literature [56]. Specifically, they propose a model applying pattern recognition to the characterization of individual users. The profiles computed with this model can be used for anomaly detection, particularly at the host level. A key aspect of their approach consists of using a variant of k-nearest-neighbor clustering algorithm for data compression and cluster discovery. According to them, this would facilitate real-time detection. Unfortunately, even though the main goals sought by the authors consist of improving detection efficiency and achieving real-time detection, it is not clear whether their model can achieve these goals, since no evaluation data are actually provided.

Staniford et al. propose a *portscan* detector, which is particularly effective against stealthy scans [57]. They propose an architecture that combines an anomaly sensor with a correlator. Firstly, the anomaly sensor computes an anomaly score for packets. Secondly, based on the anomaly scores the correlator uses simulated annealing to cluster packets into activities that are similar. This architecture is similar to ours, particularly since our IP weight metrics play the same role as their anomaly score. Our model, however, is broader and targets a wider range of network attacks, while their model focuses only on *portscan*. As indicated by the authors, formal experiments to measure the detection performance of their model have yet to be completed.

Another early attempt to unsupervised anomaly detection is presented by Eskin [3]. A mixture probability distribution model D is proposed to identify anomalous behaviors. Eskin assumes that each element in the model falls into one of two cases: elements are either anomalous with small probability (λ) or are normal with majority probability $(1-\lambda)$. D is thus composed by normal distribution M and alternate distribution A according to the probability λ, giving $D = (1-\lambda) \times M + \lambda \times A$. EM algorithm is used to estimate D. This method shows a strong ability to detect intrusive system calls when the

amount of anomalous system calls consists of less than 5% of the total data. However, it has high false alarm rate when the number of anomalies is much larger than the number of normal instances in the dataset.

Portnoy et al. extended the unsupervised anomaly detection approach suggested by Eskin [5]. They use a simple distance-based metrics and a simple variant of single-linkage to cluster the unlabelled data. In order to identify intrusions, they make the following assumptions:

1. The number of normal data instances largely outnumbers the number of intrusions.
2. The data instances include two clusters: intrusive cluster and normal cluster.

The direct consequence of these two assumptions is that normal instances are expected to form larger clusters compared to intrusive ones. And thus, large clusters will be flagged as normal, while small clusters will be considered intrusive. After testing their methodology using 1999 KDDCUP intrusion detection dataset, they report an average detection rate of 40-55% with a 1.3-2.3% false positive rate.

Three clustering algorithms are later suggested to improve the accuracy of clustering-based detection in [4]. The first algorithm calculates the density of points near the data point being analyzed. If this point does not belong to a dense region, it will be considered an anomaly. The second algorithm calculates the sum of all the distances to the k-nearest neighbors. If the sum is greater than a threshold, the data point is flagged as an anomaly. The third algorithm is based on the support vector machine paradigm [58]. It solves convex optimization problems to classify the lower regions of support in a probabilistic distribution. Two different datasets are used to evaluate these algorithms and the evaluation result shows that the clustering algorithms have strong ability to detect anomalies. However, it is claimed that more experiments need to be done with different datasets, although the evaluation results are promising.

Yamanishi et al. propose an online unsupervised outlier detection approach for

network intrusion detection [59]. Their model uses a finite probabilistic mixture model to represent the network data. An online discounting learning algorithm is used to learn the probabilistic model. Based on the learned model, a score is generated for the data, with a high score indicating an outlier. Intrusion detection, in this case, consists of identifying outliers and reporting them as intrusions. Evaluation of the model is performed using a subset of 1999 KDDCUP intrusion detection dataset and reveals detection ratios ranging between 55% to 82%.

There are several other works that attempt to achieve unsupervised anomaly detection using techniques other than clustering or outlier detection. Examples of those include works by Sekar et al [60] and Shyu et al. [61]. Sekar et al. define state machine specifications of network protocols, augmented with statistical information that needed to be maintained to detect anomaly. They claim that the learning component in their method is robust enough to operate without human supervision and thus, can be considered as unsupervised anomaly detection. However, they do not provide any experimental evaluation of these claims in their paper. Shyu et al. propose a novel anomaly detection scheme based on principle component classifier. They evaluate their algorithm using 1999 KDDCUP intrusion detection data. The experimental result shows that their approach has better performance than other approaches, such as LOF approach [62] and Canberra metrics [63]. Based on this robust result, they claim that their method would also work with unlabelled training data. However, they have not conducted any experimental evaluation to support this assertion.

2.2.3 Specification-based Detection

Specification-based detection approaches premise that every well-behaved system execution shall conform to the specification of its intended behavior. An execution sequence performed by the subject that violates the specification of programs will be

considered as an attack. This approach, in theory, can detect unseen attacks that may be exploited in the future [64]. However, specifying the behavior of large number of privileged programs running in real operating environments is a daunting and difficult task. Even though inductive logic programming is used to automatically synthesize specifications of programs in [65], rigorous validation of so many programs' specifications is still an open issue. Moreover, the formal methods community has been struggling with the same issue for decades without significant success. Hence, in spite of its appealing principles, specification-based detection still remains in infancy.

Chapter 3

Case Study: Supervised Intrusion Detection Using Genetic Programming

How to detect unknown attacks is one of the most important challenges faced by current intrusion detection schemes. To address this issue, we propose in this chapter a new supervised intrusion detection scheme using genetic programming (GP) [66]. Intrusions are decided based on specific rules. GP algorithm is used to evolve these rules. New rules are generated using genetic operators. Specifically, four genetic operators are used in our detection framework, namely reproduction, mutation, crossover and dropping condition operators. We use the new rules to detect new and known attacks. Our detection scheme is evaluated with 1999 KDDCUP intrusion detection dataset and the result shows a potential capability to find new intrusions. In the remainder of this chapter, we briefly summarize the GP algorithm, illustrate the proposed approach, discuss the evaluation result, and make some concluding remarks.

3.1 Overview of GP Algorithm

Genetic Programming is an extension of Genetic Algorithm (GA) [66]. It is a general search method that uses analogies from natural selection and evolution. The main difference between GP and GA is the solution encoding method. GA encodes potential

solutions for a specific problem as a simple population of fixed-length binary strings named chromosomes and then applies reproduction and recombination operators to these chromosomes to create new chromosomes. GP encodes multi potential solutions for specific problems as a population of programs or functions. The programs are represented as parse trees, which are usually composed of internal nodes and leaf nodes. Internal nodes are called primitive functions, and leaf nodes are called terminals. Terminals are viewed as inputs to the specific problem. They might include the independent variables and the set of constants. Primitive functions are combined with the terminals or simpler function calls in order to form more complex function calls.

GP randomly generates the initial population of solutions. A population is manipulated using various genetic operators to produce new population. The whole process of evolving from one population to the next population is called a generation. The main steps of the GP algorithm can be described as follows:

1. Create a random population of programs, or rules and use the symbolic expressions to represent the initial population.
2. Evaluate each program or rule using the fitness value, which is calculated by a pre-defined fitness function; the fitness function measures the capability of the rule or program to solve specific problem.
3. Use reproduction operator to copy existing programs into the new generation.
4. Generate the new population with crossover, mutation or other operators from a randomly chosen set of parents.
5. Repeat steps 2 for an evolving new population until a pre-defined termination criterion has been satisfied, or a fixed number of generations have been completed.

Usually, the solution to the problem is the new population with the best fitness value within all generations.

3.2 Proposed Approach

The basic idea of the proposed intrusion detection approach is based on a belief that new rules will have better performance than initial ones. Initial rules are based on known attacks and new rules evolve from initial ones. As a result, new rules not only can cover known attacks, but also have possibly the potential to detect novel attacks.

3.2.1 Rule Expression

GP can be used to evolve rules. In this case, rules are represented using parse trees. For example, a rule is described as *if* (*c or* (*d and e*) *or* (*a and b*)) *then consequence*. The primitive functions are "*and*" and "*or*" and terminals are *a*, *b*, *c*, *d*, and *e*. Figure 3.1 illustrates the parse tree representing this rule. A and O means operators "*and*" and "*or*", respectively.

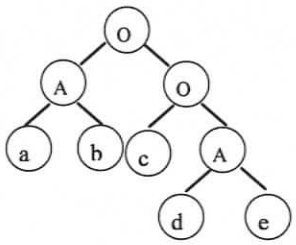


Figure 3.1. *Parse Tree Representation for a Rule*

More specifically, in our detection framework, rules are derived from known attacks. For instance, an empirical description about an intrusion is: "*if the type of a packet is TCP protocol and its source IP address is the same as its destination IP address then it is intrusive*". The rule *Aab* can be used to define this intrusive behavior, in which *a* stands for *PROTOCOL_TYPE = TCP* and *b* stands for *SIPA = DIPA* (*SIPA* = source IP

address; *DIPA* = destination IP address).

3.2.2 Genetic Operators

The genetic operators used in GP include crossover, mutation, reproduction and dropping condition. Crossover operations involve two parent trees. Two crossover points are randomly selected in the parent trees. Then, offspring trees are generated by exchanging sub-trees, which are selected according to the crossover point in the parent trees. Figure 3.2 shows an example of a crossover operation between rule "*if* ((*a and b*) *or c or* (*d and e*)) *then intrusion*" and rule "*if* (*g and h and k*) *then intrusion*", which produces two offspring rules "*if* ((*a and b*) *or c or k*) *then intrusion*" and "*if g and h and d and e then intrusion*".

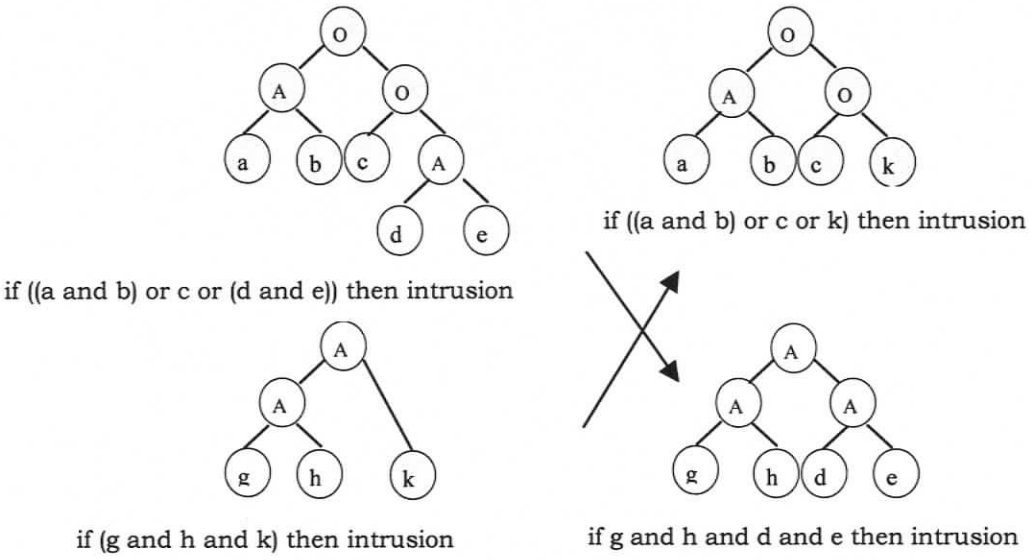
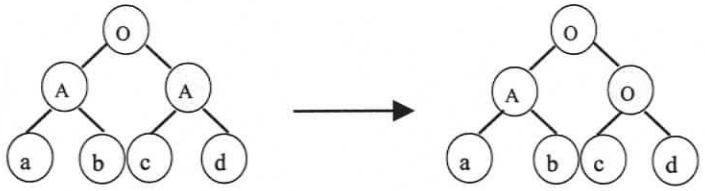


Figure 3.2. *Example of Crossover in GP*

Mutation operations involve a single parent tree. A mutation point is either a leaf node or a sub-tree, which can be randomly selected from the parent tree. We use a new point to replace the old leaf node or sub-tree and thus the new offspring tree is generated. Figure

3.3 shows an example of a mutation operation of rule "*if* (*a and b and c and d*) *then intrusion*". The produced mutation offspring rule is "*if* ((*a and b*) *or c or d*) *then intrusion*".



if(a and b and c and d) then intrusion if((a and b) or c or d) then intrusion

Figure 3.3. *Example of Mutation in GP*

Reproduction simply consists of copying the same rule. Dropping condition is a new operator. It randomly selects a condition in the rule, and then turns it into *any*, which means that this particular condition is no longer considered in the rule. For example, the rule "*if condition1 and condition2 and condition3 then intrusion*" can be changed to "*if condition1 and condition2 and any then intrusion*".

3.2.3 Fitness Function

Fitness functions ensure that the evolution is toward optimization by calculating the fitness value for each individual in the population. The fitness value evaluates the performance of each individual in the population. We use a fitness function defined in [67] that is based on a support-confidence framework. Support is a ratio of the number of records covered by the rules to the total number of records. Confidence factor *cf* represents the accuracy of rules, which is the confidence of the consequent to be true under the conditions. It is the ratio of the number of records matching both the consequent and the conditions to the number of records matching only the conditions. If a rule is represented as *if A then B* and the size of the training dataset is *N*, then we have:

$$cf = \frac{|A \ \ and \ \ B|}{|A|}; support = \frac{|A \ \ and \ \ B|}{N}$$

Where $|A|$ stands for the number of records that only satisfy condition A. $|B|$ stands for the number of records that only satisfy consequent B. $|A \ \ and \ \ B|$ stands for the number of records that satisfy both condition A and consequent B.

A rule with a high confidence factor does not necessarily behave significantly different from the average. Thus, normalized confidence factor is defined to consider the average probability of consequent denoted by *prob*.

$$normalized\_cf = cf \times \log\frac{cf}{prob}, \text{ where } prob = \frac{|B|}{N}$$

To avoid wasting time to evolve those rules with low support values, a strategy is defined: if support is below a user-defined minimum threshold (*min_support*), the confidence factor of the rule should not be considered. Thus, the fitness function is defined as follows:

$$raw\_fitness = \begin{cases} support & if\ support < min\_support \\ w_1 \times support + w_2 \times cf & otherwise \end{cases}$$

Where the weights w_1 and w_2 are user-defined and used to control the balance between the confidence and the support during the search.

Token competition is used to increase the diversity of solutions [68]. The idea is as follows: in the natural environment, once some individuals find a good place to live, then they will try to protect this environment and prevent newcomers from using it, unless the newcomers are stronger than the individuals. Other weaker individuals are hence forced to search for their own place. In this way, the diversity of the population is increased. In this approach, a token is allocated to each record in the training dataset. If a rule matches a record, its token will be seized by the rule. The priority of receiving the token is

determined by the strength of the rules. Thus, a rule with high *raw_fitness* score can acquire as many tokens as possible. The modified fitness is defined as follows:

$$modified\_fitness = raw\_fitness \times \frac{count}{ideal}$$

Where *count* is the number of tokens that the rule has actually seized, *ideal* is the total number of tokens that it can seize, which is equal to the number of records that the rule matches.

3.3 Experimental Evaluation

We evaluate our approach with 1999 KDDCUP intrusion detection dataset. Rules are represented by the attributes defined in KDDCUP dataset. In the rest of this section, we introduce the 1999 KDDCUP intrusion detection dataset, illustrate how to use GP to generate and evolve rules, and assess the detection effectiveness of the new rule base.

3.3.1 1999 KDDCUP Intrusion Detection Dataset

The KDD intrusion detection dataset is derived from the 1998 DARPA evaluation benchmark, which is the first standard corpus for evaluating intrusion detection approaches offline [12]. Thirty-three different attack types are simulated in nine weeks divided into seven weeks of training data and two weeks of testing data. The 1999 KDDCUP intrusion dataset, as a variant of DARPA dataset, reassembles the raw DARPA packets data into a sequence of connection records by using the Bro program [69]. A connection record is defined as a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol. To distinguish normal connections from attacks, three groups of higher-level features are defined, namely basic features, content features and traffic features.

The basic features of an individual TCP connection are described in Table 3.1, which include some basic information about TCP connection such as the connection's duration, protocol type, number of bytes transferred, etc. Table 3.2 illustrates the content features, which are obtained using domain knowledge. Content features normally involve only a single connection; their main purpose is to identify suspicious behaviour in the data portions. Traffic features, defined in Table 3.3, fall into two categories, namely host-based traffic features and time-based traffic features. Host-based traffic features are constructed using a window of 100 connections to the same host. Time-based features are constructed using the connections involved in a two seconds time window.

3.3.2 Rule Evolution Using GP

An individual solution in a population is represented as a parse tree using a string data structure. For example, a tree can be represented as "*AabAcdAceI*". As indicated earlier, *A* means operator "and"; *a,b,c,d* and *e* correspond to the conditions in the rules. *I* is the consequence, which means intrusion. Redundant conditions in the rule are removed, and thus rule "*AabAcdAceI*" can be interpreted as "*if a and b, and, c and d and e, then intrusion*". The attribute values of *a,b,c,d,e* are selected from known attacks.

New rules are generated in two phases. Firstly, temporary new rules are composed of new rules generated using the four genetic operators and additional rules directly generated from previous populations. Thus, the number of temporary new rules is doubled. Secondly, half of the temporary new rules with the highest fitness scores after token competition are retained and passed to the next generation.

To assess the feasibility and efficiency of GP for intrusion detection, we selected 10,000 connection records provided by the KDDCUP intrusion detection dataset for rule training. Eleven parameters are used to describe the attacks, namely *protocol_type*, *land*, *wrong_fragment*, *synflood*, *num_compromised*, *same_srv_rate*, *diff_srv_rate*, *count*,

Table 3.1. *Basic Features of Individual TCP Connections*

Feature Name	Description
duration	length (number of seconds) of the connection
protocol_type	type of the protocol, e.g. tcp, udp, etc.
service	network service at the destination, e.g., http, telnet, etc.
src_bytes	number of data bytes from source to destination
dst_bytes	number of data bytes from destination to source
flag	normal or error status of the connection
land	1 if connection is from/to the same host/port; 0 otherwise
wrong_fragment	number of wrong fragments
urgent	number of urgent packets

Table 3.2. *Content Features within a Connection Suggested by Domain Knowledge*

Feature name	Description
hot	number of hot indicators
num_failed_logins	number of failed login attempts
logged_in	1 if successfully logged in; 0 otherwise
num_compromised	number of compromised conditions
root_shell	1 if root shell is obtained; 0 otherwise
su_attempted	1 if su root command attempted; 0 otherwise
num_root	number of root accesses
num_file_creations	number of file creation operations
num_shells	number of shell prompts
num_access_files	number of operations on access control files
num_outbound_cmds	number of outbound commands in an ftp session
is_hot_login	1 if the login belongs to the hot list; 0 otherwise
is_guest_login	1 if the login is a guest login; 0 otherwise

Table 3.3. *Traffic Features Computed using a Two-second Time Window*

Feature name	**Description**
	Following features referring to same-host connections during two seconds period
count	number of connections to same host as current connection
srv_count	number of connections to same service as current connection
serror_rate	% of connections that have "SYN" errors
rerror_rate	% of connections that have "REJ" errors
same_srv_rate	% of connections to the same service
diff_srv_rate	% of connections to different services
	Following features referring to same-service connections
srv_serror_rate	% of connections that have "SYN" errors
srv_rerror_rate	% of connections that have "REJ" errors
srv_diff_host_rate	% of connections to different hosts
	Following features referring to same-host connections during past 100 connections
dst_host_count	number of connections to same host as current connection
dst_host_srv_count	number of connections to same service as current connection
dst_host_same_srv_rate	% of connections to the same service
dst_host_diff_srv_rate	% of connections to different services
dst_host_serror_rate	% of connections that have "SYN" errors
dst_host_rerror_rate	% of connections that have "REJ" errors
dst_host_same_src_port_rate	% of connections from same source port
	Following features referring to same-service connections
dst_host_srv_diff_host_rate	% of connections to different hosts
dst_host_srv_serror_rate	% of connections that have "SYN" errors
dst_host_srv_rerror_rate	% of connections that have "REJ" errors

srv_count, *dst_host_count*, and *dst_host_srv_count*. The meaning of these parameters is illustrated in Table 3.1, 3.2 and 3.3, respectively. Parameter *synflood* is a new parameter introduced in this thesis, which is not the original KDDCUP dataset parameter. Rules are composed by a set of attribute descriptors, whose representation and meaning are illustrated in Table 3.4. We randomly generate 40 initial rules using these descriptors. Table 3.5 illustrates these initial rules.

Table 3.4. *Representation of Attribute Descriptors (R1, R2 and R3 are random values)*

Terminal	Meaning	Terminal	Meaning
s	num_compromised=0	i	protocol_type=udp
b	count>R1	f	land=1
c	srv_count>R1	o	land=0
d	dst_host_count>R1	j	count<R2
k	srv_count<R2	q	synflood=0
p	wrong_fragment=0	r	synflood>1
e	dst_host_srv_count>R1	a	protocol_type=tcp
l	dst_host_count<R2	h	protocol_type=icmp
m	dst_host_srv_count<R2	u	same_srv_rate=0.00
t	num_compromised>1	v	same_srv_rate=1.00
g	wrong_fragment>1	w	diff_srv_rate>R3

We calculate the fitness value for each rule based on training dataset, which are labeled as either normal or intrusive. The test dataset is similar to training dataset. The only difference is the test data includes in addition some unknown attacks not occurring in training data. The rules in the initial population are evolved using four genetic operators. The rates of crossover, mutation and dropping condition operations are respectively 0.6, 0.01 and 0.001, which are selected based on experience. Forty offspring rules are evolved from previous forty parent rules. Temporary new rules are generated by combining offspring rules with parent rules. Half of the temporary new rules with the highest fitness scores after token competition are selected as the new rules.

Table 3.5. *Initial Rules*

Rules	**Meaning**
Afp	if land=1 and wrong_fragment=0 then intrusion
AAgg	if wrong_fragment>1 then intrusion
Aab	if protocol_type=tcp and count>3 then intrusion
bAbA	if srv_count>3 then intrusion
AhAg	if protocol_type=icmp and wrong_fragment>1then intrusion
rA	if synflood=1.00 then intrusion
AatA	if protocol_type=tcp and num_compromised>1 then intrusion
Aaav	if protocol_type=tcp and same_srv_rate=1.00 then intrusion
wwwwA	if diff_srv_rate>0.33 then intrusion
Aajt	if count<3 and num_compromised>1then intrusion
AfpAqA	if land=1 and wrong_fragment=0 and synflood=0.00 then intrusion
AaggqA	if wrong_fragment>1 and synflood=0.00 then intrusion
Aigq	if protocol_type=udp and wrong_fragment>1 and synflood=0.00 then intrusion
AabrA	if protocol_type=tcp and count>3 and synflood>1 then intrusion
Aarc	if srv_count>3 and synflood>1 then intrusion
Ahcr	if protocol_type=icmp and srv_count>3 and synflood>1 then intrusion
AaaAfj	if protocol_type=tcp and land=1 and count<3 then intrusion
AaAoc	if protocol_type=tcp and srv_count>3 and land=0 then intrusion
gaAjA	if protocol_type=tcp and count<3 and wrong_fragment>1 then intrusion
capAA	if protocol_type=tcp and srv_count>3 and wrong_fragment=0 then intrusion
Abc	if count>3 and srv_count>3 then intrusion
AatA	if protocol_type=tcp and num_compromised>1 then intrusion
Akvq	if srv_count<3 and synflood=0 and same_srv_rate=1.00 then intrusion
uagjA	if protocol_type=tcp and same_srv_rate=0.00 and wrong_fragment>1 and count<3 then intrusion
AAvA	if protocol_type=tcp and same_srv_rate=1.00 then intrusion
AAwv	if protocol_type=tcp and diff_srv_rate>0.33 and same_srv_rate=1.00 then intrusion
AqsujaA	if protocol_type=tcp and count<3 and synflood=0 and num_compromised=0 and same_srv_rate=0.00 then intrusion
Aasf	if num_compromised=0 and land=1 then intrusion
AsAg	if wrong_fragment>1 and num_compromised=0 then intrusion
AAAtfg	if num_compromised>1 and land=1 and wrong_fragment>1 then intrusion
Aaib	if protocol_type=udp and count>3 then intrusion
Attt	if num_compromised>1 then intrusion
Avfrg	if land=1 and wrong_fragment>1 and same_srv_rate=1.00 and synflood>1 then intrusion
AAAtiA	if protocol_type=udp and num_compromised>1 then intrusion
AAAvA	if same_srv_rate=1.00 then intrusion
cwAA	if srv_count>25 and diff_srv_rate>0.33 then intrusion
Aaiog	if protocol_type=udp and land=0 and wrong_fragment>1 then intrusion
Ar	if synflood>1 then intrusion
Aagg	if wrong_flagment>1 then intrusion
AaffA	if land=1 then intrusion

Rules evolution is terminated either after 5000 runs have been completed or the fitness value for each rule is bigger than a threshold equal to 0.95. Table 3.6 describes the new rules obtained during the experiment.

3.3.3 Performance Evaluation

Evaluation of intrusion detection approaches for detecting novel attacks is an important and multi-faceted problem. The training dataset we use is one day's connection records provided by KDDCUP dataset, consisting of 10000 connection records. Eight kinds of network attacks are included in the training dataset, namely *land, synflood, pod, teardrop, back, neptune, ipsweep* and *portsweep*[1]. The testing dataset we use is another one-day activity consisting of 10000 connection records. Ten kinds of network attacks are included in the testing dataset, namely *smurf, udpstorm, land, synflood, pod, teardrop, back, neptune, ipsweep* and *portsweep*. *Smurf* and *udpstorm* attacks are seen as the novel attacks since they are absent from the training dataset. Detection of attacks involved in the test dataset, and not occurring in the training dataset, assesses the potential ability to detect unknown attacks. We use three performance metrics to evaluate our new rules, namely false positive rate (FPR), false negative rate (FNR) and unknown attack detection rate (UADR). A false positive occurs when a rule classifies normal traffic as intrusive. A false negative occurs when a rule characterizes an intrusion as normal. UADR measures the capability of a new rule to detect novel attacks.

For each rule, we calculate its FPR, FNR and UADR independently. We find that every time we use GP to evolve the rules, the number of generated rules is different and thus the FPR, FNR and UADR for each rule is also different. Therefore, to statistically evaluate the efficiency of our GP based approach, FPR, FNR and UADR are defined as the arithmetical average of the sum of all new rules' rates:

---

[1] See Appendix for brief description of the attacks.

Table 3.6. *New Rules*

Rules	**Meaning**
Ag	if wrong_fragment>1 then intrusion
Afpq	if land=1 and wrong_fragment=0 and synflood=0 then intrusion
Aav	if protocol_type=tcp and same_srv_rate=1.00 then intrusion
Ahcq	if protocol_type=icmp and dst_host_srv_count>160 and synflood=0 then intrusion
Aikq	if protocol_type=udp and srv_count>367 and synflood=0 then intrusion
Aat	if protocol_type=tcp and num_compromised>1then intrusion
At	if num_compromised>1 then intrusion
Ag	if wrong_fragment>1 then intrusion
Ajlpr	if count<412 and dst_host_count<810 and wrong_fragment=0 and synflood>1 then intrusion
Ail	if protocol_type=udp and dst_host_count>203 then intrusion
Agr	if wrong_fragment>1 and synflood>1 then intrusion
Agq	if wrong_fragment>1 and synflood=0 then intrusion
Agiq	if protocol_type=udp wrong_fragment>1 and synflood=0 then intrusion
Aw	if diff_srv_rate>0.33 then intrusion
Aagq	if protocol_type=tcp and wrong_fragment>1 and synflood=0 then intrusion
Aqv	if synflood=0 and same_srv_rate=1.00 then intrusion
Agu	if wrong_fragment>1 and same_srv_rate then intrusion
Aadr	if protocol_type=tcp and dst_host_count>88 and synflood>1 then intrusion
Ahmq	if protocol_type=icmp and dst_host_srv_count<160 and synflood=0 then intrusion
Aicq	if protocol_type=udp and srv_count>367 and synflood=0 then intrusion
Aha	if protocol_type=icmp and count>160 then intrusion
Afs	if land=1 and num_compromised=0 then intrusion
Aags	if protocol_type=tcp and wrong_fragment>1 and num_compromised=0 then intrusion
Av	if same_srv_rate=1.00 then intrusion
Af	if land=1 then intrusion
Afg	if land=1 and wrong_fragment>1 then intrusion
Aflp	if land=1 and sat_host_coun<203 and wrong_fragment=0 then intrusion
Afq	if land=1 and wrong_fragment=0 then intrusion
Agh	if protocol_type=icmp and wrong_flagment>1then intrusion
Abc	if count>120 and srv_count>250 then intrusion
Aad	if protocol_type=tcp and dst_host_count>320 then intrusion
Aijv	if protocol_type=udp and count<10 and same_srv_rate=1.00 then intrusion
Akr	if srv_count<60 and synflood>1 then intrusion
Aabo	if protocol_type=tp and count>450 and land=0 then intrusion
Aeh	if protocol_type=icmp and dst_host_srv_count>255 then intrusion
Aopt	if land=0 and wrong_fragment>1 and num_compromised>1 then intrusion
Agr	if wrong_fragment>1 and synflood>1 then intrusion
Ahfg	if protocol_type=icmp and land=1 and wrong_fragment>1 then intrusion
Afw	if land=1 and diff_srv_rate>0.5 then intrusion
Act	if srv_count>46 and num_compromised>1 then intrusion

$$FPR = \frac{\sum_{i=1}^{N}(FPR)_i}{N};\quad FNR = \frac{\sum_{i=1}^{N}(FNR)_i}{N};\quad UADR = \frac{\sum_{i=1}^{N}(UADR)_i}{N}$$

Where N is the number of evolved rules; FPR_i, FNR_i, and $UADR_i$ stand for the FPR, FNR and UADR of the i^{th} rule.

Since the number of new rules is different in each run, the average FPR, FNR and UADR for each run are also different. We execute 10,000 runs and plot respectively the probability distribution and the log scale probability distribution of FPR, FNR and UADR. Figures 3.4-a and 3.4-b illustrate FPR's probability distribution and its log scale probability distribution. The FNR's probability distribution and the corresponding log scale probability distribution are illustrated in Figures 3.5-a and 3.5-b. Figures 3.6-a and 3.6-b plot the probability distribution of UADR and its log scale probability distribution. Table 3.7 describes some statistical information for FPR, FNR, and UADR, such as their average value and standard deviation over 10000 runs.

The detailed information about scale distribution for FPR, FNR and UADR over 10000 runs is summarized in Tables 3.8, 3.9 and 3.10 respectively. In most cases, the evolved new rules have a low FPR and FNR. Moreover, the UADR for new rules in about 20% of the runs is bigger than 0.9, which shows a good potential to find new attacks.

In practice, we usually use the rule base instead of single rule to test the performance of a rule-based intrusion detection system. The rule base includes all evolved new rules and intrusion decisions are made based on the rule base. This evaluation approach avoids the redundancy of rules and is supposed to be more accurate. We execute 10000 runs to evaluate the statistical performance of our rule base. Two evaluation metrics are used, namely false positive rate (FPR) and detection rate (DR). Generally a good intrusion detection approach should have high detection rate with low false positive rate. The probability distribution of FPR for the rule base over 10000 runs is illustrated in Figure

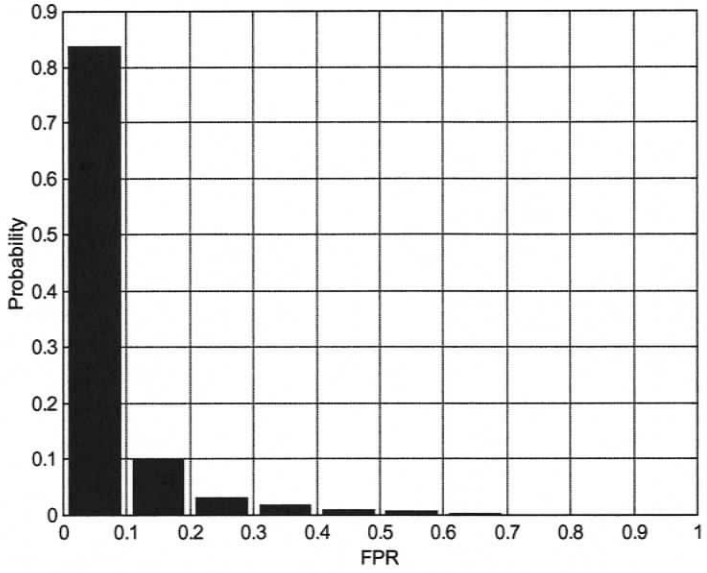


Figure 3.4-a. *Distribution of FPR*

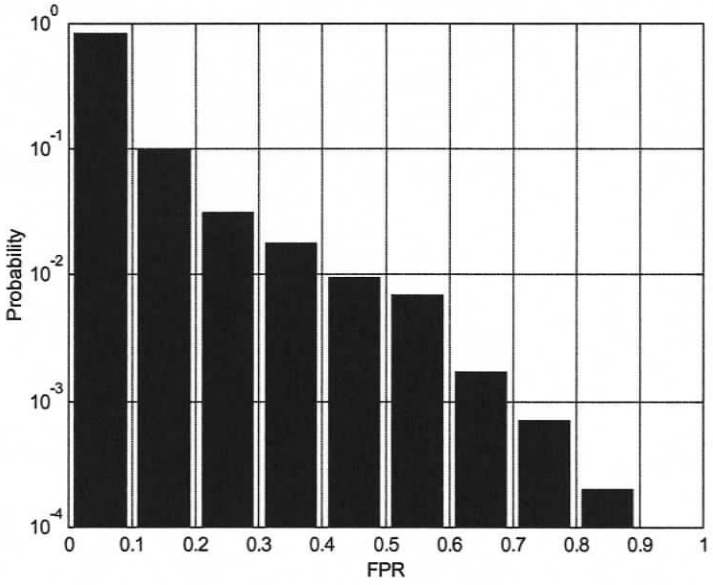


Figure 3.4-b. *Log Scale Distribution of FPR*

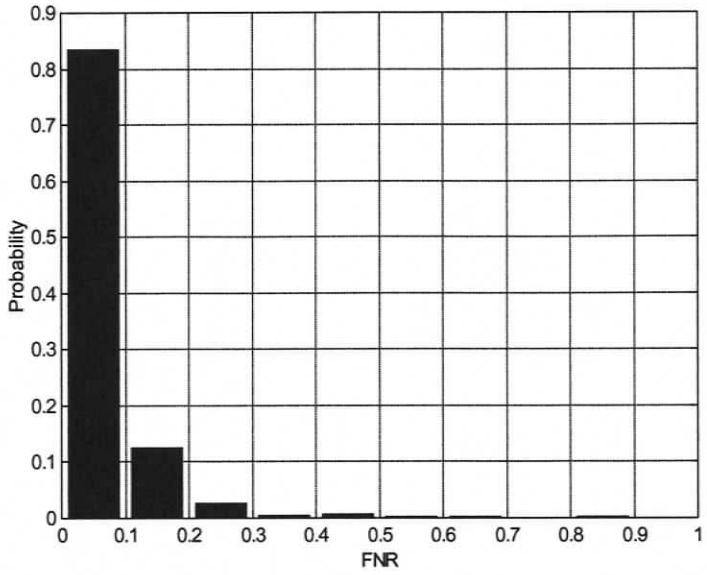


Figure 3.5-a. *Distribution of FNR*

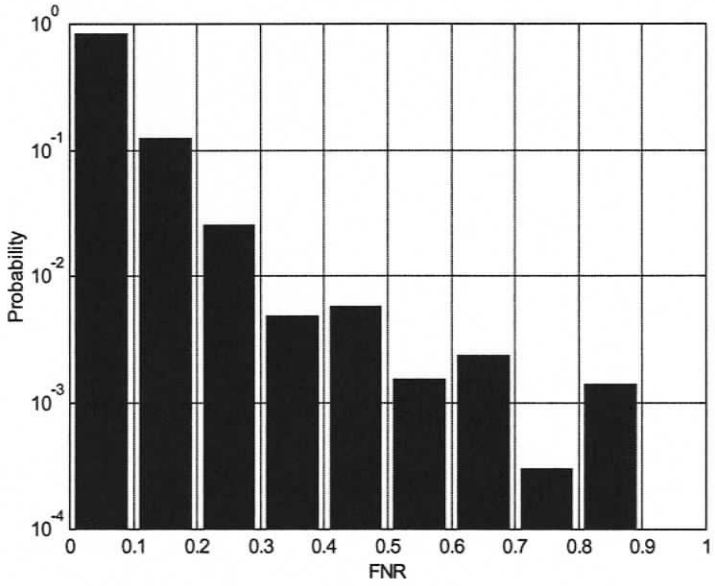


Figure 3.5-b. *Log Scale Distribution of FNR*

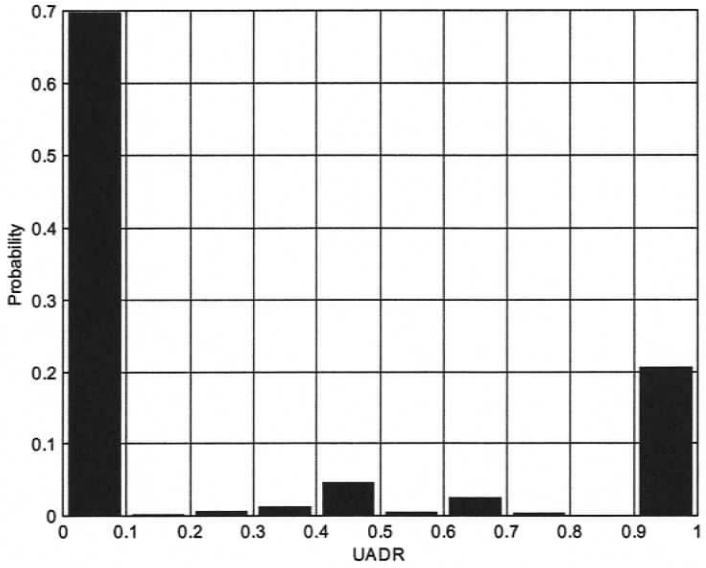


Figure 3.6-a. *Distribution of UADR*

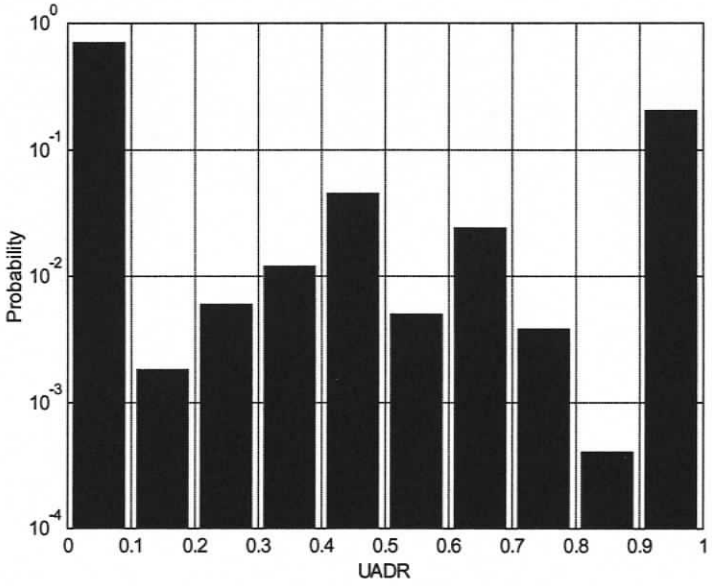


Figure 3.6-b. *Log Scale Distribution of UADR*

Table 3.7. *Statistical Information over 10000 Runs*

FPR		FNR		UADR	
Avg.	Std.	Avg.	Std.	Avg.	Std.
0.0523	0.0944	0.0504	0.0801	0.2509	0.397

Table 3.8. *Scale Distribution Over 10000 Runs for FPR*

FPR	0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1.0
Number Of Run	8200	1000	400	200	99	70	20	8	3	0

Table 3.9. *Scale Distribution Over 10000 Runs for FNR*

FNR	0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1.0
Number Of Run	8300	1100	300	100	100	30	40	5	25	0

Table 3.10. *Scale Distribution Over 10000 Runs for UADR*

UADR	0-0.1	0.1-0.2	0.2-0.3	0.3-0.4	0.4-0.5	0.5-0.6	0.6-0.7	0.7-0.8	0.8-0.9	0.9-1.0
Number Of Run	6835	25	75	105	550	60	250	50	5	2055

3.7-a. Figure 3.7-b illustrates the same information for FPR between 0 and 0.1. The average value of FPR over 10000 runs is 0.41% and the corresponding standard deviation is 0.0063. Figure 3.8-a and 3.8-b illustrate the probability distribution of DR for the rule base over 10000 runs and its log scale probability distribution. The average value and the standard deviation of DR over 10000 runs are 0.5714, and 0.4068 respectively.

Figure 3.9 plots a graph describing the relation between false positive rate and corresponding detection rate. The DR increases as the FPR increases. The DR is close to 100% when the FPR is in the range between 1.4% and 1.8%. However, when the

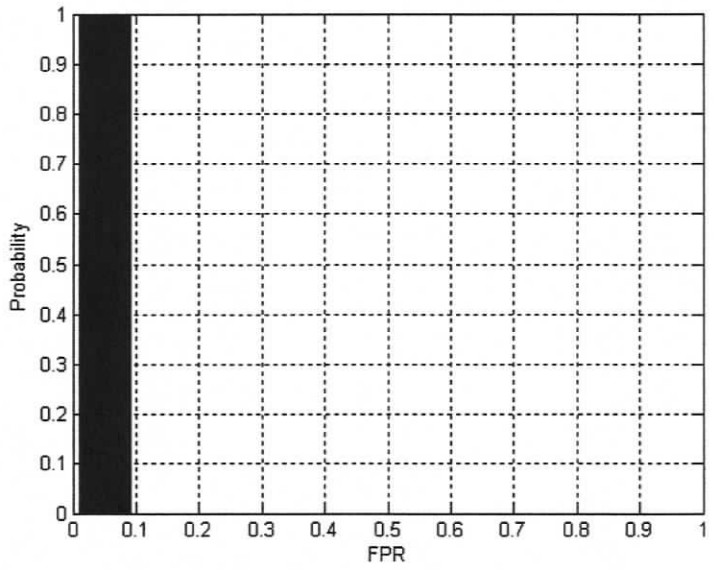


Figure 3.7-a. *Distribution of FPR*

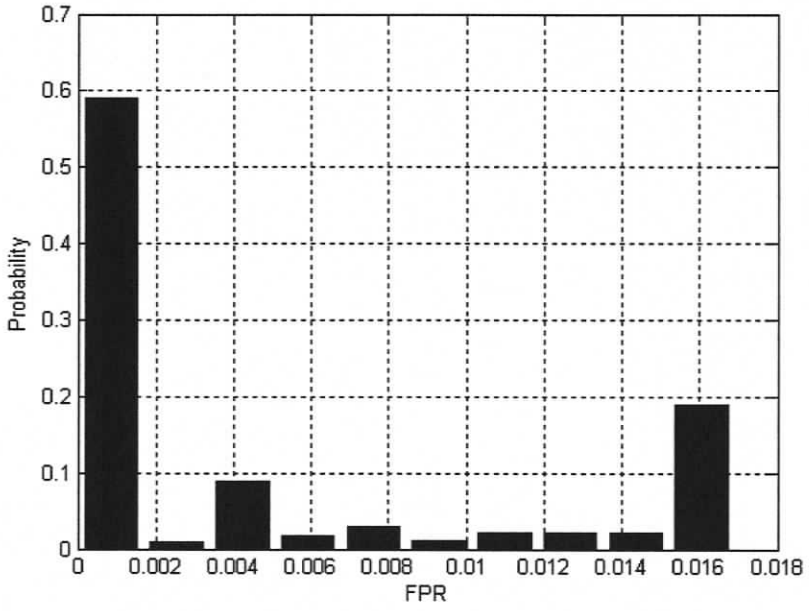


Figure 3.7-b. *Distribution of FPR between 0 and 0.1*

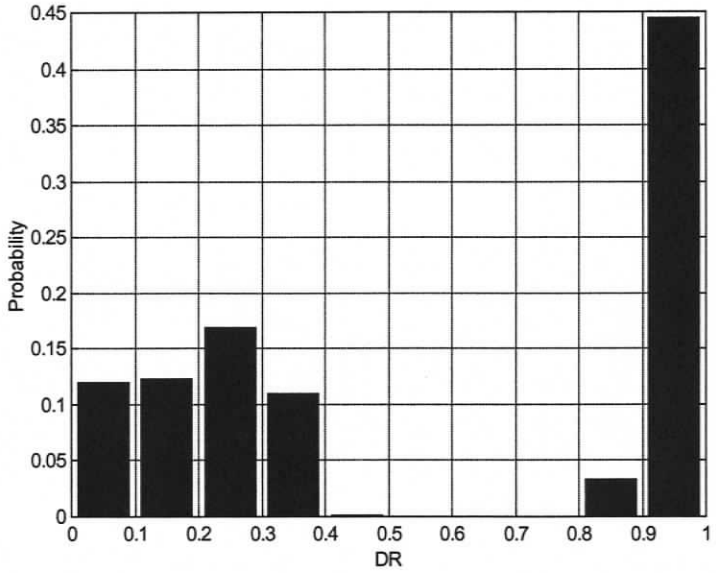


Figure 3.8-a. *Distribution of DR*

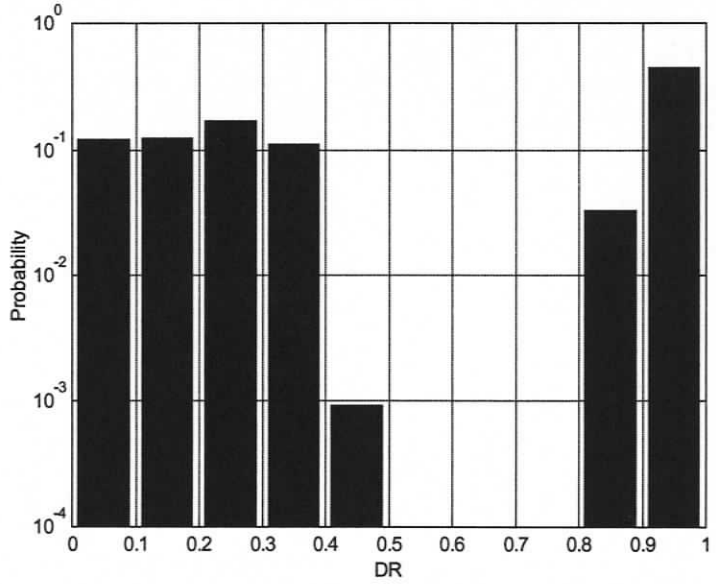


Figure 3.8-b. *Log Scale Distribution of DR*

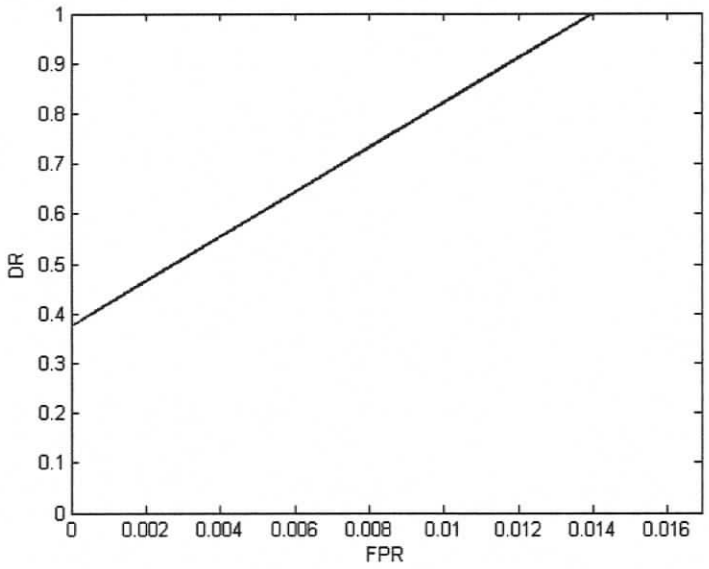


Figure 3.9. *Relation Curve between FPR and DR*

false positive rate is close to 0%, the DR is only about 40%. The DR falls in a broad range from 40% to 100% because the number of rules in the rule base is different for each run. When there are more rules in the rule base, we have a high detection rate and thus, will possibly have a low false positive rate.

3.4 Discussion

A novel intrusion detection scheme is proposed in this chapter to find unknown attacks. The approach is based on the GP algorithm. Initial rules are randomly generated according to the existing domain knowledge about attacks. GP is used to evolve the rules and new rules are supposed to have better performance than initial rules, and might have the potential to find new attacks. The approach is then evaluated with 1999 KDDCUP intrusion detection dataset. The experiment shows that the average FPR, FNR, and UADR in 10000 runs are 5.23%, 5.04% and 25.09% respectively, and the FPR and DR for the rule base in 10000 runs are 0.41% and 57.14% respectively. Although the

approach represents a new scheme for detecting new attacks, it still has several limitations:

Firstly, the approach relies on labeled training dataset. Labeling the training dataset, which is usually carried out manually, can be time consuming and error-prone since the data always include noises. Moreover, the training dataset needs constantly to be updated as new attacks are discovered. This is a deterrent for online learning.

Secondly, the detection result is not good for some runs because the selection of crossover and mutation points in corresponding operations is random. In addition, deciding the probability of genetic operators selection is based on experiences. Moreover, this approach belongs to the category of supervised anomaly detection and it naturally shares the inherent limitations of supervised anomaly detection scheme, which are described as follows:

1. Supervised anomaly detection does not consider the completeness of the training and testing dataset. Completeness refers to the extent to which the dataset represents all types of attacks. For instance, the 1999 KDDCUP's training dataset only contains a total of 24 attack types with an additional 14 types included in the testing dataset. Even if the supervised learning approaches perform well under the testing dataset, they may possibly carry errors that are not identified due to incompleteness of the testing dataset.
2. In supervised anomaly detection scheme, labeling the training data manually is time consuming and also error-prone since training data usually includes noises.
3. The high dimensions of features vector used in training and testing data tend to consume a large amount of system resources. For instances, the 1999 KDDCUP intrusion detection dataset uses 41 features to identify a connection record.

In principle, these limitations could be addressed by investigating approximate unsupervised learning model. This is the topic covered in subsequent chapters.

Chapter 4

I-means: A Novel Hybrid Clustering Algorithm

I-means is a hybrid-clustering algorithm that is built around the k-means algorithm. Specifically, I-means extends k-means by addressing inherent limitations. An important feature of I-means is the ability to determine systematically the optimal number of clusters for a given dataset. The I-means algorithm combines three distinct but complementary functionalities:

1. Cluster number estimation and clusters' centers initialization, using an evolutionary approach to mixture resolving.
2. Outlier detection and removal based on mixture resolving.
3. Data clustering using plain k-means.

In this chapter, we briefly overview clustering analysis, highlight the limitations of k-means algorithm which is one of the most popular algorithms available, and then present the I-means algorithm.

4.1 Cluster Analysis

4.1.1 Overview

Clustering is the organization of data patterns into groups or clusters based on some

measure of similarity [7]. There is a clear distinction between discriminant analysis called supervised classification and clustering, which is also referred to as unsupervised classification. In discriminant analysis, an initial set of labeled patterns is used to classify unseen data. Specifically, labeled patterns are used as training data to learn the classes of unseen data, which in their turn are used to label new data. In contrast, in clustering unseen data patterns are labeled using solely information extracted from the data; no initial labeled patterns are used for training. A typical data clustering process maybe structured according to the following stages:

1. Pattern representation: involves feature definition, selection, and extraction. Features consist of main characteristics of the data set to be used for clustering. Feature selection consists of identifying a subset of the existing features to be used for the clustering process. Feature extraction consists of generating new features, more salient and effective from the original features set by conducting some transformation on the selected features.
2. Similarity measure definition: provides objective criteria for grouping or discriminating patterns. A popular similarity measure consists of a distance function on pairs of patterns such as the Euclidean distance.
3. Grouping: consists of the cluster formation; there are various categories of grouping techniques including hierarchical, partitional, probabilistic and graph-theoretic. Hierarchical and partitional clustering are the most popular approaches. Hierarchical clustering generates a nested series of partitions by merging or splitting clusters based on a similarity measure; in contrast, partitional clustering generates a single partition of the data by optimizing similarity criteria.
4. Data abstraction: consists of producing a simple and compact representation of the clusters from either processing or readability perspective.
5. Evaluation: consists of establishing the validity of the clustering algorithm or procedure. Using statistical techniques may achieve this.

4.1.2 Clustering Approaches

The main goal of clustering is grouping data objects in classes, so that intra-class similarity is maximized and inter-class similarity is minimized. Examples of clustering approaches include distance-based clustering, probability-based clustering and hierarchical-based clustering.

The distance-based clustering describes the similarity of two patterns by defining a distance measure between them. Three commonly used distance measures are Euclidean distance, Manhattan distance and Minkowshi distance.

Euclidean distance is defined as:

$$d(i,j)=\sqrt{(\left|x_{i1}-x_{j1}\right|^2+\left|x_{i2}-x_{j2}\right|^2+\cdots+\left|x_{ip}-x_{jp}\right|^2)}$$

Manhattan distance is defined as:

$$d(i,j)=\left|x_{i1}-x_{j1}\right|+\left|x_{i2}-x_{j2}\right|+\cdots+\left|x_{ip}-x_{jp}\right|$$

Minkowski distance is defined as:

$$d(i,j)=(\left|x_{i1}-x_{j1}\right|^q+\left|x_{i2}-x_{j2}\right|^q+\cdots+\left|x_{ip}-x_{jp}\right|^q)^{1/q}$$

Where $i=(x_{i1},x_{i2},...,x_{ip})$, and $j=(x_{j1},x_{j2},...,x_{jp})$; i and j represent two different p-dimensional objects in the dataset; q is a positive integer. When $q=1$, Minkowski distance is equivalent to Manhattan distance. When $q=2$, Minkowski distance is equivalent to Euclidean distance.

A typical process for the distance-based clustering starts by randomly generating a number, say R, of initial clusters, and then distances from every object to the central points of the R clusters are calculated. According to the shortest distance rule, initial clusters are reorganized and new clusters are generated. This process is repeated until the clusters remain stable or a given termination condition is satisfied.

The probability-based clustering assigns an object to a cluster according to some

conditional probability instead of a distance. Assume the components of pattern X, $X = \{x_1, x_2, ..., x_n\}$ are independent and the R initial clusters are denoted by C_1, C_2 ,..., C_R . According to Bayes rule, the similarity of X for a cluster C_i is represented by the joint probability $p(X, C_i)$, which can be calculated as:

$$p(X, C_i) = p(x_1 \mid C_i) \times p(x_2 \mid C_i) \times ..... \times p(x_n \mid C_i)$$

Probability $p(x_j \mid C_i)$ can be estimated from sample statistics of patterns in the clusters. If $p(X, C_i)$ is bigger than a threshold δ, X will be added to C_i, otherwise a new cluster including X will be generated. Then, all $p(x_1 \mid C_i)$, $p(x_2 \mid C_i)$ ...and $p(x_n \mid C_i)$ are updated and the process is repeated until the clusters remain unchanged or a given termination condition is reached.

The hierarchical-based clustering creates a tree of clusters. Each node in the tree is linked to one or several children clusters while sibling clusters share a common parent node. According to the method used for trees building, hierarchical clustering methods are divided into two categories, agglomerative and divisive. Agglomerative methods recursively merge central points of different clusters while divisive methods start from a cluster including all data and then gradually split them into smaller clusters. Both agglomerative and divisive methods require calculating the distance between clusters. Generally, four distance measures are usually used in hierarchical clustering, namely minimum distance, maximum distance, mean distance and average distance.

Minimum distance is defined as:

$$d_{\min}(C_i, C_j) = \min_{p \in C_i, q \in C_j} |p - q|$$

Maximum distance is defined as:

$$d_{\max}(C_i, C_j) = \max_{p \in C_i, q \in C_j} |p - q|$$

Mean distance is defined as:

$$d_{mean}(C_i, C_j) = |m_i - m_j|$$

Average distance is defined as:

$$d_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{p \in C_i} \sum_{q \in C_j} |p - q|$$

Where m_i is the mean of cluster C_i; n_i is the number of components in cluster C_i; p and q are objects in the dataset.

4.1.3 Issues

Many clustering approaches have been proposed in past years. The main reason is, as indicated in the first chapter, that "there is no clustering technique that is universally applicable in uncovering the variety of structures present in multidimensional data sets... each new clustering algorithm performs slightly better than existing ones on a specific distribution of patterns" [7]. Furthermore, the thorough knowledge of the domain and the data gathering process is essential in achieving higher quality feature extraction, similarity computation, grouping, and data abstraction.

Some empirical studies in the literature have shown that k-means is one of the most efficient in terms of execution time compared to other existing clustering algorithms. It has also been reported that k-means is one of the few algorithms that have been applied successfully to large data sets; most of the existing algorithms cannot handle large data sets. The appeal of k-means is related to its time and space complexities, which are linear, and the fact that it is order-independent. Therefore, k-means appears to be a good candidate for addressing the challenge of intrusion detection efficiency.

However, there are several weaknesses in k-means, which may represent serious impediments to both its effectiveness and efficiency. Important shortcomings of k-means include cluster convergence, number of clusters dependency and degeneracy. The

k-means algorithm may fail to converge if initial partitions are not properly chosen, a situation that occurs often and is not an obvious task. The number of clusters dependency is related to the difficulty of identifying the optimal number of clusters needed; this dependency is a serious problem. Clusters degeneracy refers to the fact that the clustering process may lead to empty clusters, which are useless for the classification process. Another significant limitation is that the k-means algorithm cannot process noisy data points or outliers.

In order to address all the weaknesses mentioned above and achieve more versatile clusters without sacrificing performance, a solution consists of developing a hybrid technique, which combines the strength of k-means with some other approaches that address these issues. Based on this consideration, we propose a new algorithm that combines k-means with an evolutionary approach to mixture-resolving and outlier detection. In the rest of this chapter, we propose the evolutionary algorithm for estimating the optimal number of clusters, present the outlier detection strategy based on mixture resolving and formalize the I-means algorithm.

4.2 An Evolutionary Approach to Mixture Resolving

4.2.1 Context

Determining the optimal number of clusters for a dataset is one of the most difficult issues in cluster analysis [70]. Identifying the optimal number of clusters is essential for effective and efficient data clustering [71]. A popular clustering approach sensitive to this problem is based on Gaussian mixture model (GMM). GMM is based on the assumption that the data to be clustered are drawn from one of several Gaussian distributions. It is suggested that Gaussian mixture distribution can approximate any distribution up to arbitrary accuracy, as long as a sufficient number of components are used [9].

Consequently, the entire data collection is seen as a mixture of several Gaussian distributions, and their corresponding probability density functions can be expressed as a weighted finite sum of Gaussian components with different parameters and mixing proportions [10]. A common approach for estimating the parameters of GMM is Expectation-Maximization (EM) algorithm [72]. However, EM may easily converge into local maximum after a limited number of iterations. Moreover, EM usually assumes that the number of mixing components is known in advance, which is not always the case in practice.

Previous attempts for estimating the number of mixing components for the GMM are mainly based on statistical techniques. Examples of these previous works suggested in the literature includes [73, 74, 75, 76, 77]. Figueiredo and Jain propose a new criterion named *minimum message length* to fit the Gaussain mixture model using the EM algorithm [73]. Starting with a high number of components, their approach iteratively decreases the number of components until the optimal number is obtained. McLachlan uses the likelihood ratio test statistic λ to test the smallest number of components k for the Gaussian mixture fitting with the real data [74]. Penalvar et al. introduce a new criterion based on the probability density function of the components, and then propose a modified version of the EM algorithm in order to estimate the optimal number of components [75]. Richardson and Green propose the reversible Markov chain Monte Carlo method for sampling univariate normal mixtures with a varying number of components [76]. Vlassis et al. use the kurtosis as a measure of non-gaussianity to estimate dynamically the optimal number of components [77]. Their method starts with a small number of components k, such as $k=1$, and then increases the number of components dynamically through splitting operations based on the optimization criterion.

In this dissertation we tackle the issue of number of clusters estimation using a novel evolutionary approach, which combines the Gaussian mixture and the EM algorithm. Although the previous approaches based on statistical techniques have proved their

ability to estimate the optimal number of clusters, they are prone to converge into local optima since they usually stop performing further search when the corresponding criteria reach certain thresholds. Also, there are some other local search algorithms such as simulated annealing approaches, randomized hill-climbing techniques, etc., which share the same limitations with statistical techniques. In contrast, the evolutionary computation scheme proposed in our work has the inherent capability to escape from local maximum since the search space for optimal solutions can be extended by genetic operations and optimization.

In the remainder of this section, we illustrate our evolutionary algorithm for determining the optimal number of clusters for a set of data and present the results obtained in the empirical validation of the algorithm.

4.2.2 Evolutionary Entities and Functions

4.2.2.1 Representation of Evolutionary Individuals

A random variable x is said to follow a finite mixture distribution when its probability density function $p(x)$ can be represented as a weighted sum of kernels:

$$p(x) = \sum_{i=1}^{k} \alpha_i f_i(x; \mu_i, \nu_i)$$

Where k is the number of mixture components and α_i $(1 \le i \le k)$ are the mixing proportions or mixing coefficients satisfying the constraints $\alpha_i \ge 0$ and $\sum_{i=1}^{k} \alpha_i = 1$; $f_i(x; \mu_i, \nu_i)$ is the component density or kernel, where μ_i refers to the mean of x and ν_i stands for the variance of x. In the case where each component is Gaussian, we say that variable x follows a Gaussian mixture model, where the density function

$f_i(x;\mu_i,\nu_i)$ can be a multivariate Gaussian or a univariate Gaussian distribution.

Each individual in the evolutionary population is composed by a *three-tuple* vector $<\alpha_i,\mu_i,\nu_i>$ $(1 \le i \le k)$. It was established that EM was an effective algorithm to estimate the parameters of the mixture model [72]. Given the representation of GMM, the estimation problem can be described as: given a dataset $X \sim \{x_n \mid n = 1,2,...,N\}$, we estimate the unknown parameters θ, such that $\theta = <\alpha_i,\mu_i,\nu_i>$, by maximizing the likelihood function $\ell(x \mid \theta) = p(x_1, x_2 ... x_n \mid \theta)$. Suppose θ^* is the estimation value which can maximize the $\ell(x \mid \theta)$, we have:

$$\theta^* = \arg\max \ell(x \mid \theta) = \arg\max(p(x_1, x_2 ... x_n \mid \theta))$$

EM algorithm first estimates the posterior probability $p(i \mid x_n)$ for every n and i using the current parameters and then uses these posteriors to re-estimate the parameters by maximizing the likelihood function. The first step is called E-step and the second is M-step. Suppose the mixture component is the univariate Gaussian distribution, the EM algorithm starts with some initial random parameters α_i^0, μ_i^0 and ν_i^0 and then repeatedly applies the E-step and M-step to generate better parameter estimates α_i^1, μ_i^1, and ν_i^1. The complete EM algorithm used to estimate the set of parameters $\{\alpha_i,\mu_i,\nu_i\}$ is described as follows:

1. Initialize the parameter vector $\theta^0 = \{\alpha_i^0,\mu_i^0,\nu_i^0\}$
2. E-step: for every data sample in $X \sim \{x_n \mid n = 1,2,...,N\}$ and for every mixture component k, compute the posterior probability $\alpha_i = p(i \mid x_n)$ as follows:

$$p(i \mid x_n) = \frac{\alpha_i N(x_n;\mu_i,\nu_i)}{\sum_{i=1}^{k} \alpha_i N(x_n;\mu_i,\nu_i)}$$

3. M-step: re-estimate the parameters using the samples weighted by the posterior probabilities $p(i \mid x_n)$ as follows:

$$\alpha_{inew} = \frac{1}{N}\sum_{n=1}^{N} p(i \mid x_n)$$

$$\mu_{inew} = \sum_{n=1}^{N} \left(\frac{p(i \mid x_n)}{\sum_{n=1}^{N} p(i \mid x_n)}\right) x_n$$

$$\nu_{inew} = \sum_{n=1}^{N} \left(\frac{p(i \mid x_n)}{\sum_{n=1}^{N} p(i \mid x_n)}\right) (x_n - \mu_{inew})(x_n - \mu_{inew})^T$$

4. Go to step 2 until the algorithm converges.

An important attribute of EM algorithm is that it always converges to at least a local maximum of the likelihood function $\ell(x \mid \theta)$ after a limited number of iterations [72].

EM algorithm generates an estimate for the set of parameters $\{\alpha_i, \mu_i, \nu_i\}$ $(1 \le i \le k)$ and posterior probabilities $p(i \mid x_n)$ $(1 \le n \le N)$. The posterior probability describes the likelihood that the data pattern x_n approximates to a specified Gaussian component i. The greater the posterior probability for a data pattern x_n belonging to a specified Gaussian component i, the higher the approximation is. As a result, each data x_n is assigned to the corresponding Gaussian component i according to $p(i \mid x_n)$ and final clustering results are statistically represented by the set of parameters $\{\alpha_i, \mu_i, \nu_i\}$.

4.2.2.2 Evolutionary Operators

During the evolution, we randomly split, as well as, merge or delete components. Therefore we propose three new evolutionary operators called *splitting*, *merging* and

deletion operators, used as their names indicate for splitting, merging and removing the components.

Splitting Operator: The splitting operation leads to two new components and thus the total number of components will be increased by 1. For univariate Gaussian component, the means, variances and mixing proportions for the two new components are calculated as follows:

$$\mu_{1,split} = \mu_i + \nu_i, \quad \mu_{2,split} = \mu_i - \nu_i$$

$$\nu_{1,split} = \nu_{2,split} = \nu_i$$

$$\alpha_{1,split} = \alpha_{2,split} = \frac{\alpha_i}{2}$$

For multivariate Gaussian component, all means, covariance matrixes, and mixing proportions are re-estimated by the EM algorithm according to the new number of clusters.

Merging Operator: The merging operation leads to a new component by combining two components and thus the total number of components will be decreased by 1. For univariate Gaussian component, the means, variances and mixing proportions for the new component is as follows:

$$\mu_{merge} = \frac{\alpha_1\mu_1 + \alpha_2\mu_2}{\alpha_1 + \alpha_2}$$

$$\nu^2_{merge} = \frac{\alpha_1}{\alpha_1 + \alpha_2}[\nu_1 + (\mu_1 - \mu_{merge})]^2 + \frac{\alpha_2}{\alpha_1 + \alpha_2}[\nu_2 + (\mu_2 - \mu_{merge})]^2$$

$$\alpha_{merge} = \alpha_1 + \alpha_2$$

Otherwise, for multivariate Gaussian component, all means, covariance matrixes, and mixing proportions are re-computed by the EM algorithm according to the new number of clusters.

Deletion Operator: The delete operation eliminates a component entirely from the

total number of components. After applying this operator, all other components update their mixing proportions to sum one and also keep corresponding means and covariance matrices unchanged.

4.2.2.3 Fitness Function

We define an entropy-based fitness function as the criterion for measuring how well the mixture model approximates the unknown density underlying the data samples: *the higher the fitness function value, the better the approximation*. The entropy-based fitness function EF_T is defined as follows:

$$EF_T = \sum_{i=1}^{k} \alpha_i |ef_i|$$

Where k is the number of mixture components and α_i $(1 \le i \le k)$ are the mixing proportions satisfying the constraints $\alpha_i \ge 0$ and $\sum_{i=1}^{k} \alpha_i = 1$; ef_i denotes the individual fitness for each of the mixture component, which is defined as follows:

$$ef_i = \frac{H(C_i)}{H_{\max}(C_i)}$$

Where C_i stands for the set of data whose probabilities of belonging to the i^{th} component are the highest compared to other components, and $H(C_i)$ represents the entropy of these data. According to results from information theory, the theoretical maximum entropy for an individual component, denoted by $H_{\max}(C_i)$, can be calculated as follows:

$$H_{\max}(C_i) = \frac{1}{2}\log((2\pi e)^n |\Sigma|)$$

Where Σ is the covariance matrix of C_i and n is the number of dimensions for

data cluster C_i. It is established in information theory that Gaussian variables have maximum entropy among all the variables with equal variance. Hence, a high value of the individual entropy value ef_i for the i^{th} kernel corresponds to high probability that this component is truly Gaussian and also represents a better match between probability distribution function and data. As a result, the higher the entropy-based fitness value EF_T, the better the approximation of the entire dataset.

4.2.3 Proposed Evolutionary Algorithm

The proposed evolutionary algorithm is described in Table 4.1. The algorithm starts with an initial number of components, which is empirically selected based on the size of the data. A population denoted by $population_j$ corresponds to a set of new individuals created during the j^{th} generation; we denote by k_j the size of $population_j$. An individual is denoted by $<\alpha_i^j, \mu_i^j, \sum_i^j>$, where i represents the i^{th} component generated ($1 \le i \le k_j$); j stands for the number of generations. We use $population_{optimal}$ and $k_{optimal}$ to denote the optimal individuals and the optimal number of clusters, respectively. New individuals are composed by $population_j$ and $population_{optimal}$. The entropy-based fitness value associated with $population_j$ is denoted by EF_T^j, while $EF_{Toptimal}$ refers to the current optimal fitness value during evolution.

Table 4.1. *The Evolutionary Algorithm*

Function: GA_Mixture_Model (population) **returns** a *population with optimal number of clusters*
Inputs: *population*

Initialization: $j \leftarrow 0$; $population_j \leftarrow population$;

$population_{optimal} \leftarrow EM(population_j)$;

$EF_{Toptimal} \leftarrow EF_T^j$;

Repeat: $j \leftarrow j+1$;

Apply genetic operators to $population_{optimal}$ yielding $population_j$;

$population_j \leftarrow EM(population_j)$; Compute EF_T^j;

If $\left(EF_T^j > EF_{Toptimal}\right)$

do $population_{optimal} \leftarrow population_j$, $EF_{Toptimal} \leftarrow EF_T^j$;

Until: $EF_{Toptimal} > th_3$ or $j \geq th_4$ or $k = 0$

Return $population_{optimal}$

The algorithm combines iteratively EM steps with evolutionary operations; table 4.2 illustrates the EM sub-function. EM is first used to estimate the best parameters for an initial number of clusters. The evolutionary algorithm is then used to estimate the best number of clusters given the (best) parameters obtained after the EM steps. The outcome of the evolutionary algorithm is re-submitted to the EM algorithm, and the same two-stages process is repeated until the termination conditions are satisfied.

Table 4.2. *EM Algorithm Implementation*

Sub_Function: EM (population) **returns** a population
Inputs: a population of size k
Initialization: $r \leftarrow 0$;

$population_r \leftarrow population$; Compute the log-likelihood L_r;

Repeat:
$r \leftarrow r+1$; r stands for the number of iterations for EM;

Generate a new group of components $\{(\alpha_i^r, \mu_i^r, \Sigma_i^r) \mid 1 \le i \le k\}$ using E-step and M-step;

$population_r \leftarrow \{<\alpha_1^r, \mu_1^r, \Sigma_1^r>, <\alpha_2^r, \mu_2^r, \Sigma_2^r>, \ldots, <\alpha_k^r, \mu_k^r, \Sigma_k^r>\}$;

Calculate the log-likelihood L_r for $population_r$;

Until: $|L_r - L_{r-1}| < th_1$ or $r > th_2$

Return $population_r$

The termination conditions occur when the generated number of new clusters becomes 0 or specified maximum entropy fitness value or numbers of iterations are reached. Thresholds th_1 and th_2 correspond to the termination conditions associated with the EM algorithm: th_1 is a measure of the absolute precision required by the algorithm. If the variation in log likelihood between two consecutive steps of the EM algorithm is less than this value, then the algorithm will terminate. Threshold th_2 is the maximum number of iterations of EM. Thresholds th_3 and th_4 correspond to the termination conditions associated with the evolutionary algorithm: th_3 is the maximum entropy fitness value and th_4 is the maximum number of iterations for the evolutionary algorithm.

Previous work by Rudolph shows that a canonical genetic algorithm (CGA) will never converge to the global optimum, but "variants of CGAs that always maintain the best solution in the population, either before or after selection", will converge to the

global optimum [78]. Our algorithm ensures that the best population is always maintained after each selection and at the same time, increases the diversity of solutions by randomly selecting the genetic operations. As a result, the proposed evolutionary algorithm has inherently the capability to converge into the global optimal solution in theory. The empirical validation also confirms this in practice.

4.2.4 Empirical Validation

We conducted an empirical validation of our evolutionary algorithm, in which, we used two different datasets. In this evaluation, we investigated the relationship between the probability of genetic operations and convergence speed. The selection of genetic operation probabilities is based on experience. In this section, we introduce three criteria to decide the probability associated with genetic operations used to validate the proposed evolutionary algorithm.

We studied three performance metrics as suggested in [79], namely the Average number of Evaluations on Success (AES), the Success Rate (SR), and the Mean Best Fitness (MBF). AES measures the speed of optimization; more specifically it measures the number of iterations required for the successful runs to converge to the global optimum. SR measures the number of runs for the algorithm to successfully converge into the optimum solution; this evaluates the reliability of the algorithm. When the algorithm is unsuccessful in some runs, the MBF is used to measure how close the solution can reach the optimum. MBF is usually used to evaluate the pre-maturity of the solutions. In practice, if the algorithm converges to the global optimum, MBF is set to zero; otherwise MBF is calculated as the difference between the best fitness value and the value of the fitness function for the pre-mature solution. In this case, the best fitness value is set to 1 since the upper bound of the fitness function is 1.

The first dataset, generated synthetically, contains 300 data items. The data items

were independently generated using three bi-dimensional normal distributions $N_1(\mu_1,\Sigma_1)$, $N_2(\mu_2,\Sigma_2)$ and $N_3(\mu_3,\Sigma_3)$ with prior probabilities $\alpha_1=\alpha_2=\alpha_3=0.33$ and means $\mu_1=[6,3]^T$, $\mu_2=[3,6]^T$ and $\mu_3=[2,2]^T$. Corresponding covariance matrices are $\Sigma_1=\left[\begin{smallmatrix}0.6 & 0.15\\ 0.15 & 0.6\end{smallmatrix}\right]$, $\Sigma_2=\left[\begin{smallmatrix}0.4 & 0.0\\ 0.0 & 0.25\end{smallmatrix}\right]$ and $\Sigma_3=\left[\begin{smallmatrix}0.6 & 0.0\\ 0.0 & 0.3\end{smallmatrix}\right]$. We studied 10 different sets of genetic operation probabilities, specifically: merging probabilities varying from 0.3 to 0.99, splitting probabilities ranging from 0.699 to 0.009, and a constant deletion probability set to 0.001. For each set of genetic operation probabilities, we run the evolutionary algorithm 10 times during the test. The maximum number of iterations was set to 500 and the fitness threshold was set to 0.98. During the evolution, the initial number of clusters was empirically set to 30.

Table 4.3 summarizes the performance results obtained for the 100 test runs. With merging probability equal or greater to 0.70, the algorithm converges to the global optimum for all the test runs, by identifying the exact number of clusters contained in the test dataset. Figures 4.1 to 4.4 plot the algorithm's convergence speed under different sets of genetic operation probabilities.

Table 4.3. *Convergence Results for 100 Test Runs of the Evolutionary Algorithm for the Synthetic Dataset*

Probability for Genetic Operator			**MBF**	**AES**	**SR**
p_{mr}	p_s	p_d	Avg./Std.	Avg./Std.	
0.99	0.009	0.001	0/0	27/1	10
0.95	0.049	0.001	0/0	29/3	10
0.90	0.099	0.001	0/0	35/4	10
0.85	0.149	0.001	0/0	38/5	10
0.80	0.199	0.001	0/0	48/13	10
0.70	0.299	0.001	0/0	66/11	10
0.60	0.399	0.001	0.001/0.003	182/133	9
0.50	0.499	0.001	0.416/0.219	444/120	2
0.40	0.599	0.001	0.012/0.037	500/0	0
0.30	0.699	0.001	0.109/0.003	500/0	0

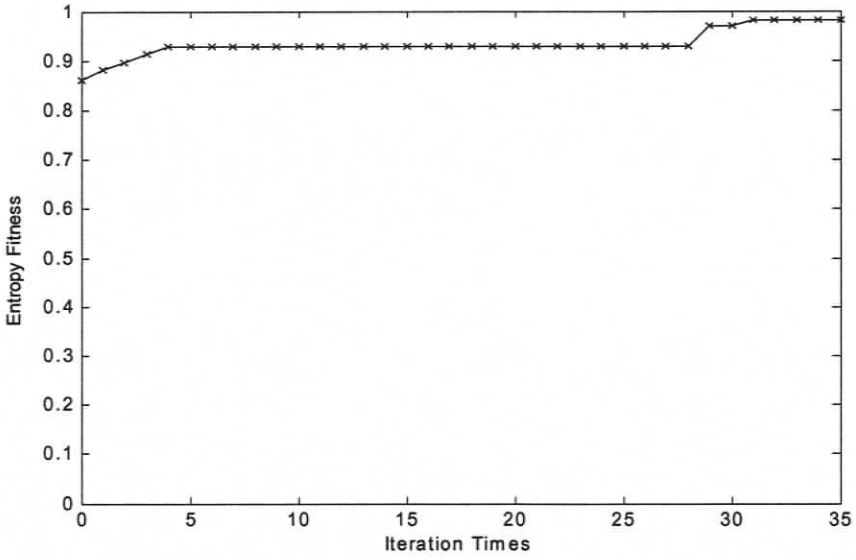


Figure 4.1. *Convergence Speed for* $P_{mr} = 0.95$.

The algorithm converges after 31 iterations.

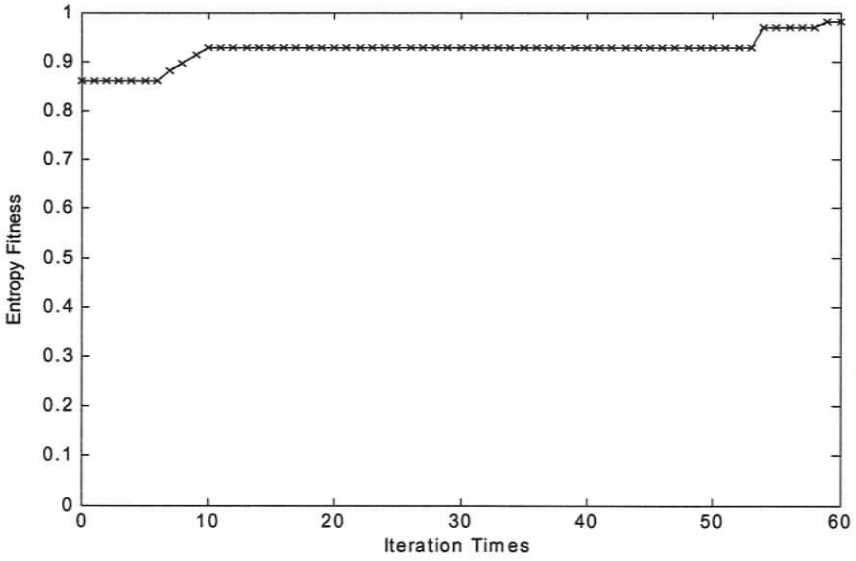


Figure 4.2. *Convergence Speed for* $P_{mr} = 0.80$.

The algorithm converges after 59 iterations.

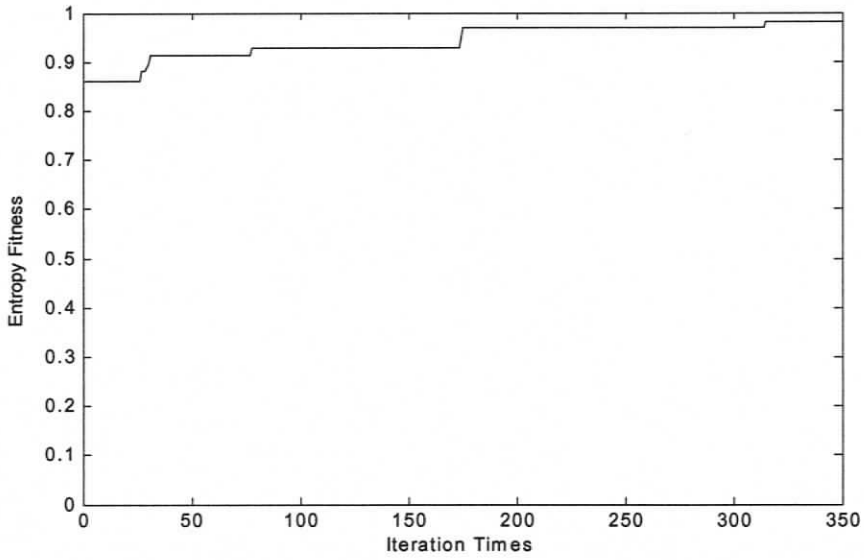


Figure 4.3. *Convergence Speed for* $P_{mr} = 0.60$.

The algorithm converges after 315 iterations.

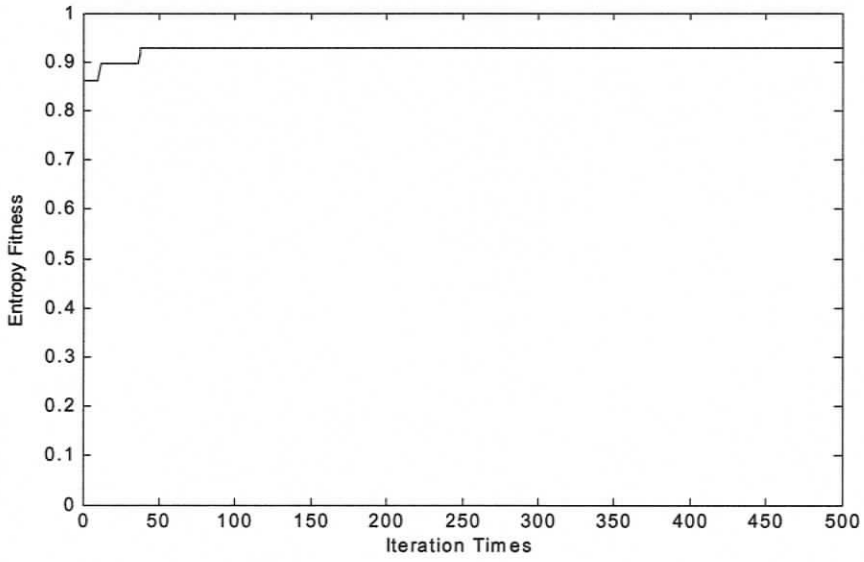


Figure 4.4. *Convergence Speed for* $P_{mr} = 0.50$.

The algorithm cannot converge.

The second dataset is the famous *Iris* dataset [80], which contains 3 clusters corresponding to different types of Iris plant, namely *Versicolor*, *Virginica* and *Setosa*. Each cluster contains 50 *4-dimensional* data items. Like with the first dataset, we studied 10 different sets of genetic operation probabilities, specifically merging probabilities varying from 0.3 to 0.99, splitting probabilities ranging from 0.699 to 0.009, constant deletion probability set to 0.001. Table 4.4 summarizes the performance results obtained for the 100 test runs for the Iris dataset. The algorithm converges to the global optimum for all the test runs for merging probability equal or above 0.60. Figures 4.5 to 4.8 plot the algorithm's convergence speed under different set of genetic operation probabilities.

The evaluation results show that our algorithm has the capability to converge into the global optimal solution with a high merging probability and a low splitting probability. It estimates exactly the optimal number of clusters for the synthetic dataset and the real dataset.

Table 4.4. *Convergence Results for 100 Test Runs of the Evolutionary Algorithm for the Iris Dataset*

Probability for Genetic Operator			**MBF**	**AES**	**SR**
p_{mr}	p_s	p_d	Avg./Std.	Avg./Std.	
0.99	0.009	0.001	0/0	30/1	10
0.95	0.049	0.001	0/0	34/4	10
0.90	0.099	0.001	0/0	35/3	10
0.85	0.149	0.001	0/0	45/10	10
0.80	0.199	0.001	0/0	49/9	10
0.70	0.299	0.001	0/0	74/19	10
0.60	0.399	0.001	0/0	124/45	10
0.50	0.499	0.001	0.231/0.087	500/0	0
0.40	0.599	0.001	0.249/0.00	500/0	0
0.30	0.699	0.001	0.249/0.00	500/0	0

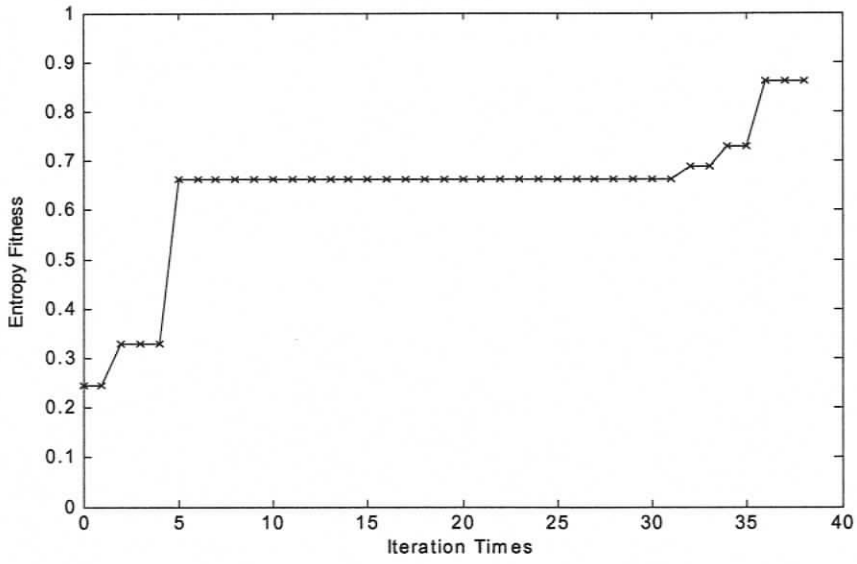


Figure 4.5. *Convergence Speed for* $P_{mr} = 0.95$.

The algorithm converges after 36 iterations.

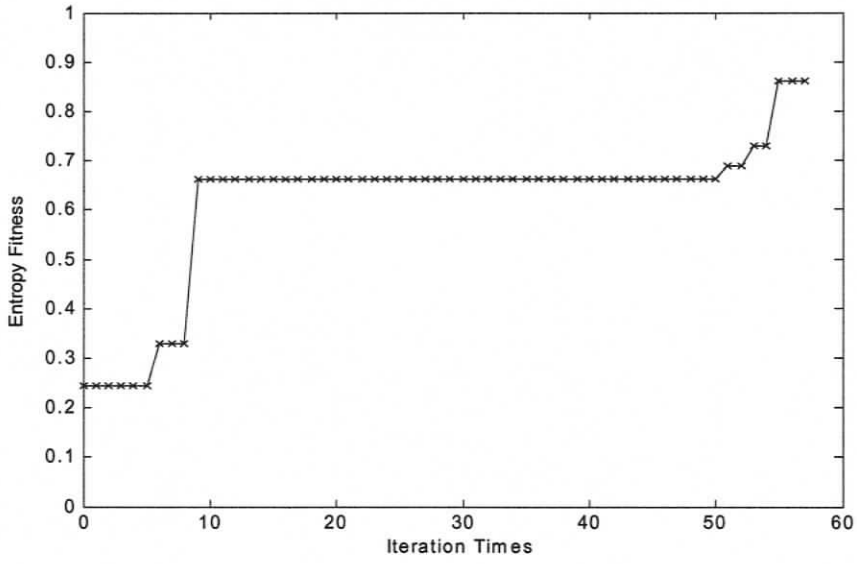


Figure 4.6. *Convergence Speed for* $P_{mr} = 0.80$.

The algorithm converges after 55 iterations.

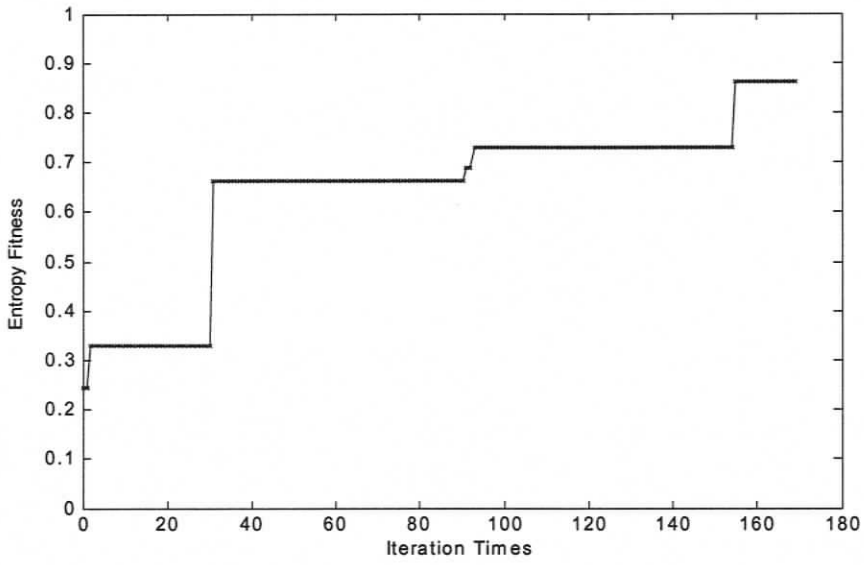


Figure 4.7. *Convergence Speed for* $P_{mr} = 0.60$.

The algorithm converges after 155 iterations.

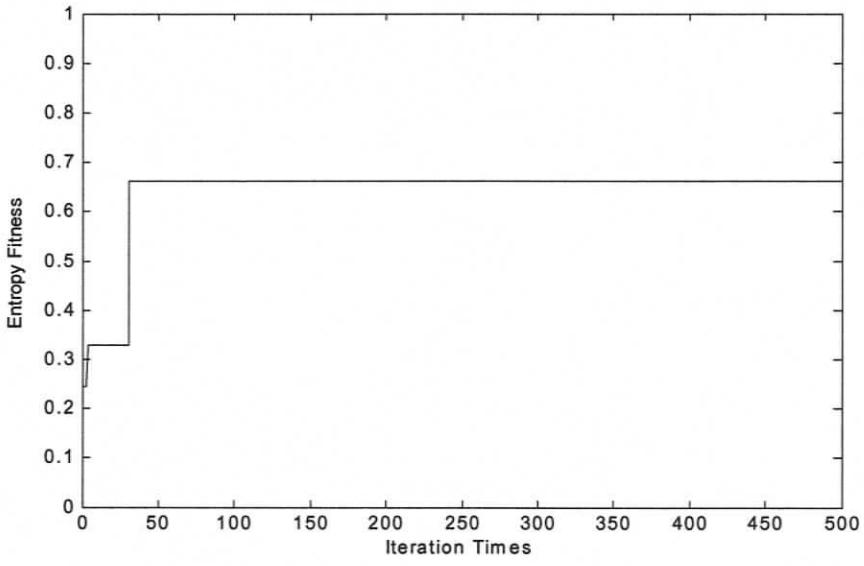


Figure 4.8. *Convergence Speed for* $P_{mr} = 0.50$.

The algorithm fails to converge.

4.2.5 Case Study on Computer Intrusion Detection

The purpose of this case study is to validate the performance of the algorithm with a large-scale dataset, which happens usually in intrusion detection. Using 1999 KDDCUP intrusion detection dataset, the case study also confirms that estimating exactly the optimal number of clusters improves significantly the performance of unsupervised anomaly detection.

As indicated in the first chapter, current unsupervised anomaly detection approaches assume that the data consists of two clusters, namely intrusion cluster and normal cluster, and also that the number of intrusion instances is less than the number of normal instances. These assumptions unavoidably will lead to a high false alert rate when all data instances in the dataset are normal.

We selected two days' data from the entire KDDCUP intrusion detection dataset. The first day includes only normal instances and the second day includes both intrusive and normal instances. As indicated in chapter 3, KDDCUP intrusion dataset consists of three groups of features, namely basic features, content features and traffic features. For our case study, we selected the traffic features subspace whose size is 19. More information about these features can be found in chapter 3. The number of connection records in the first day is 4904 normal instances. The number of connection records in the second day is 4972 instances, in which 4904 are normal and 68 are intrusive.

In practice, clustering a *19*-dimensional dataset is time consuming, hence, before clustering, it is necessary to reduce the feature space size by eliminating redundant information. We use Principal Component Analysis (PCA) to reduce the size of the feature space. Specifically, suppose $X \sim \{x_1, x_2, ..., x_{19}\}$ is the original *19*-dimensional data and its covariance matrix is denoted by S, the eigenvectors E and eigenvalues λ for the original dataset can be calculated by solving equation $(S - \lambda I)E = 0$. The new projected data set denoted by U is defined as: $U = E \times X$. The corresponding principal

component has the highest eigenvalues. After reducing the feature space, clustering techniques are used to classify the data and make intrusion decisions. Initially, we apply the plain k-means clustering algorithm to our two days data by assuming that the dataset consists of exactly two clusters, normal and intrusion; note that with k-means the number of clusters is selected randomly. Table 4.5 illustrates the results obtained with this approach. As illustrated in Table 4.5, when there is no intrusion in the dataset, the assumption that the number of clusters is 2 is not valid, and thus the detection system generates a false positive rate (FPR) of 18.1%, which is very high. The false positive rate is defined as the ratio of the number of normal instances detected as intrusions to the total number of normal instances.

Table 4.5. *Detection Result using Plain K-means Clustering Algorithm*

Detection Result / **Day**	Number of detected intrusive instances	Number of normal instances detected as intrusions	FPR (%)
1st Day	0	888	18.1
2nd Day	68	0	0.0

In order to address this limitation, we improve the detection framework by inserting our evolutionary algorithm before k-means clustering stage. In the evolutionary algorithm, we set the merging probability, splitting probability and deletion probability to 0.95, 0.049 and 0.001, respectively, according to the previous empirical validation results. Also, we set the maximum number of iterations and the fitness threshold to 500 and 0.98, respectively. Our algorithm exactly estimates the optimal number of clusters for both two days' dataset after a limited number of iterations. Figures 4.9 and 4.10 plot the algorithm's convergence speed. As a result, after applying k-means clustering, the FPR for the first day's data is decreased from 18.1% to 0%.

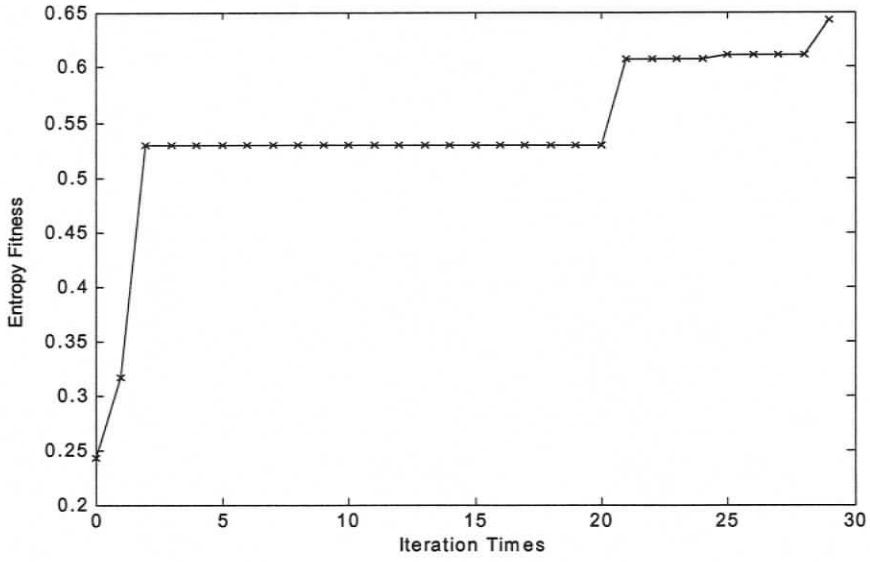


Figure 4.9. *Convergence Speed with 1st Day Data*

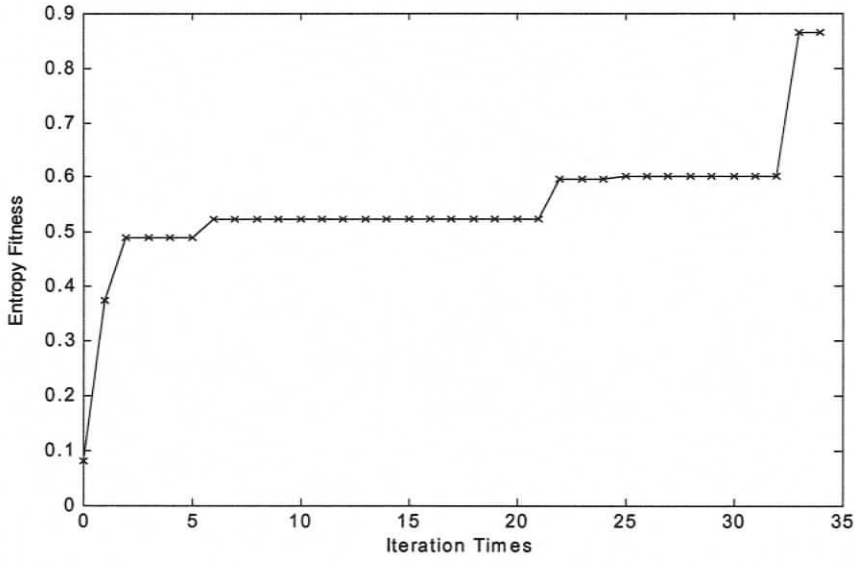


Figure 4.10. *Convergence Speed with 2nd Day Data*

4.3 Outlier Detection based on Mixture Resolving

We use Gaussian mixture model (GMM) to detect outliers from a given dataset. GMM assumes that data patterns in a dataset can be drawn from one of several distributions. As a result, the entire data collection is seen as a mixture of several Gaussian distributions. However, in some cases, there are few data patterns that are categorized into none of the components of the mixture distribution. These data patterns are naturally seen as the outliers (noisy data) for the dataset. We describe the outlier detection strategy using an example depicted in Table 4.6. Each element in Table 4.6 refers to a posterior probability for data pattern approximating to a Gaussian component. Based on the table, we can naturally infer that $data_1$ belongs to Gaussian distribution 1 denoted by N_1, because the posterior probability that $data_1$ belongs to N_1 is 1. Similarly, $data_2$ belongs to N_2 and $data_3$ belongs to N_3. In the case of $data_4$, the decision depends on the values of the p_i. For instance, if $p_4 > p_3 > p_2 > p_1$ we can conclude that $data_4$ belongs to N_4. For $data_5$ the posterior probability approximating to each of the GMM components is null; hence $data_5$ is the outlier.

Table 4.6. *Posterior Possibility for an Example Dataset*

Gaussian Component / Data Pattern	N_1	N_2	N_3	N_4
$data_1$	1	0	0	0
$data_2$	0	1	0	0
$data_3$	0	0	1	0
$data_4$	p_1	p_2	p_3	p_4
$data_5$	0	0	0	0

The proposed outlier detection algorithm is based on the posterior probability generated by EM algorithm. The posterior probability describes the likelihood that the data pattern approximates to a specified Gaussian component. The greater the posterior

probability for a data pattern belonging to a specified Gaussian component, the higher the approximation is. As a result, data are assigned to the corresponding Gaussian components according to their posterior probabilities. However, in some cases there are some data patterns whose posterior probability of belonging to any component of GMM is very low or close to zero. These data are naturally seen as the outliers or noisy data. We illustrate the detailed outlier detection algorithm in Table 4.7.

Thresholds th_5 and th_6 correspond to the termination conditions associated with the outlier detection algorithm: th_5 is a measure of the absolute precision required by the algorithm and th_6 is the maximum number of iterations required by the algorithm. Threshold $outlier_{thres}$ refers to the minimum mixing proportion. Once the mixing proportion corresponding to a specified Gaussian component is below $outlier_{thres}$, the posterior probability for the data pattern to belong to this Gaussian component will be set to 0.

4.4 I-means Algorithm

Table 4.8 depicts the I-means algorithm. The algorithm combines the evolutionary approach to mixture modeling, the outlier detection strategy, and k-means clustering. The evolutionary approach generates an optimal estimate of the initial number of clusters and the cluster centers, which are used by k-means for actual data clustering. Outlier detection based on mixture modeling is used to find and remove the noisy data from the original dataset before the data are processed by k-means.

Thresholds th_7 and th_8 correspond to the termination conditions for k-means algorithm: th_7 is a measure of the absolute precision required by k-means. The k-means algorithm terminates when the variation in distance between center points in two consecutive steps of the algorithm are less than threshold th_7. Threshold th_8 is the maximum number of iterations.

Table 4.7. *The Proposed Outlier Detection Algorithm*

Function: GMM_Outlier_Detection (dataset and k) **returns** outlier data set

Inputs: *dataset* $X \sim \{x_n \mid n = 1,2,...,N\}$, *and the estimated number of components k*

Output: Outlier Data Set

Initialization:

Outlier Data Set = ϕ; $j \leftarrow 0$;

Initial parameters $\{\alpha_i^j, \mu_i^j, \nu_i^j\}$, $1 \le i \le k$, are randomly generated;

Calculate the initial log-likelihood L_j;

Repeat:

For $1 \le i \le k$, $1 \le n \le N$

If ($\alpha_i^j \ge outlier_{thres}$) **then** compute posterior probability $p_j(i \mid x_n)$;

Else $p_j(i \mid x_n) = 0$;

$j \leftarrow j+1$;

Re-estimate $\{\alpha_i^j, \mu_i^j, \nu_i^j\}$ by using $p_{j-1}(i \mid x_n)$, $1 \le i \le k$, $1 \le n \le N$;

Calculate the current log-likelihood L_j;

Until: $|L_j - L_{j-1}| < th_5$ or $j > th_6$

For $1 \le i \le k$, $1 \le n \le N$

If ($p_{j-1}(i \mid x_n) = 0$), assign x_n to Outlier Data Set

Return Outlier Data Set;

Table 4.8. *The I-means Algorithm*

Function I-means (data) **returns** clusters

Inputs: Collection of data samples

Initialization: Randomly generate an initial population $population_0$ with initial number of components k; $q \leftarrow 0$;

$population_{optimal} \leftarrow GA\_Mixture\_Model(population_0)$

$OutlierData = GMM\_Outlier\_Detection(population_{optimal}, k_{optimal})$

Remove outlier data from the original dataset and initialize data clustering based on $population_{optimal}$: the number of clusters corresponds to $k_{optimal}$, and the means of each component in $population_{optimal}$ corresponds to the initial cluster centers c_i, $1 \le i \le k_{optimal}$

Repeat: $q \leftarrow q+1$

Assign data samples to clusters by determining the closest cluster center points. Calculate the new center point c_{inew} for each cluster i.

Until: $|c_{inew} - c_i| < th_7$ or $q > th_8$

Return the clusters obtained.

4.5 Conclusions

We propose in this chapter a new hybrid clustering algorithm, named I-means. I-means algorithm improves the performance of the classical k-means algorithm by optimally adjusting the pre-determined parameters. I-means is composed by an evolutionary approach to mixture resolving and a revised k-means algorithm. The evolutionary approach is used to search the optimal number of clusters, and generate the center points for initial clusters. Empirical validations with synthetic data as well as real dataset show that the evolutionary algorithm can exactly estimate the global optimal number of clusters after few generations. This improves significantly the performance of the unsupervised anomaly detection process.

Chapter 5

Characterizing Anomalous Network Activities

In this dissertation we primarily focus on the detection of multiple-connection based attacks, and thus the main data source for our approach consists of network packets. There is a wide range of packet information. In order to achieve effective and efficient detection, we need to select a limited but sufficient number of features. In 1999 KDDCUP intrusion detection dataset [8], Lee et al. characterize intrusion data into three sets of features, namely basic features, content features and traffic features [27]. They describe network connections using a total of 41 features. These, of course, cover the attacks as much as possible. However, defining packets with so many features makes online intrusion detection almost impossible.

In this context, we derive a collection of utility functions through empirical observations of network traffic behaviors. We name these utility functions IP Weight metrics. IP Weight metrics assess the degree of anomalousness of IP packet flows, allowing substantial reduction of the feature space. In the remainder of this chapter, we summarize our empirical observations, and derive the initial set of features. Then we define IP Weight metrics, and study their feature space reduction capability by comparing them with PCA technique, which is one of the most popular feature extraction techniques.

5.1 Empirical Observations and Features Selection

5.1.1 Feature Space

Based on the standard characteristics of IP packets on networks, we define a set of features to describe a single IP packet. Let us denote by P the set of IP packets. A packet $p \in P$ can be represented as a feature vector consisting of 13 fields described in Table 5.1.

Table 5.1. *Basic Features for an IP Packet*

Representation	Meaning
t	time stamp corresponding to the appearing time of the packet in a certain time window
dip	destination IP address; this usually corresponds to the address of a host we want to protect
dp	destination port
sip	source IP address
sp	source port
ihl	length of IP header for the IP packet
pktl	packet length including header and data
ident	an integer that identifies the current data in a packet, which can be used to piece together data fragments
fragoff	the offset of the IP packet indicating the position of the fragment's data relative to the beginning of the fragment data in the original data
pro	upper-layer program receiving the incoming IP packets after IP processing is complete
thl	length of TCP header
seq	data location of the TCP segment
ack	number of data received by the destination host
$\text{traffic}_{\text{in}}$	the appearing frequency of packets flowing to the destination IP address over the observation time window
$\text{traffic}_{\text{out}}$	the appearing frequency of packets outgoing from the destination IP address over the observation time window

We define a packet flow as a group of packets flowing to a specified destination during a specified observation period. Let us denote by F the set of all packet flows. A packet flow $f \in F$ is defined as a *6*-dimensional vector:

$f =< g, t, \delta_t, dip, nop, nodp_{\max} >$, where

- $g \in \wp(P)$ is the set of packets observed and $\wp(P)$ denotes the power set of P;
- t is the starting time of the observation;
- δ_t is the observation time window;
- dip refers to the destination IP address that we want to protect;
- nop stands for the total number of packets in the flow;
- $nodp_{\max}$ means the maximum number of packets over all destination ports in the packet flow.

5.1.2 Empirical Observations

The study of anomalous network activities could be achieved either empirically through pilot studies or based on intuitive analysis and understanding of the domain. We use both approaches in order to make some observations that serve as basis for characterizing anomalous activities. Our goal is not to provide a complete description of anomalous activities, which is unrealistic; instead, we are interested in identifying a limited number of facts that can be used for effective and efficient detection. Specifically, we base our work on four intuitive observations:

1. Network traffic involving a high frequency of packets flowing to the same destination address within a very short time period, or network traffic involving a high frequency of packets flowing to the same destination address with same destination port during a very short time period, is most likely anomalous.

2. During normal network usage, for those network traffic flowing to the same destination over a given time period, the likelihood of their corresponding destination ports to be randomly distributed is low. The same observation applies for their corresponding source IP addresses and source ports.
3. Network traffic containing one or several packets that violate basic structural rules of packets is likely anomalous.
4. For a normal host, its incoming traffic and (matching) outgoing traffic are most likely similar.

Observation 3 simply derives from the TCP/IP protocol specification. To confirm other three observations, we conducted a pilot study, in which network data are collected over three weeks: two weeks for normal network usage and one week for anomalous network usage including some known attacks. Figure 5.1 illustrates the live networking topology for our pilot study. The destination server was deployed behind the firewall. Network traffic flowing to the server includes the internal traffic from internal LAN and the public traffic from the Internet. We ensured that the traffic over two weeks' normal usage was normal by auditing the after-event logs of the firewall. During the anomalous network usage over one week, the server operated as a honey pot [81]. Several utilities including known vulnerabilities were purposely installed on the server and exposed to the public.

Figures 5.2a to 5.7b illustrate the analysis made from the collected data. In these figures, we denote the IP address of the server by *dip* and the size of the observation time window by δ_t. The frequency of IP packets flowing to the same destination IP address during the observation time window in two weeks normal network usage and one week anomalous network usage are plotted in Figures 5.2-a and 5.2-b respectively. Figures 5.3-a and 5.3-b plot the maximum frequency of packets flowing to same destination IP address with same destination port during the observation time window, respectively. In these figures, the frequency of normal packet flows follow a regular

pattern, while the frequency of anomalous packet flows is persistently high. These confirm observation 1.

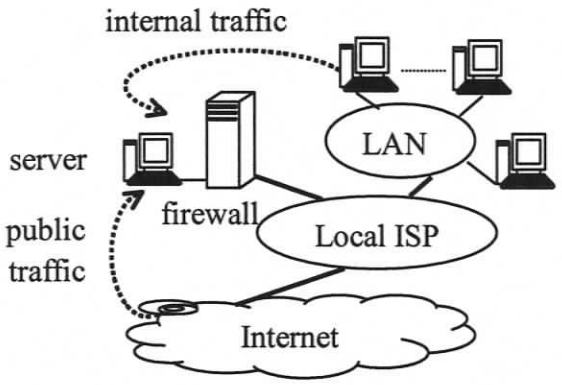


Figure 5.1. *Networking Environment for Empirical Observations*

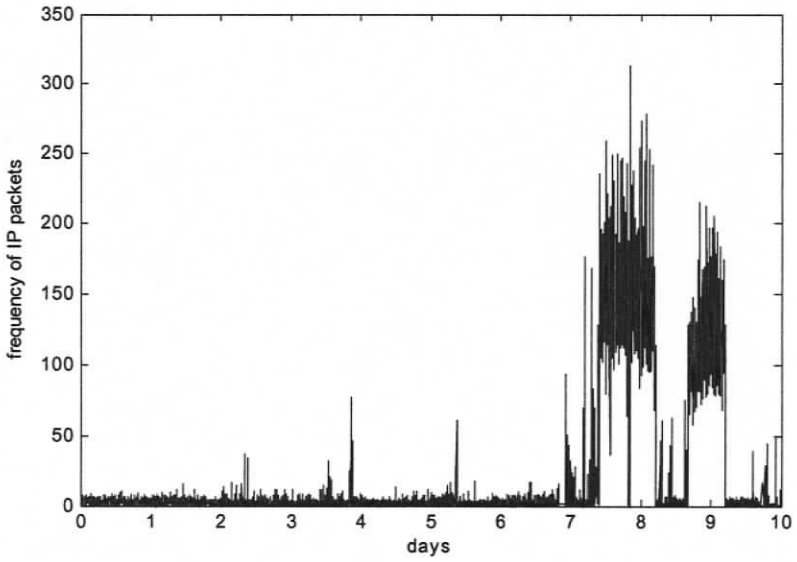


Figure 5.2-a. *Frequency of Normal Packets Flowing to Same Destination IP Address During the Observation Time Window*

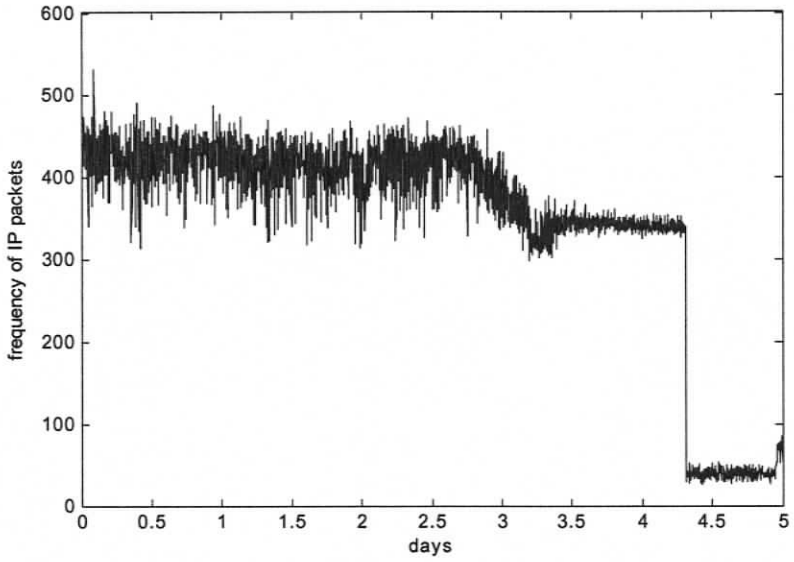


Figure 5.2-b. *Frequency of Anomalous Packets Flowing to Same Destination IP Address During the Observation Time Window*

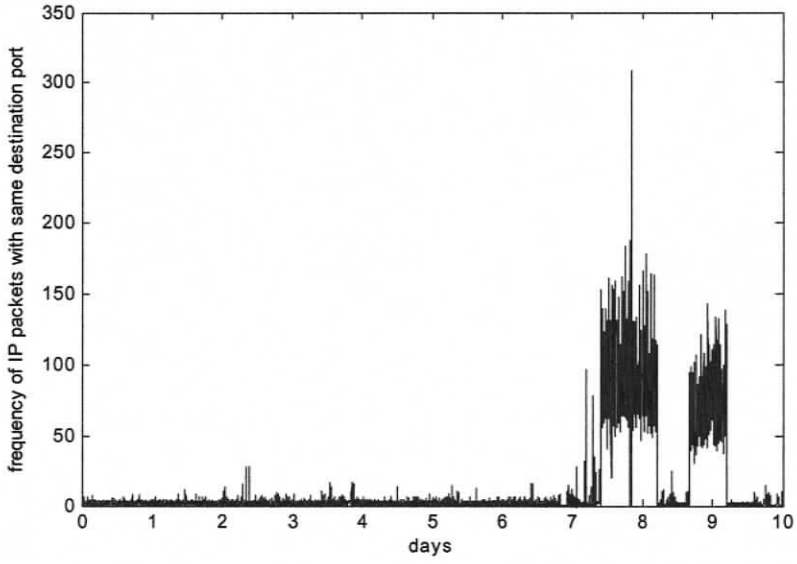


Figure 5.3-a. *Maximum Frequency of Normal Packets Flowing to Same Destination IP Address with Same Destination Port During the Observation Time Window*

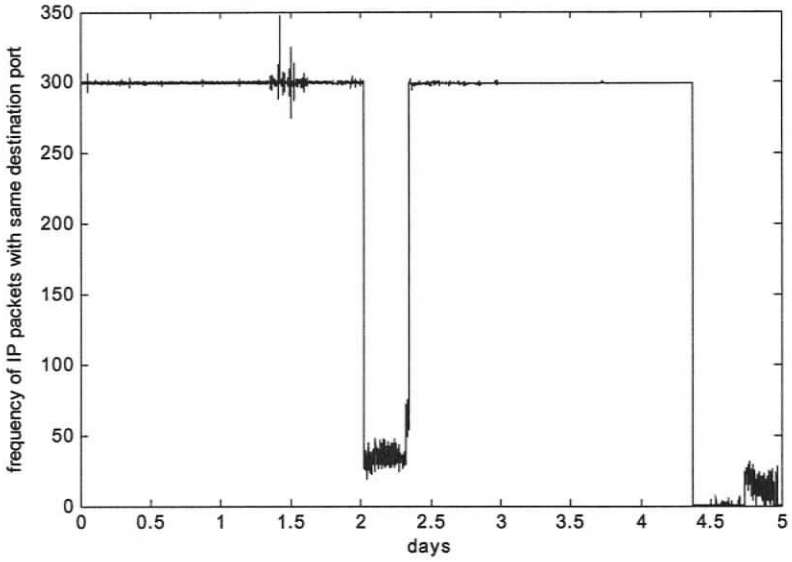


Figure 5.3-b. *Maximum Frequency of Anomalous Packets Flowing to Same Destination IP Address with Same Destination Port During the Observation Time Window*

The randomness of destination ports in packet flows with same destination port during the observation time window over two weeks' normal network usage and one week's anomalous network usage are plotted in Figures 5.4-a and 5.4-b, respectively. Similarly, Figures 5.5-a and 5.5-b plot the randomness of source ports in corresponding packet flows. Figures 5.6-a and 5.6-b plot the randomness of source IP addresses. In these graphs, the randomness of destination ports, source IP addresses and source ports of anomalous packet flows is more often higher than those of normal packet flows, which supports observation 2.

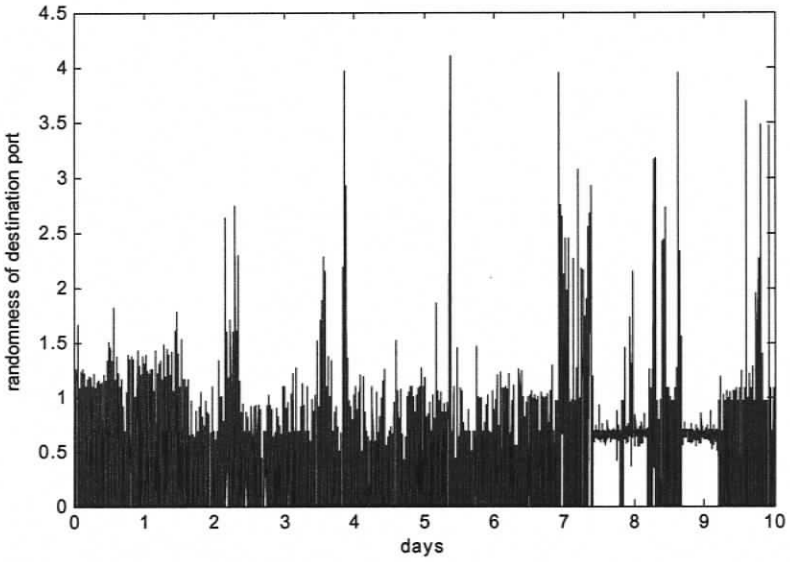


Figure 5.4-a. *Randomness of Destination Port for Normal Packets Flowing to Same Destination IP Address During the Observation Time Window*

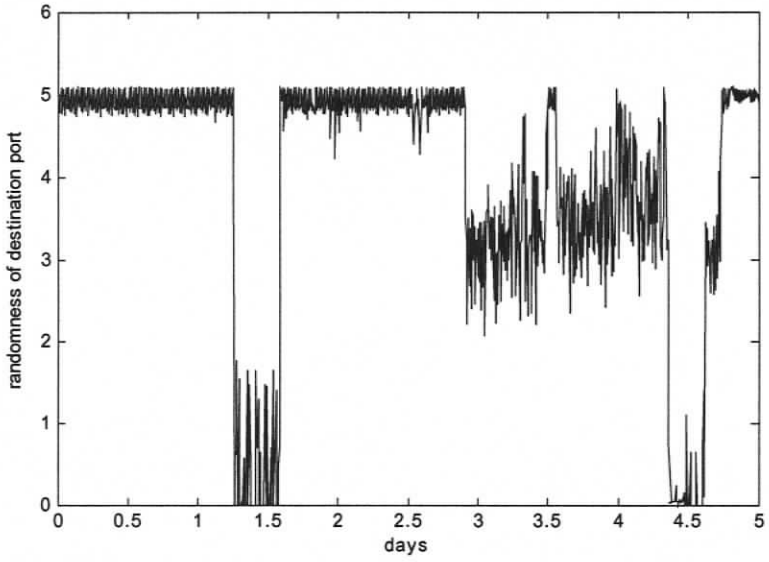


Figure 5.4-b. *Randomness of Destination Port for Anomalous Packets Flowing to Same Destination IP Address During the Observation Time Window*

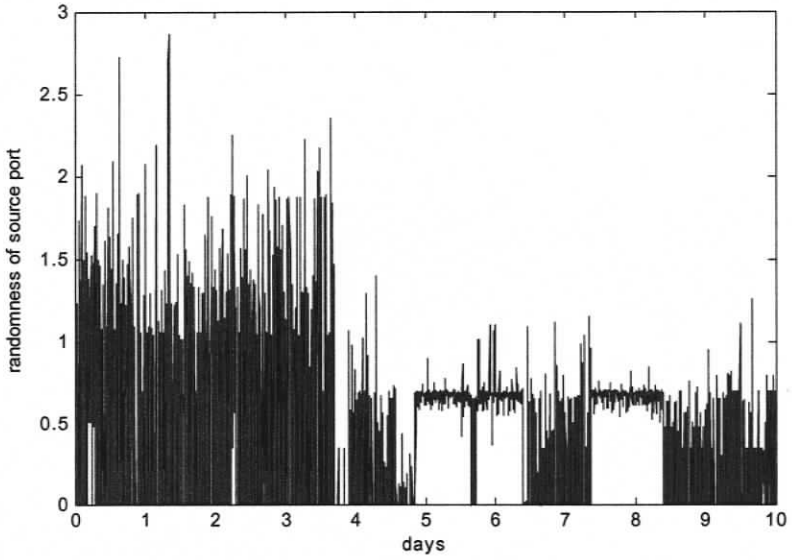


Figure 5.5-a. *Randomness of Source Port for Normal Packets Flowing to Same Destination IP Address During the Observation Time Window*

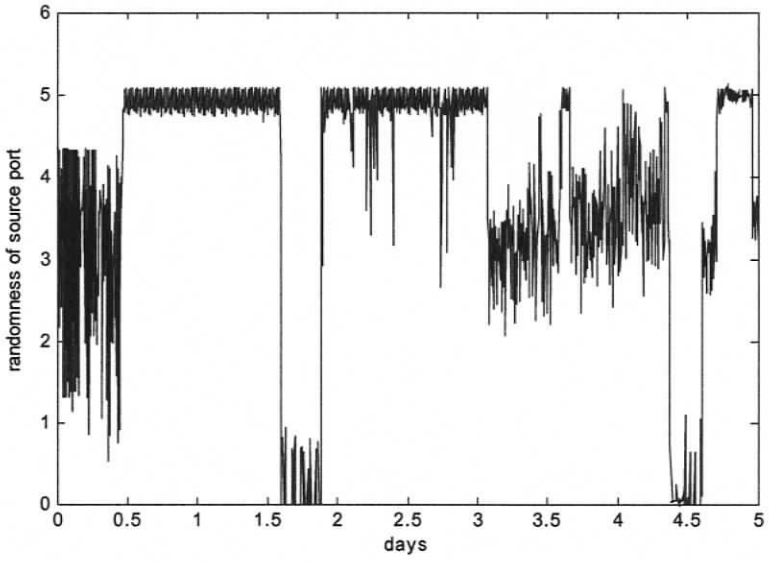


Figure 5.5-b. *Randomness of Source Port for Anomalous Packets Flowing to Same Destination IP Address During the Observation Time Window*

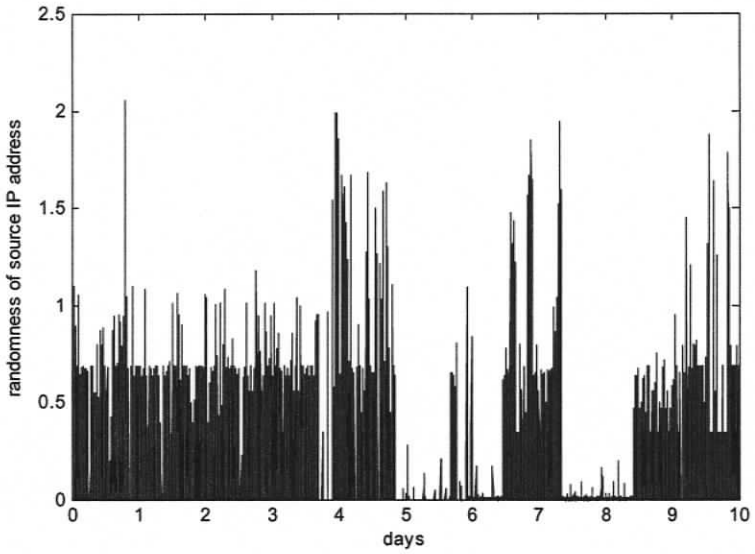


Figure 5.6-a. *Randomness of Source IP Address for Normal Packets Flowing to Same Destination IP Address During the Observation Time Window*

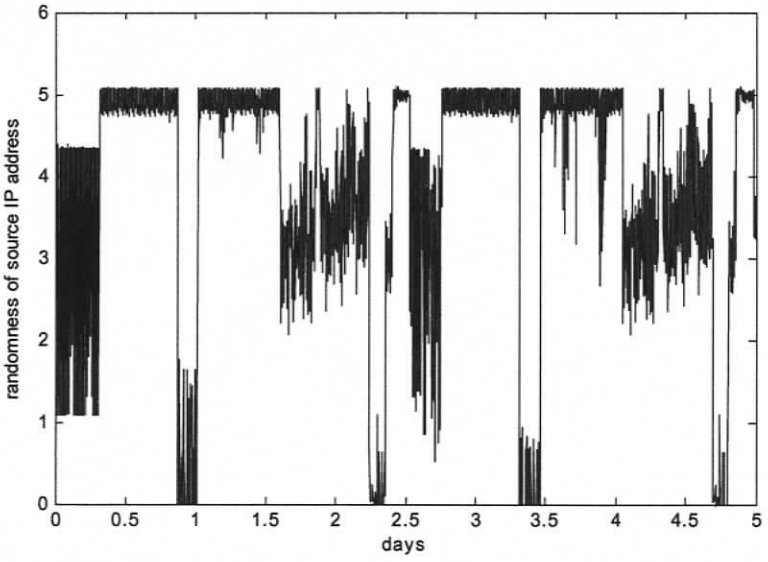


Figure 5.6-b. *Randomness of Source IP Address for Anomalous Packets Flowing to Same Destination IP Address During the Observation Time Window*

For the server we protect, Figure 5.7-a and 5.7-b plot the load ratio of its corresponding incoming traffic to outgoing traffic over two weeks normal network usage and one week anomalous network usage. Both the incoming traffic and outgoing traffic correspond to the same destination IP address. The load ratio for the specified destination IP address of anomalous traffic is much higher than those of normal traffic, which confirms the observation 4.

Exceptions for these observations may occur in some special cases. For instance, a normal web sever working on high traffic load obviously violate the first observation. However, this kind of violations has a slight impact on the final intrusion decisions, which is confirmed by later experimental evaluation. We derive empirical utility functions of network traffic by considering only the most generic cases.

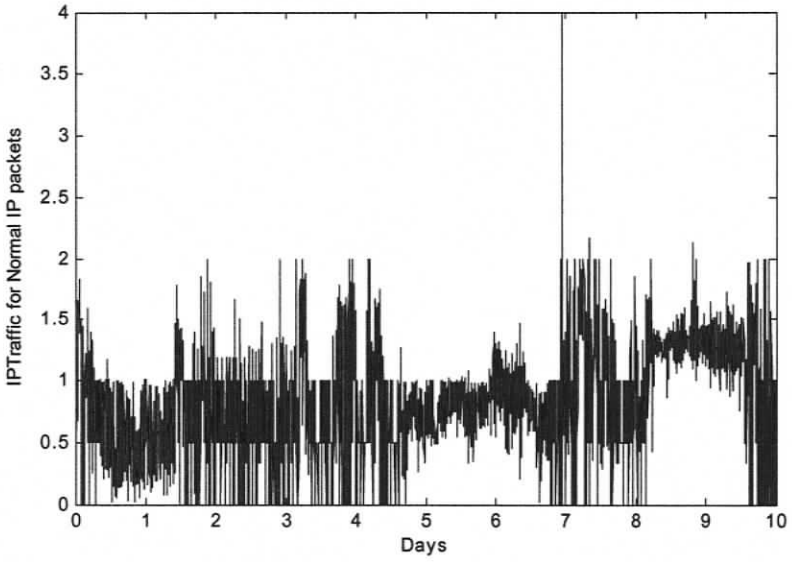


Figure 5.7-a. *Load Ratio for Normal Packets with Same Destination IP Address During the Observation Time Window*

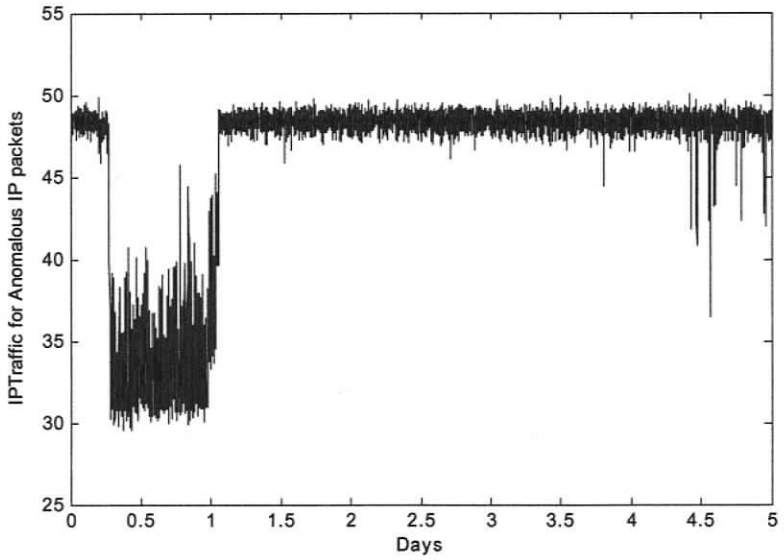


Figure 5.7-b. *Load Ratio for Anomalous Packets with Same Destination IP Address During the Observation Time Window*

5.2 Feature Extraction Using IP Weight Metrics

In order to achieve efficient and effective detection, we extract a limited feature set consisting of four dimensions by applying some transformations on the initial feature set denoted by F. Specifically, four ordinal utility functions are defined to characterize the degree of anomalousness of network activities, and each of them maps several features in F into a single numerical feature [82]. We name the four utility functions IP Weights. Each of the functions measures empirically anomalousness along one of four dimensions, namely frequency, randomness, structure and load. We denote by $ipw_{freq} : F \to R$, $ipw_{ran} : F \to R$, $ipw_{str} : F \to R$ and $ipw_{load} : F \to R$ respectively the frequency, randomness, structure and load component of IP Weights, where R is the set of real numbers. The underlying consequence of the empirical observations made earlier is that the greater the IP Weights, the more anomalous the packet flows. In the rest of this section, we introduce the IP weight measures and present the results of the comparison of

our approach with PCA, which shows that our approach provides the same capability as PCA but in a faster and simpler way.

5.2.1 Frequency-based Feature Extraction

Given a packet flow $f \in F$ with same dip during δ_t (dip stands for destination IP address of the host we want to protect), we denote by x_1 and x_2 the appearing frequency of IP packets during δ_t and the maximum appearing frequency of IP packets over all dp during δ_t, respectively. Thus, we have:

$$x_1 = \frac{nop}{\delta_t} \text{ and } x_2 = \frac{nodp_{\max}}{\delta_t}$$

Empirical observations show that the greater the value of x_1 and x_2, the more likely the corresponding network packet flow is anomalous. Both x_1 and x_2 contribute to some extent to the anomalousness of network traffic. As a result, we define ipw_{freq} by adopting a polynomial representation, which is expressed as follows:

$$ipw_{freq}(f) = (x_1 f_1(x_1,x_2) + x_2 f_2(x_1,x_2)) g(x_1,x_2) \qquad [5.1]$$

Where $f_1 : R \times R \to [0,1]$ and $f_2 : R \times R \to [0,1]$ are two numerical functions that represent the contributions of x_1 and x_2 respectively; $g : R \times R \to R$ is a numerical function adjusting the value of ipw_{freq}, which is selected as $\frac{x_2}{x_1}$. Given x_1 and x_2, we always have $x_2 \le x_1$ and $f_1(x_1,x_2) + f_2(x_1,x_2) = 1$. By selecting f_2 as $\frac{x_2}{x_1}$, we can derive f_1 as $1 - \frac{x_2}{x_1}$. By substituting f_1, f_2 and g, we can express the empirical

utility function ipw_{freq} as follows:

$$ipw_{freq}(f) = \left(x_1 - x_2 + x_1^{-1}x_2^2\right)\frac{x_2}{x_1} \qquad [5.2]$$

5.2.2 Randomness-based Feature Extraction

The entropy is used to measure the randomness of variables according to information theory. Given a packet flow $f \in F$ with same dip during δ_t, the randomness of their corresponding source IP addresses, ports, and destination ports is denoted by H_{sip}, H_{sp} and H_{dp}, respectively. This is formulated by the following three expressions:

$$H_{sip}(f) = -\sum_{sip} p(sip)\log_2 p(sip)$$

$$H_{sp}(f) = -\sum_{sp} p(sp)\log_2 p(sp)$$

$$H_{dp}(f) = -\sum_{dp} p(dp)\log_2 p(dp)$$

Where $p(sip)$ refers to the appearing probability associated with source IP addresses sip, which is computed by taking the ratio of number of packets with specified source IP address sip by the total number of packets observed in the flow f. Using the same approach, we can compute the $p(sp)$ and $p(dp)$, which refer to the probabilities associated with source port sp and destination port dp, respectively.

Since each of these features (i.e. the randomness) has the same contribution to the anomalousness of network traffic, we combine them into a single feature by selecting their maximum value. Consequently, the utility function ipw_{ran} is defined as follows:

$$ipw_{ran}(f) = \max(H_{sip}(f), H_{sp}(f), H_{dp}(f)) \qquad [5.3]$$

5.2.3 Load-based Feature Extraction

For a normal target host dip, the magnitudes of its incoming and (matching) outgoing traffic are most likely similar. However, when denial of service (DoS) attacks are used to compromise this host, its outgoing traffic is usually low compared to its incoming traffic. Empirical observations made earlier confirm this.

Given a packet flow $f \in F$, we define two new features $traffic_{in}$ and $traffic_{out}$ to represent respectively the appearing frequency of packets flowing to and outgoing from dip over δ_t. The ratio between $traffic_{in}$ and $traffic_{out}$ describes the load balance of the host we want to protect. Thus the utility function ipw_{load} is defined as follows:

$$ipw_{load}(f) = \frac{traffic_{in}}{traffic_{out}} \qquad [5.4]$$

5.2.4 Structure-based Feature Extraction

Normal IP packets must satisfy some basic structural rules. In most cases, TCP/IP implementation will check the structures of packets. However, in some cases, the structure violation is hardly detected by the TCP/IP stack implementation. Based on general knowledge of the structure of IP packets, we define a limited number of rules, which are usually satisfied by any pair of packets belonging to the same TCP/UDP connection. Table 5.2 describes the corresponding rules.

Table 5.2. *Rule-base for Packets in the Same Connection*

Index	Rule
1	*if* $ident_1 = ident_2$, *then* $fragoff_1 \neq fragoff_2$
2	*if* $ident_1 > ident_2 + 1$ *then* $seq_1 \geq seq_2$ *and* $ack_1 \geq ack_2$
3	*if* $ident_2 > ident_1 + 1$ *then* $seq_2 \geq seq_1$ *and* $ack_2 \geq ack_1$
4	*if* $ident_1 = ident_2 + 1$ *then* $seq_1 = seq_2 + pktl_2 - ihl_2 - thl_2$ *and* $ack_1 \geq ack_2$
5	*if* $ident_2 = ident_1 + 1$ *then* $seq_2 = seq_1 + pktl_1 - ihl_1 - thl_1$ *and* $ack_2 \geq ack_1$

Given a packet flow $f \in F$ with same *dip* during δ_t and a pair of packets $p_1, p_2 \in g$ observed during δ_t, we may consider the following three scenarios:

1. Both packets belong to the same connection and violate the rules.
2. Both packets belong to the same connection and satisfy the rules.
3. The two packets belong to different connections.

In the second and third case, we cannot conclude on the anomalousness of the traffic; we assign a null value as their contribution to the IP weight; if all the packets pairs fall in these cases, we obtain $ipw_{str}(f) = 0$. In the first case, we can conclude that the traffic is anomalous. However, establishing online that two packets belong to the same connection with certainty remains a difficult task.

So we use in this work an approximate notion of connection-similarity, which is based on the following precondition: IP packets belonging to the same TCP/UDP connection must have *the same source IP addresses and ports, the same destination IP addresses and ports, and the same protocol types*. We can be certain IP packets that do not satisfy this precondition do not belong to the same TCP/UDP connection. However, we cannot be certain the IP packets satisfying the precondition belong to the same connections. The standard specification and empirical observation show that given two packets p and q which satisfy the connection-similarity precondition, the smaller the difference between identification fields of packets p and q, the higher the probability

of two packets p and q belonging to the same connection. Based on this consideration, we give the following definition for ipw_{str}:

$$ipw_{str}(f) = \frac{1}{nop(nop-1)} \sum_{p \in g} \sum_{\substack{q \in g, \\ p \neq q}} \varepsilon_{pq} e^{-|ident(p)-ident(q)|} \qquad [5.5]$$

Where $|ident(p) - ident(q)|$ stands for the absolute value of the difference between identification fields of packets p and q; ε_{pq} is a positive integer, which takes the following values:

- $\varepsilon_{pq} = 1$ if packets p and q belong to the same TCP/UDP connection (i.e. satisfying the connection-similarity precondition) and violate the rule base simultaneously.
- $\varepsilon_{pq} = 0$ otherwise.

Coefficient $nop(nop-1)$ is a normalizing factor, which corresponds to the total number of packets pairs considered; note that nop stands for the total number of packets in the flow f.

5.3 Examples of IP Weights Computation

We present an example illustrating how to calculate IP weight measures. Figure 5.8 illustrates the IP packets flowing to destination IP address "192.168.1.106" during 2 seconds. We compute IP weights for these packets.

From Figure 5.8, we know that the total number of packets during 2 seconds is 17, that is $nop = 17$ and $\delta_t = 2$. In the total of 17 packets, 12 packets are directed to destination port 135 and 5 packets are directed to destination port 137. As a result, the maximum number of packets over all destination ports is 12, that is $nodp_{\max} = 12$. Thus

[Time]	[SIP]	[SP]	[DIP]	[DP]	[Protocol]	[Identification]
8:30:51	24.108.249.169	3712	192.168.1.106	135	TCP	36283
8:30:51	24.108.249.169	3712	192.168.1.106	135	TCP	36296
8:30:51	24.108.249.169	3712	192.168.1.106	135	TCP	36297
8:30:51	24.108.249.169	3719	192.168.1.106	135	TCP	36298
8:30:51	24.108.249.169	3712	192.168.1.106	135	TCP	36309
8:30:51	24.108.249.169	3719	192.168.1.106	135	TCP	36311
8:30:52	24.108.249.169	3719	192.168.1.106	135	TCP	36312
8:30:52	24.108.249.169	3719	192.168.1.106	135	TCP	36320
8:30:52	24.108.249.169	3719	192.168.1.106	135	TCP	36321
8:30:52	24.108.249.169	3726	192.168.1.106	135	TCP	36322
8:30:52	24.108.249.169	3719	192.168.1.106	135	TCP	36325
8:30:52	24.108.249.169	3726	192.168.1.106	135	TCP	36328
8:30:52	24.108.249.169	3726	192.168.1.106	137	TCP	36329
8:30:52	24.108.249.169	3726	192.168.1.106	137	TCP	36333
8:30:52	24.108.249.169	3726	192.168.1.106	137	TCP	36334
8:30:52	24.108.249.169	3726	192.168.1.106	137	TCP	36348
8:30:52	24.108.249.169	3726	192.168.1.106	137	TCP	36349
.....						

Figure 5.8. *Examples of IP Packets Flowing to Protected Host During 2 Seconds*

we have:

$$x_1 = \frac{nop}{\delta_t} = \frac{17}{2} = 8.5 \quad \text{and} \quad x_2 = \frac{nodp_{\max}}{\delta_t} = \frac{12}{2} = 6$$

We compute the frequency component of IP weight metrics according to equation [5.2], that is $ipw_{freq}(f) = \left(x_1 - x_2 + x_1^{-1}x_2^2\right)\frac{x_2}{x_1} = 4.76$.

Only one source IP address "24.108.249.169" appears in the packet flows, thus $p(sip) = 1$ and $H_{sip}(f) = 0$; Three source ports appear in the packet flows, namely 3712, 3719 and 3726. Therefore, $p(sp = 3712) = \frac{4}{17}$, $p(sp = 3719) = \frac{6}{17}$ and $p(sp = 3726) = \frac{7}{17}$. As a result, $H_{sp}(f) = -\sum_{sp} p(sp)\log_2 p(sp) = 1.5476$. Similarly, two

destination ports appear in the packet flows, namely 135 and 137. Hence, $p(dp=135)=\frac{12}{17}$, $p(dp=137)=\frac{5}{17}$ and $H_{dp}(f)=-\sum_{dp} p(dp)\log_2 p(dp)=0.874$. As a result, according to equation [5.3], $ipw_{ran}(f)=1.5476$.

We also record that the number of outgoing packets from 192.168.1.106 during 2 seconds is 18. Therefore, $traffic_{in}=\frac{17}{2}=8.5$ and $traffic_{out}=\frac{18}{2}=9$. According to equation [5.4], we have $ipw_{load}(f)=\frac{traffic_{in}}{traffic_{out}}=0.944$.

We use $\{p_1, p_2 ... p_{17}\}$ to label these 17 packets. From Figure 5.8, we find that there are 30 pairs of packets satisfying the connection-similarity precondition. We enumerate these pairs of packets as follows:

$<p_1,p_2>,<p_1,p_3>,<p_1,p_5>,<p_2,p_3>,<p_2,p_5>,<p_3,p_5><p_4,p_6>,<p_4,p_7>,$
$<p_4,p_8>,<p_4,p_9>,<p_4,p_{11}>,<p_6,p_7><p_6,p_8>,<p_6,p_9>,<p_6,p_{11}>,<p_7,p_8>,$
$<p_7,p_9>,<p_7,p_{11}>,<p_8,p_9>,<p_8,p_{11}>,<p_9,p_{11}>,<p_{10},p_{12}>,<p_{13},p_{14}>,<p_{13},p_{15}>,$
$<p_{13},p_{16}>,<p_{13},p_{17}>,<p_{14},p_{15}>,<p_{14},p_{16}>,<p_{14},p_{17}>,<p_{15},p_{16}>,<p_{15},p_{17}>,<p_{16},p_{17}>$

All these packet pairs satisfy the rules in the rule base in Table 5.2. Thus, $ipw_{str}(f)=0$. Consequently, the *4*-dimensional vector of IP weight measures is $<4.76,1.5476,0.944,0>$

5.4 Comparison with PCA

PCA is a widely used method for reducing the number of dimensions of a data set [83]. Suppose $X \sim \{x_1, x_2, ..., x_n\}$ is the original *n*-dimensional data and its covariance matrix is denoted by S, the eigenvectors E and eigenvalues λ for the original dataset can be calculated by solving equation $(S-\lambda I)E=0$. The new projected data set denoted by U is defined as: $U=E\times X$. The principal component is selected by sorting the

eigenvalues from highest to lowest, and then taking the m vectors with the largest eigenvalues. More detailed information about the PCA techniques can be found in [84].

We compared experimentally PCA with our feature extraction technique based on IP Weight metrics. To conduct the comparison, we selected a limited number of features consisting of two components of IP Weight metrics. Frequency and randomness components and the following five features are selected:

- Feature 1 denoted by f_1 is the appearing frequency of IP packets flowing to the same dip during δ_t;
- Feature 2 denoted by f_2 is the maximum appearing frequency of IP packets flowing to the same dip with same dp during δ_t;
- Features 3, 4 and 5 denoted by f_3, f_4, and f_5, respectively, stand for the randomness of the destination port, the source port and the source IP address. The packet flows considered have the same dip during δ_t.

The original data set is described by a 5-dimensional feature vector $< f_1, f_2, f_3, f_4, f_5 >$, which includes 8,000 records collected over two weeks' of normal networking traffic. Feature extraction is carried out using both PCA and IP Weights. In the case of IP Weights, the features f_1 and f_2 are mapped into a new numerical value ipw_{freq} by using the following utility function:

$$ipw_{freq}(f) = \left(f_1 - f_2 + f_1^{-1} f_2^2\right)\frac{f_2}{f_1}$$

The utility function $ipw_{ran}(f) = \max(f_3, f_4, f_5)$ maps features f_3, f_4 and f_5 into another numerical value ipw_{ran}. Table 5.3 illustrates the covariance coefficients, which measure the relativity between new features and original features.

Table 5.3. *Comparison between PCA and IP Weight Metrics*

Output / Input	PCA Covariance Matrix					IP Weight Metrics Covariance Matrix	
	f_{new1}	f_{new2}	f_{new3}	f_{new4}	f_{new5}	ipw_{freq}	ipw_{ran}
f_1	**0.9988**	0.0484	**0.0002**	0.0001	0.0000	**0.9680**	**0.0852**
f_2	**0.9913**	0.1315	**0.0004**	0.0003	0.0000	**0.9963**	**0.0502**
f_3	**0.2461**	0.3076	**0.6714**	0.5938	0.2034	**0.1886**	**0.7240**
f_4	**0.1377**	0.0172	**0.8299**	0.5386	0.0441	**0.1282**	**0.7581**
f_5	**0.1560**	0.1313	**0.7251**	0.3756	0.5399	**0.1714**	**0.5747**

In the case of PCA, the original feature vector $< f_1, f_2, f_3, f_4, f_5 >$ is the input and the output new features are denoted f_{new1}, f_{new2}, f_{new3}, f_{new4} and f_{new5}. The new features f_{new1} and f_{new3} stand for the principal components since their corresponding eigenvalues are the two highest. The covariance matrix between original feature vector and new feature vector is calculated and illustrated in Table 5.3.

By comparing the value of the covariance matrix for both approaches, we note that our approach provides the same capability as PCA using a simpler model.

5.5 Conclusions

We propose in this chapter new anomalousness metrics named IP weights. The IP weight metrics are derived from domain knowledge and empirical observations. Four utility functions are used to map several features into a set of numerical values along four dimensions, namely frequency, randomness, structure and load. IP weights measures the level of anomalousness of network packets: *the higher the IP weights, the more anomalous packets are*. Our clustering and outlier detection algorithm are based on this important property of IP weights. Moreover, the application of the utility functions reduces significantly the dimensionality of the feature space. Empirical comparison with

PCA technique shows that our utility functions exhibit the same capability as PCA algorithm, using, however, a faster and simpler model.

Chapter 6

Evaluation

We conduct two levels of validation for our unsupervised anomaly detection framework: the detection effectiveness offline with 1998 DARPA intrusion detection dataset and the detection efficiency online in a live networking environment. Although the primary objective of the online evaluation is to study detection efficiency, it also gives us the opportunity to confirm the results obtained offline in terms of detection effectiveness. More specifically, we analyze the detection performance of our framework by comparing I-means with k-means, which as claimed is being improved by I-means. Besides clustering, outlier detection is the most popular scheme used in unsupervised anomaly detection. Since outlier detection is one of the components of I-means clustering, we start our evaluation by analyzing two outlier detection schemes namely a univariate outlier detection (UOD) algorithm and a multivariate outlier detection (MOD) algorithm. The outcome of this study is that outlier detection is not powerful enough to achieve effective detection, which justifies the use of clustering as further step.

In our evaluation, we study the detection performance of each of the algorithms separately using the general unsupervised architecture shown in Figure 6.1. This allows comparing each of the algorithms under the same conditions.

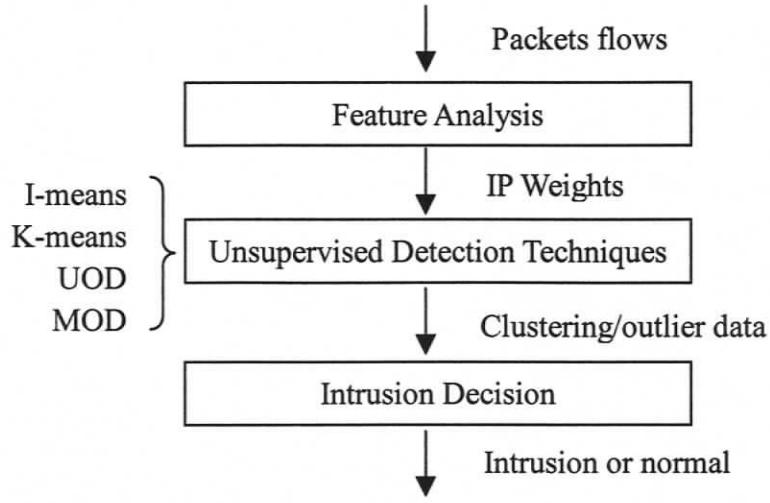


Figure 6.1. *General Unsupervised Detection Architecture*

6.1 Offline Evaluation

6.1.1 1998 DARPA Intrusion Detection Dataset

The first standard corpus used for evaluating intrusion detection approaches offline is the 1998 DARPA intrusion detection dataset [12]. Over 300 attacks are simulated in nine weeks. Training data are generated in the first seven weeks and testing data are created in the remaining two weeks. In total thirty-three different attack types are divided into four different attack categories, namely DoS, R2L, U2R and Probing. In this dissertation, since the proposed IP Weight metrics characterize the anomalousness of IP packets flows, we are interested specifically in multiple-connection based attacks. Consequently, for the evaluation, we extract these kinds of attacks from the DARPA dataset and establish a multiple-connection based intrusion dataset, which contains only DoS, R2L, and Probing attacks.

We select 17 days' data from the total 45 days' (nine weeks) dataset. All types of multiple-connection based network attacks are included in the 17 days' data. The location

of these data in the DARPA dataset and their corresponding attack types are illustrated in Table 6.1. A detailed description of these attacks can be referred to the appendix.

Table 6.1. *Attack Types and their Location in DARPA Dataset*

Location	Week1			
	Wednesday		Thursday.	Friday
Attack	*synflood*	*smurf*	*pod*	*teardrop*
Location	Week2			
	Monday	Tuesday	Wednesday	Friday
Attack	*portsweep*	*ipsweep*	*land*	*back*
Location	Week2	Week3	Week4	Week6
	Monday	Friday	Tuesday	Wednesday
Attack	*guest*	*nmap*	*satan*	*imap*
Location	Week6	Week7	Week8	
	Thursday	Friday	Tuesday	Friday
Attack	*dict*	*imap*	*apache2*	*process table*
Location	Week9			
	Monday		Tuesday	Thursday
Attack	*saint*	*udpstorm*	*mscan*	*mailbomb*

Figure 6.2 illustrates some examples of network traffic records from the DARPA dataset. Each record includes nine fields, namely index, traffic start date, traffic start time, traffic duration, service type, source port, destination port, source IP address and destination IP address.

Based on the traffic information provided by DARPA dataset, we can calculate three kinds of IP Weight metrics, namely ipw_{freq}, ipw_{ran} and ipw_{str}. The fourth metric ipw_{load} cannot be derived from the offline DARPA dataset. The DARPA dataset doesn't provide the required incoming or outgoing traffic information for the protected target host. For the evaluation, we compute the IP weights using a fixed time window $\delta_t = 2s$, which is small enough for online detection. More importantly, this corresponds to the time window used in KDD cup benchmark, and as such it allows comparing our results with existing

ones. As a result, the computed IP weights consist of 3600 instances along three dimensions, namely ipw_{freq}, ipw_{ran} and ipw_{str}: 3493 instances represent the normal network traffic and 107 instances correspond to intrusive traffic consisting of 19 types of multiple-connections based attacks. Figure 6.3 shows sample IP weight instances.

```
1  01/27/1998 00:00:01 00:00:23 ftp 1755 21 192.168.1.30 192.168.0.20
2  01/27/1998 05:04:43 07:59:01 telnet 1042 23 192.168.1.30 192.168.0.20
3  01/27/1998 06:04:36 00:00:59 smtp 43590 25 192.168.1.30 192.168.0.40
4  01/27/1998 08:45:01 00:00:01 finger 1050 79 192.168.0.40 192.168.1.30
5  01/27/1998 09:23:45 00:01:34 http 1031 80 192.168.1.30 192.168.0.40
6  01/27/1998 21:53:17 00:00:45 exec 2032 512 192.168.1.30 192.168.0.40
7  01/27/1998 21:58:21 00:00:04 http 1031 80 192.168.1.30 192.168.0.20
.....
```

Figure 6.2. *Examples of Traffic Records from DARPA Dataset*

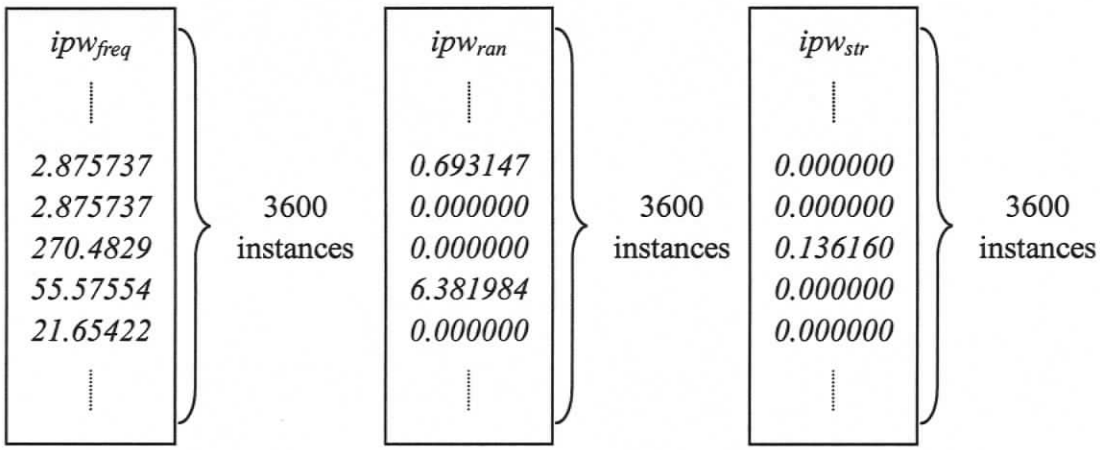


Figure 6.3. *Sample IP Weight Measures*

We use two performance metrics to evaluate the detection effectiveness, namely detection rate (DR) and false positive rate (FPR). We calculate the DR and FPR according to the following formulas:

$$DR = \frac{number\ of\ attack\ instances\ detected}{total\ number\ of\ attack\ instances}$$

$$FPR = \frac{number\ of\ normal\ instances\ detected\ as\ alerts}{total\ number\ of\ normal\ instances}$$

6.1.2 Outlier Detection

6.1.2.1 UOD and MOD Algorithm

Based on the dimensionality of the Gaussian model, the outlier detection algorithm proposed in chapter 4 consists of two versions: univariate outlier detection (UOD) and multivariate outlier detection (MOD). UOD detects outliers in *single*-dimensional data sets, while MOD detects outliers in *multi*-dimensional data sets.

The outlier detection algorithm proposed in Section 4.3 is used to detect the outliers among the IP Weights. The intrusion decision strategy is based on the outcome of outlier detection: if no outlier is detected, the network traffic is considered normal; otherwise, the outlier is reported as an intrusion.

UOD and MOD use different ways of detecting intrusions based on the IP Weight metrics. UOD detects each component of IP Weight metrics independently and then makes separate decisions. Final intrusion decision is a disjunction of the individual decisions made through the different IP Weight components. Figure 6.4 illustrates the detection strategies for UOD; ipw_{freq}, ipw_{ran}, ipw_{str} and ipw_{load} represent the different IP weight components; UOD generates from these inputs four independent outputs. An intrusion is finally reported if at least one output reports an intrusion. Using MOD, all four components of IP Weight metrics are combined into a *4*-dimensional vector. Intrusion decisions are based on the outliers of the *4*-dimensional vector space $<ipw_{freq}, ipw_{ran}, ipw_{str}, ipw_{load}>$.

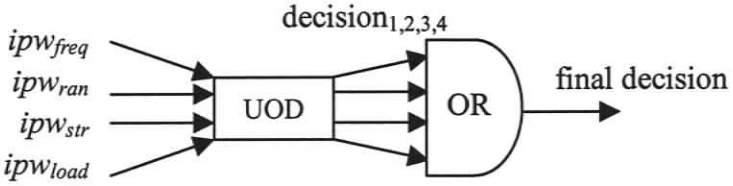


Figure 6.4. *Intrusion Detection Strategies for UOD*

6.1.2.2 Evaluation using UOD

In UOD based detection, the instances of ipw_{freq}, ipw_{ran} and ipw_{str} are concurrently evaluated by the UOD detector and then three intrusion decisions are made by the detector. As indicated earlier, the final decision is a disjunction of the three individual decisions. For an intrusion to be finally reported, at least one of the individual decisions must correspond to an intrusion.

We calculate DR and FPR for different values of $outlier_{thres}$ and plot accordingly the receiving operator characteristic (ROC) curves in Figure 6.5. Table 6.2 shows the DR and FPR for selected points from the ROC curves. The different points of the ROC curve are obtained by varying the $outlier_{thres}$.

Since initially intrusion decisions are made separately, redundant decisions could be made. An intrusive instance may be detected by all three metrics or a normal instance may be falsely reported by all of three metrics. Table 6.3 illustrates the impact of redundant decisions.

Table 6.2. *Selected Points from ROC Curve for UOD*

$outlier_{thres}$	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
DR(%)	100	95.33	95.33	85.05	81.31	81.31	66.36	65.42	62.62	61.68
FPR(%)	97.03	12.11	11.57	6.64	5.81	4.58	0.74	0.46	0.43	0.37

Table 6.3. *Detection Redundancy of UOD*

$outlier_{thres}$	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
number of redundant intrusive instances	92	60	60	58	57	56	9	7	6	5
number of redundant false positive instances	591	78	68	17	5	1	0	0	0	0

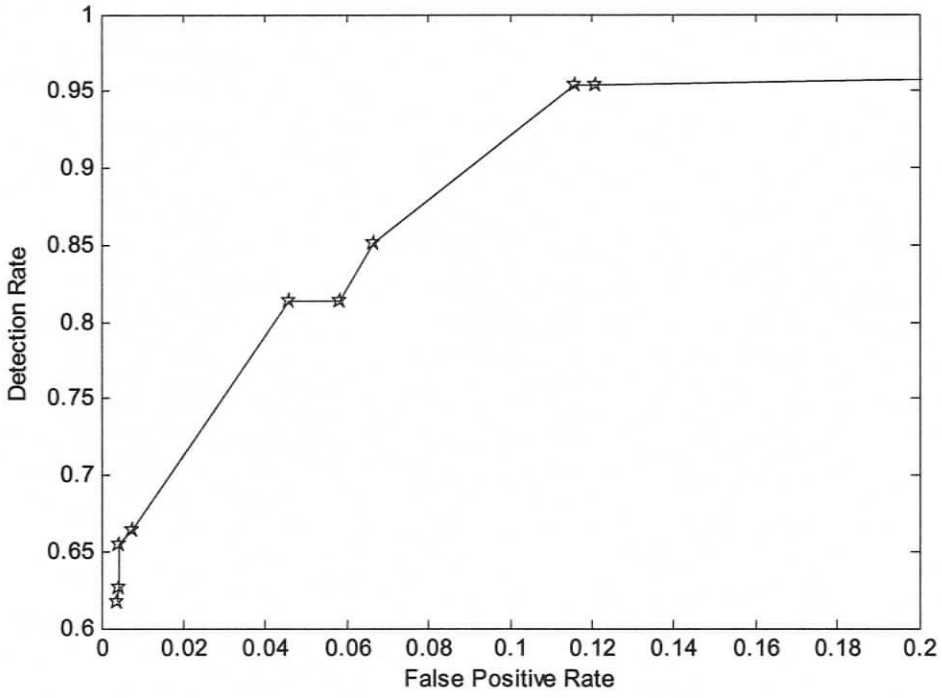


Figure 6.5. *ROC Curves for UOD*

6.1.2.3 Evaluation using MOD

In contrast with UOD, the input to MOD is a set of *three*-dimensional vectors < ipw_{freq}, ipw_{ran}, ipw_{str} >. By varying the value of $outlier_{thres}$, we calculate the corresponding DR and FPR and plot the receiving operator characteristic (ROC) curves in Figure 6.6. Table

6.4 shows the DR and FPR for selected points from the ROC curves.

Table 6.4. *Selected Points from ROC Curve for MOD*

$outlier_{thres}$	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
DR(%)	100	95.33	95.33	95.33	85.05	81.31	81.31	80.37	64.49	56.08
FPR(%)	97.03	19.38	12.71	11.48	7.07	4.92	4.67	2.03	0.86	0.8

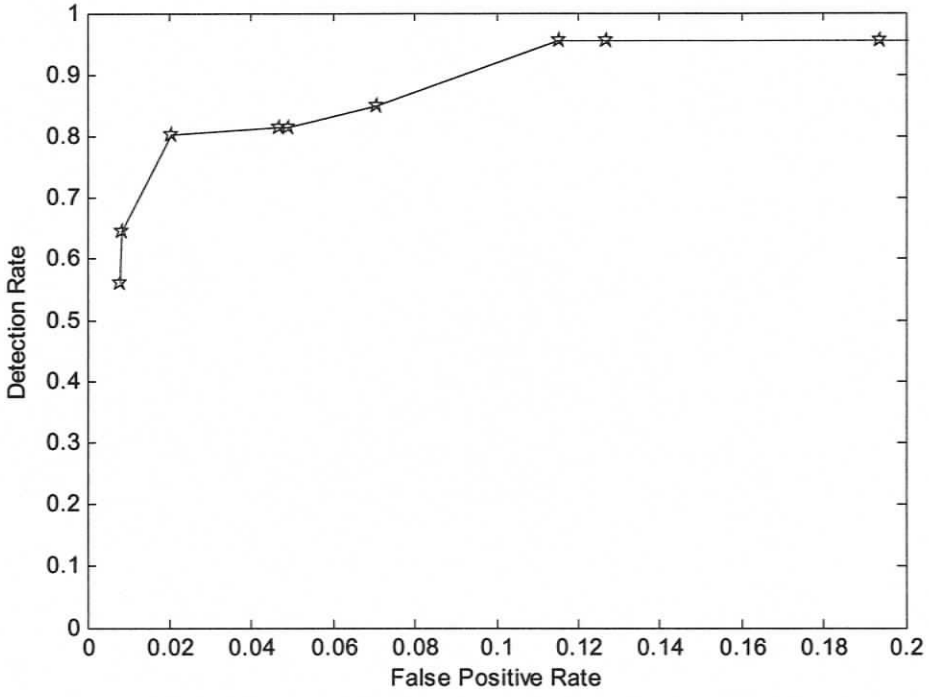


Figure 6.6. *ROC Curves for MOD*

6.1.2.4 Discussion

Figure 6.7 plots the ROC curves for both UOD and MOD. Overall the experiment results show that, highest detection rates are achieved with MOD, while lowest false positive rates are obtained with UOD. For instance, some of the best operating points at a

threshold of 10^{-7}, correspond to <DR = 66.36%, FPR = 0.74%> with UOD, and <DR=80.37%, FPR=2.03%> with MOD. The UOD strategy, which seems more attractive in achieving low FPR, detects 16 types of attacks out of a total of 19 multiple-connection based attack types at a threshold of 10^{-7}. Attacks *processtable*, *dictionary*, and *guest* (variant of *dictionary*) are completely missed by the UOD system. In practice, the goal targeted for effective intrusion detection is to detect more than 99% of the attacks with a false alarm rate of less than 1% (sometimes claimed as less than 0.1%) [31]. So in either case, the detection performance is far from the goal set for effective intrusion detection. This justifies why in I-means, outlier detection is augmented by plain clustering.

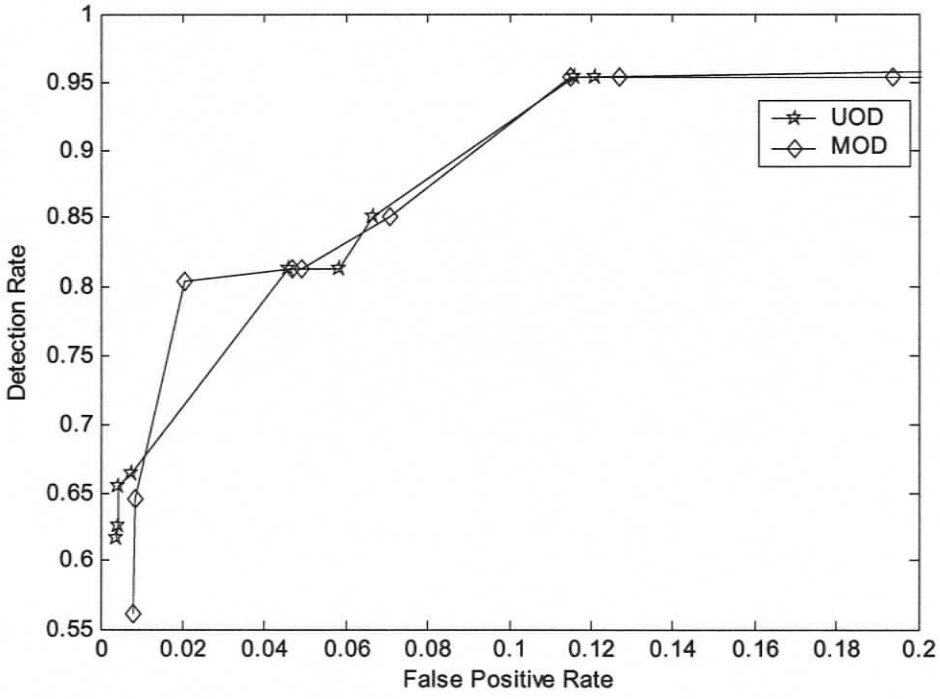


Figure 6.7. *ROC Curves for UOD and MOD*

6.1.3 Clustering

6.1.3.1 Detection Strategies

6.1.3.1.1 I-means Algorithm

The detection framework based on I-means is illustrated in Figure 6.1. Packet flows are first collected on networks and then IP weights are generated through feature analysis. I-means clustering algorithm is used to cluster the IP weights. Intrusions are decided based on the outcome of the clustering. The decision strategy is based on the simple consideration that while some intrusion instances are clearly different from normal data, others are not. Outliers fall in the first category; hence outliers are systematically labeled as intrusions. In order to identify data belonging to the second category, we consider clusters with high IP weights. We know that the higher the IP weight, the more anomalous the data. In this regard, we consider the cluster with the greatest center point, and isolate in this cluster the data whose values are greater than the center point; let us denote by G the set of all these elements. In order to achieve effective detection, we should take into account the maximum acceptable false positive rate ($MAFPR$) and the number N of IP Weights currently computed. The $MAFPR$ is selected by the security officer or by the management. Given values of $MAFPR$ and N, if the cardinality of G is less or equal than $MAFPR \times N$, then all the elements of G will be labeled as intrusive. Otherwise, only the $MAFPR \times N$ elements of G with the highest values will be labeled as intrusions. This permits keeping the number of false alarm under control, and helps improving detection effectiveness. Setting a maximum acceptable value for FPR is important in intrusion detection systems, because beyond a certain value, the operator (e.g. SSO), might simply ignore the alarms regardless of whether they are true or false.

6.1.3.1.2 K-means Algorithm

The detection framework based on k-means is illustrated in Figure 6.1. Similarly with I-means, IP weights are calculated and then k-means algorithm is used to cluster the IP weights. In contrast with the intrusion decision strategy for I-means, as k-means cannot find outliers in IP weights, only clusters with high IP weights are considered in deciding intrusions. We isolate such clusters and select the data whose values are greater than the center point; G denotes the set of all these elements. Using the same notation as above, if the cardinality of G is less or equal than $MAFPR \times N$, then all the elements of G are labeled as intrusive. Otherwise, only the $MAFPR \times N$ elements of G with the highest values are labeled as intrusions.

6.1.3.2 Evaluation

6.1.3.2.1 Evaluation using K-means

When evaluating the k-means detection framework, we predefine the number of clusters as 2. The initial center point for each cluster is arbitrarily selected, specifically we use the two first instances of the total 3600 instances as the center points for the clusters. The selection of the number of clusters for k-means is based on the assumption that the IP Weight dataset is composed of two clusters: normal clusters and intrusive clusters. This assumption is reasonable. Actually using the I-means algorithm in the next section, we establish that the optimal number of clusters for the IP Weight dataset is two. We calculate the DR and FPR using k-means by varying the MAFPR and plot the corresponding ROC curve in Figure 6.8. Table 6.5 show the DR and FPR for selected points from the ROC curves for k-means. The different points of the ROC curve are obtained by varying the MAFPR. While the FPR varies within acceptable ranges, the DR barely exceeds 50%, which confirms the poor detection effectiveness of k-means hypothesized earlier.

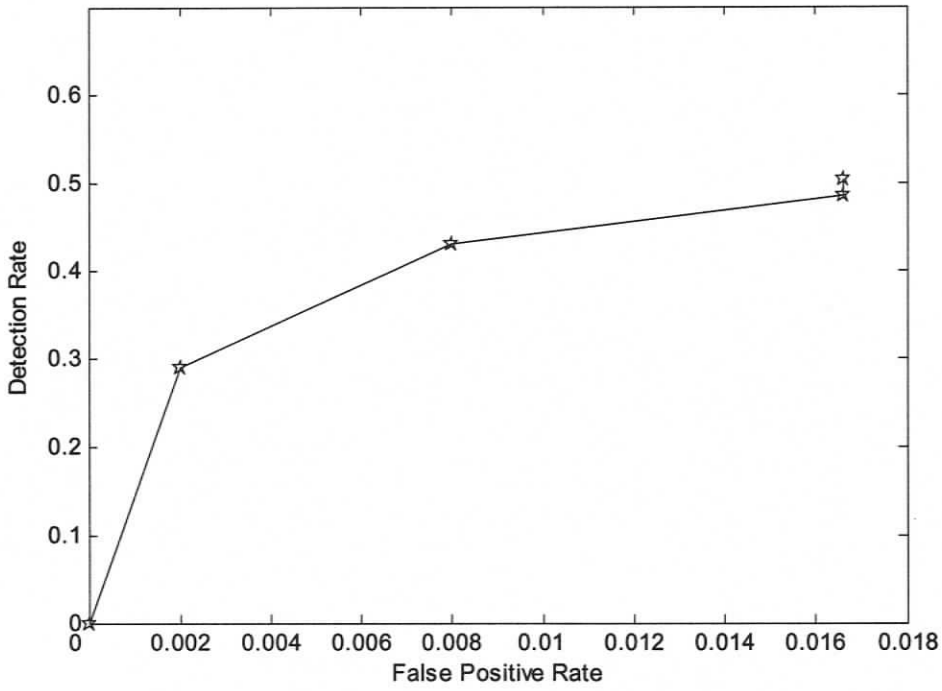


Figure 6.8. *ROC Curves for K-means*

Table 6.5. *Selected Points from ROC Curve for K-means*

MAFPR(%)	1	2	3	4	5	6	7	8	9	10
DR(%)	28.97	42.99	48.60	50.47	50.47	50.47	50.47	50.47	50.47	50.47
FPR(%)	0.20	0.80	1.66	1.66	1.66	1.66	1.66	1.66	1.66	1.66

6.1.3.2.2 Evaluation using I-means

In contrast with k-means algorithm, the I-means algorithm can determine the optimal number of clusters for a given dataset. The initial center point for each cluster is not randomly selected. Instead, I-means define the initial center points according to a preliminary clustering result based on GMM. Moreover, I-means has the capability to identify and isolate potential outliers in the dataset. This obviously improves the

performance of the clustering.

In the evaluation of I-means detector, the detection result consists of two components: outlier IP Weight instances and the clustered IP Weight instances. The clustering results do not include any outlier since they are isolated from the dataset before the clustering process. We calculate the DR and FPR by varying the MAFPR and plot the corresponding ROC curve in Figure 6.9. Table 6.6 show the DR and FPR for selected points from the ROC curves for I-means. The different points of the ROC curve are obtained by varying the MAFPR. There is a notable improvement in effectiveness compared with the k-means results given earlier.

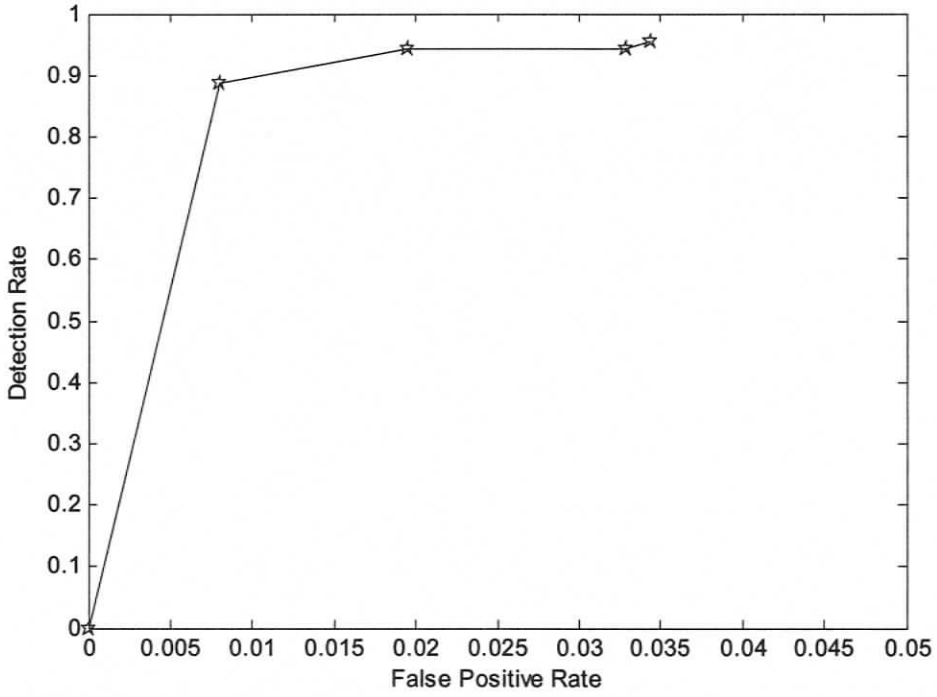


Figure 6.9. *ROC Curves for I-means*

Table 6.6. *Selected Points from ROC Curve for I-means*

MAFPR(%)	1	2	3	4	5	6	7	8	9	10
DR(%)	88.79	94.39	94.39	95.33	95.33	95.33	95.33	95.33	95.33	95.33
FPR(%)	0.8	1.95	3.29	3.44	3.44	3.44	3.44	3.44	3.44	3.44

Since the I-means detection involves three separate decisions based on individual IP weight component, detection redundancy is also an issue. We calculate the DR and FPR with different values of the *MAFPR* for each metrics. Specifically the DR and FPR of the metrics ipw_{str} is 0.94% and 0% over ten different values of *MAFPR*. Figure 6.10 is the ROC curves illustrating the individual contributions of the metrics ipw_{freq}, ipw_{ran} and ipw_{str} respectively. Table 6.7 and 6.8 shows the DR and FPR for selected points from the ROC curves in Figure 6.10. The comparison among the three metrics shows that ipw_{ran} yields a highest detection rate (DR=77.57%) with a low false alarm rate (FPR=1.17%).

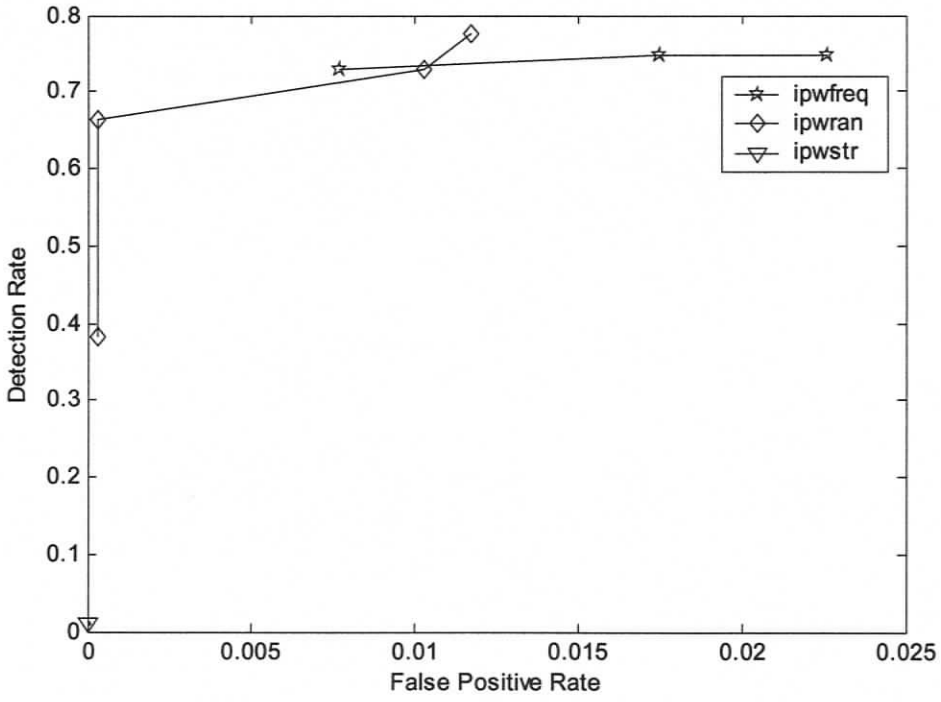


Figure 6.10. *ROC Curves for ipw_{freq}, ipw_{ran} and ipw_{str} in I-means*

Table 6.7. *Selected Points from ROC Curve of ipw_{freq} for I-means*

MAFPR(%)	1	2	3	4	5	6	7	8	9	10
DR(%)	72.9	74.77	74.77	74.77	74.77	74.77	74.77	74.77	74.77	74.77
FPR(%)	0.77	1.75	2.26	2.26	2.26	2.26	2.26	2.26	2.26	2.26

Table 6.8. *Selected Points from ROC Curve of ipw_{ran} in I-means*

MAFPR(%)	1	2	3	4	5	6	7	8	9	10
DR(%)	38.32	66.36	72.9	77.57	77.57	77.57	77.57	77.57	77.57	77.57
FPR(%)	0.03	0.03	1.03	1.17	1.17	1.17	1.17	1.17	1.17	1.17

Table 6.9. *Detection Redundancy for I-means*

MAFPR (%)	1	2	3	4	5	6	7	8	9	10
number of redundant intrusive instances	25	51	58	62	62	62	62	62	62	62
number of redundant false positive instances	0	0	0	0	0	0	0	0	0	0

6.1.3.2.3 Discussion

Overall, the ROC curves show that the I-means detector achieves high detection rate with very low FPR values. For instance, one of the best operating points in this curve corresponds to (DR = 88.79%, FPR=0.8%, MAFPR=1%) and validation results in this point show that I-means can detect 18 types of attacks out of a total of 19 types of multiple-connection based network attacks. Only attack *processtable* is completely missed by the system. The attack *processtable* prevents any other commands from being executed in the victim host by filling the process table of the victim host with a large number of TCP/IP connections.

For comparison, one of the best operating points for the detection system based on

k-means corresponds to (DR = 42.99%, FPR=0.8%, MAFPR=2%). At this point, k-means detector completely missed 8 types of attacks, namely *processtable*, *back*, *dictionary*, *guest*, *imap*, *mailbomb*, *teardrop and udpstorm*. Furthermore, it detected only part of the 11 remaining types of attacks instances, which explain the relatively poor detection rates it achieved.

Thus as illustrated by Figure 6.11, which show ROC curves for both I-means and K-means, I-means improves significantly on k-means in terms of detection effectiveness.

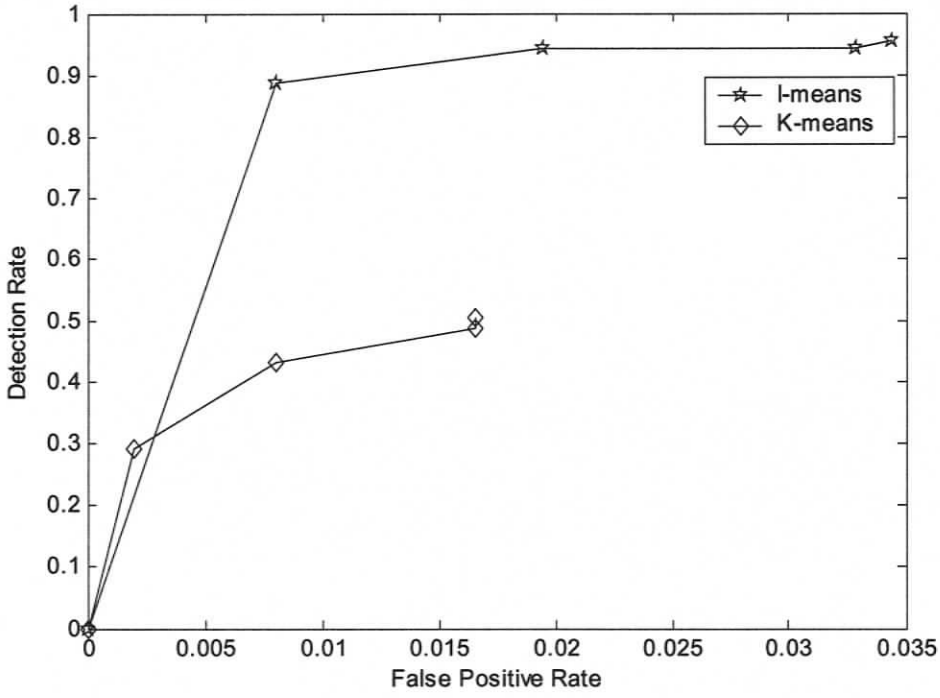


Figure 6.11. *ROC Curves Comparing the Effectiveness of I-means with K-means*

6.1.4 Summary

In this section, we evaluate four detectors based on four different detection mechanisms, namely UOD, MOD, K-means and I-means. The 1998 DARPA intrusion detection dataset is used during the evaluation. We extract a subset of the DARPA dataset, which include 19 types of multiple-connection based attacks. Generally, our evaluation includes two

layers, namely grading layer and detection layer. In the grading layer, IP weight metrics are used to extract and quantify the features defined in DARPA dataset; a total of 3600 IP Weight instances are generated. In the detection layer, four different detectors are evaluated. DR and FPR are used to evaluate the performance of the detectors. Figure 6.12 illustrates four ROC curves for UOD, MOD, K-means and I-means respectively. Table 6.10 describes one of the best operation points for each of the four detectors. From Table 6.10, we see that I-means detector achieves the highest detection rate with very low false alarm rate compared with the other three detectors.

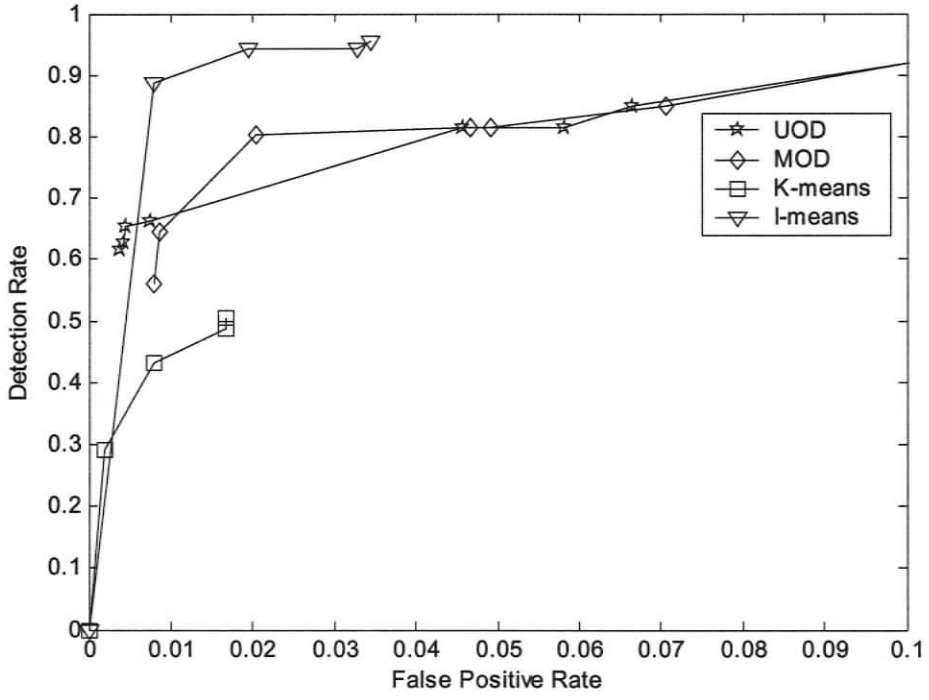


Figure 6.12. *ROC Curves for UOD, MOD, I-means and K-means Respectively*

Table 6.10. *Sample Good Performance with UOD, MOD, K-means and I-means Respectively*

Detector	DR (%)	FPR (%)	Operating Point
I-means	88.79	0.8	MAFPR=1%
UOD	66.36	0.74	$Outlier_{thres}=10^{-7}$
K-means	42.99	0.8	MAFPR=2%
MOD	80.37	2.03	$Outlier_{thres}=10^{-8}$

Although UOD and MOD can achieve very good detection rates, they create higher false alarm rates ranging above 10%. While k-means can achieve low false alarm rates, it can only find half of the attacks. This is because the outliers degrade the clustering result for k-means algorithm. Moreover, it is not enough to make intrusion decision based only on the outliers. Many intrusive instances involve normal values, as such they cannot be seen as outliers by using UOD or MOD. I-means clustering algorithm solves these problems due to its capability for identifying outliers and intelligently determining the initial parameters for clustering.

6.2 Online Evaluation with Real Traffic

Online evaluation in a real networking environment is conducted to assess the efficiency and effectiveness of our detection system. All the four IP weights components are used in the online evaluation. As illustrated by Figure 6.13, the implementation of our detection system includes two modules, namely grader and detector. Collected IP packets are submitted to the grader, which computes corresponding IP Weights. The IP Weights are then stored in a database. Synchronously, the detector reads the IP Weights from the database, identifies the intrusive data and then makes intrusion decisions accordingly.

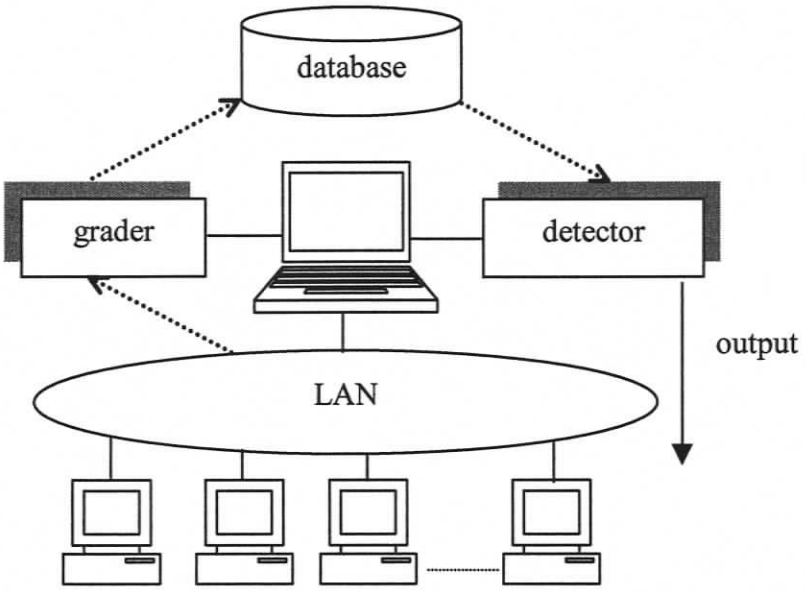


Figure 6.13. *Deployment of I-means Based Detection System*

Figure 6.14 illustrates the hardware topology of the live networking environment. The attack traffic is generated from LAN1; LAN2 is the victim network. Our detection system is deployed on LAN2. During the evaluation, we executed six types of multiple-connection based attacks, four of which, categorized as distributed denial of service (DDoS) attacks, include *synflood*, *smurf*, *udpflood* and a mixture attack of *synflood* and *udpflood*. The other two attacks fall in the category of stealthy probing attacks, namely *xscan* [85] and *fluxay* [86]. The attacks and observations were made during a one-day real network activity test session.

Attacks were initiated randomly during the day, and each attack was repeated ten times. We recorded ten random *starting times* for each attack. We also recorded the *detection times* for each attack and computed the corresponding system *response times* by taking the difference between *detection time* and *starting time* as follows:

$$response\ \ time = det ection\ \ time - starting\ \ time$$

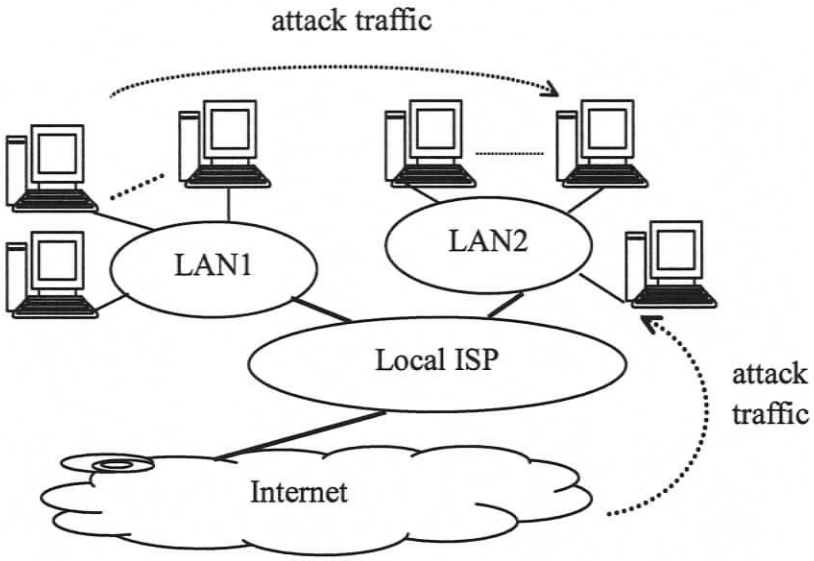


Figure 6.14. *Network Topology*

We conducted the same experiment separately using I-means and k-means. According to the results, in a short time frame the I-means based detection system was able to detect all six types of attacks. Table 6.11 illustrates the average response time for each attack; detailed timing information is provided in Table 6.12. From Table 6.11 and 6.12, we see that the average response time for the detector for *udpflood* attack is the fastest and the average response time to identify *synflood* attack is the longest. Generally the response times for all six attacks fall within few seconds ranges.

Table 6.11. *System's Response Time for Real Attacks using I-means Algorithm*

Performance \ Attack	*udpflood*	*synflood*	*smurf*	*mixture flood*	*fluxay*	*xscan*
Average Response Time (s)	2.4	7.5	3.5	4	4.5	5.3
Standard Deviation	1.075	0.9718	0.8498	2	1.5811	0.9487

Table 6.12. *Response Time Obtained Using I-means Algorithm*

udpflood			*synflood*			*smurf*		
ST	DT	RT	ST	DT	RT	ST	DT	RT
08:07:53	08:07:54	1s	08:43:15	08:43:22	7s	08:17:13	08:17:17	4s
09:09:51	09:09:52	1s	09:44:07	09:44:15	8s	09:18:09	09:18:12	3s
10:10:50	10:10:52	2s	10:44:47	10:44:53	6s	10:19:14	10:19:18	4s
11:11:50	11:11:52	2s	11:45:22	11:45:31	9s	11:38:19	11:38:23	4s
12:12:40	12:12:43	3s	12:46:09	12:46:17	8s	12:39:03	12:39:05	2s
13:13:20	13:13:24	4s	13:46:52	13:46:58	6s	13:40:42	13:40:45	3s
14:13:55	14:13:57	2s	14:47:37	14:47:45	8s	14:41:13	14:41:18	5s
15:14:30	15:14:33	3s	15:48:14	15:48:22	8s	15:41:50	15:41:54	4s
16:15:15	16:15:17	2s	16:48:52	16:49:00	8s	16:42:16	16:42:19	3s
17:15:50	17:15:54	4s	17:50:18	17:50:25	7s	17:42:42	17:42:45	3s
mixture of udpflood and synflood			*fluxay*			*xscan*		
ST	DT	RT	ST	DT	RT	ST	DT	RT
08:58:42	08:58:46	4s	18:54:57	18:5502	5s	18:23:05	18:23:09	4s
09:59:53	10:00:00	7s	19:56:47	19:56:49	2s	19:24:06	19:24:13	7s
10:02:44	10:02:49	5s	20:58:02	20:58:05	3s	20:24:47	20:24:52	5s
11:03:24	11:03:27	3s	21:59:24	21:59:29	5s	21:25:30	21:25:35	5s
12:04:05	12:04:07	2s	22:00:26	22:00:29	3s	22:26:12	22:26:16	4s
13:04:53	13:04:54	1s	23:01:20	23:01:26	6s	23:26:47	23:26:53	6s
14:05:41	14:05:48	7s	00:02:04	00:02:10	6s	00:27:22	00:27:28	6s
15:06:24	15:06:27	3s	01:02:43	01:02:49	6s	01:28:23	00:28:28	5s
16:07:12	16:07:17	5s	02:03:25	02:03:31	6s	02:29:06	02:29:12	6s
17:07:52	17:07:55	3s	03:04:10	03:04:13	3s	03:29:51	03:29:56	5s

Legend: ST: starting time;
DT: detection time;
RT: response time

Table 6.13 illustrates the average response times computed per attack for k-means algorithm. Comparing Table 6.11 and 6.13, we see that the response times achieved using I-means are far better than those obtained using k-means. Moreover, using live networking data, k-means failed to detect *synflood* attacks, although it did detect this attack using DARPA dataset.

Table 6.13. *System's Response Time for Real Attacks using K-means Algorithm*

Performance \ Attack	*udpflood*	*smurf*	*mixture flood*	*fluxay*	*xscan*
Average Response Time (s)	5.4	11.0	4.7	4.6	22.3
Standard Deviation	0.966	10.81	2.751	0.966	1.767

6.3 Conclusion

As indicated in the introduction, the main goal of this work is to develop an *effective* and *efficient* unsupervised anomaly detection framework. To achieve this objective, we propose the I-means detection framework. The evaluation shows that I-means clearly improves on algorithms such as k-means. Based on the results of the online evaluation, we can claim that detection efficiency is achieved considering that response times are within few seconds' ranges. As reported by McHugh, the goal targeted in DARPA competitions for effective intrusion detection is to detect more than 99% of the attacks with a false alarm rate of less than 1% [31]; in other words FPR < 1 % and DR > 99%. I-means is able to achieve a FPR =0.8%, but with a DR = 88.79% (see Table 6.6). So, although the false alarm target is achieved, the detection rate still needs to be improved for full effectiveness. We are convinced that this can be achieved by increasing the number of metrics, and plan to address that in future research. It is still important to highlight the fact that (FPR =0.8%, DR = 88.79%) is a significant improvement over existing anomalies detection approaches, which are plagued by high false alarm rates and low detection rates. This is highlighted in Table 6.14, which summarizes the best and most flattering results obtained with other unsupervised detection approaches published in the literature [4, 61, 87]. I-means achieve by far the best results among all these approaches. Unsupervised SVM is close to I-means in DR but with much higher FPR.

Table 6.14. *Comparison with Other Unsupervised Intrusion Detection Approaches*

Approach	Dataset	DR	FPR
Euclidean-based [4]	1999 KDDCUP dataset [8]	66%	2%
Mahalanobis-based [87]	1998 DARPA dataset [12]	57.9%	2%
NN(nearestneighbor) [87]	1998 DARPA dataset [12]	73.7%	2%
k-NN (k=5) [61]	1999 KDDCUP dataset [8]	61.59%	2%
LOF (local outlier factor) [87]	1998 DARPA dataset [12]	68.4%	2%
One class SVM [4]	1999 KDDCUP dataset [8]	67%	4%
Canberra Metrics [61]	1999 KDDCUP dataset [8]	5.17%	2%
Unsupervised SVM [87]	1998 DARPA dataset [12]	84.2%	4%
I-means	1998 DARPA dataset [12]	88.79%	0.8%

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Learning models play a significant role in intrusion detection because of their potential for discovering and finding unknown behaviors. According to the different learning processes, learning models can be divided into two categories: supervised detection models and unsupervised detection models. Using supervised models, normal system behaviors are automatically formed by learning labeled training data. Deviations from learned normal behaviors are flagged as anomalous behaviors.

There are, however, some limitations when applying supervised learning techniques for intrusion detection. First, supervised learning techniques do not consider the completeness of training data and testing dataset. Even if the supervised learning approaches perform well under the testing dataset, they may possibly carry errors that are not identified due to incompleteness of the testing dataset. Moreover, labeling the training data manually is time consuming and also error-prone since training data usually includes noises.

Unsupervised detection models address these limitations. Unsupervised models automatically form opinion of normal system behaviors by grouping a set of unlabelled data patterns according to some measures of similarities. In this case, two essential issues must be addressed to achieve efficient and effective detection, namely proper

characterization of data patterns and proper selection of the clustering algorithm.

Characterizing data patterns is important for intrusion detection because real-time detection, which is desirable, becomes impossible beyond a certain data dimension. As a result, in order to address this issue, we define a set of utility functions to characterize network packets in chapter 5. As shown in chapter 5, the proposed utility functions reduce the dimensions of the features space at level comparable with principal component analysis (PCA) technique using a simpler model. The utility functions also provide a grading scheme for the anomalousness of IP packet flows. The underlying assumption for these utility functions is that the higher the IP weights, the more anomalous the packet flows.

Selecting a proper clustering algorithm is another important issue for unsupervised intrusion detection. It has been reported that k-means clustering algorithm is one of the few algorithms that have been applied successfully to large data sets. Because of the large amounts of data processed by intrusion detection systems, k-means seems to be a good candidate. However, there are several weaknesses in k-means which represent serious impediments for effective intrusion detections.

In order to address these shortcomings, we propose in chapter 4 a new hybrid clustering algorithm, named I-means. I-means combines an evolutionary approach to mixture resolving, some outlier detection strategies, with plain k-means algorithm. The evolutionary approach using mixture model can estimate intelligently the optimal number of clusters for a set of data. Empirical evaluation with three different datasets confirms that the proposed algorithm can converge into the global solutions (i.e. the optimal number of clusters) after only a few iterations. Outlier data can also be detected by the mixture model. After discriminating the optimal number of clusters for the dataset and the corresponding cluster center points, and removing the possible outlier data from original dataset, the plain k-means algorithm is used to cluster the data. As a result, I-means largely improves the performance of k-means algorithm.

In chapter 6, we compare I-means with k-means using the 1998 DARPA intrusion detection dataset. The evaluation results show that I-means achieves by far the best performance. Specifically, the detection framework based on I-means uncovers 18 types of attacks out of a total of 19 types of multiple-connection based network attacks and yields 88.79% detection rate with 0.8% false positive rate. Furthermore, an online evaluation deploying I-means based detection framework in a live networking environment not only confirms the detection effectiveness observed with DARPA dataset, but also shows a strong run time efficiency, with response times falling within few seconds range.

7.2 Future Work

Although the proposed unsupervised detection framework based on I-means performs very well on detecting multiple-connection based intrusions, two substantial works are necessary in the future.

1. The proposed IP weight metrics evaluate the anomalousness of IP packets flowing to a protected destination IP address during a specified time window. Consequently, the IP weight metrics can only characterize and discriminate multiple-connection based intrusions. They are helpless in characterizing single-connection based intrusions. Therefore, in order to detect more attacks, it is necessary to derive new metrics covering single-connection based attacks.
2. The proposed I-means based detection framework is one of three detectors in a larger research project named SPIDER for Systematic and Pro-active Intrusion Detection System. The other two components include biometrics based detector and host based detector. The main purpose of SPIDER is to provide a complete intrusion detection solution for monitoring computing systems. In the future, we will integrate all three detectors by using agent technologies. How to implement

efficient communications among different agents (detectors) and how to evaluate and test the performance of SPIDER are two important open issues in this work.

Bibliography

[1] J.P. Anderson, "Computer security threat monitoring and surveillance", Technical Report, James P. Anderson Co., Fort Washington, Pennsylvania, April 1980.

[2] D. E. Denning, "An intrusion detection model", *IEEE Transactions on Software Engineering*, No. 2, Pages: 222-232, February 1987.

[3] E. Eskin, "Anomaly detection over noisy data using learned probability distributions", *Proceedings of 17th International Conference on Machine Learning*, Pages: 255-262, June 2000.

[4] E. Eskin, A. Arnold, M. Prerau, L. Portnoy and S. Stolfo, "A geometric framework for unsupervised anomaly detection: detecting intrusions in unlabeled data", *Data Mining for Security Aapplications (DMSA-2002)*, Kluwer Academic Publisher, 2002.

[5] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering", *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*, Philadelphia, PA, November 2001.

[6] A.K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall Advanced Reference Series, 1988.

[7] A.K. Jain, M.N. Murty and P.J. Flynn, "Data clustering", *ACM Computing Surveys*, Vol. 31, No. 3, Pages: 264-296, September 1999.

[8] KDD CUP 1999, KDD CUP 1999 intrusion detection dataset "http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html", 1999.

[9] B.D. Ripley, *Pattern Recognition and Neural Networks*, Cambridge, U.K., Cambridge University Press, 1996.

[10] D. Titterington, A. Smith and U. Makov, *Statistical Analysis of Finite Mixture Distributions*, John Wiley & Sons, New York, 1985.

[11] P. G. Neumann and D. B. Parker, "A summary of computer misuse techniques", *Proceedings of the 12th National Computer Security Conference*, Pages: 396–407, October 1989.

[12] K. Kendall, "A database of computer attacks for the evaluation of intrusion detection systems", Master's Thesis, Massachusetts Institute of Technology, 1998.

[13] Rootshell,http://www.rootshell.com/archive-j457nxiqi3gq59dv/199711/sol_syslog .txt.html, November 1997.

[14] M. Sebring, E. Shellhouse, M. Hanna, and R. Whitehurst, "Expert systems in intrusion detection: a case study". *Proceedings of 11th National Computer Security*

Conference, Pages: 74-81, October 1988.

[15] T. Lunt, R. Jagannathan, R. Lee, S. Listgarten, D. Edwards, P. Neumann, H. Javitz and A. Valdes, "IDES: the enhanced prototype, a real-time intrusion detection system", Technical Report, SRI Project 4185-010, SRI-CSL-88-12, CSL SRI International, Computer Science Laboratory, Menlo Park, CA, October 1988.

[16] D. Anderson, T. F. Lunt and A. Valdes, "Next-generation intrusion-detection expert system (NIDES)", Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, Menlo Park, CA, May 1995.

[17] D. Anderson, T. F. Lunt, H. Javitz, A. Tamaru and A. Valdes, "Detecting unusual program behavior using the statistical component of the next-generation intrusion detection system (NIDES)", Technical Report SRI-CSL-95-06, Computer Science Laboratory, SRI International, Menlo Park, CA, May 1995.

[18] U. Lindqvist and P.A. Porras, "Detecting computer and network misuse through the production-based expert system toolset (P-BEST)", *Proceedings of the 1999 IEEE Symposium On Security and Privacy*, Pages: 146-161, May 1999.

[19] K. Ilgun, "USTAT: A real-time intrusion detection system for UNIX", *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, Pages: 16-28, May 1993.

[20] K. Ilgun, R. A. Kemmerer and P. A. Porras, "State transition analysis: A rule-based intrusion detection approach", *IEEE Transactions on Software Engineering*, 21(3), Pages: 181-199, March 1995.

[21] S. Kumar and E. H. Spafford, "An application of pattern matching in intrusion detection", Technical Report CSD-TR-94-013, The COAST Project, Department of Computer Sciences, Purdue University, West Lafayette, IN, June 1994.

[22] S. Kumar and E. H. Spafford, "A pattern matching model for misuse intrusion detection", *Proceedings of the 17th National Computer Security Conference*, Pages: 11-21, 1994.

[23] S. Kumar and E. H. Spafford, "A software architecture to support misuse intrusion detection", Technical Report, The COAST Project, Department of Computer Sciences, Purdue University, West Lafayette, IN, March 1995.

[24] S. Kumar, "Classification and detection of computer intrusions", PhD Thesis, Purdue University, West Lafayette, Indiana, August 1995.

[25] W. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection", *Proceedings of the 7th USENIX Security Symposium*, January 1998.

[26] W. Lee, S. J. Stolfo and K. W. Mok, "Mining audit data to build intrusion models", *Proceedings of the 4th International Conference On Knowledge Discovery and Data Mining*, Pages: 66-72, August 1998.

[27] W. Lee, S. J. Stolfo and K. W. Mok, "A data mining framework for building intrusion detection models", *Proceedings 1999 IEEE Symposium on Security and Privacy*, May 1999.

[28] S. Axelsson, "Intrusion detection systems: A survey and taxonomy", Technical

Report 99-15, Department of Computer Engineering, Chalmers University of Technology, SE-412 96 Goteborg, Sweden.

[29] J. Lee, S. Moskovics and L. Silacci, "A survey of intrusion detection analysis methods", http://www.cs.ucsd.edu/classes/sp99/cse221/projects/IDS.pdf.

[30] A. K. Jones and R. S. Sielken, "Computer system intrusion detection: a survey", http://www.cs.virginia.edu/~jones/IDS-research/Documents/jones-sielken-survey-v11.pdf.

[31] J. McHugh, "Intrusion and intrusion detection", *International Journal of Information Security*, Volume 1, Number 1, Pages: 14-35, 2001.

[32] H. Debar, M. Dacier and A. Wepsi, "A revised taxonomy for intrusion-detection systems", IBM Research Report, 1999.

[33] T. F. Lunt, "Detecting intruders in computer systems", 1993 Conference on Auditing and Computer Technology, http://www.raptor.com/lib/canada93.ps.

[34] A. Sundaram, "An introduction to intrusion detection", http://www.aitel.hist.no/fag/dsh-m/lek01/An%20Introduction%20to%20Intrusion%20Detection.doc.

[35] S. E. Smaha, "Haystack: An intrusion detection system", *Proceedings of the IEEE Fourth Aerospace Computer Security Applications Conference*, December 1988.

[36] H. S. Vaccaro and G. E. Liepins, "Detection of anomalous computer session activity", *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, Pages: 280-289, May 1989.

[37] T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood and D. Wolber, "A network security monitor", *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, Pages: 296-304, 1990.

[38] H.S. Teng, K.Chen and S.C-Y Lu, "Adaptive real-time anomaly detection using inductively generated sequential patterns", *Proceedings of 1990 IEEE Symposium on Security and Privacy*, Pages: 278-284, May 1990.

[39] K. A. Jackson, D. H. DuBois and C. A. Stallings, "An expert system application for network intrusion detection", *Proceedings of the 14th National Computer Security Conference*, Pages: 215-225, October 1991.

[40] J. Hochberg, K. Jackson, C. Stallings, J. F. McClary, D. DuBois and J. Ford, "NADIR: An automated system for detecting network intrusion and misuse", *Computers & Security*, 12(3), Pages: 235-248, 1993.

[41] S. Forrest, A.S. Perelson, L. Allen and R. Cherukuri, "Self-nonself discrimination in a computer", *Proceedings of 1994 IEEE Symposium on Security and Privacy*, Pages: 202-212, May 1994.

[42] S. Forrest, S.A. Hofmeyr and T.A. Longstaff, "A sense of self for Unix processes", *Proceedings of 1996 IEEE Symposium on Security and Privacy*, Pages: 120-128, May 1996.

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

[44] P. D'haeseleer, S. Forrest and P. Helman, "An immunological approach to change detection: algorithm, analysis and implications", *Proceedings of 1996 IEEE Symposium on Security and Privacy*, Pages: 110-119, May 1996.

[45] C. Warrender, S. Forrest and B. Pearlmutter, "Detecting intrusions using system calls: Alternative data models", *Proceedings of 1999 IEEE Symposium on Security and Privacy*, Pages: 133-145, May 1999.

[46] B. Yu, E. Byres and C. Howey, "Monitoring controller's 'DNA sequence' for system security", *Emerging Technologies Conference* (Instrumentation Automation and Systems Society), September 2001.

[47] A. Ahmed and I. Traore, "Detecting computer intrusions using behavioral biometrics", *Third Annual Conference on Privacy, Security and Trust*, October 2005.

[48] H. Debar, M. Becker and D. Siboni, "A neural network component for an intrusion detection system", *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, Pages: 240-250, May 1992.

[49] A. K. Ghosh, J. Wanken and F. Charron, "Detecting anomalous and unknown intrusions against programs, *Proceedings of the 14th Annual Computer Security Applications Conference*, Pages 259-267, December 1998.

[50] L. Mé, "GASSATA, a genetic algorithm as an alternative tool for security audit trails analysis", *Proceedings of the First International Workshop on the Recent Advances in Intrusion Detection*, September 1998.

[51] A. Chittur, "Model generation for an intrusion detection system using genetic algorithm", High School Honors Thesis, 2002.

[52] J. Gomez, D. Dasgupta, O. Nasaroui and F. Gonzalez, "Complete expression trees for evolving fuzzy classifiers systems with genetic algorithms and application to network intrusion detection", *Proceedings of NAFIPS-FLINT joint Conference*, Pages: 469-474, 2002.

[53] B. Balajinath and S. Raghavan, "Intrusion detection through learning behavior model", *Computer Communications*, Vol. 24, No. 12, Pages: 1202-1212, 2001.

[54] S. M. Bridges and R. M. Vaughn, "Fuzzy data mining and genetic algorithms applied to intrusion detection", *Proceedings of the Twenty-third National Information Systems Security Conference*, 2000.

[55] M. Crosbie and G. Spafford, "Applying genetic programming to intrusion detection", Technical Report, FS-95-01, AAAI Fall Symposium Series, AAAI Press, 1995.

[56] L. B. Lankewicz and M. Benard, "Real-time anomaly detection using a non-parametric pattern recognition approach", *Proceedings of the 7th Annual Computer Security Applications*, 1991.

[57] S. Staniford, J. Hoagland and J. Mcalerney, "Practical automated detection of stealthy portscans", *Journal of Computer Security*, Vol. 10, No. 1-2, Pages: 105-126, 2002.

[58] B. Scholkopf, J. C. Platt, J. S. Shawe-taylor, A. J. Smola and R. C. Williamson, "Estimating the support of a high-dimensional distribution", *Neural Computation*, 13(7), Pages: 1443-1471, 2001.

[59] K. Yamanishi, J. Takeuchi, G. Williams and P. Milne, "On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms", *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2000)*, ACM Press, Pages: 320-324, 2000.

[60] R. Sekar and A. Gupta, "Specification-based anomaly detection: A new approach for detecting network intrusions", *ACM Computer and Communication Security Conference*, 2002.

[61] M. L. Shyu, S.C. Chen, K. Sarinnapakorn and L.W. Chang, "A novel anomaly detection scheme based on principal component classifier", *Proceedings of the IEEE Foundations and New Directions of Data Mining Workshop*, in conjunction with the *Third IEEE International Conference on Data Mining*, Pages: 172-179, November 2003.

[62] M. M. Breunig, H. P. Kriegel, R. T. Ng and J. Sander, "LOF: Identifying density-based local outliers", *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, Pages: 93-104, May 2000.

[63] S. M. Emran and N. Ye, "Robustness of Canberra metric in computer intrusion detection", *Proceedings of the IEEE Workshop on Information Assurance and Security*, Pages: 80-84, June 2001.

[64] C. Ko, M. Ruschitzka and K. Levitt, "Execution monitoring of security-critical programs in distributed systems: A specification-based approach", *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, Pages: 175–187, May 1997.

[65] C. Ko, "Logic induction of valid behavior specifications for intrusion detection", *Proceedings of 2000 IEEE Symposium on Security and Privacy*, May 2000.

[66] J. R. Koza, *Genetic Programming*, MIT Press, 1992.

[67] M. L. Wong and K. S. Leung, *Data Mining using Grammar based Genetic Programming and Applications*, Kluwer Academic Publishers, 2000.

[68] K. S. Leung and K. F. Yam, "Rule learning in expert systems using genetic algorithms: 1, concepts", *Proceeding of the 2nd International Conference on Fuzzy Logic and Neural Networks*, Pages: 201-204, 1992.

[69] V. Paxson, "Bro: A system for detecting network intruders in real-time", *Proceedings of the 7th USENIX Security Symposium*, 1998.

[70] B.S. Everitt, "Unresolved problems in cluster analysis", *Biometrics*, Vol. 35, Pages: 169-181, 1979.

[71] R. C. Dubes, *Cluster Analysis and Related Issues in Handbook of Pattern Recognition and Computer Vision*", Eds. C.H. Chen, L.F. Pau and P.S.P. Wang, World Scientific Publishing Company, Pages: 3-32, 1993.

[72] A. P. Dempster, N. M. Laird and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm (with discussion)", *Journal of the Royal*

Statistical Society B, Vol. 39, Pages: 1-38, 1977.

[73] J. M. N. Figueiredo and A. K. Jain, "Unsupervised selection and estimation of finite mixture models", *Proceedings of the International Conference on Pattern Recognition* (ICPR2000), 2000.

[74] G. J. McLachlan, "On bootstrapping the likelihood ratio test statistic for the number of components in a normal mixture", *Applied Statistics*, Vol. 36, Pages: 318–324, 1987.

[75] A. Penalver and J. M. S. F. Escolano, "An entropy maximization approach to optimal model selection in Gaussian mixtures", *Proceedings of the 8th Iberoamerican Congress on Pattern Recognition*, CIARP 2003, Havana, Cuba, November 2003.

[76] S. Richardson and P. J. Green, "On bayesian analysis of mixtures with an unknown number of components (with discussion)", *Journal of the Royal Statistical Society: Series B*, Vol. 59, No. 4, Pages: 731-792, 1997.

[77] N. Vlassis and A. Likas, "A kurtosis-based dynamic approach to Gaussian mixture modeling", *IEEE Transaction on Systems, Man, and Cybernetics, Part A*, 29(4), Pages: 393-399, 1999.

[78] G. Rudolph, "Convergence analysis of canonical genetic algorithms", *IEEE Transactions on Neural Networks*, 5 (1), Pages: 96-101, 1994.

[79] T. Back, A. E. Eiben and N.A.L. Van Der Vaart, "An empirical study on GAs 'without parameters'", *Lecture Notes in Computer Science*, Vol. 1917. Springer-Verlag, Pages: 315–324, 2000.

[80] C. L. Blake and C. J. Merz, "UCI repository of machine learning databases", University of California, Irvine, Department of Information and Computer Sciences, 1998.

[81] F. Pouget, M. Dacier and H. Debar, "Honeypot: a comparative survey". Eurecom Report, RR-03-81, September 2003.

[82] F.S. Roberts, *Measurement Theory*, Addison-Wesley Publishing Company, 1979.

[83] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, Second Edition, Academic Press, 2003.

[84] PCA Tutorial, http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf.

[85] X-scan, http://www.xfocus.org.

[86] Fluxay, http://www.netxeyes.com.

[87] A. Lazarevic, A. Ozgur, L. Ertoz, J. Srivastava and V. Kumar, "A comparative study of anomaly detection schemes in network intrusion detection", *Proceedings of the Third SIAM International Conference on Data Mining*, May 2003.

Appendix

Brief Description of DARPA Attacks

In this appendix, we briefly describe a variety of multiple-connection based network attacks in 1998 DARPA intrusion detection dataset.

A.1 Apache2

An apache2 is a denial of service attack, in which an attacker sends a request containing many http headers to the targeted apache web server. The server will use an increasing amount of CPU time to process these headers, and thus may deny service to other user. The apache2 attack can cause the web server to become inaccessible, or at least painfully slow.

A.2 Back

The back attack is a variant of the apache2 attack, in which an attacker sends a URL request containing may slashes (i.e. '/') to the victim apache web server. As such, the server will be busy to process these malicious requests and deny services to other legitimate requests.

A.3 Land

A Land attack, named after the published exploit of that name, is an attack whereby a TCP SYN packet is sent with a spoofed source IP address and port number which matches that of the destination IP address and port. This causes some TCP implementations to go into a loop that crashes the machine. This attack can crash the

target system, or consume the CPU resources of the target to the point where no other activity can take place.

A.4 Mailbomb

A mailbomb attack occurs when an attacker sends a large number of messages to a mail server (i.e. pop3, smtp), overflowing the mail queue and causing mail system failure.

A.5 SYN Flood

A SYN Flood attack is a denial of service attack that exploits vulnerabilities of TCP/IP stack implementation. A detailed description regarding this attack refers to Figure 2.1 in chapter 2.

A.6 Ping of Death

The ping of death attack is a denial of service attack, in which an attacker sends an ICMP echo request packet containing a very large amount of information, such as oversized data to a target host. The kernel buffer of the target host will then be overflowed if it attempts to respond to this ICMP request. As a result, the host will crash following this attack.

A.7 Process Table

The process table exploits the vulnerability that network services will allocate a new process for each incoming TCP/IP implementation. It is possible that the process table of a target host will be filled with multiple instantiations of network servers, and thus other commands cannot be executed on the target host.

A.8 Smurf

A smurf attack is carried out by using ICMP protocol. In normal conditions, source host *A* sends an echo request message to destination host *B*, and then *B* makes an echo reply message to *A*. When a smurf attack occurs, *A* spoofs the IP address by using *C* and then sends an echo request message to a broadcast address so that the victim host *C* will receive all echo reply messages. As a result, links and routers to *C* may get clogged by

traffic, and *C* cannot receive requests from other users.

A.9 Teardrop

This attack crashes a system using an unusual fragmentation of IP packets that will cause a complete crash of a machine (blue screen) or a loss of network connectivity on vulnerable machines. The attacker sends two or more fragments that cannot be reassembled properly by manipulating the offset value of the packet and causes the victim's computer to reboot or halt its activities.

A.10 Udpstorm

The udpstorm attack is a denial of service attack, in which an attacker generates a never-ending data stream between two UDP ports in the same or different hosts. As a result, the host cannot offer services to other users and the network may be congested or slowdown.

A.11 Ipsweep

The Ipsweep attack determines which hosts are listening on networks through a surveillance sweep. The collected information can be used by attackers in staging attacks and searching for vulnerable hosts.

A.12 Mscan

Mscan stands for a probing tool set that uses both DNS zone transfers and brute force scanning of IP addresses in order to locate active hosts and test them for vulnerabilities.

A.13 Nmap

Nmap is a general-purpose scanning tool set. It provides a variety of portscans, such as SYN, FIN and ACK scanning with both TCP and UDP protocol. The nmap toolset also allows users to specify a set of parameters (i.e. which ports are scanned, how much time to wait between each port and whether the ports are scanned in a sequential order or a random order).

A.14 Saint

SAINT is the acronym of Security Administrator's Integrated Network Tool. It gathers a large amount of information from networking services, such as finger, ftp, telnet, etc. The information contains misconfiguration of network services, well-known vulnerabilities in operating systems or network utilities, and weak policy decisions. The attacker can exploit the useful information for further intrusions.

A.15 Satan

Satan is a previous version of saint. They are quite similar in purpose and design. The main difference between them is the particular vulnerabilities that they scan for.

A.16 Portsweep

The portsweep is used to scan which ports of a specified host are opened on networks. This attack determines the service types running on a specified host and then exploits these vulnerabilities to attack the host.

A.17 Dictionary

The dictionary attack is a remote to local user attack, in which an attacker attempts to gain access to a specified host by repeatedly guessing the possible usernames and passwords. Many services are easy to be attacked by this attack, such as ftp, rlogin, etc.

A.18 Guest

The guest attack is a variant of the dictionary attack. It exploits the vulnerabilities of a falsely configured system, in which the password of guest accounts is null. When the guest account of most operating systems is activated by default, the guest attack easily cracks these kinds of systems.

A.19 Imap

The imap attack can gain the root access to an imap server by exploiting a buffer overflow bug. The attacker usually sends a carefully crafted text to a system running a vulnerable IMAP server in order to cause a buffer overflow. Then the attacker gains the root access and can execute arbitrary instructions on the victim's server.