Seems interesting

DOI 10.12694/scpe.v20i1.1465
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 20, Number 1, pp. 113–160. http://www.scpe.org © 2019 SCPE

A DETAILED DESCRIPTION ON UNSUPERVISED HETEROGENEOUS ANOMALY BASED INTRUSION DETECTION FRAMEWORK

ASIF IQBAL HAJAMYDEEN* AND NUR IZURA UDZIR†

Abstract. Observing network traffic flow for anomalies is a common method in Intrusion Detection. More effort has been taken in utilizing the data mining and machine learning algorithms to construct anomaly based intrusion detection systems, but the dependency on the learned models that were built based on earlier network behaviour still exists, which restricts those methods in detecting new or unknown intrusions. Consequently, this investigation proposes a structure to identify an extensive variety of abnormalities by analysing heterogeneous logs, without utilizing either a prepared model of system transactions or the attributes of anomalies. To accomplish this, a current segment (clustering) has been used and a few new parts (filtering, aggregating and feature analysis) have been presented. Several logs from multiple sources are used as input and this data are processed by all the modules of the framework. As each segment is instrumented for a particular undertaking towards a definitive objective, the commitment of each segment towards abnormality recognition is estimated with various execution measurements. Ultimately, the framework is able to detect a broad range of intrusions exist in the logs without using either the attack knowledge or the traffic behavioural models. The result achieved shows the direction or pathway to design anomaly detectors that can utilize raw traffic logs collected from heterogeneous sources on the network monitored and correlate the events across the logs to detect intrusions.

Key words: Anomaly detection, Clustering, Heterogeneous logs, Filtering, Feature analysis

AMS subject classifications. 68T10, 68T05

1. Introduction.

The contemporary IDS are incompetent in taking advantage on the benefit of heterogeneous data sources for investigation to identify intrusions [1]. The significance of using several logs for intrusion detection was presented and emphasized by Abad et al. [2], and the outcome of attacks on various logs was demonstrated. As a proof of concept, the existence of an intrusion was determined by correlating the system calls with network logs and the experiments were conducted to detect a particular anomaly. Artificial data were used to train and test the model projected. Detection accuracy improved with log correlation, but then predict the next system call method underperformed for this problem. Apart from that, the description of attack traces and the way such information were extracted from various log sources used in the study were not described.

UCLog, a unified logging architecture [3] correlates events from various logs for intrusion detection. The correlations between the activities were utilized to achieve better accuracy and also trace the origin of intrusion exempting the administrator to examine such information. This was further extended as UCLog+ [4] to parse and store alerts and incident records. Cross-Layer Based Intrusion Detection and Prevention [5] detects intrusions by manipulating the data obtainable through multiple layers of the protocol stack. Most importantly, Denning (1987) [6] suggested to construct a framework for a general-purpose intrusion-detection expert system (IDES) which is independent of any system, application environment, system vulnerability, or type of intrusion needs to be extensively explored and unsupervised heterogeneous anomaly detection framework (UHAD) [7] is a step towards this directive.

2. Dataset Description.

The sensitivity of the data urges to remain proprietary affecting the availability of datasets regularly [8]. This creates a bigger challenge in gathering the appropriate data for the experiments. The most accepted log source utilized for intrusion detection nowadays is network traffic and the widespread use of network traffic was due to its availability and standardization [9]. Selecting useful data is a significant task in the pre-processing stage [10] and the most essential concern is the form of data sources to consider [4] as the selection of right data sources helps to identify various kinds of intrusions or attacks. To assess the capability of the framework proposed in detecting anomalies, the logs captured by heterogeneous sources are needed. One of the challenges of the Honeynet project i.e., Scan of the Month #34 (SOTM#34) [11], was used as input data that contains both intrusive and non-intrusive events.

*Faculty of Information & Sciences Engineering, Management & Science University, 40100 Shah Alam, Selangor, Malaysia (asif@msu.edu.my).

†Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia .

The dataset used for the experiments described in this paper has been chosen for the following reasons: appropriateness of the data and real traffic data.

Appropriateness of the data. Testing the proposed framework requires logs collected directly by heterogeneous sources, i.e., operating systems, applications and network devices. SOTM#34 dataset consists of logs collected from heterogeneous sources and hence appropriate. The KDDCup1999 and DARPA IDS evaluation datasets were commonly used for testing intrusion detectors consist of data sniffed from a particular location on the network (Homogeneous data) and not collected directly from the heterogeneous sources where the actions are destined to and therefore the usage of this data to test the framework is not applicable.

Real traffic data. The framework intends to detect a wide variety of attacks in the absence of a learned traffic model and therefore requires raw data (naturally unlabelled) recorded directly by multiple sources. While both KDDCup1999 and DARPA IDS evaluation dataset are synthetic, SOTM#34 contains real log data from heterogeneous sources and hence suitable. This dataset has been examined by several participants [12, 13, 14, 15] of the challenge, and also scrutinised by Panichprecha [16] and Herreras and Gomez [17], hence providing better confidence on the analysis results to assess the capability of the proposed framework. The dataset comprises of several logs i.e., Apache Server, Linux syslogs, Snort NIDS, and IPTables firewall, which were captured by dissimilar sources in a Honeynet system.

The Honeynet constitutes three key systems:

- Bridge, a multi-homed host operating an unknown distribution of Unix/Linux and performs routing/filtering using Netfilters IPTables kernel module.
- Bastion is a Network Intrusion Detection System running Snort with the Bleeding Snort rule sets in conjunction with those provided with the basic Snort package.
- Combo is the victim system running Red Hat Linux with a 2.4.20-8 kernel, an Intel Pentium III 740 MHz processor, and 128 MB RAM. It runs multiple virtual IP addresses on the 11.11.79.0/24 network.

The framework makes two assumptions about the data used:

Assumption 1: The data from the logging sources has logged every action and is free from unauthorised alteration(s) or fabrication(s).

Assumption 2: The majority of the logged events are usual traffic with a minority percentage of malicious traffic.

2.1. Data Pre-processing. Logs were written in a proprietary fashion, since there were no establishments to standardize log formats [18]. Therefore, the relevant features in a log needed for processing has to be extracted beforehand. Parsing is the process of extracting meaningful data [19], and translating the data into a format to ease and enhance further processing. Existing log parsing tools introduce new features derived from existing features in the resulting parsed log; and this log have to be parsed again to remove the additional features in order to retain the integrity of the logs. To avoid the overhead of double parsing and also to maintain logs integrity, we wrote our own parsers rather than using existing tools.

The significance of the custom written parser is as follows:

1. Able to extract all the features as exactly recorded in the log, irrespective of the difference in the delimiters.
2. Able to extract features precisely without losing any part of the value for the feature.
3. Does not use any kind of log format specifications to extract the features of a specific log.
4. Does not introduces new features derived from the existing features into the parsed log.
5. Able to adjust the timestamp of the events in a log for the time difference between the logs to support correlation, if the time difference is provided.
6. Able to remove unwanted feature names stored together with the values and also removes unnecessary punctuations, i.e., white space, comma, equal (=), appearing in between feature values. This is to avoid misinterpretations of treating a single feature as multiple features by the subsequent process.

However, due to difficulties in precisely extracting the features from Linux syslogs, Sawmill [20] was used for initial parsing. The parsed log contained '(empty)', symbolizing the absence of value for a particular feature; and this was parsed again to replace '(empty)' with '-' to maintain uniformity across the logs. The timestamp in all the logs was separated as date and time while parsing to ease synchronization. The parser is a part of our extractor which is integrated with supplementary functions, i.e., Isolator and Timestamp Synchronizer. The feature selection and clustering methods necessitate the number of features in an event should be the same throughout the log in order to be processed. Therefore, the isolator separates the events of IPTables firewall log in separate files, as there were differences in the number of features for different connections (TCP, UDP, ICMP) having 24, 22 and 21 features, respectively. Time is one of the important factors to correlate events between logs, hence the synchronizer adjusts the timestamp for the time difference between the logs.

Feature selection was used in intrusion detection for recognizing and eliminating insignificant features [21], and was applied on logs to assist the progressing process to enhance the accuracy in predicting and categorising abnormal events. Knowing the predictive ability of every feature in relation to every other feature in a log will assist in precisely identifying the important features. Therefore, feature selection was accomplished by setting all feature in the log as class attributes to recognise the connection of the set feature with other features. Since each subset contains events of different durations, feature selection was applied on all datasets to verify, whether the features selected for a particular log was uniform for all subsets. The selected features of all class attribute settings were summarized to find the unselected features, and every such unselected feature was removed from the respective parsed log before further processing. The same strategy was followed for various logs used for the experiments.

Keeping the evidence in a single common setup [18] by transporting all events from various logs together facilitates the analysis process in identifying most of the anomalies. The ability to distinguish normal behaviour from an attack can be better accomplished by analysing more features, but not every feature is relevant to the detection task [22]. Including all features in the logs will make the schema size bigger leading to difficulty in managing. Every log records features that they deem as important. Due to this nature, there were variations in the number of features recorded by a particular log and the values contained in these features. Especially with the logs from heterogeneous sources, the variations between the features were more diverse. Therefore, the process of framing a generic schema for a given set of logs becomes critical. To analyse the events from various logs together, a generic format (GF) was outlined, with common and significant features existing in the considered logs.

This features, i.e., timestamp, source IP, destination IP, source port, destination port, protocol, were also used by other researchers [23, 24, 25, 26] for intrusion detection. Moreover, Message was also incorporated in GF as it states the action performed by the event and their patterns were fully dissimilar for benign and malicious events. No additional features were introduced in GF to represent the log source an event belongs to, as it may mislead the clustering method. The framed GF not only states the features for GFL, but also the relevant features from various logs, that can fit in.

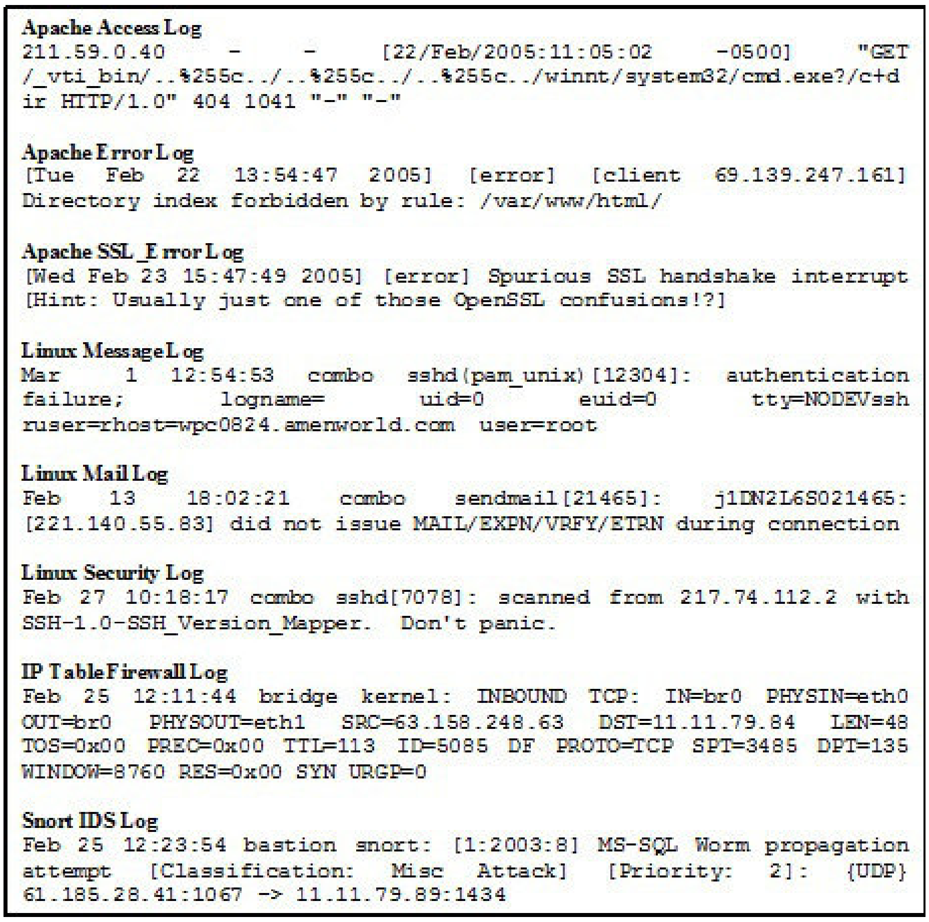
2.2. Data Treatment. This section describes the details on the treatment of data to make it viable for processing by the framework components. SOTM#34 log data was subjected to two processes namely feature extraction and feature selection.

2.2.1. Feature Extraction. Five operations were performed on the logs in this phase namely parsing, synchronizing, relocating, isolating and dividing and are defined as follows:

1. Parse: Extracting the relevant feature(s) from the log events and storing the extracted features using a common delimiter.
2. Synchronize: Adjusting the time stamp of a particular log to match the time difference with other logs, enabling the discovery of relationship between events across the logs.
3. Relocating: Moving the events from a subset to another after synchronizing to follow the duration covered by subsets.
4. Isolate: Separating the events in IPTables firewall log that differs in the number of features it contains.
5. Divide: Separating the IPTables firewall and Snort IDS events in several files to match the time duration covered by each subset.

Sample events from various logs are provided in Figure 2.1. The followings are the difficulties faced, while parsing the logs to extract the features precisely:

1. White space or blank space separates the features in an event by all the logs, while it also appears inside a feature value, e.g., Apache SSL-Error Log.



Apache Access Log
```
211.59.0.40    -    -    [22/Feb/2005:11:05:02    -0500]    "GET
/_vti_bin/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+d
ir HTTP/1.0" 404 1041 "-" "-"
```

Apache Error Log
```
[Tue   Feb   22   13:54:47   2005]   [error]   [client   69.139.247.161]
Directory index forbidden by rule: /var/www/html/
```

Apache SSL_Error Log
```
[Wed Feb 23 15:47:49 2005] [error] Spurious SSL handshake interrupt
[Hint: Usually just one of those OpenSSL confusions!?]
```

Linux Message Log
```
Mar    1   12:54:53   combo   sshd(pam_unix)[12304]:   authentication
failure;      logname=      uid=0      euid=0       tty=NODEVssh
ruser=rhost=wpc0824.amenworld.com  user=root
```

Linux Mail Log
```
Feb   13   18:02:21   combo   sendmail[21465]:   j1DN2L6S021465:
[221.140.55.83] did not issue MAIL/EXPN/VRFY/ETRN during connection
```

Linux Security Log
```
Feb 27 10:18:17 combo sshd[7078]: scanned from 217.74.112.2 with
SSH-1.0-SSH_Version_Mapper.  Don't panic.
```

IP Table Firewall Log
```
Feb  25  12:11:44  bridge  kernel:  INBOUND  TCP:  IN=br0  PHYSIN=eth0
OUT=br0   PHYSOUT=eth1   SRC=63.158.248.63   DST=11.11.79.84   LEN=48
TOS=0x00  PREC=0x00  TTL=113  ID=5085  DF  PROTO=TCP  SPT=3485  DPT=135
WINDOW=8760 RES=0x00 SYN URGP=0
```

Snort IDS Log
```
Feb 25 12:23:54 bastion snort: [1:2003:8] MS-SQL Worm propagation
attempt   [Classification:   Misc   Attack]   [Priority:   2]:   {UDP}
61.185.28.41:1067 -> 11.11.79.89:1434
```

FIG. 2.1. *Samples of Raw Logs*

2. Some of the features have delimiters to mark the start and end of a value, while others do not have such delimiters, e.g., Timestamp and Status Code in the Apache Access Log.
3. Delimiters are dissimilar with different features in the same log, e.g., [ ] for Timestamp and for ClientRequestLine in Apache Access log, [ ] for Classification and for Protocol in Snort IDS Log.
4. Delimiter used for a particular feature in the logs from a particular source was also not even, e.g., ClientIP in Apache access log is delimited by white spaces whereas Apache error log is delimited by [ ].
5. The number of features in an event for a particular log is not similar for all the events in that log, e.g., TCP, UDP and ICMP connections in IPTables firewall Log.
6. The name of the feature precedes most of the values for all the events in the log, e.g., IPTables firewall and Linux Message Log.
7. Time stamp format across the logs are not even, e.g., Apache logs includes year in time stamp while others do not, and the time stamp format of Apache access log differs from the Apache error and Apache SSL-error log.

There are no establishment to standardize log formats [18], and therefore every logging source writes log in its own format with the features based on what they consider as important. So, every log was parsed separately to extract the relevant features. Even though there are five operations in this phase, not all the logs are subject to all these operations. Variations in the package of data and the lack of time synchronization between the log events recorded by multiple sources which is very common in production environments necessitates separate treatment. Hence, this module provides three different apparatus to handle the logs as illustrated in Figures 2.2, 2.3 and 2.4. The common operation available in all the setup is parsing and the application of synchronization, relocation, isolation and division may vary with logs depending on the package and time difference of the collected data.

The first step in feature extraction is to identify the boundary of a feature and consequently extracting the feature values for every log event. To rectify the differences in the delimiters used with various features in the raw logs, the features in an event has to be given a common delimiter which in our case is a comma (,). Therefore, all the logs in text files has been parsed to transform the events with a common punctuation and was recorded in a Comma Separated Value (CSV) file that will ease and enhance further processing. Existing log parsing tools introduce new features derived from existing features in the resulting parsed log, and this log has to be parsed again to remove the additional features in order to retain the integrity of the logs. Moreover these tools do not offer facilities to synchronize the time difference across logs and to segregate events within a log, if there is a difference in the number of features recorded by the events. Therefore, custom written Perl scripts for different logs were used to implement this step. But, due to difficulties in parsing Linux Syslog, Sawmill [20] was used to parse this log.

Apache Server Logs and Linux Syslog. Apache server provides three logs which are access, error and SSL-error whereas Linux provides three logs that are message, mail and security. All these logs undergo three operations that are parsing, synchronizing and relocating (Figure 2.2). The number of features and the content

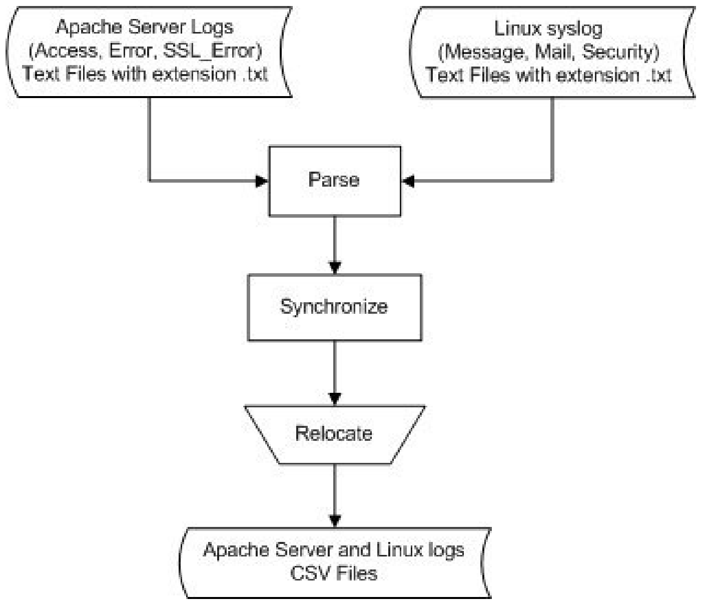

Apache Server Logs
(Access, Error, SSL_Error)
Text Files with extension .txt
Linux syslog
(Message, Mail, Security)
Text Files with extension .txt
Parse
Synchronize
Relocate
Apache Server and Linux logs
CSV Files

Fig. 2.2. *Apache and Linux Logs Parsing Apparatus*

of features varies with logs, therefore separate Perl scripts were written to parse each log. The logs provided in text file were parsed to extract the features and stored in a CSV file. During parsing, the timestamp was separated as date and time to facilitate synchronization. The year in which the events are recorded was not available in Linux Syslog and therefore introduced during parsing. The time between the logging sources i.e., bridge and bastion, was synchronized and thus IPTables firewall and Snort IDS events were in sync. But then, the time between the combo (victim machine) and the bridge was not synchronized.

Subsequently the Apache server and Linux syslog events contrasted by 4 hours 47 minutes against IPTables firewall log [15, 14, 13]. This was taken as the standard to synchronize and subsequently, the synchronizer balanced Apache server logs and Linux syslogs for the distinction towards the IPTables firewall log. Because of the adjustment in date and time after synchronization, a portion of the log events have been shifted to the particular subsets to maintain the time duration covered for every subset.

IPTables Firewall Log. The events of IPTables firewall log contains the name of the feature for most of the features in all the events recorded. Only the value for every feature in an event is needed for processing and on the other hand, keeping the feature name for every event increases the size of this log. Moreover, it causes additional overhead while processing events in the subsequent phases. Hence, the feature name preceding the value and the = that separates the feature name and feature value were removed during parsing. Moreover the timestamp was separated as date and time to match the format in other logs. After parsing, the events were isolated according to connections, i.e., TCP, UDP and ICMP, having 24, 22 and 21 features, respectively. The isolated events were maintained in three separate files and every such file is subdivided into four separate log files matching the duration of the other log subsets. This will allow the following process to handle different connections separately with subsets of moderate size. Since the Apache server log and Linux Syslog were synchronized towards IPTables firewall log, the timestamp of IPTables firewall log does not need synchronization. The pre-processing flow of IPTables firewall log is illustrated in Figure 2.3.

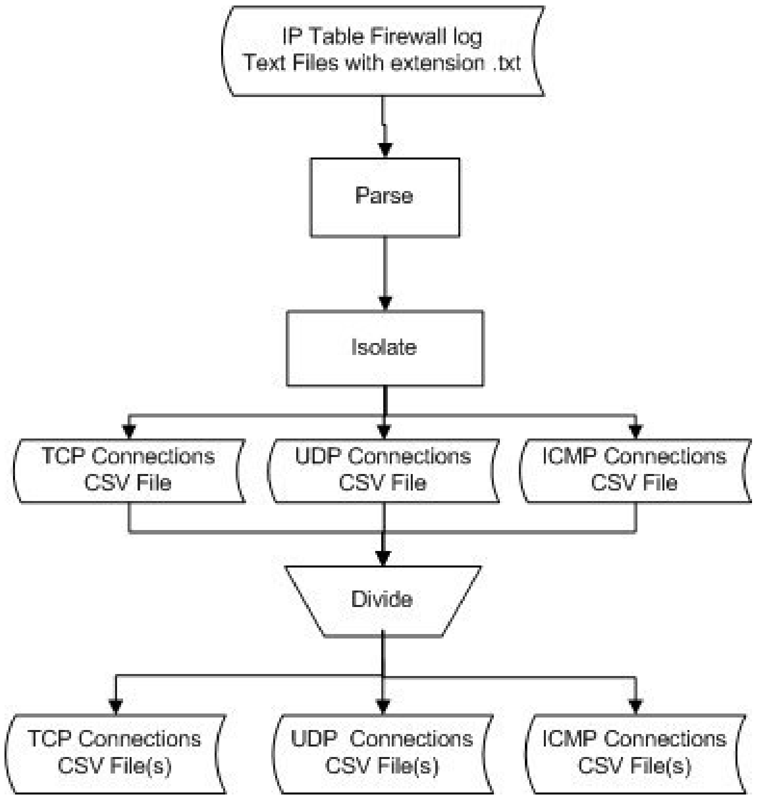

IP Table Firewall log
Text Files with extension .txt
Parse
Isolate
TCP Connections CSV File
UDP Connections CSV File
ICMP Connections CSV File
Divide
TCP Connections CSV File(s)
UDP Connections CSV File(s)
ICMP Connections CSV File(s)

Fig. 2.3. *IPTables Firewall Log Parsing Apparatus*

Snort IDS Log. Several Snort IDS log events did not have a value for the features classification and priority and those events that has the value was preceded by the feature name. Hence the name of these features in these log events were removed during parsing and a hyphen (-) was introduced for those events which do not have a value for this feature. Moreover, the timestamp was separated as Date and Time to match the format followed by other logs. The parsed log events were divided into four separate files to match the duration of the respective subsets tested. Synchronization and isolation were not needed for this log, as there was no time difference with IPTables firewall log and no difference in the number of features in between the events. The pre-processed logs were stored in CSV files, and the flow of process is illustrated in Figure 2.4.

Outcomes of Feature Extraction. Irrespective of the variations in the setup to handle different logs, the output is provided in CSV files that are ready to be processed by the following modules. The extracted features

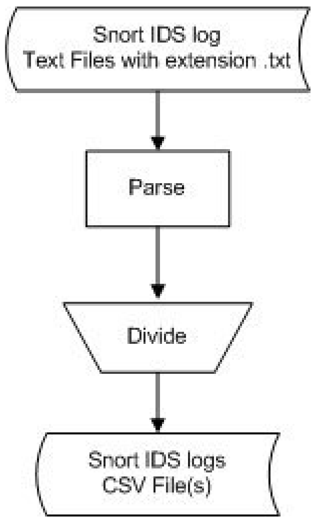


FIG. 2.4. *Snort IDS Log Parsing Apparatus*

of the logs belonging to particular subset is maintained separately. The time taken to parse and synchronize was less than a second for smaller sized logs, i.e., less than 1000 events, and a maximum of 27 seconds to parse and isolate 179752 events in the IPTables firewall log. The pre-processed logs were maintained in CSV files and the volume of events in every log by subset is provided in Table 2.1.

TABLE 2.1
Details of the Log Subsets

Log Type	**Subset-1**	**Subset-2**	**Subset-3**	**Subset-4**
Access	256	539	1280	464
Error	433	604	1270	484
SSL Error	74	64	29	48
TCP	9632	32646	34748	13210
UDP	613	1996	2387	1546
ICMP	171	789	1303	474
Message	112	149	208	103
Mail	42	62	54	31
Security	138	263	232	122
Snort	4423	17049	5013	10293
Total Events	15894	54161	46524	26775

2.2.2. Feature Selection and Retention. Selection and retention of features for every log is accomplished in two steps as illustrated in Figure 2.5 which includes:

1. Selecting features and summarizing the selected features identified for a particular log.
2. Removing unselected features from the respective log.

Weka's CfsSubsetEval estimates features by considering the specific predictive capacity of each feature and BestFirst scans for intricate dealings between features [27]. Since the target was to increase the precision by using only the features assisting in predicting the anomalous events, Weka's [28] CfsSubsetEval and BestFirst methods were used to implement this step. The log for which the features need to be selected is loaded into Weka Explorer. The attribute evaluator is set to CfsSubsetEval and the search method is set to Best-First. To perform the selection process, a feature in the log is set as class attribute to recognise the association of the fixed feature with further features and the features selected are noted. Knowing the predictive ability of every

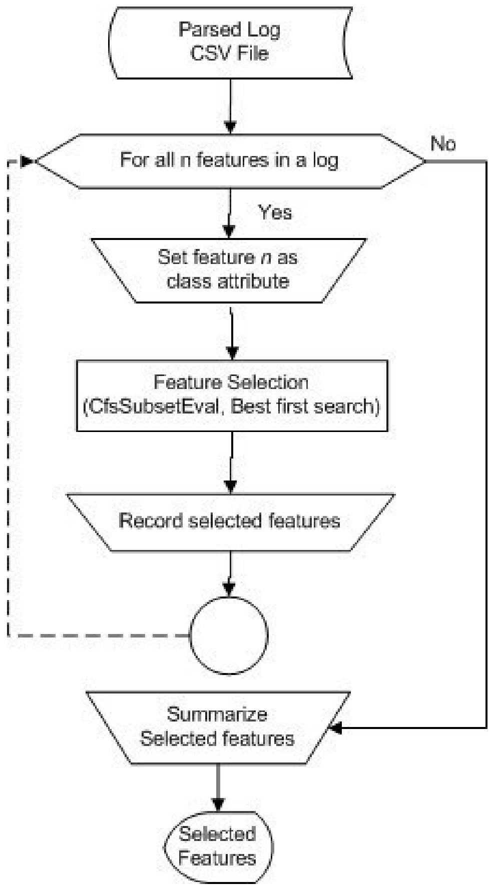

Parsed Log CSV File
For all n features in a log
No
Yes
Set feature n as class attribute
Feature Selection (CfsSubsetEval, Best first search)
Record selected features
Summarize Selected features
Selected Features

FIG. 2.5. *Feature Selection Strategy*

feature in relation to every other features in a log will assist in precisely identifying the important features and therefore every feature in the log is set as class attribute and the selected features were recorded. The number of times the features are selected for a log depends on the number of features in the log. The selected features of every class attribute setting were summarized to find the unselected features of a particular log. The same apparatus is used for different logs and every log used for the experiment is subject to this process to find the optimal set of features for a particular log. Subsequently, the features selected for a particular log across different subsets were summarized to find the unselected features. Every such unselected feature was removed from the respective parsed log before further processing. The selected features used for further processing is provided in Table 2.2.

Most importantly, this final set of features selected for every log consists of all significant features planned to be extracted for GFL except IPTables firewall log. The feature protocol in the IPTables firewall log was not selected for any class attribute setting. This is because the events of this log were previously isolated according to connections (TCP, UDP, ICMP), due to the difference in the number of features recorded for each of these connections. Therefore, the isolated events recorded in separate logs contained the same value for the feature protocol. Moreover the feature selection algorithm also requires the number of features in all the events to be the same. Eventually, applying feature selection on this isolated log events have not selected this feature for any class attribute setting, due to the similarity of the values for this feature, i.e. protocol.

2.2.3. Framing Generic Format. The features for GFL were identified by examining the features available in the logs. The only feature available in all the logs was the Timestamp (Date f_d, Time f_t). Features like Source IP Address (f_{sip}) existed in all logs except Apache SSLError and Linux mail log, whereas Destination

Table 2.2
Features Selected from Parsed Logs

Log		Features
Apache Server	Access	Date, Time, IPclient, Clid, Userid, clientRequestLine, Status Code, Object-Size, Referrer, Agent
	Error	Date, Time, clientIP, Msg
	SSL-Error	Date, Time, Severity, Msg
IPTables firewall	TCP	Date, Time, Bridge, Direction, Protocol, IN, PHYSIN, OUT, PHYSOUT, LEN, TOS, PREC, TTL, ID,DF, SPT, DPT, WINDOW, RES, STATUS, URGP, SRC, DST
	UDP	Date, Time, Bridge, Direction, Protocol, IN, PHYSIN, OUT, PHYSOUT, LEN, TOS, PREC, TTL, ID,DF, SPT, DPT, LEN, SRC, DST
	ICMP	Date, Time, Bridge, Direction, Protocol, IN, PHYSIN, OUT, PHYSOUT, LEN, TOS, PREC, TTL, ID,DF, TYPE, CODE, id, seq, SRC, DST
Linux Syslog	Message	Date, time, Logging device, Daemon, PID, Operation, User, Tty, UID, EUID, Remote host, System message
	Mail	Date, time, Logging device, From, To, Daemon, Mailer, Stat, Class, Priority, Protocol, Message ID, Relay, Control address, DSN, Queue ID, Messages queued, Messages delivered, Bytes queued, Bytes delivered, Delay, Xdelay
	Security	Date, time, Logging device, Daemon, PID, Operation, User, Source, System message, Messages
Snort IDS		Date, time, Logging device, Destination IP, Source port, Destination port, Classification, Snort priority

IP Address (f_{dip}), Source port (f_{sp}), Destination port (f_{dp}) and Protocol (f_{pr}) were available only in IPTables firewall log and Snort IDS logs. Initially GFL (Table 2.3) was constructed with these seven features from various

Table 2.3
Generic Format Specification - Version 1

Generic Format	Date	Time	Source IP	Destination IP	Source Port	Destination Port	Protocol	Log Source
Access	Date	Time	IPclient					Log name
Error	Date	Time	ClientIP					Log name
SSL-Error	Date	Time						Log name
TCP	Date	Time	SRC	DST	SPT	DPT	PROTO	Log name
UDP	Date	Time	SRC	DST	SPT	DPT	Protocol	Log name
ICMP	Date	Time	SRC	DST				Log name
Mail	Date	time						Log name
Message	Date	time	Remote host					Log name
Security	Date	time						Log name
Snort IDS	Date	time	SourceIP	Destination IP	Source port	Destination port	Protocol	Log name

logs. Additionally a feature by name log-source was introduced in GFL to specify the name of the log an event belongs to, since it consists of events from various logs. The aforesaid features of the filtered events from all the logs (whichever GFL features available in the respective log) were brought forward to GFL. Clustering this GFL resulted in higher false negatives affecting the accuracy of clusters, which eventually resulted in detecting only 12% of the anomalous events in the log. Message (f_{me}) is a feature available in most logs except IPTables firewall log, but it has different names in various logs. Therefore, a common name (Message) was suggested, and the respective features from various logs were extracted accordingly.

The new version of GFL (Table 2.4) includes all the features from the previous version together with the Message. Clustering the GFL with these nine features has improved the accuracy (nearly 80% in average) than using the previous version of GFL. But, the subsequent detection of anomalies remains approximately the same as the previous. The improvement in the accuracy was due to the introduction of Message, but still the usage of log-source as a feature of GFL which does not exist as a feature in the actual logs affected the classification of events. In order to avoid the effect of log-source in classifying the events, log-source was removed from the GFL. The next version of GFL constitutes only eight features excluding log-source and is presented in Table 2.5. The features were extracted accordingly from the respective logs and recorded in GFL. Clustering this GFL improved the accuracy to nearly 90% and a substantial increase in detected anomalous events. IPTables firewall log merely watches and records the traffic through the network and it can be taken as an additional option to improve detection. Therefore, it was excluded from the GFL used for clustering and the events were maintained separately to be used for correlation during analysis. Many of the features stated in GFL, especially IP address

Table 2.4
Generic Format Specification - Version 2

Generic Format	**Date**	**Time**	**Source IP**	**Destination IP**	**Source Port**	**Destination Port**	**Protocol**	**Message**	**Log Source**
Access	Date	Time	IPclient					Client Request Line	Log name
Error	Date	Time	ClientIP					Msg	Log name
SSL-Error	Date	Time						Msg	Log name
TCP	Date	Time	SRC	DST	SPT	DPT	PROTO		Log name
UDP	Date	Time	SRC	DST	SPT	DPT	Protocol		Log name
ICMP	Date	Time	SRC	DST					Log name
Mail	Date	time							Log name
Message	Date	time	Remote host					System Message	Log name
Security	Date	time						System Message	Log name
Snort IDS	Date	time	SourceIP	Destination IP	Source port	Destination port	Protocol	Rule	Log name

Table 2.5
Generic Format Specification - Version 3

Generic Format	**Date**	**Time**	**Source IP**	**Destination IP**	**Source Port**	**Destination Port**	**Protocol**	**Message**
Access	Date	Time	IPclient					Client Request Line
Error	Date	Time	ClientIP					Msg
SSL-Error	Date	Time						Msg
TCP	Date	Time	SRC	DST	SPT	DPT	PROTO	
UDP	Date	Time	SRC	DST	SPT	DPT	Protocol	
ICMP	Date	Time	SRC	DST				
Mail	Date	time						
Message	Date	time	Remote host					System Message
Security	Date	time						System Message
Snort IDS	Date	time	SourceIP	Destination IP	Source port	Destination port	Protocol	Rule

and port numbers, were not available in Apache SSL-Error and Linux mail log. Therefore, the events belonging to these logs were also excluded from the GFL that was used for the following process, i.e., clustering, and were maintained separately as per the features stated in GFL (Table 2.6) to be used at some stage during analysis. Out of the eight features used to construct GFL, f_{me} has gained critical importance because it exhibits more about the action performed on the system than the other features. The Message (f_{me}) feature was available in all the logs except IPTables firewall log and therefore the STATUS feature available in TCP connections of this log was used as Message. The Source IP (f_{sip}) and Source port (f_{sp}) appears as a part of the SystemMessage in Linux security log. In order to facilitate the clustering and further analysis, both of these features has been extracted and replicated as a separate feature in this log for all the events where Source IP (f_{sip}) and Source

Table 2.6
Generic Format Specification - Proposed

Generic Format	**Date**	**Time**	**Source IP**	**Destination IP**	**Source Port**	**Destination Port**	**Protocol**	**Message**
Access	Date	Time	IPclient					client RequestLine
Error	Date	Time	ClientIP					Msg
Message	Date	time	Remote host					System message
Security	Date	time	Source		Port			System message
Snort IDS	Date	time	SourceIP	Destination IP	Source port	Destination port	Protocol	Rule
SSL-Error	Date	Time						Msg
TCP	Date	Time	SRC	DST	SPT	DPT	PROTO	STATUS
UDP	Date	Time	SRC	DST	SPT	DPT	Protocol	
ICMP	Date	Time	SRC	DST				
Mail	Date	time						

port (f_{sp}) is available in SystemMessage. The final version of GFL used for analysis was constructed with eight features as mentioned in Table 2.6. The GFL features and the corresponding features chosen from various logs that fit in GFL were tabulated (Table 2.6). The usage of GFL with these features (as stated in Table 2.6) increased the clustering accuracy to an average of 95% with various subsets which naturally improved the volume of anomalies detected.

Adding features to GFL, which is available in any one of the logs considered, decreased the clustering accuracy and also the subsequent detection of anomalies. Therefore, the proposed GFL was considered appropriate for the set of logs considered for detection. Moreover, the set of features chosen for GFL not only applies for the logs considered, but also to a wide variety of logs of similar nature. However, the reader should note that the selection of features for GFL and the appropriate features selected from different logs was based on the logs considered and it may differ with different logs.

3. UHAD Components. The main objective of the framework is to detect anomalous events by correlating the information available in various logs and the process involved to achieve that is summarised in the overall algorithm. Apart from this, every single process stated in the algorithm has been designed to achieve a specific goal that contributes to the ultimate objective. Therefore, this section describes various components of the framework used in the process of anomaly detection and also specifies the reason behind choosing these components. It also states the metrics used to evaluate the performance of each component.

3.1. Clustering Events. The objective of this step is to separate abnormal events from the normal events for various logs. As there were many patterns in the log for both normal and abnormal, the number of distinctive patterns needs to be identified. Manual analysis of log events to identify unique patterns is almost impractical as it takes longer time and prone to errors. Therefore, the usual events are to be recognized and reduced before analysis to decrease the processing overhead.

Clustering is an unsupervised learning scheme for grouping similar or identical events together. It also has the ability to decide on the existence of intrusive events in the raw logs [29] by grouping these events in separate clusters. Therefore, we make use of the clustering algorithms to group the log events according to the patterns. The traditional steps like training and testing employed in machine learning based intrusion detectors [31] and data mining based intrusion detectors [30, 32, 33, 34] by splitting or rearranging the test data to differentiate with training data to suit the requirements of a particular study was not used. Even though this method often results in better precision and accuracy, it deteriorates performance, if the tested data contain events that were not learned by the training model, making it unreliable for real time detection. To overcome these limitations of model based approaches, only the clustered events of a particular log were used for further analysis and not the cluster model which was usually used to detect anomalies in upcoming log events. Therefore, the proposed strategy makes use of the existing clustering algorithms to group the log events and the parameters required by these algorithms were manipulated based on the log examined. The clustering methods used necessitate the amount of cluster (K) to assemble the events. As several logs were used, clustering each log with different clusters (K) to select the best cluster is time consuming. In order to isolate dissimilar event patterns in detached clusters, K was manipulated based on the patterns in the log. A procedure comprising two stages was used for isolating log events, which are:

Step 1: Predicting the best number of clusters (K_{ij}) for a given E_{ij} , and

Step 2: Clustering E_{ij} using the predicted K_{ij} .

Applying the clustering method distinctly for dissimilar services improves the detection quality [35]. Every log is an outcome of a service and therefore every single log from a source was treated separately for this step. IPTables firewall log is an additional option to improve the ways of detection as it just watches and records the traffic through it. Therefore, it was excluded for clustering and will be used for correlation during analysis to support the anomalies identified.

In step 1, the ideal number of clusters (K_{ij}), i.e., the number of clusters that is appropriate for a particular log to be grouped, was manipulated according to the experimental setting stated in Experiment 1. The manipulated K_{ij} serves as the output for step 1 and accordingly K_{ij} was used to cluster the respective logs in step 2.

There are numerous clustering methods and every method generates clusters differently for the same dataset. This poses the difficulty in deciding the appropriate algorithm for a particular context. The usage of existing clustering algorithms, especially K-Means and EM clustering for intrusion detection is very common and was used in many previous works [30, 32, 35, 36, 37, 38, 39, 40, 41, 42]. Siriporn and Benjawan [43] used Farthest First (FF) and K-Means clustering to detect intrusions [44] and used FF especially to detect rare attacks. This exposes the application of these algorithms for intrusion detection and the results achieved suggest the use of these algorithms for this purpose.

K-Means [45] treats all features equally [42] and is computationally faster with the ability to handle larger datasets. Moreover, it is order independent, i.e., it generates the same clusters of data irrespective of the sequence of the data presented. Since voluminous data received from the logging sources have to be grouped with minimum time consumption to facilitate anomaly detection, the usage of K-Means is applicable and appropriate. Farthest First (FF) [46] is a fast, simple, approximate, [28] hierarchical and distance based clustering using a distance measurement analogous to K-Means [47]. The calculation of cluster centroids in consecutive iterations by FF is contrary to that of K-Means, i.e., places each cluster centroid in turn at a point farthest from the current cluster centroid. This speeds up the cluster process due to less reassignments and adjustments with most datasets. This nature of FF helps to produce accurate clusters provided the events were qualitatively different from the other with minimum time consumption. EM [48] is an iterative clustering algorithm that groups the data in a way that is different from K-Means. It has the ability to optimize large number of features and also finds good estimates of the missing values in the features of a dataset. The log events examined contains many features and the values for some of the features were not available in many of the events in the log. In this case EM provides better judgement on the missing values in the logs thereby producing better clusters. Although, K-Means and FF are faster in clustering compared to EM, all the three algorithms were tested to know its ability in accurately grouping different patterns and volume of events.

In step 2, the events from various logs were clustered using the respective K_{ij} manipulated from step 1. Three clustering algorithms namely K-Means, EM and FF were used for clustering and its accuracy in clustering events of various logs were compared. The chosen clustering methods uses K in generating clusters and seed value to resolve on initial cluster centres. Even the same algorithm produces dissimilar clusters with different parameter initializations, led to the difficulty in selecting the suitable parameters. Because of this reason, the K value needed for clustering was predicted using EM in the previous step, i.e., step 1. No methods were available to manipulate the appropriate seed value for a given set of events. Choosing random seed values and clustering several times will increase the processing time. Therefore, the default seed value or the seed value equivalents to the number of events being clustered were used with different settings. The experiment for step 2 was conducted according to Experiment 2. Apart from that, in order to analyse the performance of the algorithm in separating normal and abnormal events in various logs, the clusters generated by various algorithms and its settings with different logs were examined.

3.2. Filtering Clustered Events. The objective of this phase is to eliminate the usual events (noise) whilst holding the anomalous events for subsequent processing. Filtering is a process of reducing events for further analysis that are unlikely to hold information of importance [19]. Hence the clustered events were filtered to remove the unneeded events. Since anomalous events were less in number compared to benign events generated by normal usage, it falls in smaller clusters. Therefore, the smaller clusters needs to be identified based on the volume of events it contains. Since, multiple logs with different event volumes were used for detection, deciding a common cluster size to identify smaller clusters is not reasonable. Hence, a threshold (E_t) for identifying sparse clusters was calculated based on the volume of events and clusters for a particular log and the threshold is defined as:

$$E_t = \{E_{ij}/k_{ij}\} \tag{3.1}$$

Threshold may vary with logs due to the difference in the volume of events and the number of clusters generated for each log. Filtering events using this calculated threshold was to remove normal events (referred as filtered-out events) and to retain abnormal events (referred as filtered-in events) for further scrutiny. Therefore, the characteristics of anomalies were not required for filtering events. The clustered logs were filtered and evaluated according to the specification stated in Experiment 3.

3.3. Aggregating Filtered-In Events. The aim of this step is to combine the redundant events thereby reducing the events in the filtered log. Though, the filtered-in events were basically abnormal, using only the unique events will reduce processing overhead and increase the accuracy in formation of clusters. Even though much work use sampling methods for data reduction, it has been noted by Tavallaee et al. [49], that sampling methods in anomaly detection introduces a significant bias that degrades the performance. Hence, we chose aggregating instead, as both were basically data reduction techniques. Aggregation merges redundant records into a single record [19]. A group of two or more events were united, if all the features in the events were accurately analogous to the directly succeeding event(s), and the accumulated event serves as the representative. No features were introduced in the aggregated log to symbolize those aggregated events. The volume of events that gets reduced due to aggregation is based on the feature values that an event contains and therefore there are chances of zero reduction, if all the events in the filtered log were unique. Therefore the percentage of events reduced was not evaluated.

3.4. Transferring Events. The objective of this step is to extract the selected features of all the events from various aggregated logs as stated in GF and appropriately placing the respective features in GFL. This is to enable the analysis process to examine the events from various logs in a single structured format. Since all the GFL features were not existing in all the logs, absence of a specific feature in a log was substituted with a hyphen (-) during transfer. The performance of this step was verified to make sure whether the stated features in GF for every single event in the aggregated logs (E_{ij}^{**}) were completely transferred to GFL.

3.5. Clustering GFE. Despite the fact that the logs were clustered and filtered earlier, there are probabilities of having a less number of usual or irrelevant events; due to the imprecision in clustering and/or the succeeding withholding by the filtering threshold. To determine such events and also to discover the association between the events in GFL which comprises events from several logs, GFE was re-clustered. The GFE was clustered using the same two-step strategy, algorithms and parameter settings used previously for the individual logs. GFL was maintained separately according to the algorithms and settings used in the previous phases. This is to enable separate treatment of GFL accordingly in this phase and also the subsequent phases with respect to the algorithms and settings. The step 1 of the clustering strategy was carried out as per the setup provided in Experiment 4. Experiments for step 2 was conducted according to the algorithms and settings provided in Experiment 5. The clustered events (GFE) were stored in ARFF format and maintained separately with respect to algorithm and the respective settings.

3.6. Detecting Anomalous Events by Analysing Features. The aim of this step is to analyse clustered events to identify the relation between them with respect to the features it contains and thereby detecting the anomalous events from various logs. A mixed approach that uses multiple features of anomalies might be an eligible solution for different circumstances [50]. Therefore, the analysis process concentrates on features namely IP address (f_{sip}, f_{dip}) and port numbers (f_{sp}, f_{dp}). Additionally cluster number (f_k) was also used to get a clear picture of the intrusions.

3.6.1. IP Address Analysis. Normally an intrusion leaves multiple signs of its presence in various logs [2, 4]. As such, the events related to an intrusion may have been captured by various logs. To discover the presence of such intrusive events in multiple logs, this step identifies the relation between the IP address, i.e., source IP address (f_{sip}) and destination IP address (f_{dip}), of the events from various logs and the clustered GFL events (GFE) were used for analysis. Every abnormal activity that was captured by Apache server logs and Linux syslog must have raised an alert in Snort IDS log, as IDS has this capacity by design. So, the IP address that exists in Linux and Apache events that also exist in Snort IDS events were identified primarily. Not all IP address in Apache and Linux log has an matching with Snort IDS log, since some of the anomalous activities may have been missed by Snort IDS; but then, the anomalous patterns must have joined together in the same cluster during clustering. Therefore, to detect also those abnormal activities which were missed by Snort IDS log, all the events in those clusters to which the identified IP address belongs to were extracted and considered as anomalous. The experiments for this step were conducted and the detected anomalous events IPAE were evaluated as per Experiment 6. Attacks are frequently launched from an IP address or from an IP subnet [37] and therefore capturing every abnormal action originating from various IP address becomes significant. This is the reason behind evaluating the coverage of IP address in the detected anomalous events apart from the anomalous events itself.

3.6.2. Port Number Analysis. The objective of this step is to identify anomalous events based on port numbers. This is not a substitute to IP Address analysis, but to recognise and retain those anomalous events which were available in those clusters that were not identified by IP analysis. The findings of Kim et al. [51] also reveals that the usage of port numbers for classifying network traffic is still applicable and also suggest to use port based analysis methods. Most of the anomalous activities were launched from a host by exploiting the unassigned and dynamic ports, since no fixed service was running on these ports. Therefore, the source port number (f_{sp}) and the destination port number (f_{dp}) of the events were checked against the listing of the dynamic and unassigned port numbers as per Internet Assigned Numbers Authority (IANA). As most of the normal events have been filtered out in the previous phases, those events in (GFE) having port numbers matching with the IANA listing of dynamic and unassigned port numbers were detected and considered as anomalous. The experiments for this step were conducted and the detected anomalous events (PAE) were evaluated as per Experiment 7.

3.7. Consolidating Anomalous Events. The objective of this step is to consolidate the anomalous events discovered by both analysis methods to serve as the output of the framework. This was achieved by performing three operations namely combining, correlating and concentrating the events. Some of the anomalous events which were recognized during IP analysis may also have been recognized during port analysis. Therefore, these anomalous events, i.e., IPAE and PAE were compared to identify the distinct events and every event was brought together for correlation. To recognize the relation between the events of Linux Syslog, Apache logs and Snort IDS log with IPTables firewall log, the IP address (f_{sip} and f_{dip}), excluding internal IP address) of these events were compared with IPTables firewall log; and the matching events from IPTables firewall log were extracted and appended with the previously combined events. To concentrate on the most critical anomalous events, less significant events were reduced using a threshold (A_t) on the occurrence of events pertaining to an IP address. This was accomplished by identifying IP addresses of events which satisfies a specific threshold (A_t), and consequently the events pertaining to these IP address were extracted. These extracted events are deemed as anomalous and serves as the result of the framework. The experiments for this step were conducted and the consolidated anomalous events (AE) were evaluated as per Experiment 8.

4. Experimental Design for UHAD. This section illustrates the Unsupervised Heterogeneous Anomaly Detection Framework (UHAD) and the data pre-processing steps were discussed in detail followed by the abstract description of the framework. The inner details of the framework components were described together with the algorithm or strategy used to implement the components. Additionally, the flow of data, the input received and the output generated by each of the components were also presented. Finally, the contribution of the framework components towards anomaly detection was also discussed. The details of the experiments done at each step, i.e., clustering, filtering and analysis, of the framework are described in this section. This includes the respective algorithms used, together with the parameter settings or the needed parameters of the algorithm and the parameters evaluated.

4.1. Experiment 1: Manipulating Ideal Clusters for Individual Logs. Prediction of the best number of clusters by clustering the logs using EM clustering with the default parameter values, i.e., K = -1 and seed = 100 (K is used by the clustering algorithm to classify the given set of events into K clusters and seed is used to identify the initial cluster centre for every cluster). Apache server log constitutes three logs namely access, error and SSL-error log whereas Linux Syslog constitutes three logs namely message, mail and security log. All the events recorded by Snort IDS were provided in a single log. Every log was treated separately by EM clustering to manipulate the ideal number of clusters that a log can be clustered.

4.2. Experiment 2: Clustering Individual Logs. To capture the impact of K and seed in clustering, the algorithms were tested with four different parameter settings. The default seed value of K-Means, EM and FF are 10, 100 and 1, respectively. The settings used are as follows: Setting-1 (S1): Ideal clusters (K_{ij}) manipulated in step 1.1 of the overall algorithm with the default seed of the respective clustering algorithm. Setting-2 (S2): Ideal clusters (K_{ij}) manipulated in step 1.1 of the overall algorithm with the seed value set to the total number of events in a particular log (L_{ij}). Setting-3 (S3): Doubling up ideal clusters (K_{ij}) manipulated in step 1.1 of the overall algorithm with the default seed of the respective clustering algorithm. Setting-4 (S4): Doubling up ideal clusters (K_{ij}) manipulated in step 1.1 of the overall algorithm with the seed value set to the total number of events in a particular log (L_{ij}). The experimental setup for the step 2 of the clustering strategy is provided in Table 4.1. Every log was clustered with all the three algorithms and four settings and therefore the experiments were conducted 12 times on every log. Subsequently the clustering accuracy achieved with different algorithms and settings were calculated separately. The ability of the clustering algorithm in identifying and

Table 4.1
Experiment 2 - Clustering Individual Logs

Algorithms	Settings	Evaluated Parameter
K-Means, EM, FF	1, 2, 3, 4	Accuracy

placing the events in the respective cluster gains significance, as it helps the following component to identify and remove those clusters which are insignificant. Hence, the quality of clusters produced by these algorithms was calculated using Weka Experimenter with 10 fold cross validation and 10 iterations to allow every part of the log to be tested. A true positive (TP) decision assigns two similar events in the same cluster whereas a true negative (TN) decision assigns two dissimilar events to different clusters. Failure to assign the events in the appropriate cluster is measured using false positive (FP) and false negatives (FN). FP decision assigns two dissimilar events to the same cluster whereas FN decision assigns two similar events to different clusters. All these four measurements decides the cluster goodness and therefore the accuracy of clustering is calculated using the following formula:

$$Accuracy = \frac{TP + TN}{FP + FN + TP + TN}$$

The accuracy achieved by various algorithms and settings with various logs were evaluated.

4.3. Experiment 3: Filter-in Events of Sparse Clusters. Every clustered log with the respective algorithms and settings were filtered separately according to the threshold calculated for the respective log. During the calculation of the threshold, if the calculated threshold turns to be a decimal number, e.g., 20.3, it was rounded to the next ascending integer, e.g., 21. The number of events in the cluster, i.e., cluster size, is a whole number and the cluster size must be less than the threshold to be filtered-in, the threshold was rounded to the next integer. The percentage of events reduced with different logs with the algorithms and settings due to the application of the threshold is evaluated and the volume of abnormal events retained for further examination were analysed. This is to ensure that, the application of filtering threshold is able to reduce the log events and at the same time, retains the abnormal events.

4.4. Experiment 4: Manipulating Ideal Clusters for Individual Logs. Prediction of the best number of clusters for GFL using EM clustering with the default parameter values, i.e., K = -1 and seed = 100. As GFL was maintained separately for each algorithm and setting, the clusters appropriate for that GFL were manipulated separately.

4.5. Experiment 5: Clustering GFL. Using the K manipulated in the first step (Experiment 4), GFL was clustered with the configuration provided in Table 4.2. TP, TN, FP and FP were measured to calculate the accuracy of clusters formed using the formula stated in Experiment 2.

Table 4.2
Experiment 5 - Clustering GFL

Algorithms	**Settings**	**Evaluated Parameter**
K-Means, EM, FF	1, 2, 3, 4	Accuracy

4.6. Experiment 6: Anomaly Detection using IP Address. Every clustered GFL, i.e., GFE, with the respective algorithms and settings were analysed separately based on the relationship between the events with respect to IP Address and Cluster Number. The volume of anomalous events (IPAE) detected and the volume of IP addresses covered by these anomalous events was evaluated. Additionally, the results of IP analysis was compared with SOTM/#34 analysis results which serves as the ground truth for the anomalies in this dataset. This is to ascertain the ability of the framework in detecting anomalous events without using a knowledge-base or traffic models.

4.7. Experiment 7: Anomaly Detection using Port Number. Every clustered GFL with the respective algorithms and settings were analysed separately based on port numbers. The volume of anomalous events detected and the volume of IP addresses covered by these anomalous events were evaluated.

4.8. Experiment 8: Consolidating Anomalous Events. The anomalous events detected by IPA and PA with the respective algorithms and settings were consolidated separately. The threshold was used to focus on the most significant anomalies that needs to be immediately addressed and the impact of the variable threshold (A_t), i.e., 3, 6, 9, in retaining the anomalous events from various IP addresses were evaluated. A three-way handshake (also called as three message handshake and/or SYN-SYN-ACK) is a method used to set up a connection over an Internet Protocol based network. In other words, there must be at least three events related to an IP address in a traffic log to signify the establishment of a connection between two machines. Hence the threshold was chosen as three, which requires at least three events from an IP address to exist in the log to be selected as consolidated. This will also show whether the victim machine has responded to the request of the attacker. The threshold is increased by six and nine to focus especially on those events which may provide enough evidence on the anomalous actions between attack source and victim.

To overcome the limitation of unsupervised anomaly detection approaches, we propose UHAD (Unsupervised Heterogeneous log-based Anomaly Detection), a knowledge independent framework that uses unsupervised clustering algorithms to detect anomalous events from heterogeneous logs. The grouped log events are further examined by several knowledge independent functions to detect anomalies. Every component receives the processed log events from the preceding component, and manipulates the needed parameters for the process based on the log events received.

The components of the framework are implemented using the following applications:

- Weka (Waikato Environment for Knowledge Analysis) is a popular collection of machine learning algorithms written in Java which can be applied directly to a dataset or called from your own Java code and is well-suited for developing novel machine learning schemes. It supports several data mining tasks and particularly data pre-processing, classification, regression, clustering, feature selection, and visualization.
- Perl is a high-level, interpreted, dynamic programming language offering dominant text processing services with no limitations on the data size enabling straightforward manipulation of text files. Feature selection tools in Weka were utilized during data pre-processing and the clustering tools were used to

implement the two-step clustering strategy proposed in the framework. The new algorithms proposed for filtering, aggregating, transferring, consolidating and analysing features were written using Perl.

5. Design of UHAD Framework. The goal of the framework is to detect anomalies in an unsupervised fashion without using any kind of knowledge on attacks or a trained model of network behaviour. This is accomplished by correlating and analysing the event features in heterogeneous logs. The framework consists of two major phases with several components in each phase. The first phase considers individual logs from multiple sources separately, starting from clustering and passes through several components before the events from various logs were transferred to a common format (i.e., Generic Format Log (GFL)), whereas the next phase considers the events in GFL as input, which was clustered and analysed to identify anomalies. UHAD primarily relies on the pattern of log events and its features to detect anomalies, making it applicable to the evolving network traffic environment. The overall framework of UHAD is illustrated in Figure 5.1, and the first and second phases are illustrated in Figure 5.2 and Figure 5.3, respectively.

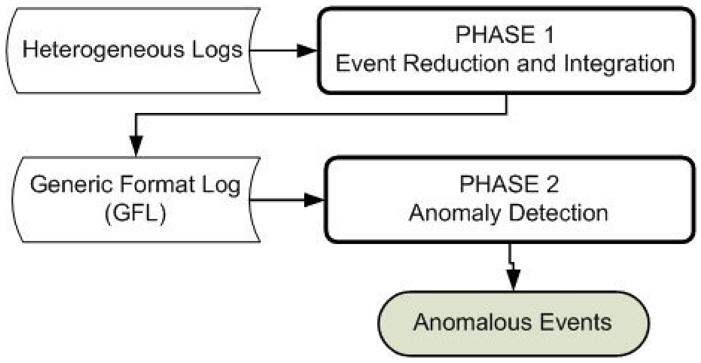


FIG. 5.1. *Overall Framework of UHAD*

Only the Cluster Events component, i.e., component (1) in Figure 5.2 and component (5) in Figure 5.3, utilize existing algorithms available in Weka, while other components are newly proposed algorithms written in Perl.

5.1. UHAD Components. The framework is composed of seven co-operating components, where every component performs a specific task in the process of anomaly detection, and the details of the tasks carried out by each component are described in the following sections.

5.1.1. Clustering Events. The log events were clustered using a two-step strategy and every log was treated separately for this step. The log is subjected to two operations as illustrated in Figure 5.4 and both the operations were implemented using Weka.

Predicting Clusters. Out of the three clustering algorithms (i.e., K-Means, EM and FF) chosen to be used in the framework, EM alone has the capacity to predict the number of cluster that is appropriate for the given dataset. This unique capacity of EM clustering is utilized to implement the first step in our clustering strategy. The Weka [28] implementation of EM algorithm manipulates the number of clusters for a given set of instances using cross validation by separating the instances into a number of partitions called folds. The steps involved in predicting the number of clusters by cross validation are as follows:

1. Number of clusters (k) is set to 1.
2. Training instances are split randomly into 10 folds.
3. EM is executed 10 times with 10 folds by the usual cross validation.
4. Log likelihood is averaged over all the 10 results.
5. If the log likelihood has increased, the number of clusters (k) is incremented by 1 and the process repeats from step 2.

For those training sets containing 10 instances or more, the fold is set to 10 otherwise the number of folds is set to the number of instances. In the first step a given set of events E_{ij} is loaded to Weka to manipulate the

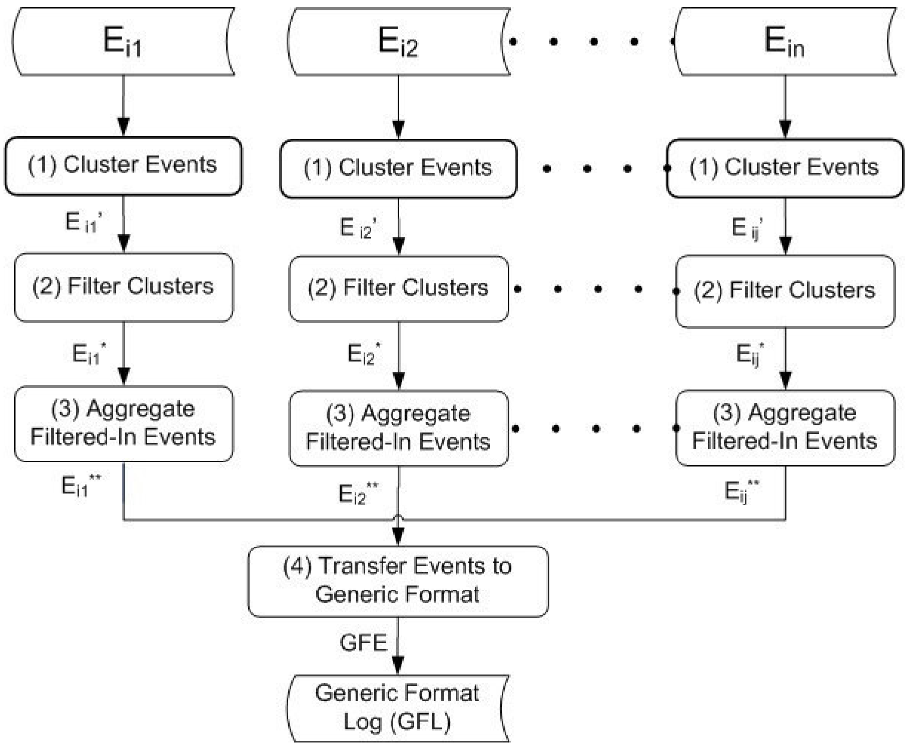


FIG. 5.2. *Event Reduction and Integration in UHAD*

best number of clusters (K_{ij}) using EM. As the intention of this step is to predict the number of clusters, the default seed value 100 was used with the number of clusters (K) set to -1 to enable EM to predict the number of clusters. The clusters (K_{ij}) thus predicted is used to cluster the events (E_{ij}) in the next step.

Generating Clusters. The details of the clustering algorithms used in the second step of the clustering strategy implemented in UHAD is as follows:

Expectation Maximization. The EM algorithm [48] comprises of two recursive steps, Expectation and Maximization, which uses a statistical model called Gaussian finite mixtures to accomplish the objective of producing the most likely set of clusters for a given dataset, given the number of clusters (K). The model includes a set of K probability distributions to provide data representation for each cluster. Each K distribution is defined by parameters like number of iterations and the difference in log likelihood between successive iterations. Initially these parameters are deduced by the algorithm based on the input data, which is subsequently determined by the probability that a particular instance belongs to specific cluster for the given data by utilizing these parameter deduced. Parameter distribution is amended again and this continues until the generated clusters have a certain level of overall cluster goodness or until the maximum number of iterations is reached.

K-Means. K-means [45] is a simple and popular clustering method that divides instances based on the attribute values into K disjoint clusters. Instances that shape the cluster have similar attribute values and K specifies the number of cluster to be generated. The steps of K-means algorithm are as follows:
1. Define the number of clusters K.
2. Initialize the K cluster centroids by randomly dividing all instances into K clusters, calculating their centroids, and verifying that all centroids differs from the other.
3. Iterate on all instances and calculate the distances of centroids for all clusters. Assign each object to the cluster with the nearest centroid.
4. Recalculate the centroids of both modified clusters.
5. Repeat step 3 until the centroids change.

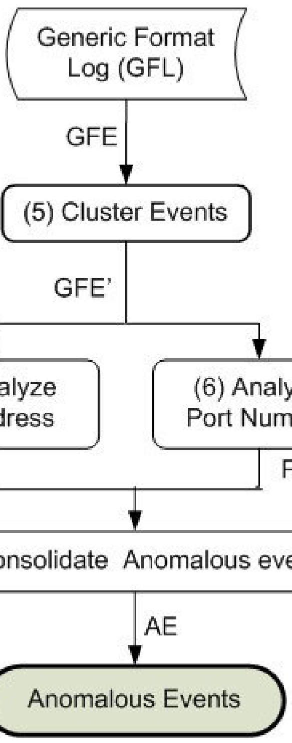


Fig. 5.3. *Anomaly Detection in UHAD*

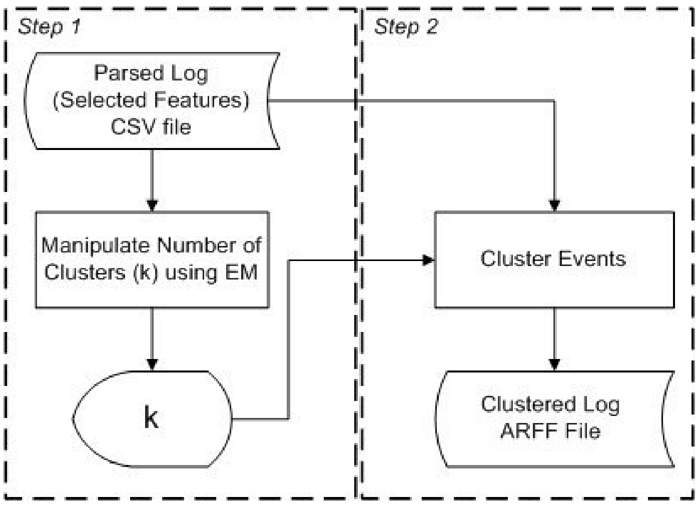
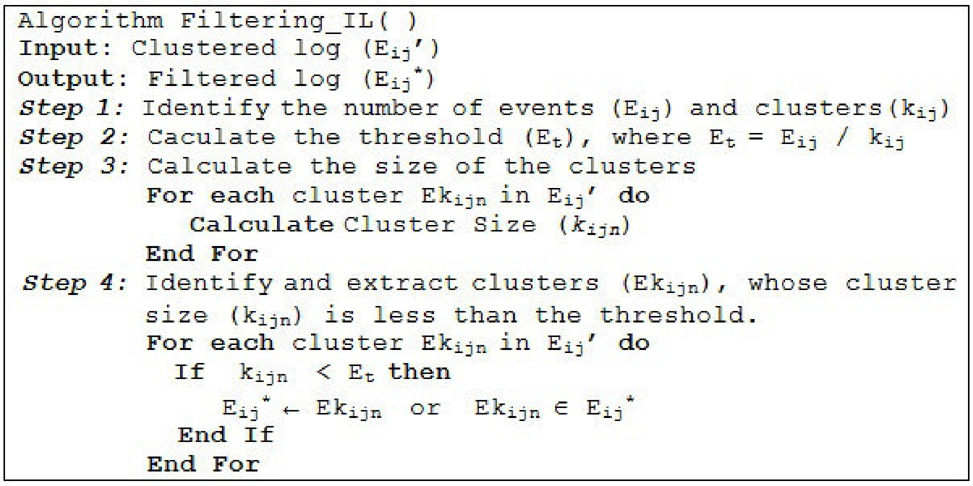

Step 1
Step 2
Parsed Log (Selected Features) CSV file
Manipulate Number of Clusters (k) using EM
k
Cluster Events
Clustered Log ARFF File

Fig. 5.4. *Two-Step Clustering Strategy*

Moreover, a distance function is needed to calculate the distance (i.e. similarity) between two instances and the most commonly used is the Euclidean distance where every attribute contributes evenly to the calculation of this value. The algorithm has the skill to treat the features in the events equally and segregates precisely in likely clusters. Additionally it has the capacity to handle larger datasets with a lesser processing time justifies the importance of using it in the second step of the clustering strategy.

Farthest First. Farthest first [46] is an alternative of K-Means that places every cluster center in turn at the point farthest from the existing cluster center and this point lies inside the data area. Since there is less reassignment, the process of clustering is faster. The logs (L_{ij}) captured from Apache server, Linux and Snort IDS were clustered according to the specification stated in Experiment 2. Therefore every log was clustered 12 times, and the clusters thus generated were maintained separately for further processing and evaluation. The clustered events were stored in its native format, i.e. Attribute Relation File Format (ARFF) for further processing and not the trained model.

5.1.2. Filtering Clustered Events. The filtering algorithm (Figure 5.5) was implemented using the script written in Perl which receives a clustered file in ARFF format as input and produces the filtered-in events in CSV format. The ARFF file was parsed to eliminate the header generated during clustering and the relevant



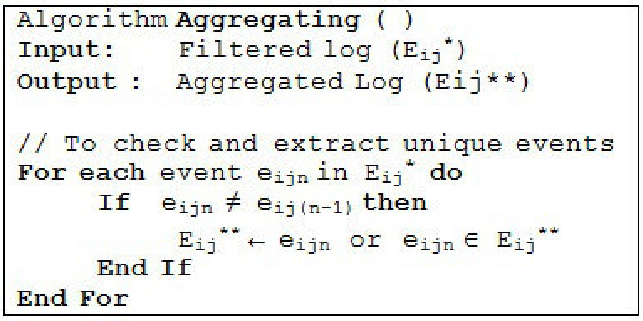
```
Algorithm Filtering_IL( )
Input: Clustered log ($E_{ij}'$)
Output: Filtered log ($E_{ij}^*$)
Step 1: Identify the number of events ($E_{ij}$) and clusters($k_{ij}$)
Step 2: Caculate the threshold ($E_t$), where $E_t = E_{ij} / k_{ij}$
Step 3: Calculate the size of the clusters
        For each cluster $Ek_{ijn}$ in $E_{ij}'$ do
           Calculate Cluster Size ($k_{ijn}$)
        End For
Step 4: Identify and extract clusters ($Ek_{ijn}$), whose cluster
        size ($k_{ijn}$) is less than the threshold.
        For each cluster $Ek_{ijn}$ in $E_{ij}'$ do
          If  $k_{ijn} < E_t$ then
             $E_{ij}^* \leftarrow Ek_{ijn}$  or  $Ek_{ijn} \in E_{ij}^*$
          End If
        End For
```

FIG. 5.5. *Filtering Strategy for Individual Logs*

features of every log together with the cluster number were extracted to a CSV file before proceeding with filtering. In some cases, the number of clusters generated was lesser than the number of clusters requested and therefore the parsed clusters were scanned to identify the number of clusters. As such, the threshold was calculated based on the identified number of clusters and the total events clustered. The volume of events and the identified number of clusters varies from log to log and therefore the threshold calculated also varies. The volume of events in every cluster was found and those clusters whose size was less than the calculated threshold (E_t) was filtered-in for further scrutiny as mentioned in the algorithm (Figure 5.5). The cluster an event belongs to was also included in the filtered log to verify the patterns of events in each cluster.

5.1.3. Aggregating Filtered-In Events. Filtered-in log was received as input and the redundant events were combined by checking every event to produce an aggregated log and the algorithm is illustrated in Figure 5.6. Even though the log was previously clustered and filtered, the order in which the log event appears in the original log was maintained. Therefore, every event in the log was compared with the event that is immediately following it. As an analogy, if the current event being scrutinized is equivalent to the previous event automatically the current event is dropped and the next event becomes the current event whereas if the current event is different from the previous event, then the current event is retained and it becomes the previous event.

5.1.4. Transferring Events to GFL. The process of extracting the events from various logs and transferring the events to GFL is automated using custom written script. The events in Apache server, Linux syslog and Snort IDS logs were transferred to GFL excluding Apache SSLError and Linux mail log, since many of the GFL features were not available in these logs; but they are maintained separately to be used at some stage in the analysis. As IPTables firewall log was also used only during analysis, these events were stored separately



```
Algorithm Aggregating ( )
Input:    Filtered log ($E_{ij}^*$)
Output :  Aggregated Log (Eij**)

// To check and extract unique events
For each event $e_{ijn}$ in $E_{ij}^*$ do
     If  $e_{ijn} \neq e_{ij(n-1)}$ then
          $E_{ij}^{**} \leftarrow e_{ijn}$ or $e_{ijn} \in E_{ij}^{**}$
     End If
End For
```

FIG. 5.6. *Aggregating Strategy for Individual Logs*

as per the features specified in GF. The transfer of events from various log sources is depicted in Figure 5.7. In some logs, source IP address was not recorded as separate feature, but then as a part of another feature

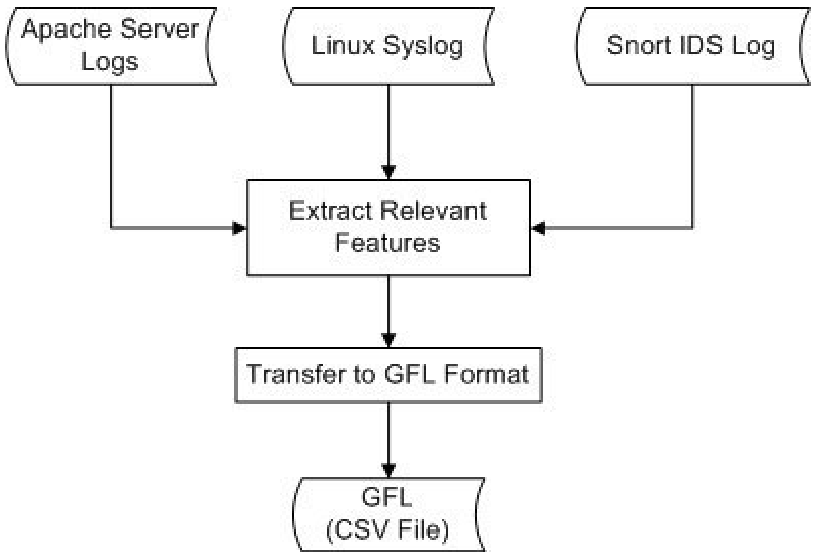


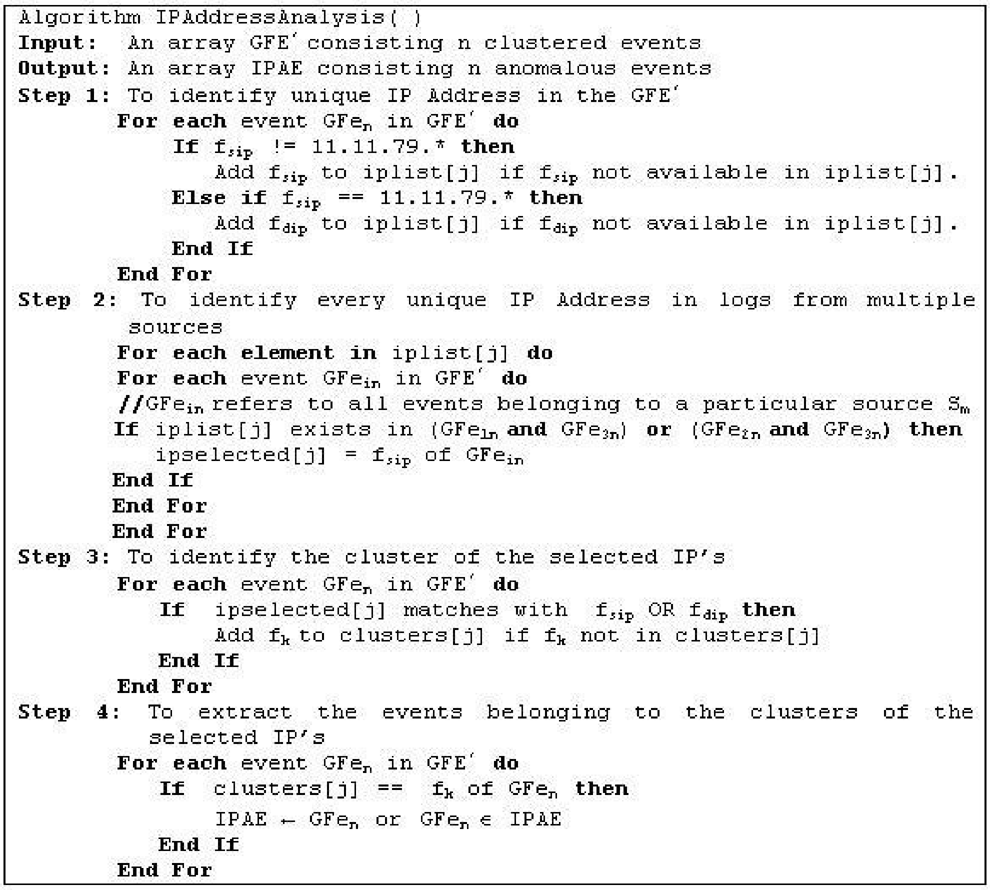
FIG. 5.7. *Transferring Events to GFL*

and therefore it was extracted and represented as a separate feature. Since we compare the performance of three algorithms with four settings, the GFL events were maintained separately according to the algorithm and parameter settings.

5.1.5. Clustering GFE. The GFL events which was maintained according to clustering algorithms and their respective settings were considered separately for this step. In the first step a given set of GFL events (GFE) is loaded into Weka to manipulate the ideal number of clusters (K_g) using EM. The default seed value 100 was used with the number of clusters (K) set to -1 to enable EM to predict the number of clusters. The clusters (K_g) thus predicted is used to cluster GFE. The respective GFL events were clustered using the same algorithm and parameter settings. The clustered events were stored in ARFF format for further processing.

5.1.6. Analyse Features to Identify Anomalous Events. The analysis process intends to discover the relationship between events pertaining to the various features represented by the events to detect anomalous events. We performed analysis using the features IP address (f_{sip}; f_{dip}), port number (f_{sp}; f_{dp}) and cluster number (f_k).

IP Address Analysis. The analysis procedure (Figure 5.8) receives clustered log events (GFE) as input which was scanned several times, before an event was decided as anomalous. During the analysis process three features were manipulated which are source IP address (f_{sip}), destination IP address (f_{dip}) and cluster number (f_k). Although checking the IP address of the events was focused, the cluster number of these events was also



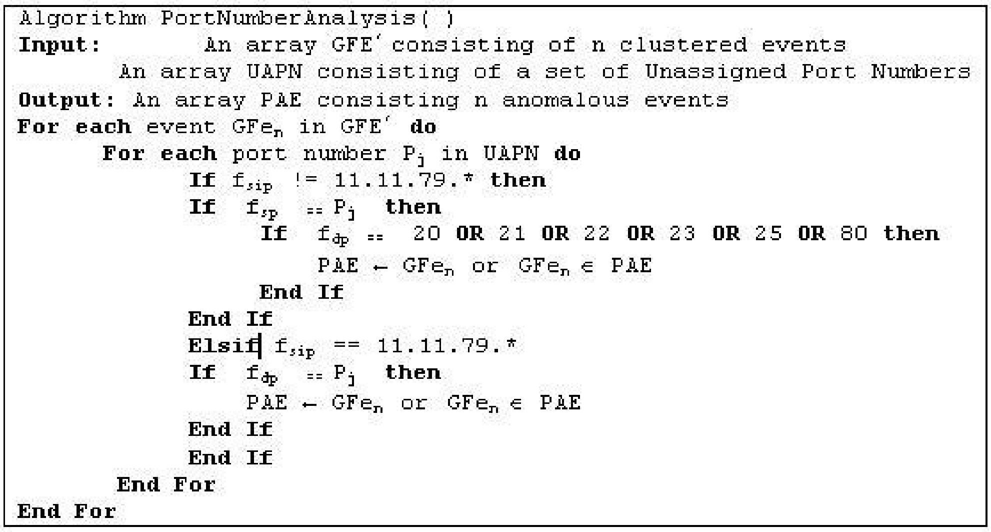
```
Algorithm IPAddressAnalysis( )
Input:  An array GFE′ consisting n clustered events
Output: An array IPAE consisting n anomalous events
Step 1: To identify unique IP Address in the GFE′
        For each event GFe_n in GFE′ do
            If f_sip != 11.11.79.* then
                Add f_sip to iplist[j] if f_sip not available in iplist[j].
            Else if f_sip == 11.11.79.* then
                Add f_dip to iplist[j] if f_dip not available in iplist[j].
            End If
        End For
Step 2: To identify every unique IP Address in logs from multiple
         sources
        For each element in iplist[j] do
        For each event GFe_in in GFE′ do
        //GFe_in refers to all events belonging to a particular source S_m
        If iplist[j] exists in (GFe_1n and GFe_3n) or (GFe_2n and GFe_3n) then
           ipselected[j] = f_sip of GFe_in
        End If
        End For
        End For
Step 3: To identify the cluster of the selected IP's
        For each event GFe_n in GFE′ do
            If  ipselected[j] matches with  f_sip OR f_dip then
                Add f_k to clusters[j] if f_k not in clusters[j]
            End If
        End For
Step 4: To extract the events belonging to the clusters of the
         selected IP's
        For each event GFe_n in GFE′ do
            If  clusters[j] ==  f_k of GFe_n then
                IPAE ← GFe_n or GFe_n ∈ IPAE
            End If
        End For
```

Fig. 5.8. *IP Address Analysis*

manipulated to get a big picture of the events related to intrusions. The first pass was to identify the unique IP address in log events, where f_{sip} was checked in case of inbound connections and f_{dip} for outbound connections. Those identified unique IP addresses existing in Apache and Linux logs were checked for its existence in Snort IDS log, and vice versa. Since there were no labels specified in GFL to identify the log which an event belongs to, the events were identified by the non-availability of feature values in that event, i.e., Apache and Linux log events do not have values for the features destination IP address (f_{dip}), source port (f_{sp}), destination port (f_{dp}) and protocol (f_{pr}).

The events containing IP addresses that were selected for its existence in multiple log sources, were checked to identify the cluster which they belong to. All the events belonging to the identified clusters were extracted and deemed as anomalous which serves as the result of the analysis. The results were evaluated based on the volume of anomalous events identified by the analysis and also the number of IP addresses covered by these events from where the intrusion have originated. To verify the validity of the events detected, it was compared with the results of SOTM#34 challenge, to substantiate the coverage of anomalies by UHAD.

Port Number Analysis (PA). Ports possess significant discriminative control in classifying certain types of traffic when used with other features [51] and most anomalous activities utilises the unused and unassigned ports. The same set of GFL events (GFE) used for IP address analysis were used for port analysis, too. Since the analysis (Figure 5.9) was based on port numbers, those events without it will be automatically excluded from analysis. In the case of inbound connections the source port number of the events was checked against



```
Algorithm PortNumberAnalysis( )
Input:       An array GFE′ consisting of n clustered events
        An array UAPN consisting of a set of Unassigned Port Numbers
Output: An array PAE consisting n anomalous events
For each event $GFe_n$ in GFE′ do
      For each port number $P_j$ in UAPN do
            If $f_{sip}$ != 11.11.79.* then
            If $f_{sp}$ == $P_j$ then
                If $f_{dp}$ == 20 OR 21 OR 22 OR 23 OR 25 OR 80 then
                    PAE ← $GFe_n$ or $GFe_n \in$ PAE
                End If
            End If
            Elsif $f_{sip}$ == 11.11.79.*
            If $f_{dp}$ == $P_j$ then
                PAE ← $GFe_n$ or $GFe_n \in$ PAE
            End If
            End If
      End For
End For
```

Fig. 5.9. *Port Number Analysis*

the listing of the dynamic and unassigned port numbers as per Internet Assigned Numbers Authority (IANA). In this manner, the destination ports of the matching events were checked for the availability of well-known port numbers, similar to 20, 21, 22, 23, 25 and 80 to generate anomalous event list, as intrusive events mostly succeed through these ports on the victim host. Then again for outbound connections, the destination port number of the events was checked against the listing of the dynamic and unassigned port numbers according to IANA. All the matching events were regarded as anomalous and was moved to a distinct CSV file.

5.1.7. Consolidate Anomalous Events. Three operations were performed, namely uniting, associating and concentrating to consolidate the anomalous events detected by the analysis methods, i.e. IPA and PA, and the process is illustrated in Figure 5.10. Initially all the anomalous events detected by IPA was copied to a CSV file referred as an intermediate anomalous events list (iAE) and every single event in the list is referred as iAe. Every anomalous event detected by PA was compared with every single anomalous event of IPA and non-matching anomalous events of PA will be appended to iAE. Unique IP addresses excluding internal IP addresses, i.e., 11.11.*.*, represented by the iAE events were identified and every such IP address was compared with IP address of IPTables firewall log events maintained in a separate log as per GF. All the IPTables firewall log events containing IP address matching with the identified IP addresses were appended to iAE. Now iAE contains the detected anomalous events (AE) captured from various sources that serves as the result of the framework. In order to concentrate on significant anomalous events, a threshold (A_t), i.e., 3, 6, 9, was used to consolidate the anomalous events based on the IP address. Since the unique IP addresses were already found before correlation, the occurrence of the iAE events containing these IP addresses were counted. Those IP addresses satisfying the threshold (A_t) were identified. The iAE events were checked for these IP addresses, and those events satisfies the threshold were extracted and maintained separately. This to verify that, whether applying the threshold at this point supported to focus on the most significant anomalous events or it reduced such events. All the new algorithms proposed to implement various components of the framework were written in Perl.

The limitation of the filtering component deployed in UHAD [10] is further improved with the refined filterer [52] by increasing the volume of retained abnormal events; hence, the other components of the framework [52] in are basically the same as UHAD [10]. The aim of the refined filterer is to retain all the abnormal events in the log for subsequent processing, irrespective of the existence of such events in larger number in the logs and the inaccuracies in clustering. The refined filterer receives a clustered log (E_{ij}) which is initially scanned to

```
Algorithm Consolidate_AnomalousEvents( )
Input: An array IPAE consisting of anomalous events of IP analysis
       An array PAE consisting of anomalous events of port analysis
       An array IPTGFE consisting of n events
Output: An array AE consisting n anomalous events
Step 1: Transfer all the IPAE  to iAE //iAE: Intermediate Anomalous
         Events
Step 2: Transfer events in PAE to iAE that is not available in IPAE
        For each $PAe_y$ in PAE do
          For each event $IPAe_x$ in IPAE do
           If $PAe_y$  != $IPAe_x$ then
              iAE ←  $PAe_y$ // iAE = Intermediate Anomalous Events
           End If
          End For
        End For
Step 3: To identify unique IP Address in iAE
         For each event $iAe_x$ in iAE do
           If $f_{sip}$ != 11.11.79.* then
              Add $f_{sip}$ to iplist[j] if $f_{sip}$ not available in iplist[j].
           Else if $f_{sip}$ == 11.11.79.* then
              Add $f_{dip}$ to iplist[j] if $f_{dip}$ not available in iplist[j].
           End If
         End For
Step 4: To extract the events form IP Tables whose IP matches with the
        iplist[]
         For each event $IPTGFe_n$ in IPTGFE do
          For each element in iplist[j] do
              If ($f_{sip}$ == iplist[j] OR $f_{dip}$ == iplist[j]) then
                   iAE ←  $IPTGFe_n$
             End If
          End For
         End For
Step 5: Count the appearance of each IP Address in the events iAE
        For each event $iAe_x$ in iAE do
         For each element in iplist[j] do
            If iplist[j] == $f_{sip}$ OR iplist[j] == $f_{dip}$ then
             ipcount[j]++;
            End If
         End For
        End For
Step 6: Select the IP's that satisfies a specific threshold ($A_t$), i.e.,
        3, 6, 9
        For each element in iplist[j] do
         If ipcount[j] >= $A_t$ then
             selectedip[k] = iplist[j];
         End If
        End For
Step 7: Extract the events of this selected IP's and deem as anomalous.
        For each event $iAe_x$ in iAE do
          If selectedip[k] == $f_{sip}$ OR selectedip[k] == $f_{dip}$ then
               AE ←  $iAe_x$
          End If
        End For
```

FIG. 5.10. *Comparing Anomalous Events*

identify the volume of events (nE_{ij}) and the number of clusters (K_{ij}) to calculate the filtering threshold (E_t), i.e., $E_t = \mathrm{n}E_{ij}/\ K_{ij}$.

The filtering threshold calculated as such is equivalent to the average cluster size. The calculated filtering threshold was used to identify the sparse and dense clusters in the clustered log, and the events of sparse ($E_{ij}{*}$) and dense clusters ($E_{ij}^{\#}$) were identified, extracted and maintained separately. But then, there are chances that some of the abnormal events may have been mixed up with normal events in dense clusters due to the inaccuracy in clustering or the similarities between the feature values of abnormal and normal events. Therefore, such abnormal events in dense clusters must be identified and included in filtered-in events for further processing.

In order to achieve that, subsequently, every event in the dense clusters was compared against every event in the sparse clusters, i.e., the respective features in these events were separately or individually compared. The cluster number in the log that was included during the process of clustering, which also appears in the filtered events is excluded from the counted features for a particular log. Those events in the dense clusters which match with any one of the events in sparse clusters (abnormal events) are considered abnormal, and the matching events ($E^{\#}_{ij}$) i.e., those events having a match of at least 50% of features in between the events, were added to the set of filtered-in events after all such events were identified. For instance, if there are six features in the log events compared, then at least three features of an event in the dense clusters must exactly match with any one of the events in the sparse clusters in order to be filtered-in for further processing. As there may be subtle difference in the patterns of abnormal events grouped in sparse and dense clusters, comparing both categories by features for exact matches is not reasonable and will not assist in retaining majority of the abnormal events. Therefore, 50% is chosen as the minimum matching between events in order to consider for further processing. This is based on the assumption that the sparse clusters contain only abnormal events and the misclassified abnormal events in dense clusters should be partially or completely matching with any of the events in sparse clusters. Therefore, the dense cluster events were checked against sparse cluster events, and the matching events of dense clusters were also filtered-in together with sparse clusters for further examination.

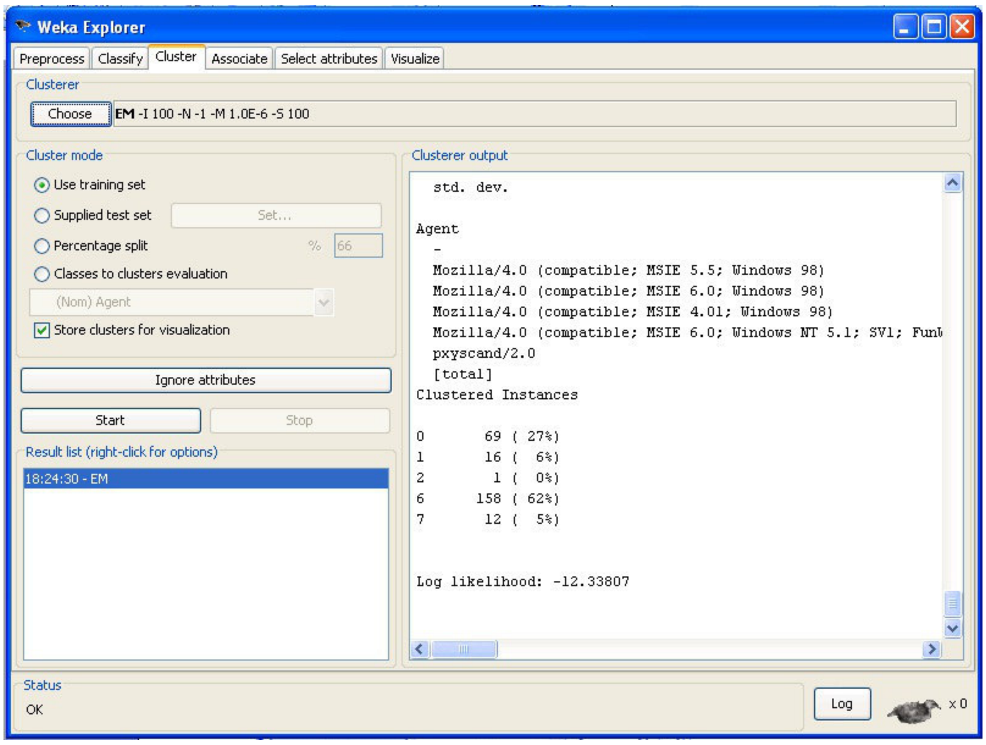
Moreover, the refined filterer strives to retain every abnormal event in the log irrespective of the inaccuracy in some of the clusters generated. The refined filterer is very much process intensive as every event in the dense clusters has to be compared with every event in sparse clusters and moreover every feature of the event is individually compared with another event. Additionally, the logs clustered with the reordered events were also filtered using the refined filterer to verify whether the filterer managed to retain the abnormal events as it does with the clustered events of original log. This is to evaluate the effect of reordering and the subsequent clustering on the refined filterer especially on the retention of abnormal events.

6. Results and Discussions. To assess the capability of the framework in discovering anomalies, the log events captured by multiple sources in a Honeynet system were used. A total of four subsets with varying duration were tested. First and foremost, the idea behind the development of this framework is to detect a wide range of anomalous events by analysing various logs without using any kind of knowledge on anomalies or the models of network traffic behaviour. All the components implemented in this framework work towards the aforesaid objective. Apart from this, every component has a specific objective towards the main objective, and hence the performance of every component was measured with different metrics as stated in Chapter 3. The performance of the framework as a whole in detecting various anomalous events were evaluated by comparing its results with the SOTM/#34 challenge results [12, 13, 14, 15] provided at Honeynet.org.

6.1. Clustered Events. This section describes the results of the two-step clustering strategy implemented in the framework. Firstly, the predicted clusters for various logs were discussed. Secondly, the accuracy achieved with various clustering algorithms with different parametric settings were compared. Additionally, the relation of clustering accuracy with the actual formation of clusters with respect to abnormal events were examined.

6.1.1. Predicted Clusters. The clusters for each log were predicted according to Experiment 1 specification. EM clustering was applied with the default values for the parameters (i.e. K = -1, seed = 100) on the selected features of the individual logs (E_{ij}) to estimate the best number of clusters (K_{ij}). For example, while predicting the number of clusters for Apache Access log, EM at the start selects eight clusters by cross validation, but concludes generating only five clusters (0, 1, 2, 6, 7) as shown in Figure 6.1.

Therefore, the best number of clusters selected for this log was five, and the similar approach was followed for all the logs in manipulating the ideal number of clusters. The cluster predicted by EM for various logs is provided in Table 6.1. This reveals the fact that the number of clusters predicted were not influenced by the number of features and events in a log, but by the patterns of the events, i.e, the more the number of distinct patterns, the higher the number of clusters predicted. Apache error log consisting of 433 events with five features resulted in seven clusters whereas Snort IDS log consisting of 4423 events with 11 features resulted in only three clusters. This also revealed the fact that there is no relation between the number of events and



Clusterer: EM -I 100 -N -1 -M 1.0E-6 -S 100

Percentage split % 66

Result list: 18:24:30 - EM

Status: OK

```
  std. dev.

Agent
  -
  Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
  Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)
  Mozilla/4.0 (compatible; MSIE 4.01; Windows 98)
  Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; FunW
  pxyscand/2.0
  [total]
Clustered Instances

0       69 ( 27%)
1       16 (  6%)
2        1 (  0%)
6      158 ( 62%)
7       12 (  5%)


Log likelihood: -12.33807
```

Fig. 6.1. *Manipulating Clusters using EM*

Table 6.1
Ideal Clusters by EM

Log Type	**Suset-1**	**Subset-2**	**Subset-3**	**Subset-4**
Access	5	5	8	6
Error	7	6	7	7
SSL-Error	2	4	4	3
Message	5	5	5	7
Mail	2	2	4	4
Security	4	4	4	6
Snort IDS	3	8	6	7

its features with the clusters predicted.

The time taken to predict the ideal number of clusters by EM for each log varied from seconds to hours depending on the volume of events and its features. The time taken to manipulate the events to predict clusters for Linux Message log with 112 events took five seconds whereas Snort IDS log with 17049 events took 1 hour 30 minutes and 12 seconds. Moreover, prediction time also depends on the existence of varying patterns in the log, eventually resulting in more number of clusters, e.g., Linux message log with 105 events took 10 seconds to predict seven clusters, whereas 112 events belonging to the same type of log took five seconds to predict five clusters. The manual effort and time spent in finding the ideal clusters for a particular log using our strategy is far lesser than that of the usual method, i.e., applying the clustering algorithm on a dataset several times with different clusters (K) and choosing the best clusters among them. Thus the clustering strategy implemented in our framework facilitates the process in finding the appropriate clusters for the logs with less time consumption and manual effort.

6.1.2. Clustering Accuracy and Cluster Analysis. Using the clusters (K_{ij}) manipulated from step 1 (Experiment 1), the logs were clustered according to the algorithms and settings stated in Experiment 2, and the clustered events (E_{ij}) were recorded in ARFF format. Every clustered log was maintained separately according to subsets, algorithms and settings. The accuracy of the clusters generated for various logs by the algorithms with different settings were evaluated. The maximum time taken to cluster 17049 events of Snort IDS with seven clusters, i.e., the log containing the highest number of events in subset-2, was 2 minutes and 5 seconds by EM whereas K-Means and FF took only 17 seconds and 2 seconds, respectively. The accuracy of K-Means, EM and FF in clustering various logs with different parameter settings was evaluated, i.e., comparing the accuracy of the default setting (setting-1) with other settings (settings 2, 3 and 4). This is to recognize the impact of seed and K on accuracy, when increased. In addition, the relation of accuracy and the parametric settings with the actual grouping of abnormal events in separate clusters were also examined. As we have tested four subsets containing various logs, the results are discussed subset by subset. Previously the results of subset-1 was presented in Hajamydeen et.al. [7], and therefore the results of the other three subsets, i.e., subset-2, subset-3, subset-4, were only discussed.

Subset-2. The accuracy achieved with various settings for this subset exhibits a similar pattern like subset-1 and is illustrated in Figure 6.2. Increasing the seed value alone (setting-2) improved the accuracy in all logs

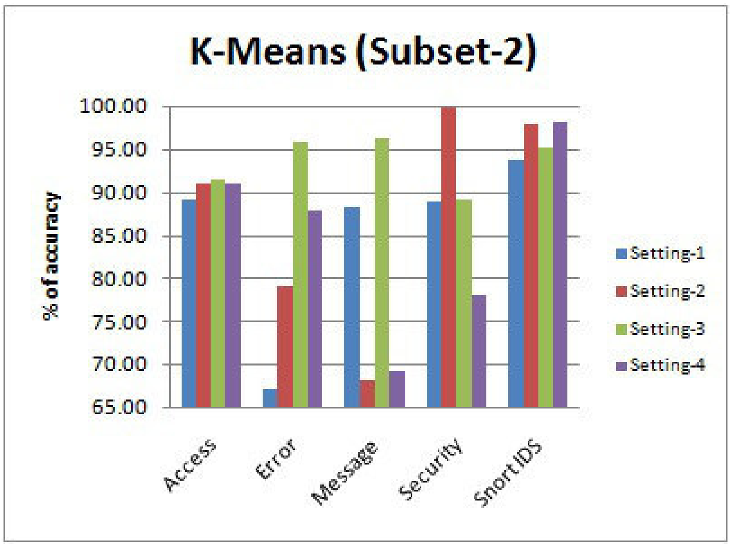


Fig. 6.2. *K-Means Clustering Accuracy (Subset-2)*

except the Linux Message log. This was because of the similarity between the feature values in this log. Doubling k with the default seed (setting-3) increased the accuracy with all the logs. But, increasing both seed and k together (setting-4) decreased the accuracy with all logs except Snort IDS logs, thus signifying the sensitive nature of K-Means to initialisation parameters. This also shows the need for customised seed value than using the default seed especially with those logs having events of bigger volume. The increase in accuracy with settings 2 and 4 for this log also shows the importance of a customised seed value especially with bigger sized logs. There were 14 successful intrusive events and 58 unsuccessful intrusive events in the Apache error log.

Analysing the clusters generated with various settings showed that setting-4 formed better clusters by grouping the intrusive events in separate clusters. There were 12 clusters generated from 0 to 11 consisting of events 120, 54, 75, 14, 1, 60, 67, 13, 1, 24, 33 and 142, respectively. All the 14 successful intrusive events were grouped in Cluster 3 and the 58 unsuccessful intrusive events with another nine abnormal events in Cluster 6. This was due to the impact of increased seed and k on clustering even though the accuracy with various settings did not reflect that. The accuracy of EM with subset-2 as illustrated in Figure 6.3 expressed a reverse accuracy pattern with respect to settings compared to subset-1. The accuracy of clustering was improved when the ideal clusters (k) were doubled (setting 3) with all the logs, and declined when the seed was increased with settings

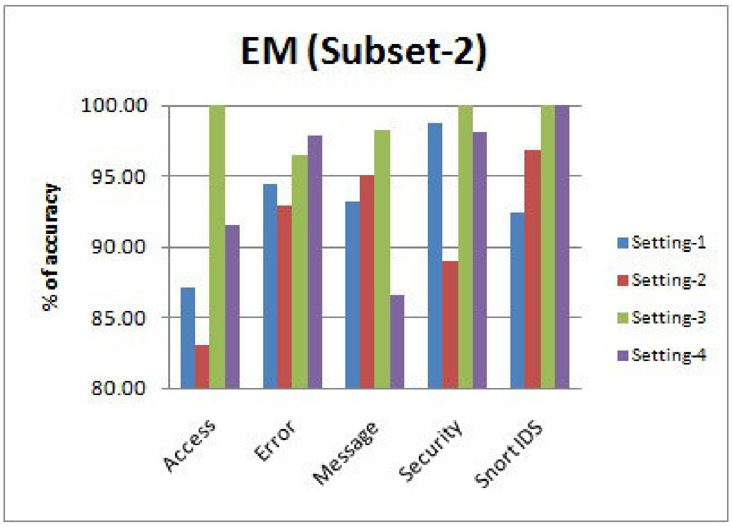


Fig. 6.3. *EM Clustering Accuracy (Subset-2)*

2 and 4 with most of the logs. This shows that the accuracy factor is not only influenced by k and seed, but also by the volume of events and their patterns. A better cluster formation was achieved by EM with setting-4 for the Apache error log. Out of the 12 clusters (107, 51,39, 54, 24, 0, 47, 60, 26, 54, 142 and 0) generated for this log, two of them were empty clusters, i.e., Clusters 5 and 11. The 58 unsuccessful intrusive events were grouped together in Cluster 7 with another two abnormal events. The 14 successful events were joined together with another 93 abnormal events. In terms of cluster formation, K-Means formed better clusters compared to EM, even though EM achieved the highest accuracy for this log.

The clustering accuracy of FF with subset-2 (Figure 6.4) expressed a similar pattern like subset-1 with settings 3 and 4, however, the pattern was opposite with setting-2 which resulted in a sharp decline in accuracy with Apache error, Linux message and Snort IDS log. Although FF is not very sensitive to seed values, the

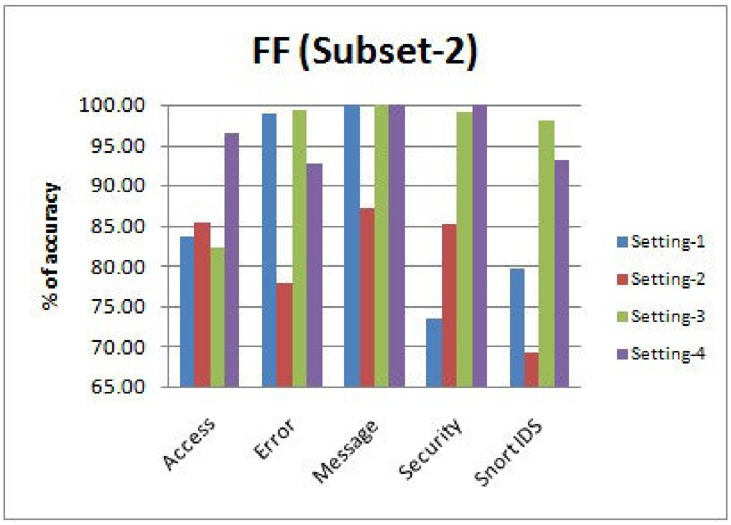


Fig. 6.4. *FF Clustering Accuracy (Subset-2)*

sharp increase and decrease in accuracy with settings 2 and 4 was due to the diverse event patterns in the logs. Moreover, the cluster formation for the logs was similar between settings 1 and 2 and also with settings 3 and 4.

The unsuccessful intrusive events of Apache error log were grouped together in Cluster 4 and successful ones in cluster 6 together with other abnormal events. Doubling k with settings 3 and 4 resulted in better formation of similar clusters. The clusters generated by FF was better than EM but inferior to K-Means. This subset contained more events in all the logs compared to subset-1 and KMeans formed better clusters for this subset. This shows the capacity of K-Means in handling voluminous log events.

Subset-3. The accuracy of K-Means with subset-3 is illustrated in Figure 6.5. A similar accuracy pattern like subsets 1 and 2 was achieved with settings 3 and 4 for this subset. But with setting-2, the increase in seed value decreased the accuracy. This shows the sensitive nature of K-Means to initial parameters. Although, the

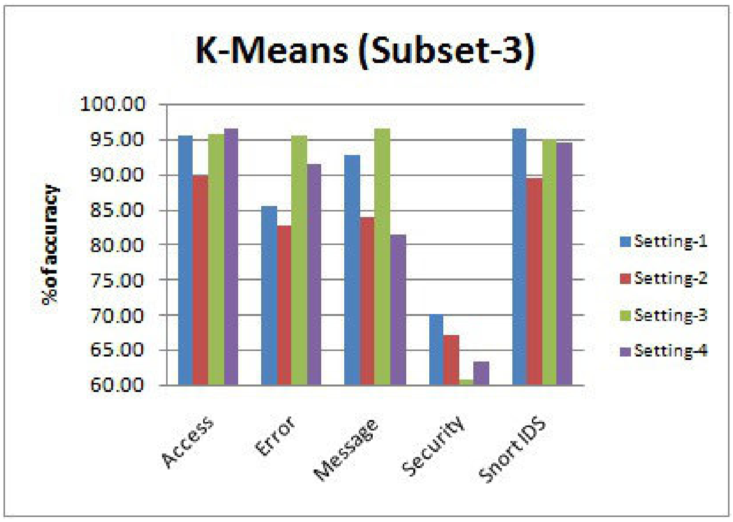


Fig. 6.5. *K-Means Clustering Accuracy (Subset-3)*

volume of events in Apache access and error log was high compared to subsets 1 and 2, K-Means achieved a better accuracy for this log than the other subsets. This exposes the capacity of K-Means in handling larger datasets. Linux security log consists of 234 events where 12 events are related to xinetd crash and the pattern of these events were very different from others.

Even though there were eight features recorded by this log, the majority of the events do not have values for all these features resulted in a lower accuracy, i.e., below 70%. This was also reflected appropriately in cluster formation. All these intrusive events were joined together in Cluster 0, i.e., the biggest cluster, for the settings 1, 2 and 4. For setting-3, these events was separated in two clusters, i.e, Cluster 0 and 4. All the 10 events in Cluster 4 was related to xinetd crash, and another two events were in Cluster 0, i.e, the biggest cluster. Although, the accuracy achieved for this setting is the lowest, the cluster formation was better. This also shows that cluster formation was not directly reflected in accuracy.

In the case of Snort IDS logs consisting of 5013 events, all the events pertaining to the intrusion were grouped in smaller clusters for all the settings. Especially, K-Means with setting-1 grouped majority of the intrusive events in the smallest cluster, i.e., Cluster 5, and in several smaller clusters for setting-3. This shows that the default seed performed better in cluster formation for this log and is also reflected in accuracy. The pattern of accuracy achieved by EM with subset-3 (Figure 6.6) is similar to subset-1 for settings 2 and 3 whereas it was similar to subset-2 for setting-4. The default setting (setting-1) achieved the highest accuracy for Apache access and Linux message log. The increase in seed and k (setting-4) improved the accuracy in Snort IDS log, exposing the contribution of seed and k together with bigger sized logs. In spite of the absence of values for the features in most of the events of Linux security log, EM formed better clusters compared to K-Means, showing its capacity in handling events of this nature.

All the 12 events related to xinetd crash were grouped together in a smaller cluster for settings 1, 2 and 3. For setting-4, 10 of these events were grouped in a separate cluster and another two events in a smaller cluster.

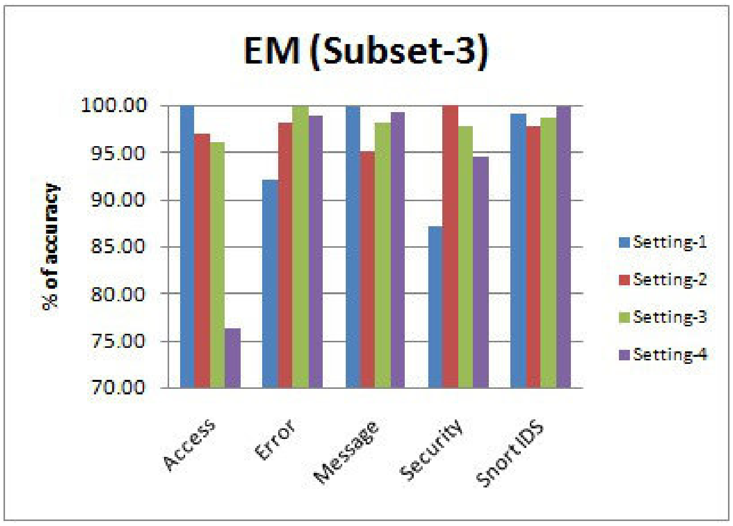


Fig. 6.6. *EM Clustering Accuracy (Subset-3)*

All the intrusive events captured by Snort IDS log were grouped in smaller cluster with setting-4. This shows the involvement of seed and K together in cluster formation. With subset-3, FF achieved accuracy (Figure 6.7) of identical pattern like subset-1. An average accuracy above 90% was achieved by FF for various settings with all the logs, except the Snort IDS log for the default setting. The accuracy achieved by FF with various settings were not appropriately reflected in cluster formation. Intrusive events related to xinetd crash in Linux security

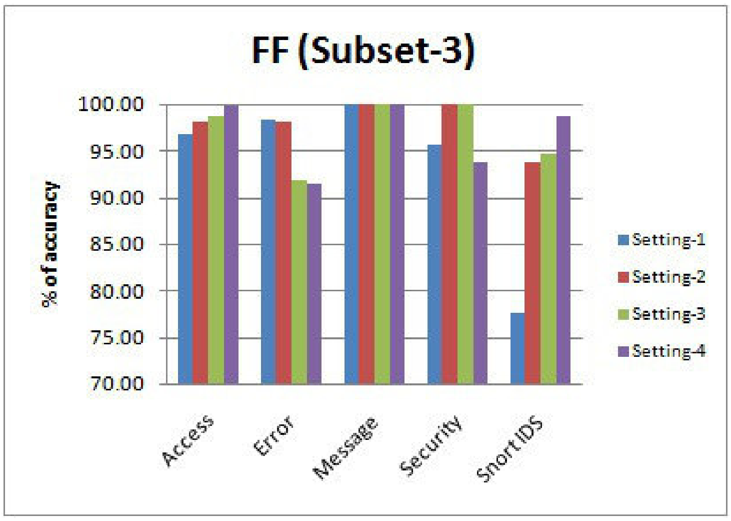


Fig. 6.7. *FF Clustering Accuracy (Subset-3)*

log that has a unique pattern than the other events were grouped in the biggest cluster for all the settings. Even in Snort IDS log, the events related to various intrusions were grouped in bigger clusters.

Subset-4. The accuracy achieved by K-Means for this subset plotted in Figure 6.8 exposed a similar pattern with subset-3. Even though doubling K (setting-3) increased accuracy, increasing the seed together with doubled K did not improve the accuracy. This is due to sensitive nature of K-Means to initialisation parameters. There were 434 events in Apache access log, out of which 78 events were the outcomes of unsuccessful intrusions originated from two IP addresses and the patterns of events from these IP addresses were different from each

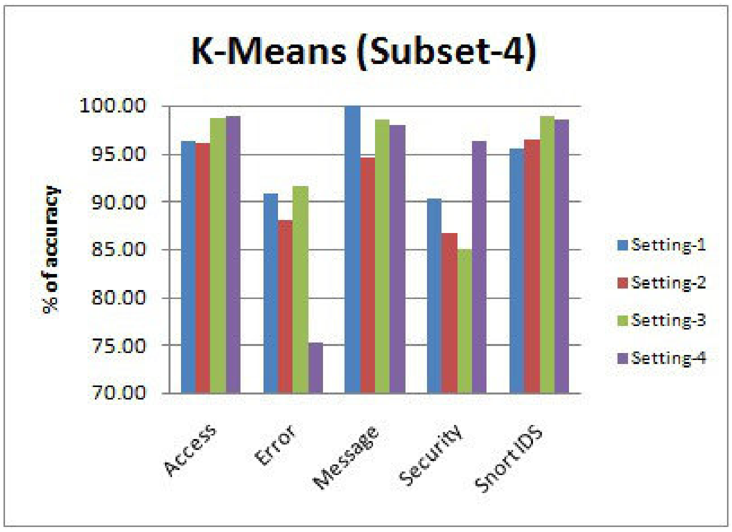


Fig. 6.8. *K-Means Clustering Accuracy (Subset-4)*

other. Settings 3 and 4 generated better clusters by separating intrusive events in two clusters according to IP address. One of these clusters contained only intrusive events and the other cluster contained intrusive events together with other events as well. In case of Snort IDS log, majority of the events related to intrusions were grouped in smaller clusters and a small number of such events gathered in big clusters together with other events for all the settings.

Better clusters were generated with settings 1 and 2 for this log, although settings 3 and 4 showed better accuracy. EM achieved the highest average accuracy of 96.89% for this subset, but the pattern of accuracy with various settings were very different from all the other three subsets and exactly opposite to subset-3. The accuracy achieved by EM for various logs is depicted in Figure 6.9. Better cluster were formed by EM with setting-4 for Apache access log, but the cluster formation was inferior to the ones generated by K-Means. In case of Snort IDS log, all the events related to intrusions were grouped in smaller clusters for the same setting. Like EM, FF also achieved the highest accuracy (Figure 6.10) with this subset, but then the pattern of accuracy

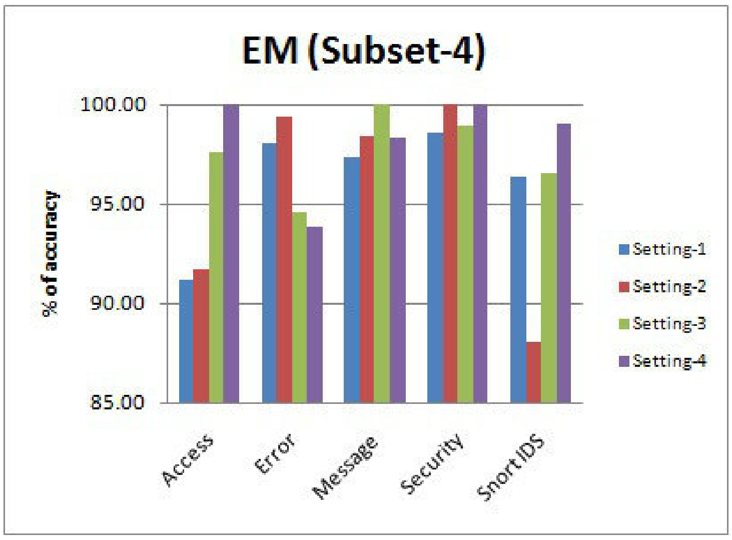


Fig. 6.9. *EM Clustering Accuracy (Subset-4)*

with respect to settings was similar to the achievement of K-Means with this subset. Analysing the clusters

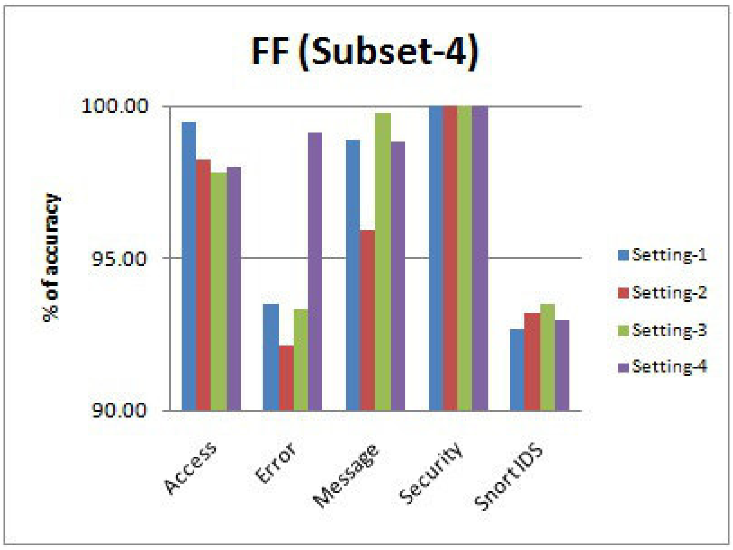


Fig. 6.10. *FF Clustering Accuracy (Subset-4)*

generated by FF with settings 3 and 4 for Apache access log revealed that all the unsuccessful intrusive events were precisely grouped in separate clusters and the cluster formation was better than K-Means. Like subset-3, all the intrusive events captured by Snort IDS log were grouped in bigger clusters for all the settings together with other events. The poor cluster generation of FF for this log with various settings were also reflected in the accuracy.

Analysing the clusters generated with the four log subsets tested provides the following conclusions:

1. Although the clustering strategy implemented achieved a better accuracy, none of the algorithm showed a consistent performance in terms of accuracy and cluster formation with all the subsets. Better clusters were formed by EM with subsets 1 and 4, and K-Means with subsets 2 and 3. This was due to the volume of events contained in these subsets for various logs.
2. Accuracy and cluster formation by a particular clustering algorithm was inconsistent with different logs in the same subset. This was due to capacity of the algorithm in handling a particular type of data as the number of features and the content of features varies with logs. Even in some logs the values for certain features were not available in most of the events.
3. Usage of settings 3 and 4 improved cluster formation with most of the logs. This was due to the increase in k which grouped the abnormal events precisely in separate clusters. EMs performance with this setting varied with subsets was due to its nature of forming clusters based on the available patterns in the log irrespective of the requested clusters. The cluster formation by FF with settings 3 and 4 were similar and this was due to the non-reactive nature of FF to seed values.
4. Examining the clusters generated shows that, the accuracy was not directly reflected in the formation of clusters. This is because calculation of accuracy is based on the context of the algorithms capacity in handling data, whereas the clusters produced were analysed based on the formation of abnormal events in separate clusters within the context of the research.
5. K-Means performed better even with the big sized logs, showing its capacity in handling larger datasets.
6. EM performed better with the logs where the values for certain features were not available in most of the events in the log.
7. FF generated better clusters with those logs where the events patterns were qualitatively different. This is due to its nature of calculating the cluster centroids in successive iterations which is opposite to that of K-Means.

Overall, the clustering strategy implemented in this phase achieved a better accuracy and cluster formation.

Due to the inconsistent performance of a particular algorithm and setting with various logs, deploying multiple algorithms and settings becomes necessary. This phase could be further extended by designing a recommender method to suggest the best clusters of various logs among those generated with various algorithms and settings, for further processing.

6.2. Filtered Events. In this phase, the clustered logs were filtered using a calculated threshold and the filtered-in events were retained for further process. The percentage of events reduced and the volume of abnormal events retained from the clustered logs with the three algorithms and its settings were evaluated (Experiment 3).

6.2.1. Event Reduction. Every clustered log was scanned to identify the number of events and clusters to calculate the threshold, and those clusters which satisfies the threshold were filtered-in.

Subset-2. The volume of events in various logs were relatively higher than that of subset-1 and especially the Snort IDS log contained 17049 events. Applying the threshold on various clustered logs for this subset reduced an average of 71% events and retained an average of 29% events for further examination and is provided in Table 6.2. Applying the threshold on Apache access log clustered by FF with setting-4 reduced 93.88% events whilst retaining 6.12% events, which was due to the better clusters formed by FF for this log.

Table 6.2
Percentage of Reduction (Subset-2)

Algorithms	Settings	Access	Error	Message	Security	Snort IDS
K-Means	Setting-1	75.51	57.28	68.87	86.09	72.79
	Setting-2	38.03	67.22	84.77	96.62	79.36
	Setting-3	77.74	85.60	67.55	79.70	74.86
	Setting-4	69.02	85.76	78.81	89.47	65.40
EM	Setting-1	57.88	60.60	64.24	81.95	55.90
	Setting-2	74.77	56.79	52.98	69.17	63.78
	Setting-3	79.78	61.42	52.98	68.42	67.44
	Setting-4	70.69	41.23	52.98	65.41	59.86
FF	Setting-1	76.25	76.32	68.87	82.71	56.57
	Setting-2	86.09	65.56	68.87	68.42	67.26
	Setting-3	85.16	77.98	76.82	92.48	82.62
	Setting-4	93.88	71.69	76.82	92.48	77.53

Applying the threshold on Apache error log clustered by K-Means with setting-4 reduced 85.76% events whilst retaining 14.24% events was due to the better cluster formation achieved by K-Means for this log. This also reveals the fact that better cluster generation will increase the volume of reduction which in turn reduces the processing overhead of the subsequent framework components.

Subset-3. The volume of events in this log subset was higher than subset-1, especially the events in Apache access and error log. Application of threshold on this log subset reduced an average of 73% events while retaining 27% events and the details of the event reduced with various logs is provided in Table 6.3. In some cases, due to the availability of more abnormal events in the log it was grouped in bigger clusters. Usage of the calculated threshold to filter the clustered events has reduced such abnormal events in bigger clusters making it unavailable for subsequent examination. For instance, FF produced better clusters for Apache access and error log, resulting in reducing more than 85% of the events from these logs. But, this reduction has removed most of the abnormal events in bigger clusters that did not satisfy the threshold.

Subset-4. The volume of events in this log subset was higher than subset-1, especially the events in Apache access and error log. Application of threshold on this log subset reduced an average of 74% events while retaining 26% events, and the details of the events reduced with various logs is provided in Table 6.4. Applying the threshold on Apache access log clustered by K-Means with setting-3 reduced 81.9% of events. But then all

Table 6.3
Percentage of Reduction (Subset-3)

Algorithms	Settings	Access	Error	Message	Security	Snort IDS
K-Means	Setting-1	72.27	75.67	74.76	81.62	65.53
	Setting-2	71.56	79.76	90.95	74.79	75.80
	Setting-3	68.67	80.00	87.14	76.92	51.53
	Setting-4	53.59	75.28	72.38	61.97	74.93
EM	Setting-1	55.70	67.40	45.71	57.69	78.48
	Setting-2	70.86	85.28	45.71	57.69	78.48
	Setting-3	68.83	78.11	60.48	57.69	62.24
	Setting-4	85.16	64.41	50.00	57.69	74.29
FF	Setting-1	89.30	89.84	60.48	84.62	75.82
	Setting-2	83.91	89.84	60.48	84.62	83.12
	Setting-3	88.91	93.46	64.29	75.21	87.65
	Setting-4	86.88	93.46	64.76	72.22	88.25

the abnormal events in this log were filtered-out by the threshold, due to the formation of such events in bigger clusters.

Table 6.4
Percentage of Reduction (Subset-4)

Algorithms	Settings	Access	Error	Message	Security	SnortIDS
K-Means	Setting-1	44.40	71.28	69.52	96.75	70.56
	Setting-2	74.57	89.67	68.57	96.75	50.33
	Setting-3	81.90	80.17	80.00	86.99	65.93
	Setting-4	76.72	80.58	76.19	91.87	65.26
EM	Setting-1	69.18	73.97	64.76	82.11	68.62
	Setting-2	69.18	64.26	71.43	91.06	54.44
	Setting-3	78.66	67.56	64.76	82.11	73.62
	Setting-4	61.64	54.55	74.29	82.11	71.09
FF	Setting-1	43.97	67.15	83.81	82.11	83.43
	Setting-2	54.09	62.81	91.43	82.11	80.24
	Setting-3	62.28	72.93	79.05	91.06	83.07
	Setting-4	72.41	73.55	79.05	91.06	83.47

6.2.2. Abnormal Event Retention. The abnormal events filtered-in by the threshold were analysed to verify whether all the anomalous events were retained. This is to ensure that, the application of threshold on the clustered events have not removed the significant events needed for further scrutiny.

Subset-2. There were two unsuccessful and one successful intrusive events in Apache access log and all these events were retained with clusters generated by all the algorithms and settings. But in the case of Apache error log, there were 14 successful intrusive events and 58 unsuccessful events. The 14 successful events were retained from K-Means clusters for setting 2 and 4, and from EM clusters with setting-2. The 58 unsuccessful intrusive events were retained from EM clusters for settings 1 and 4, and from FF clusters with settings 1 and 2. None of the algorithms retained both successful and unsuccessful events together for all the settings. This was due to the cluster formation with various settings and the placement of such events in bigger clusters which were filtered by the threshold.

Subset-3. There were 72 successful and 436 unsuccessful intrusive events in Apache access log. All the successful events were retained by the threshold from EM clusters. A maximum of 209 (47.94%) unsuccessful events were retained from K-Means cluster with setting-4. There were 484 intrusive events recorded by Apache error log, and a maximum of 102 (21.07%) such events were retained by the threshold from EM clusters with settings 1, 3 and 4. This was due to the high volume of such events in this log, which eventually joined together in bigger clusters and therefore filtered-out by the threshold.

Subset-4. There were 78 unsuccessful intrusive events in Apache error log. All these events were retained by the threshold from K-Means clusters with setting-1, and from FF clusters with settings 1 and 2. But the volume of events retained gets declined for settings 3 and 4 with K-Means and FF-clusters. This was because, the calculated threshold value decreased due to the increase in the number of clusters generated for these settings. Therefore, those unsuccessful events in a slightly bigger clusters was filtered out. But then, with EM clusters, 51 events were retained for settings 1 and 2, and 61 events for settings 3 and 4. The increase in retention was due to the nature of EM, by generating clusters depending only on the available patterns irrespective of the requested clusters.

The following are the conclusions that were drawn, analysing the filtered logs:

- There was no uniform increase or decrease in reduction of events with respect to algorithms and its settings. This was due to the varying event patterns in different logs and the influence of initialisation parameters, i.e, K and seed, on cluster formation.
- Event reduction varies with the calculated threshold and cluster size of the log being filtered. Threshold varied with the number of clusters generated whereas cluster size was influenced by the cluster formation which changes with the algorithms and settings, thereby influencing the reduction of events.
- Precise cluster formation did not always retain the maximum number of intrusive events when the volume of such events in the log was high. This was due to formation of such events in bigger clusters, which was eventually filtered out by the threshold.
- Application of calculated threshold managed to retain most or all the intrusive events in various logs, but then in some cases it failed, when such events were high or wrongly placed in the bigger clusters.

6.3. Aggregated Events. The filtered-in events were aggregated to reduce the duplicates, i.e., an event that contains similar values for all the features like the previous event. No additional feature was introduced in aggregated log to specify the number of events that were combined together to represent an aggregated event. Subsequently, all the intrusive events filtered-in were unique, accumulation did not eliminate any of these events. Fewer number of filtered-in events were reduced in logs, but then further process was not affected by this reduction. Apache access and error log were trimmed to an average of 12.75%, whereas an average of 2% in Linux syslog. Nearly 75% of the filtered in events were reduced in SSL-Error log was because of analogous event patterns recorded in this log with the same timestamp. Aggregation reduced an average of 20% filtered-in events in Snort IDS log. A similar percentage of reduction was accomplished by accumulation with the other three subsets as well. All the events that were retained by the respective logs after it has been filtered and aggregated were abnormal of some kind.

6.4. Transferred Events. The events transferred to GFL from various logs were filtered and aggregated in the previous phases. The unavailability of a feature in a log was replaced with a hyphen when transferred to the GFL. No additional features were introduced to represent the log type it belongs to, as it may mislead the clustering method. Although all the logs with the specified features in GFL were used for investigation to detect intrusions, only certain logs were transported to GFL format to be used in the clustering phase. Apache and Linux events were captured directly by the victim system and every abnormal activity expressed by Apache server logs and Linux syslog should have generated events in Snort IDS log, as it has been designed to capture such actions. As we planned to group the events to evaluate the relationship between the abnormal events of Apache server logs and Linux syslog with the events of Snort IDS, these events were transferred to GFL. Since the events of Apache SSL-Error log and Linux mail log did not have IP address and port number, these events were not transferred. As stated in GF the respective features of the excluded logs were maintained separately to be used in the analysis phase. A custom written Perl script extracted the features from various logs as stated in GF and transferred it to GFL, which took less than 5 seconds for transferring 11000 events. Since we have tested three algorithms with four settings, the volume of events transferred from individual logs to GFL varies with the algorithms and their respective settings. Therefore, the respective GFL events were maintained separately to be processed in subsequent phases and volume of GFL events for all the subsets is provided in Table 6.5. Every GFL was verified to ensure whether all the events in the aggregated logs were completely

Table 6.5
GFL Events

Algorithms	**Settings**	**Subset-1**	**Subset-2**	**Subset-3**	**Subset-4**
K-Means	Setting-1	2067	5001	2300	3258
	Setting-2	1145	3731	1773	4975
	Setting-3	1803	4566	2954	3583
	Setting-4	1966	5957	2041	3733
EM	Setting-1	1232	7808	2076	3375
	Setting-2	2664	6623	1738	4760
	Setting-3	1776	5917	2600	2881
	Setting-4	2635	7171	2021	3221
FF	Setting-1	2193	7515	1544	1971
	Setting-2	1319	5962	1222	2279
	Setting-3	946	3157	913	1969
	Setting-4	1545	3842	907	1873

transferred to GFL and the features extracted from various logs has been appropriately placed in the respective GFL features. Verification showed that the events from the respective logs were completely and appropriately placed in GFL without errors.

6.5. Clustered GFE. The GFL constitutes events from various logs that were mostly abnormal and these events were clustered to find the relationship between them using the same clustering strategy used before. Different volumes of GFL events which varies according to the algorithms and settings were clustered separately. The ideal number of clusters for GFL was manipulated using EM as stated in Experiment 4, and the resulting clusters (K) were used to group GFL as per Experiment 5 stated. As the GFL constitutes events from various logs which were clustered and filtered previously, accurate cluster formation and subsequent retention of abnormal events in those phases affected the clustering accuracy and formation in this phase.

Since the volume of events in GFL that was clustered varies with algorithms and settings, the accuracy achieved with various algorithms and settings could not be directly compared.

Subset-2. The accuracy of clustering by various algorithms and settings for subset-2 is illustrated in Figure 6.11.

The maximum accuracy 100% achieved by FF with setting-4 was due to the grouping of likely events precisely in separate clusters. Especially, the Snort IDS events with different patterns were grouped in separate clusters. K-Means with setting-1, generated clusters which contained events from various logs together. Like K-Means, FF and EM with setting-1 also generated clusters containing events from various logs together in the same cluster. This was due to the similarity between the event features contained in the GFL.

Subset-3. The accuracy of clustering by various algorithms and settings for subset-3 is illustrated in Figure 6.12. Like subset-2, the highest accuracy of 100% was achieved with FF due to the precise cluster formation with respect to the patterns. All the three algorithms generated clusters separating the events according to the patterns for this subset, i.e., Linux and Apache events joined together in the same clusters whereas Snort IDS events joined together in separate clusters. But the clusters generated by K-Means with setting-2 contained clusters with events from various logs.

Subset-4. The accuracy of clustering by various algorithms and settings for subset-4 is illustrated in Figure 6.13. Like subsets 2 and 3, the highest accuracy was achieved by FF due to the separation of events in various logs according to patterns. K-Means generated clusters that constitute events from various logs was due to its nature of giving equal importance to all the features in the log and also the feature similarity of the events in GFL.

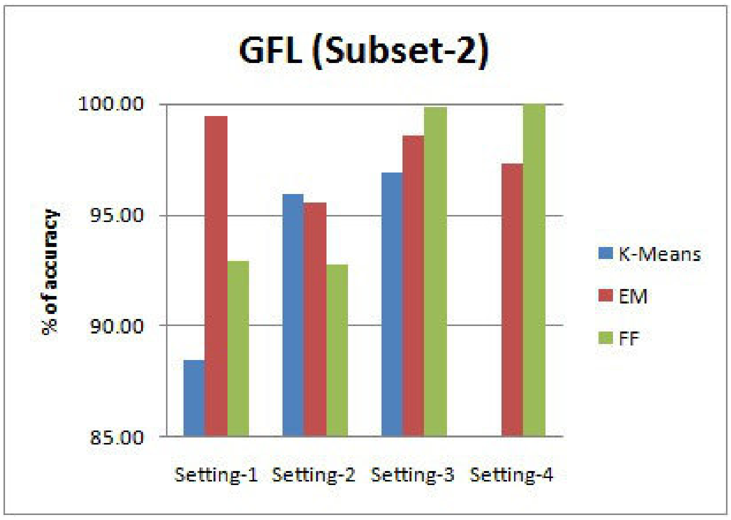


Fig. 6.11. *Clustering Accuracy with GFL (Subset-2)*

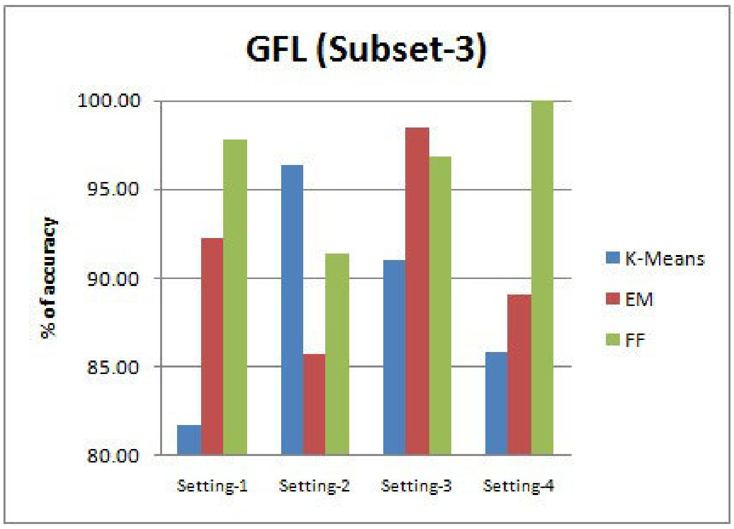


Fig. 6.12. *Clustering Accuracy with GFL (Subset-3)*

Analysing the clusters generated by various algorithms with varying volume of GFL events provides the following conclusions:

1. The variations in clustering accuracy and cluster formation with different algorithms and settings was not only due to the parameter initialisation for that setting, but also due to the variations in volume of events clustered with that algorithm and setting.

2. FF achieved the maximum accuracy with all the four subsets exposed its capacity in clustering heterogeneous log events of varying patterns, despite the absence of values for some of the features.

3. K-Means produced clusters that constitutes events from various logs together for most of the subsets. This was not only due to the similarity in the event features, but also shows its capacity in treating features in an event equally.

6.6. Anomaly Detection. In this phase, the clustered GFL events were examined to detect anomalies by finding the relationship between the features recorded by the events form various logs. The features source

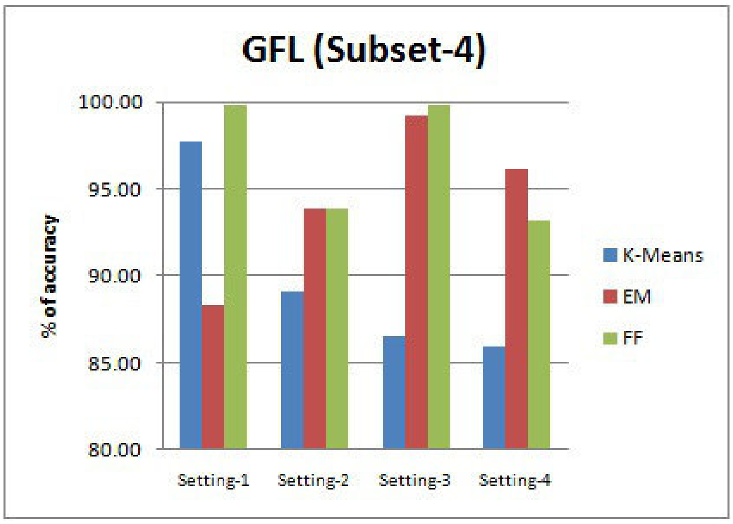


FIG. 6.13. *Clustering Accuracy with GFL (Subset-4)*

IP address (f_{sip}) and destination IP address (f_{dip}) together with cluster number (f_k) were examined during IP Address analysis and the features source port number (f_{sp}) and destination port number (f_{dp}) were examined during port number analysis. Previously the results of subset-1 for anomaly detection was presented in Hajamydeen et.al. [7], and therefore the results of the other three subsets, i.e., subset-2, subset-3, subset-4, were only discussed. Moreover, a portion of the results on the other three subsets, i.e., subset-2, subset-3, subset-4, were mentioned in Hajamydeen et.al. [52] for comparison purposes, but not detailed.

6.6.1. IP Address Analysis. Every clustered GFL with the respective algorithms and settings were analysed separately and the resulting anomalous events were evaluated as per Experiment 6. Most or all clustered events of GFL were basically anomalous and IP analysis managed to locate the anomalies by identifying the relation between events with respect to IP Address followed by cluster number. IP analysis indicated its capacity in recognising most of the anomalous events in the logs and the volume of anomalous events recognised was affected by the creation of clusters in individual logs and GFL.

Subset-2. The volume of events in this subset was larger compared to other subsets. The anomalous events detected by IP analysis using the clustered events generated by various algorithms with different settings are illustrated in Figure 6.14. The best performance for this subset was achieved with K-Means clusters by discovering most of the anomalous events through IP analysis. This was due to the capacity of K-Means in generating better clusters even with bigger datasets. There were 16 events related to successful intrusion and 59 events related to unsuccessful intrusion exploiting the AWStats vulnerability recorded in Apache access and error log which originated from two IP addresses. The events pertaining to this activity was not available in Snort IDS log, as it was missed by Snort. But IP analysis managed to detect most of these anomalous events. The volume of events detected by IP analysis from these IP addresses are illustrated in Figure 6.15. IP analysis detected all the unsuccessful intrusive events with EM clusters for settings 1 and 4, and with FF clusters for settings 1 and 2, but then, most of the successful intrusive events were not detected with these settings. All the successful events were detected with K-Means clusters for settings 2 and 4, and with EM clusters for setting-2. This was due to the unavailability of these events for analysis, since it was filtered-out in the previous phase. Including the IP addresses for which the results are plotted (Figure 6.15), there were seven IP addresses from where the anomalous activity have originated and there were 1252 events related to these IP addresses in various logs.

Over 900 anomalous events from six IP addresses were detected with the clusters generated by K-Means for settings 1 and 2. With K-Means clusters for settings 3 and 4, the events related to all the seven IP addresses were detected, but then the volume of such events were less compared to the other two settings. The volume

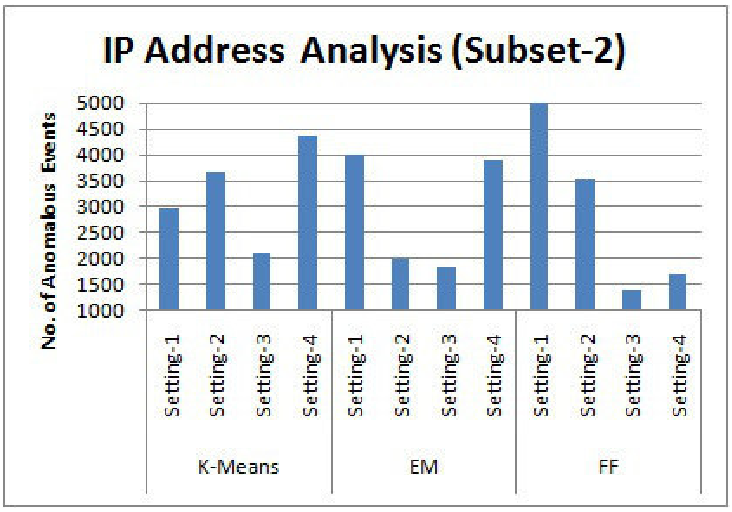


Fig. 6.14. *Anomaly Detection by IP Address Analysis (Subset-2)*

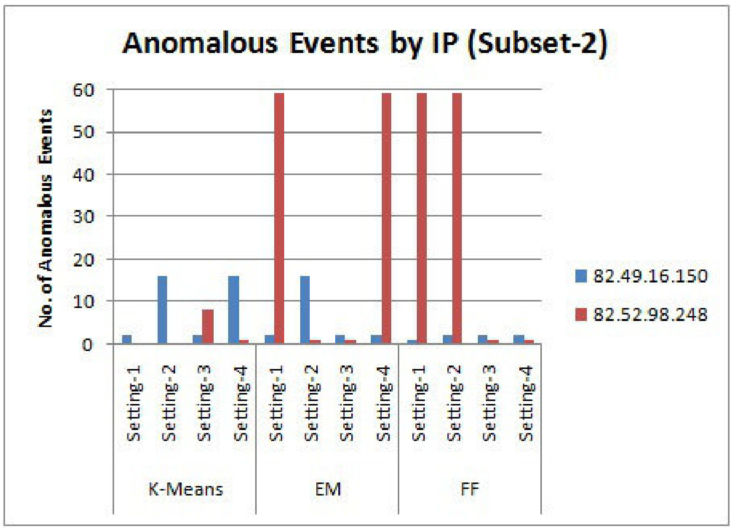


Fig. 6.15. *Anomalous Events by IP (Subset-2)*

of events or the number of IP addresses detected with the clusters of a particular setting was influenced by the cluster formation and the subsequent application filtering threshold based on cluster formation. Although IP analysis detected the most number of anomalous events with FF clusters for this subset, it failed to detect most of the significant anomalies pertaining to these seven IP addresses. This was because of the unavailability of these events due to cluster formation and subsequent application of filtering threshold in the previous phases.

Subset-3. Like subset-2, IP analysis detected majority of the anomalous events with K-Means clusters for this subset too. The anomalous events detected by IP analysis using the clustered events generated by various algorithms with different settings are illustrated in Figure 6.16. The anomalous events have originated from ten IP addresses and there were 1901 such events in this subset. A maximum of 423 events from nine IP addresses were detected by IP analysis with K-Means clusters. This was due to the volume of such events were large and most of these events were filtered out by the threshold, making it unavailable for analysis. A maximum of 126 events were detected with EM clusters and 71 events with FF clusters. This reveals the performance of the EM and FF clustering in handling bigger size logs.

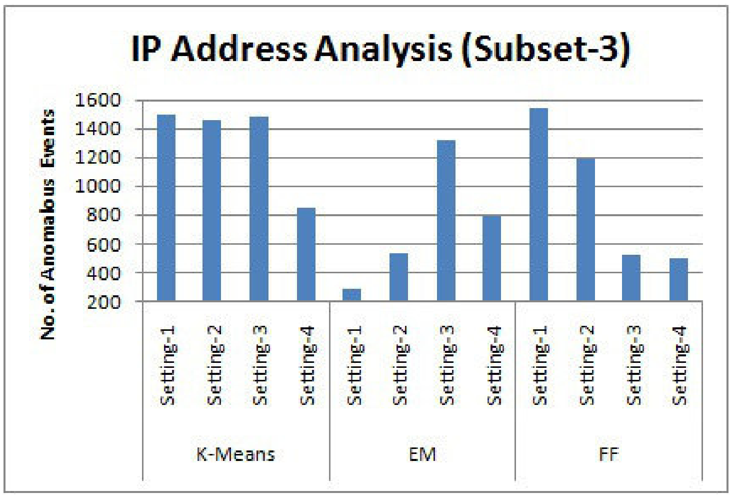


FIG. 6.16. *Anomaly Detection by IP Address Analysis (Subset-3)*

There were 30 events related to xinetd crash which originated from three IP addresses that were recorded by Linux security and Snort IDS log. A total of 20 such events were detected by IP analysis with K-Means and FF clusters as illustrated in Figure 6.17. The figure also shows that, the event from the IP address 195.22.66.28

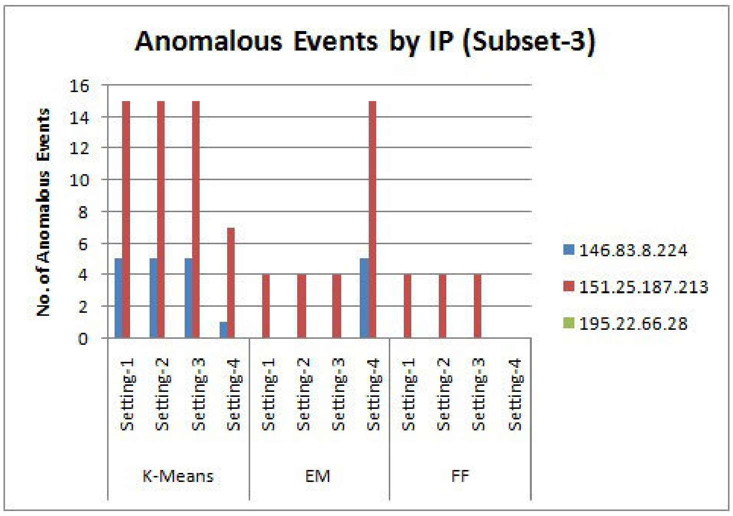


FIG. 6.17. *Anomalous Events by IP (Subset-3)*

was not detected with the clusters of any algorithms. There was only one event from the IP address 195.22.66.28 that was recorded by Linux security log and this event was filtered out for all the settings with K-Means and FF clusters, making it unavailable for analysis. Even though this event was retained after filtering with EM clusters, the failure of IP analysis to detect it was due to the cluster formation with GFL.

Subset-4. The anomalous events identified by IP analysis for various algorithms with different settings are illustrated in Figure 6.18. IP analysis detected most of the anomalous events with K-Means and EM clusters for this subset as illustrated in Figure 6.19. There were 144 events originated from three IP addresses and especially the events related to RPC attack was recorded only in Snort IDS log.

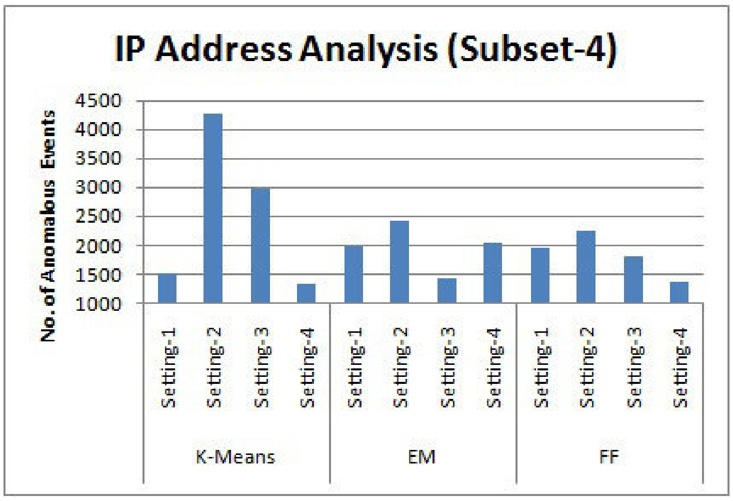


FIG. 6.18. *Anomaly Detection by IP Address Analysis (Subset-4)*

IP analysis detected majority of the anomalous events from the IP addresses 220.110.29.27 and 59.120.2.133 with the clusters generated by various algorithms and the respective settings. The RPC attack originated from 62.111.213.88 were not detected by IP analysis, which was due to the cluster formation with GFL with some of the settings.

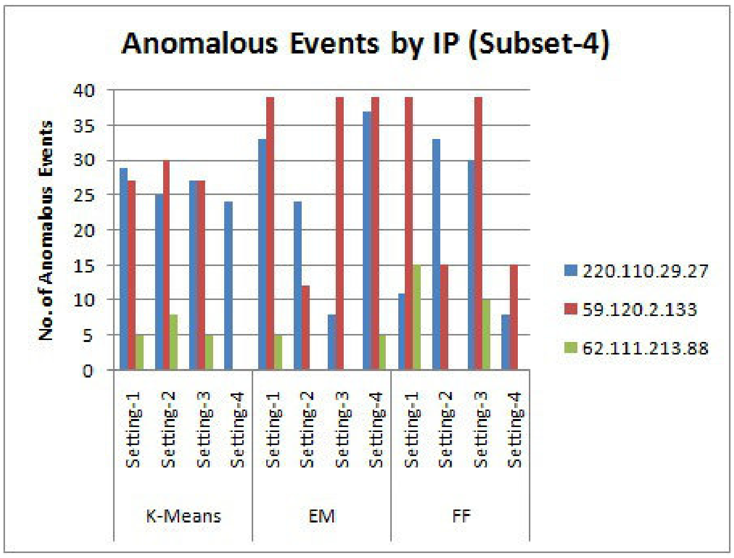


FIG. 6.19. *Anomalous Events by IP (Subset-4)*

The following are the conclusions drawn based on the detected anomalies by IP analysis:

- The detection performance of was dependent on the performance of the previous phases. As IP analysis bases its decisions from the log events received as input which was manipulated by the previous phases, the performance of these phases affected the detection process.
- The detection performance declined with those log subsets having a larger volume of anomalous events. Although, the clustering strategy implemented generated better clusters, the high volume of anomalous events in the logs were grouped in bigger clusters. The application of threshold filtered-out these anomalous events and hence not available for analysis.

- The maximum number of anomalous events from various IP addresses were detected with EM clusters for subsets 1 and 4, whereas with K-Means clusters for subsets 2 and 3. The volume of events in various logs with subsets 2 and 3 were high, and K-Means forming better clusters than EM especially with these subset shows its ability in handling larger datasets.

6.6.2. Port Number Analysis. Every clustered GFL with the respective algorithms and settings were analysed separately and the resulting anomalous events were evaluated as per Experiment 7. A simple port analysis was done by comparing the port number of the events with the IANA listing of unassigned and dynamic port numbers. Only the Snort IDS events in GFE were scrutinized, since it contains both source and destination ports.

Subset-2. The volume of anomalous events detected by port analysis with various algorithms and its settings are illustrated in Figure 6.20. A maximum of 595 events were detected by port analysis for this subset

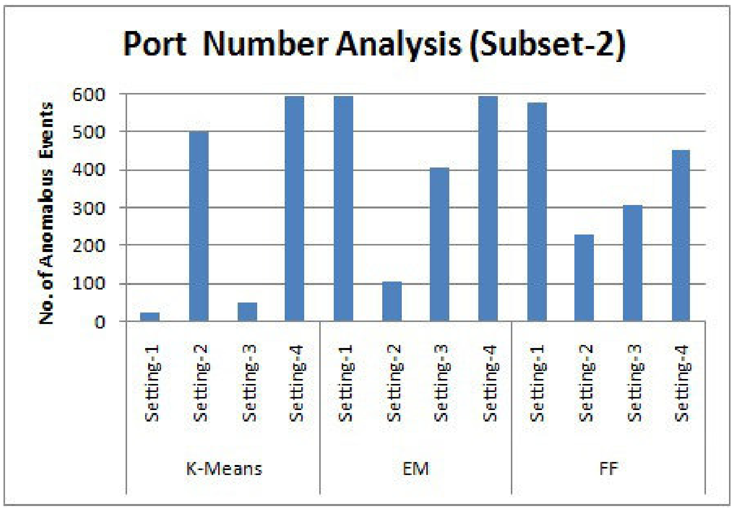


Fig. 6.20. *Anomaly Detection by Port Number Analysis (Subset-2)*

with EM clusters and the highest number 105 of anomalous events detected with setting-4 shows the impact of K on cluster formation. The anomalous events pertaining to AWStats vulnerability were not detected by port analysis which was available in this subset. The events related to this activity were recorded by Apache access and error log and was missed by Snort IDS. Since port analysis considers only those events with port numbers, these events were not detected. There were five IP addresses recorded by Snort IDS log from where the intrusions originated, out of which the events related to one IP address was detected with most of the clusters generated with various algorithms. This was due to cluster formation and the subsequent retention of events by the threshold for this log.

Subset-3. The anomalous events detected by port analysis for this subset with various algorithms and its settings are illustrated in Figure 6.21. A maximum of 111 events were detected with FF clusters (setting-2), which includes seven anomalous events from two IP addresses as stated in SOTM#34 analysis results. Similarly, a maximum of 78 events were detected with EM clusters (setting-4) which comprises 10 anomalous events from two IP addresses. This was due to cluster formation and the subsequent retention of events by the threshold for this log.

Subset-4. The anomalous events detected by port analysis for subset-4 with various algorithms and its settings are illustrated in Figure 6.22. There were 90 anomalous events originated from three IP address that were recorded by Snort IDS log. A maximum of 20 anomalous events from two IP addresses were detected by port analysis with K-Means clusters (setting-2). Although the events from these two IP addresses were detected with the clusters generated with EM and FF with certain settings, the volume of events detected were

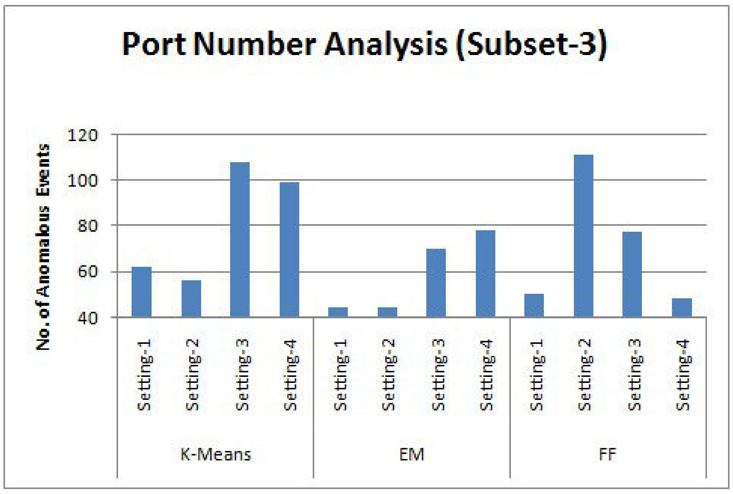


FIG. 6.21. *Anomaly Detection by Port Number Analysis (Subset-3)*

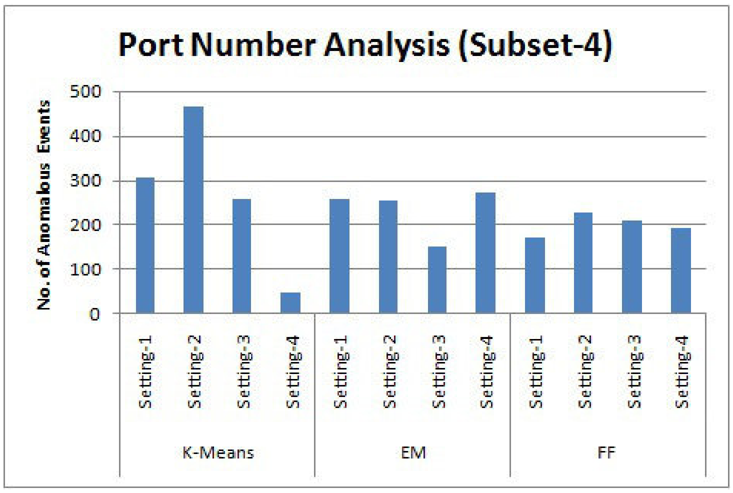


FIG. 6.22. *Anomaly Detection by Port Number Analysis (Subset-4)*

less compared to K-Means. This was due to the cluster formation of these algorithms with Snort IDS log. Moreover, the volume of events in Snort IDS log for this subset was large. The better performance of K-Means with this subset shows its ability in clustering voluminous log events, thereby supporting the detection process.

In conclusion,

- Approximately 10% to 15% of the anomalous events in the logs were detected by port analysis was due to the unavailability of port numbers in most of the log events.
- A range of 1 to 187 anomalous events that were missed during IP analysis were discovered by port analysis with various subsets. Most of the events identified by port analysis were already identified during IP analysis reduces the need for port analysis.
- The detection of anomalous events by port analysis was influenced by the cluster formation in Snort IDS log and the succeeding events retained by the filtering threshold.

6.7. Consolidated Anomalous Events. The anomalous events detected by IPA and PA with the respective algorithms and settings were consolidated separately and the resulting anomalous events were evaluated as per Experiment 8. In the process of consolidation, the distinct anomalous events identified by both analysis methods were initially combined together. To get a better picture of the successful and unsuccessful intrusion, those IPTables firewall log events whose IP address matches the IP address of the combined anomalous events were extracted and merged with the combined analysis results. The volume of events that gets added during correlation depends on the available IP addresses of the events detected during analysis and the corresponding match with IPTables firewall log. The anomalous events were concentrated using a threshold on the occurrence of events pertaining to an IP address after correlating with IPTables firewall log. This was performed to trim down the insignificant events that can be disregarded. Since varying volume of events were detected with the clusters generated with various algorithms and settings in the previous phase, combining and correlating these events also yielded varying volume of events. Concentrating these events with a specific threshold reduced a small number of events according to the occurrence of events related to an IP address. Although concentration trimmed down most of the insignificant events, there are chances of significant anomalous events being reduced, as the threshold is based on IP address. Therefore, after concentrating the events with several thresholds, the consolidated events with various threshold was verified to ensure whether the application of threshold removed or reduced the significant anomalous events, i.e., as stated in SOTM#34 analysis results, from various IP addresses. Therefore the significant anomalous events detected by UHAD which was also mentioned in SOTM/#34 analysis results were discussed in this section. But then, UHAD has detected more number of anomalous events than those mentioned in SOTM#34 analysis results.

Subset-2. Even though the volume of events in this subset was high, concentrating the events with the threshold, i.e., A_t= 9, reduced an average of 3% significant anomalous events. The details of the significant anomalous events retained from the clusters generated with various settings and algorithms is provided in Figure 6.23. A total of 1339 significant anomalous events which was launched from several IP addresses were retained

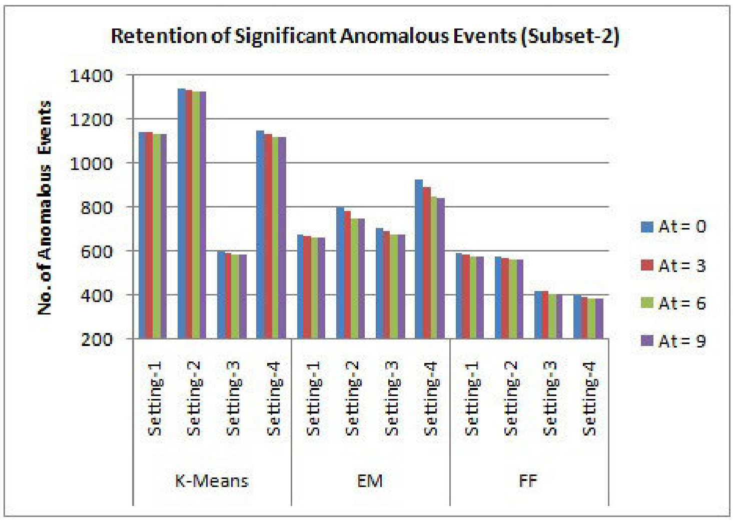


Fig. 6.23. *Retention of Significant Anomalous Events (Subset-2)*

after combining and correlating the events detected with K-Means clusters. Applying the threshold, i.e., A_t= 9, removed 31 of these events from three IP addresses.

Subset-3. The effect of threshold in retaining significant anomalous events is presented in Figure 6.24. A minimum average of 0.8% significant anomalous events were reduced with K-Means clusters and a maximum average of 4.3% events were removed with EM clusters during concentration with the maximum threshold, i.e., A_t= 9. This was due to the poor cluster formation by EM for this subset with most settings. A maximum of 2205 events were yielded after combining and correlating the detected events with K-Means clusters for setting-3. Applying the threshold, i.e., A_t= 9, removed 16 anomalous events from two IP addresses.

Subset-4. The details of the significant anomalous events retained from the clusters generated with various settings and algorithms is provided in Figure 6.25. A minimum average of 0.6% significant anomalous events

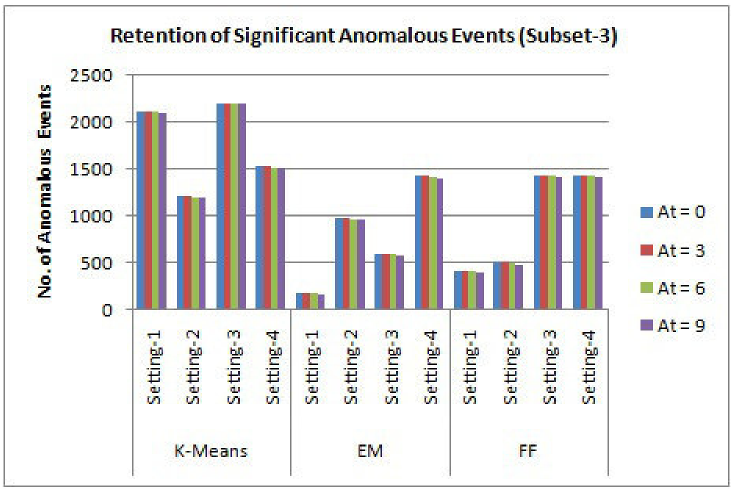


FIG. 6.24. *Retention of Significant Anomalous Events (Subset-3)*

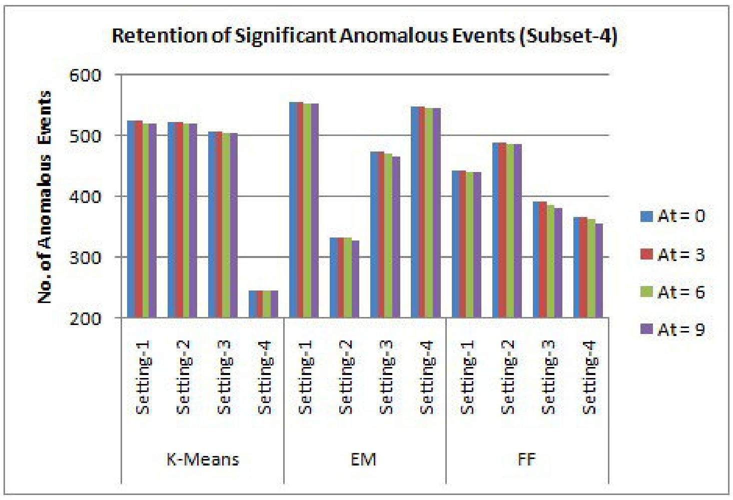


FIG. 6.25. *Retention of Significant Anomalous Events (Subset-4)*

were reduced with K-Means clusters and a maximum average of 1.8% events were removed with FF clusters during concentration with the maximum threshold, i.e., $A_t = 9$. A maximum of 555 significant anomalous events were captured after combining and correlating the detected events with EM clusters for setting-1. Application of threshold, $A_t = 9$ reduced three events from an IP address.

The following conclusions are drawn from consolidating the detected events:

- As the reduction was based on the IP addresses of those events that were detected and correlated previously, the output of this phase highly depends on the log events received from the previous phase. Therefore, the volume of events yielded after consolidation varies with the detected events of various algorithms and settings.
- Concentrating the detected events with the threshold not only reduced the insignificant events but also reduced a minimum number of significant anomalous events. Though the usage of threshold is not viable at this end, it will assist in focusing the most critical events that need to be immediately addressed.

UHAD detected a wide range of anomalies which includes Nimda scans, CodeRed worms, SSH Brute Force scans, CONNECT scans, IRC Traces, RPC attack, Xinetd crash and the AWStats vulnerability exploit. The low percentage of anomalies detected with subset-3 is due to the high volume of events and anomalies in this subset.

7. Conclusion and Future Works. The majority of the intrusion detection mechanisms available were knowledge dependent which makes use of the characteristics of anomalies or the model of traffic behaviour to detect anomalies which restricts the mechanisms to detect only known anomalies. Moreover, the existing detection methods considers a single type of log for analysis, which confines the method to detect anomalies presented only in those logs and the anomalies in the other logs were left behind. To overcome these limitations, this thesis has presented a new framework UHAD to detect a variety of anomalies by scrutinizing logs from heterogeneous sources, without using the characteristics of anomalies that hold the specification of the actions to match with events or the usual method of training and testing commonly used in anomaly detectors. Although, the three clustering algorithms tested in the framework took less time to predict and generate clusters, the accuracy of the clusters generated by an algorithm were not consistent across different logs and subsets. This was due to the capacity of the clustering algorithms in handling the event patterns and features in a particular log. Additionally, the clustering parameters used to group the events also influenced the accuracy. Subsequently, applying the filtering threshold which was automatically calculated based on these generated clusters, managed to retain majority of the abnormal events and removed the normal or insignificant events with most of the logs and subsets. With some of the logs, the filtering threshold failed to retain the abnormal events was due to the existence of more number of such abnormal events of similar patterns that were grouped in larger clusters. The usage of GFL induced the process of classifying heterogeneous log events in a single structure and enabled faster analysis by the detection algorithms. Introducing more features in GFL, which did not existed in many of the logs, affected accurate cluster formation thereby reducing the volume of the events detected during analysis. This shows the criticality in selecting the appropriate features for GFL based on the logs considered. Further analysis of the GFL based on the IP addresses and port numbers detected the events related to a wide range of anomalies and most importantly, more anomalies were detected during IP address analysis than port number analysis. Manipulating IP address (source IP and destination IP) together with the cluster number supported the analysis process in detecting majority of the anomalous events. Moreover, almost all the anomalies detected by port analysis were also detected by IP address analysis thereby reducing the need for port analysis. Consolidating the detected anomalies assisted in focusing the most significant anomalous events, but then failed to retain a few of the anomalous events, when the occurrence of such events did not satisfy the threshold applied.

The proficiency and precision of UHAD in recognizing intrusions were scrutinized using three clustering algorithms with four parametric settings and the results achieved were compared with the ground truth available at Honeynet.org for this dataset, i.e. SOTM#34. The detected anomalies were analogous with the output of other methods, therefore demonstrating the accuracy of UHAD in detecting anomalies. Among the three algorithms used by the framework, EM and K-Means generated better clusters supporting the investigation process to identify the majority of the anomalies in all the four subsets. The coverage of anomalies with FF clusters were marginally lower than EM and K-Means, but is also appropriate for this framework due to its faster clustering even with larger datasets.

All the parameters used in UHAD were manipulated based on the tested dataset only. Therefore, the accuracy of clusters and subsequent retention of anomalies by the threshold were influenced by these parameters. None of the algorithm showed a consistent accuracy with a specific parameter setting; and also the performance of a particular algorithm was not steady with different subsets. Because of this, some of the anomalous events were filtered-out by the threshold, making it unavailable for further investigation. Since this is the first step towards building an unsupervised anomaly detector using heterogeneous logs that calculates all the required parameters based on tested data itself; the mechanism of manipulating the clustering parameters and filtering threshold could be refined to improve the precision of anomaly detection. A recommender method could be designed to select the most accurate clusters for a particular log, among those produced by various algorithms with different parametric settings. This method need to evaluate various output parameters based on the clusters generated which requires criteria and threshold to be framed in order to evaluate such parameters to choose the accurate clusters. Subsequently, these selected clusters can be used for further investigation to detect almost all the anomalies. UHAD can also be tested with various datasets collected from heterogeneous sources to substantiate its capacity in discovering the anomalous events pertaining to a wide range of intrusions.

REFERENCES

[1] S. More, M. Matthews, A. Joshi and T. Finin, *A Knowledge-Based Approach To Intrusion Detection Modeling.* In 2012 IEEE Symposium on Security and Privacy Workshops (SPW),2012, pp. 75–81. IEEE.

[2] C. Abad,J. Taylor, C. Sengul, W. Yurcik, Y. Zhou and K. Rowe, *Log correlation for intrusion detection: A proof of concept*, In Proceedings of 19th Annual Computer Security Applications Conference., 2003, pp. 255–264.

[3] Z. Li, J. Taylor, E. Partridge,Y. Zhou, W. Yurcik, C. Abad, J. Barlow and J. Rosendale, *UCLog: A unified, correlated logging architecture for intrusion detection*, In the 12th International Conference on Telecommunication Systems-Modeling and Analysis (ICTSM)., 2004.

[4] W. Yurcik, C. Abad, R. Hasan, M. Saleem and S. Sridharan, *UCLog+: A Security Data Management System for Correlating Alerts, Incidents, and Raw Data From Remote Logs*, Arxiv preprint cs/0607111., 2006.

[5] R. Kumari and K. Sharma, *Cross-Layer Based Intrusion Detection and Prevention for Network*, In Handbook of Research on Network Forensics and Analysis Techniques, 2018, pp. 38-56. IGI Global.

[6] D. Denning, *An intrusion-detection model*, IEEE Transactions on software engineering., 2(1987), pp. 222-232.

[7] A. Hajamydeen, N. Udzir, R. Mahmod and A. Ghani, *An unsupervised heterogeneous log-based framework for anomaly detection*, Turkish Journal of Electrical Engineering and Computer Sciences., 24(2016), pp. 1117-1134.

[8] S. Peisert and M. Bishop, *How to design computer security experiments*, In Fifth World Conference on Information Security Education., 2007 , pp. 141–148. Springer.

[9] E. Barse and E. Jonsson, *Extracting attack manifestations to determine log data requirements for intrusion detection*, In 20th Annual Computer Security Applications Conference., 2004, pp. 158–167. IEEE.

[10] X. Wang, A. Abraham and K. Smith, *Intelligent web traffic mining and analysis*, Journal of Network and Computer Applications., 28.2(2005), pp. 147–165.

[11] A. Chuvakin, *Scan of the Month 34*, http://www.honeynet.org/scans/scan34/, 2005.

[12] Andrew, *Scan of the month 34-Solution*, http://www.honeynet.org/scans/scan34/sols/3/sotm/, 2005.

[13] M. Richard and M. Ligh, *Project Honeynet Scan of the Month 34*, http://project.honeynet.org/scans/scan34/sols/1/index.html, 2005.

[14] C. Kronberg, *Analysis of the log files given in SOTM34*, http://project.honeynet.org/scans/scan34/sols/2/proc.pdf, 2005.

[15] A. Chuvakin, *Scan of the Month Challenge 34- Official Solution*, http://project.honeynet.org/scans/scan34/sols/sotm34-anton.html, 2005.

[16] S. Panichprecha, *Abstracting and Correlating Heterogeneous Events to Detect Complex Scenarios*, PhD thesis, Queensland University of Technology, Brisbane, Australia., 2009.

[17] J. Herrerias and R. Gomez, *Log analysis towards an automated forensic diagnosis system*, In International Conference on Availability, Reliability, and Security, ARES'10., 2010, pp. 659–664.

[18] J. Herrerias and R. Gomez, *A log correlation model to support the evidence search process in a forensic investigation*, In Second International Workshop on Systematic Approaches to Digital Forensic Engineering, SADFE., 2007, pp. 31–42.

[19] K. Kent and M. Souppaya, *Guide to computer security log management*, NIST Special Publication, 800–92.,2006.

[20] G. Ferrar, *Sawmill (Version 8.1.10)*, http://www.sawmill.met, 2011.

[21] F. Amiri, M. Yousefi, C. Lucas, A. Shakery and N. Yazdani, *Mutual information-based feature selection for intrusion detection systems*, Journal of Network and Computer Applications., 34.4(2011), pp. 1184–1199.

[22] T. Chou, K. Yen and J. Luo, *Network intrusion detection design using feature selection of soft computing paradigms*, International Journal of Computational Intelligence., 4.3(2008), pp. 196–208.

[23] C. Sinclair, L. Pierce and S. Matzner, *An application of machine learning to network intrusion detection*, In Proceedings. 15th Annual Computer Security Applications Conference., 1999, pp. 371–377. IEEE.

[24] D. Barbara, N. Wu and S. Jajodia, *Detecting novel network intrusions using bayes estimators*, In Proceedings of the First SIAM International Conference on Data Mining., 2001.

[25] Y. Li, N. Wu, X. Wang and S. Jajodia, *Enhancing profiles for anomaly detection using time granularities*, Journal of Computer Security., 10.1-2(2002). pp.137– 157

[26] S. Staniford, J. Hoagland and J. McAlerney, *Practical automated detection of stealthy portscans*, Journal of Computer Security., 10.1-2(2002), pp.105–136 .

[27] H. Tribak, I. Rojas and O. Valenzuela, *Comparison of Soft-Computing Techniques for classification of Intrusion-Detection*, In Proceedings of the 2010 International Conference on Mathematical Models for Engineering Science., 2010, pp.284–288. World Scientific and Engineering Academy and Society (WSEAS).

[28] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I. Witten, *The Weka data mining software: An update*, ACM SIGKDD Explorations Newsletter., 2009, pp.10–18.

[29] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez and E. Vazquez, *Anomaly-based network intrusion detection: Techniques, systems and challenges*, Computers & Security., 28.1-2(2009), pp. 18–28.

[30] J. Erman, M. Arlitt and A. Mahanti, *Traffic classification using clustering algorithms*, In Proceedings of the 2006 SIGCOMM workshop on Mining network data., 2006, pp.281–286.ACM.

[31] M. Aneja, T. Bhatia, G. Sharma and G. Shrivastava, *Artificial Intelligence Based Intrusion Detection System to Detect Flooding Attack in VANETs*, In Handbook of Research on Network Forensics and Analysis Techniques., 2018, pp. 87-100. IGI Global.

[32] I. Syarif, A. Prugel-Bennett and G. Wills, *Unsupervised clustering approach for network anomaly detection*, Networked

Digital Technologies, 2012, pp. 135–145.

[33] J. Song, H. Takakura, Y. Okabe and K. Nakao, *Toward a more practical unsupervised anomaly detection system*, Information Sciences., 231(2011), pp. 4-14.

[34] G. Wang, J. Hao, J. Ma and L. Huang, *A new approach to intrusion detection using artificial neural networks and fuzzy clustering*, Expert Systems with Applications., 37.9(2010), pp. 6225–6232.

[35] G. Munz, S. Li and G. Carle, *Traffic anomaly detection using K-Means clustering*, GI/ITG Workshop MMBnet., 2007.

[36] Y. Liu, W. Li and Y. Li, *Network traffic classification using kmeans clustering*, In Second International Multi-Symposiums on Computer and Computational Sciences., 2007, pp.360–365. IEEE.

[37] R. Smith, N. Japkowicz, M. Dondo and P. Mason, *Using unsupervised learning for network alert correlation*, Advances in Artifcial Intelligence., 2008, pp.308–319.

[38] U. Zurutuza, R. Uribeetxeberria, E. Azketa, G. Gil, J. Lizarraga and M. Fernndez, *Combined data mining approach for intrusion detection*, In International Conference on Security and Criptography., 2008.

[39] M. Panda and M. Patra, *A novel classification via clustering method for anomaly based network intrusion detection system*, International Journal of Recent Trends in Engineering., 2(2009), pp. 1–6.

[40] M. Siraj, M. Maarof and S. Hashim, *Intelligent alert clustering model for network intrusion analysis*, Int. J. Advance. Soft Comput. Appl., 1(2009), pp. 33–48.

[41] U. Zurutuza, R. Basagoiti and A. Aztiria, *Behavior analysis of domain servers through windows security event log mining*, J. Inform. Assurance Security., 5.4(2010), pp.418–425.

[42] G. Tjhai, S. Furnell, M. Papadaki and N. Clarke, *A preliminary two-stage alarm correlation and filtering system using SOM neural network and K-means algorithm*, Computers & Security., 29.6(2010), pp.712–723.

[43] O. Siriporn and S. Benjawan, *Anomaly detection and characterization to classify traffic anomalies case study: TOT public company limited network*, In World Academy of Science, Engineering and Technology., 2008, pp.407–415.

[44] M. Panda and M. Patra, *A hybrid clustering approach for network intrusion detection using cobweb and FFT*, Journal of Intelligent Systems., 18.3(2009), pp.229–246.

[45] J. MacQueen, *Some methods for classification and analysis of multivariate observations*, In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability., 1967.

[46] S. Dasgupta, *Performance guarantees for hierarchical clustering*, In Computational Learning Theory., 2002, pp.351–363.

[47] X. Zheng, Z. Cai and Q. Li, *An experimental comparison of three kinds of clustering algorithms*, In Proceedings of International Conference on Neural Networks and Brain., 2005, pp. 767–771. IEEE.

[48] A. Dempster, N. Laird and D. Rubin, *Maximum likelihood from incoming data via the EM algorithm*, J. Royal Stat. Soc., 1977, pp.1–38.

[49] M. Tavallaee, N. Stakhanova and A. Ghorbani, *Toward credible evaluation of anomaly-based intrusion-detection methods*, IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews., 40.5(2010), pp. 516–524.

[50] X. Yu, L. Tang and J. Han, *Filtering and refinement: A two-stage approach for efficient and effective anomaly detection*, In 2009 Ninth IEEE International Conference on Data Mining., 2009, pp. 617–626.

[51] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos and K. Lee, *Internet traffic classification demystified: Myths, caveats, and the best practices*, In Proceedings of the 2008 ACM CoNEXT Conference., 2008.

[52] A. Hajamydeen and N. Udzir, *A refined filter for UHAD to improve anomaly detection*, Security and Communication Networks., 9(2016), pp. 2434-2447.

Edited by: Khaleel Ahmad
Received: Nov 19, 2018
Accepted: Feb 11, 2019