

Anomaly Detection Using S Language Framework: Clustering and Visualization of Intrusive Attacks on Computer Systems

Khaled Labib and V. Rao Vemuri

Department of Applied Science, University of California, 3001 Engineering III, One Shields Avenue, Davis, California 95616, U.S.A.

E-Mail: {[kmlabib](mailto:kmlabib@ucdavis.edu), [rvemuri](mailto:rvemuri@ucdavis.edu)}@ucdavis.edu

Many intrusion detection projects employ a multitude of statistical methods and machine learning techniques to achieve their goal. However there is a lack of a unified framework for developing, testing and comparing the results. This study introduces the S Language and its environment as a potential candidate for this unified framework, which can also be used to develop new methods, alter existing ones or create hybrid solutions. The S Language, originally developed at Bell Laboratories, provides a powerful environment for statistical and graphical analysis of data. This study illustrates the power of S in computer security research. Three clustering algorithms are used to detect selected Denial-of-Service and Network Probe attacks from the 1998 DARPA Intrusion Detection Evaluation data sets. The results of applying agglomerative, hierarchical and k -means clustering methods to the data sets are presented. Visualization of the clustering results provides a simple, yet powerful, means for interpreting the data.

Keywords: Intrusion Detection, S Language, Clustering Methods, Visualization

1. Introduction

The field of intrusion detection has been the focus of much research, resulting in several successful implementations of Intrusion Detection Systems (IDS). Each of the implementations generally uses its own set of homegrown software tools, scripts and programs in order to construct and validate the new IDS concepts and methods. The steps involved in developing and testing new IDS include data collection, pre-processing, algorithm development, data storage and visualization. In addition, the research and development process typically spans multiple disciplines including statistics, artificial intelligence (AI), mathematics and visualization.

However, there is a lack of a *unified framework* for developing and testing these systems that result in difficulties in comparing the results. Moreover, the software components developed for one system are not reusable by another system that is under development due to the lack of a common framework for reusability.

The S Language, developed at Bell Laboratories, provides a flexible and powerful environment for statistical and graphical analysis of data. It provides the tool set to implement many statistical ideas made possible by the widespread availability of workstations having advanced graphics and number crunching capabilities. The tools available in S allow for the implementation of many known algorithms from statistics, AI and mathematics with advanced visualizations and graphics. Implementations of S are available commercially in S-Plus [1] and as the free open-source R [2] software packages for a wide range of computer systems. This integrated suite of software facilities for data analysis and graphical display allows for the creation of an extensive and coherent collection of tools. This object-oriented programming language allows the user community to effectively extend the capabilities of their models [3].

This study introduces the S Language as a viable tool for studying, developing, testing and comparing the results of various techniques for the implementation of intrusion detection systems. A sample anomaly-based detection scheme for detecting selected Denial-Of-Service (DoS) and Network Probe attacks from the 1998 DARPA Intrusion Detection Evaluation data sets [4] is presented in detail. Results from three clustering methods namely, agglomerative, k -means and hierarchical clustering are

compared within a unified framework. Visualization of the clustering results is made possible through the advanced graphical capabilities of S.

2. Introduction to Intrusion Detection

With the widespread use of computer networks and telecommunication devices, network security has become a primary concern for the developers and users of these networks.

Currently, two approaches have gained some degree of popularity. The first approach, signature-based intrusion detection, attempts to create a database of known intrusion signatures. The primary disadvantage of this model is its inherent inability to detect new attacks or known ones that have significantly changed their behavior or signature.

The second approach, anomaly detection, attempts to establish what the normal traffic patterns look like for a given network, and flags out any variations of traffic from this norm. Unlike signature-based detectors, anomaly detectors do not compare the traffic against any signature database, but rather attempt to identify anomalies in the traffic that can suggest a possibility of an attack or intrusion that is taking place. The disadvantages of this model are high false alarm rates and the lack of ability to easily cope with normal changes in network activity. In addition, anomaly detectors can flag out abnormal behavior, but may not be able to specify the exact type of attack or its nature.

Common to these approaches is a set of tasks that are performed in order to achieve the desired goal of detecting intrusions. These tasks can be summarized as follows:

- ❖ Data collection and processing: For example, sniffing data off the network and processing it to extract the desired portions of packet data for network-based IDS or analysis of system logs in case of host-based IDS.
- ❖ Application of detection algorithm: The desired detection algorithm or method is applied to the data previously collected.
- ❖ Evaluation of results: By generating reports and using advanced visualization to assess the results obtained.

In practice, each of these tasks may be implemented using one or more programs. Many of the current projects evaluating new IDS concepts use a variety of different programs ranging from scripts and compiled executables to third-party tools and maybe certain portions of code from an older project. All of the above tasks can be achieved within a single framework using S.

3. Related Work

Much of the focus of this study is on evaluating the use of clustering methods as applied to anomaly detection, which are widely used in the field of computer security [5]. Various techniques for modeling normal and anomalous data have been developed [6].

Portony [7] presents a method for clustering similar data instances together and uses distance metrics on clusters to determine an anomaly. The author makes two basic assumptions: First, data instances having the same classification should be close to each other in feature space under some reasonable metric, while instances with different classifications should be far apart. Second, the number of instances in the training set that represent normal traffic is overwhelmingly larger than the number of intrusion instances. Clusters were labeled automatically, and were later used to classify unseen network data instances. Both training and testing was done using subsets of KDD CUP 99 data [8]. On average, the detection rate was around 40%-55% with a 1.3%-2.3% false positive rate.

In the above cited works a heterogeneous set of tools and software packages were used to develop and test each method leading to difficulties in comparing the results obtained and accurately assessing the method's performance. Different tools generate different output formats, reports and graphics making it hard to compare their results. In addition, the preprocessing phase of data using different programs and techniques can lead to variable performance numbers amongst the different implementations which makes the process of evaluating comparative performance difficult. Using S would create a unified

framework for evaluating the results and associated performance of each method. In addition, reusability of software components can become a much easier task when using a single framework.

4. The S Language and Environment

An S environment is an integrated suite of software facilities for data analysis and graphical display. One of the strengths of S is that functions implementing new statistical methods can be developed on top of the low-level facilities.

For example, to create a single function to perform the three basic tasks of an IDS, as described in section 2, the S code would look like:

```
evalIDS ← function( indata ) {  
  pd ← procData( indata );  
  intrusion ← detectIntrusion( pd );  
  evalResult ← evalResult( intrusion );  
}
```

The top-level function `evalIDS` accepts one argument, `indata`, and calls three functions namely `procData`, `detectIntrusion` and `evalResult` representing the three basic tasks. Each of the three functions in turn calls other lower level functions to implement their details. Specific examples of these functions will be discussed in details in section 8.

Using S, it is quite easy to change or experiment with design decisions made by the original implementers in order to explore new methods. For example, an existing library function can be using linear interpolation model of values. This behavior could be changed to reflect a non-linear model by re-writing the function or modifying its sources. In the previous example the function `detectIntrusion` could be made as a library function with some default algorithm to detect intrusions. This default behavior could easily be modified, by modifying the library function source, to implement variations of the default algorithm or a completely new algorithm while maintaining the same structure of the rest of the program. This flexibility is even more evident in the open-source R implementation where all the details of implementation are open for exploration.

The commercial implementation of S, called S-Plus, has an extensive Graphical User Interface (GUI), which provides menus and dialogues for many simple statistical and graphical operations. A full-featured student edition is available at no cost for students at accredited universities. This student edition was used to report the results of this study. The open-source R package can be downloaded directly from the project web site and is installable on many platforms including Windows and Linux. Almost all S scripts developed for S-Plus will run on R and vice versa. The main difference between S-Plus and R is that S-Plus, including the student edition, has a sophisticated GUI that is especially helpful for new users.

Both S-Plus, including the student edition, and open-source R implementations provide a command-line interface for entering S commands. Once the program is started, this command line allows the user to enter S commands, create variables, call functions, draw graphs, create and manipulate data tables, and save and print results. Since S is also a full programming language in its own right, it provides for assignment statements, control structures, arithmetical expressions, array and matrix operations and calling conventions for functions, amongst other capabilities.

Some of the language features found in S are found in other scientific analysis languages like Matlab and Mathematica, but S provides for some key features that make it more applicable for use in intrusion detection research. First, S is designed to be a statistical analysis tool and thus many of the specialized statistical functions are already available in its libraries and need not be written from scratch. These functions are typically the core functions that are used to develop intrusion detection engines. On the other hand, Matlab is essentially a numerical simulation tool that is designed to do linear algebra, computation, and simulation.

Second, S's ability to run on many platforms makes it ideal for use in intrusion detection research where the hosts under study are running a variety of operating systems and hardware. For example, R is available for over 14 different processor architectures and operating systems including the most

common ones such as Linux, Windows, MAC OS and many Unix flavors. This is a key feature for distributed intrusion detection systems where detection *sensor* devices need to be installed on a network of hosts with different processor architectures that run different operating systems. In this case, each sensor binary executables (an R installation) can be built from sources according to the processor architecture used at each host. On the other hand Matlab is available for five different platforms, and only in binary executables form.

Third, S has a powerful object-oriented language structure that can implement quite complex algorithms and their variants. Finally, S-Plus student edition is free for use by students and the open-source R is free to everyone. Open-source software packages have proved to be effective for use by research institutions that cannot afford costly software licenses.

Several speed and feature comparisons are available which compare S-Plus and R to other data analysis packages including Matlab, being one of the most common ones. A detailed comparison of the features available in these packages including a comparison of mathematical functionality, graphical functionality, programming environment functionality, data handling, available operating systems and speed comparison can be found in [9].

Using S for intrusion detection enables a researcher to study and compare the results of several detection methods using a single tool. Since S runs from a command-line interface, the processes of data collection, preprocessing, conversion to S objects (such as arrays and matrices), manipulation of data using a method of choice, generating the required statistics and plotting the results can all be done using a single tool.

5. Clustering Methods

Cluster analysis is concerned with discovering groupings among the cases of an n by p matrix, where n is the number of observations and p is the number of variables in each observation. A comprehensive general reference can be found in [10].

Generally speaking, clustering algorithms fall into two categories [11]:

(a) **Partitioning Algorithms:** A partitioning algorithm describes a method that divides the data set into k clusters, where the integer k needs to be specified. Typically, the algorithm is run for a range of k -values. For each k , the algorithm carries out the clustering and also yields a quality index, which allows the selection of the best value of k afterwards. The S-Plus functions `kmeans`, `pam`, `clara`, and `fanny` implement algorithms of this type.

(b) **Hierarchical Algorithms:** A hierarchical algorithm describes a method yielding an entire hierarchy of clustering for the given data set. There are two types of hierarchical methods: *agglomerative* and *divisive*. *Agglomerative* methods start with the situation where each object in the data set forms its own little cluster, and then successively merges clusters until only one large cluster remains - which is the whole data set. The S-Plus functions `agnes`, `mclust`, and `hclust` implement agglomerative methods. *Divisive* methods start by considering the whole data set as one cluster, and then splits up clusters until each object is separate. Algorithms of this type are used in the S-Plus functions `diana` and `mona`. The seven functions `daisy`, `pam`, `clara`, `fanny`, `agnes`, `diana`, and `mona` make up the cluster library. Algorithms to implement these functions are described in [12].

5.1. Partitioning Methods

Partitioning methods are based on specifying an initial number of groups, and iteratively reallocating observations among groups until some equilibrium is attained.

5.1.1. *k*-means Clustering

One of the best-known partitioning methods is the k -means. In the k -means algorithm the observations are classified as belonging to one of k groups. Group membership is determined by calculating the *centroid* for each group (the multidimensional version of the mean) and assigning each observation to the group with the closest centroid.

The k -means clustering algorithm chooses a pre-specified number of cluster centers to minimize the within-class sum of squares of the vectors for those centers. Since the algorithm needs a starting point, it chooses the mean of the clusters identified by group-average clustering. The k -means clustering algorithm needs access to the data matrix and uses Euclidean distance.

The S-Plus function `kmeans` performs k -means clustering. The main arguments to `kmeans` are dissimilarities as produced by `daisy` or `dist` and the number of clusters. Alternatively, a matrix of starting centroids may be specified in place of the number of centroids. If starting values are not specified, the initial centroids are obtained using the hierarchical clustering algorithm in `hclust`.

5.2. Hierarchical Methods

The partitioning algorithms discussed in the previous section are based on specifying an initial number of groups, and iteratively reallocating observations between groups until some equilibrium is attained. In contrast, hierarchical algorithms proceed by combining or dividing existing groups, producing a hierarchical structure displaying the order in which groups are merged or divided.

5.2.1. Agglomerative Clustering

Agglomerative methods start with each observation in a separate group, and proceed until all observations are in a single group. S-Plus has three functions for agglomerative hierarchical clustering: `hclust`, `mclust`, and `agnes`. Compared to other agglomerative clustering methods such as `hclust`, `agnes` has the following features: (a) it yields the agglomerative coefficient which measures the amount of clustering structure found; (b) apart from the usual clustering tree, it also utilizes the banner plot.

5.2.2. Divisive Clustering

While agglomerative clustering starts with many groups and combines them to form one group, divisive analysis starts with one group and repeatedly divides groups to form many groups.

The S-Plus function `diana` implementation, of a divisive hierarchical method, is probably unique in computing a divisive hierarchy, because most other software for hierarchical clustering is agglomerative. Moreover, `diana` provides (a) the divisive coefficient, which measures the amount of “clustering structure”, and (b) the banner plot.

In `diana`, the initial clustering (at step 0) consists of one large cluster containing all n objects. In each subsequent step, the largest available cluster is split into two smaller clusters, until finally all clusters contain but a single object.

6. Denial-of-Service and Network Probe Attacks

In a DoS attack, the attacker makes some computing or memory resource too busy, or too full, to handle legitimate users' requests. But before an attacker launches an attack on a given site, the attacker typically probes the victim's network or host by searching these networks and hosts for open ports. This is done using a sweeping process across the different hosts on a network and within a single host for services that are up by probing the open ports. This process is referred to as *Probe Attacks*.

Table 1 : Description of DoS and Probe Attacks

Attack Name	Attack Description
Smurf (DoS)	Denial of Service ICMP echo reply flood
Neptune (DoS)	SYN flood Denial of Service on one or more ports
IPsweep (Probe)	Surveillance sweep performing either a port sweep or ping on multiple host addresses
Portsweep (Probe)	Surveillance sweep through many ports to determine which services are supported on a single host

Table 1 summarizes the types of attacks used in this study. The attacks are described in more detail below.

Smurf attacks, also known as directed broadcast attacks, are a popular form of DoS packet floods. Smurf attacks rely on directed broadcast to create a flood of traffic. The attacker sends a ping packet to the broadcast address for some network on the Internet that will accept and respond to directed broadcast messages, known as the Smurf amplifier. These are typically mis-configured hosts that allow the translation of broadcast IP addresses to broadcast Medium Access Control (MAC) addresses. The attacker uses a spoofed source address of the victim. For Example, if there are 30 hosts connected to the Smurf amplifier, the attacker can cause 30 packets to be sent to the victim by sending a single packet to the Smurf amplifier [13].

Neptune attacks can make memory resources too full for a victim by sending a TCP packet requesting to initiate a TCP session. This packet is part of a three-way handshake that is needed to establish a TCP connection between two hosts. The SYN flag on this packet is set to indicate that a new connection is to be established. This packet includes a spoofed source address, such that the victim is not able to finish the handshake but had allocated an amount of system memory for this connection. After sending many of these packets, the victim eventually runs out of memory resources.

IPsweep and Portsweep, as their names suggest, sweep through IP addresses and port numbers for a victim network and host respectively looking for open ports that could potentially be used later in an attack.

7. Data Collection and Preprocessing

7.1. Data Collection

The 1998 DARPA Intrusion Detection data sets were used as the source of all traffic patterns in this study. The training data set includes traffic collected over a period of seven weeks and contains traces of many types of network attacks as well as normal network traffic.

This data set has been widely used in the research in intrusion detection, and has been used in comparative evaluation of many IDS'. Attack traces were identified using the time stamps published on the DARPA project web site.

7.2. Data Preprocessing

Data sets were preprocessed by extracting the IP packet header information to create the feature vectors. The resulting feature vectors were used as input vectors to be processed by the clustering algorithms. The feature vector chosen has the following format:

SIP _x	SPort	DIP _x	DPort	Prot	PLen
------------------	-------	------------------	-------	------	------

Where

- SIP_x = Source IP address nibble, where $x = [1-4]$. Four nibbles constitute the full source IP address
- SPort = Source Port number
- DIP_x = Destination IP address nibble, where $x = [1-4]$. Four nibbles constitute the full destination IP address
- DPort = Destination Port number
- Prot = Protocol type: TCP, UDP or ICMP
- PLen = Packet length in bytes

This format represents the IP packet header information. Each feature vector has 12 components corresponding to dimension p in the original data vector x . The IP source and destination addresses are broken down to their network and host addresses to enable the analysis of all types of network addresses.

Four data sets were created, each containing $n = 600$ feature vectors of the form described above. Each data set represents one of the attack types shown in Table 1, mixed with normal traffic. For each of the four data sets, the first 300 feature vectors represent the normal traffic packets that preceded the attack

packets. The following 300 feature vectors represent packets of one of the attack types. It is desired to see if the clustering algorithms can clearly distinguish between the first and second 300 feature vectors in each data set. When creating the data sets, it is assumed that normal traffic is flowing on the network and an attack would start sometime afterwards, which is a typical scenario of how attacks take place.

8. Results

The three clustering methods described in section 5 were applied to the data sets described in section 6. The objective is to evaluate the ability of each clustering method to cluster the first 300 feature vectors (containing normal traffic) into one set of clusters and the next 300 feature vectors (containing attack traffic) into a different cluster. If the method can isolate all attack feature vectors into one or more clusters consistently, we then compare its performance to other methods in terms of detection rates and computational performance. The goal is to do all these steps within S-Plus.

Using S-Plus the three common tasks discussed in section 2 are performed as follows:

8.1. Data collection and processing for the Smurf data set

The following code snippet shows how the feature vector data sets are loaded into the S-Plus student edition software, and how the different calls to the clustering methods are made. The same code could easily be used in R as well.

For example, the Smurf data set containing a mixture of normal traffic and Smurf attack packets is loaded into S as follows:

```
data.restore("C:\\smurfregular600.zeros.sdd" )
guiOpenView("data.frame",Name="smurfregular600.zeros")
```

This creates a *data frame*, called *smurfregular600.zeros*, which is a data structure in memory containing the 600 feature vectors read from the file named "smurfregular600.zeros.sdd".

8.2. Application of detection algorithm

(a) Agglomerative clustering is applied to the data set with the following parameters:

```
MenuAgnes(data = smurfregular600.zeros, variables = " SrcIP1 ,SrcIP2 ,SrcIP3 ,SrcIP4 ,SrcPort
,DstIP1 ,DstIP2 ,DstIP3 ,DstIP4 ,DstPort ,Protocol ,PacketLength ", na.rm = T, diss = F, metric =
"euclidean", stand = F, method = "average", save.x = T, save.diss = T, print.type = "Short",
save.cluster.p = T, save.num.groups = 4, plot.p = F, pltree.p = T)
```

The S function *agnes* performs agglomerative clustering as discussed in section 5.2.1. In S-Plus this function can be used from a drop-down menu and is called *MenuAgnes*. When using command-line mode (without the GUI) in S-Plus or R, the function can simply be called *agnes*.

The variables passed to *MenuAgnes* include all 12 features of the feature vector discussed in section 7. The remaining arguments to the function call include specifying Euclidean metric to be used as a distance metric (another option is to use Manhattan metric), setting the number of clusters to 4 and plotting a dendrogram, amongst other parameters. A full description of the function of each parameter in the argument list can be found in the S-Plus documentation.

Figure 1 shows the dendrogram created by the *MenuAgnes* function call. A *dendrogram* is a convenient method used to visualize the clustering results. It is a tree graph that is used to examine how clusters are formed in hierarchical cluster analysis. The vertical axis indicates a distance or dissimilarity measure. The height of a node represents the distance of the two clusters that the node joins. The larger the height, the more dissimilar the two clusters are. The horizontal axis lists all the 600 observations and their cluster assignments. Dendrograms have two limitations: First, because each observation must be displayed as a leaf they can only be used for a small number of observations. This is clear in this figure where the text of the observations on the horizontal axis is not readable. Second, the vertical axis represents the level of the criterion at which any two clusters can be joined. Successive joining of

clusters implies a hierarchical structure, meaning that dendrograms are only suitable for hierarchical cluster analysis [14].

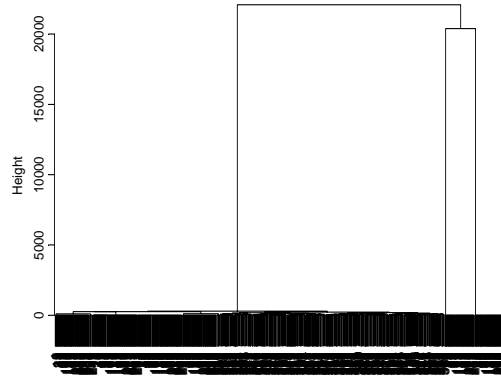


Figure 1: Dendrogram resulting from applying Agglomerative clustering to Smurf data set

8.3. Evaluation of results

(b) A plotting function is called to plot the assignment of each input vector to an output cluster. The number of output clusters are set to 4 and are given by the `save.num.groups = 4` above:

```
guiPlot( PlotType = "Line", DataSet = "DS41", Columns = "cluster.id")
```

The S function plot is a generic plotting function and can take several arguments. In S-Plus, this function is called `guiPlot` and can be used from a drop-down menu in case the GUI is used. The command-line counterpart is simply called `plot`. The function `guiPlot` is called with arguments to draw a line plot using column `cluster.id` from the DS41 data set. The DS41 data set was automatically generated by the previous call to `MenuAgnes` and contains the allocation of each input feature vector, of the 600 vectors, to the output cluster number, out of four clusters, that the vector was assigned to.

The resulting plot of executing `guiPlot` after applying agglomerative clustering to the Smurf data set is shown in Figure 2 (top). It is clearly seen that the first 300 feature vectors representing normal traffic for this data set were assigned to all clusters 1, 2, 3, and 4, while the remaining 300 feature vectors representing the Smurf attack were assigned exclusively to cluster 1.

It is clear from Figure 1 that there are two very dissimilar cluster sets representing normal and Smurf attack traffic.

(c) Similarly, *k*-means clustering is applied to the data set as follows:

```
menuKmeans(data = smurfregular600.zeros, variables = "SrcIP1, SrcIP2, SrcIP3, SrcIP4, SrcPort, DstIP1, DstIP2, DstIP3, DstIP4, DstPort, Protocol, PacketLength", na.rm = T, centers = 4, iter = 10, save.cluster.p = T)
```

The function `menuKmeans` and its command-line counterpart `kmeans` performs *k*-means clustering using all the 12 features in the input feature vectors. The function arguments specify 4 cluster centers to be used and to use a maximum of 10 iterations of the algorithm. The resulting plot showing the assignments of the input feature vectors to the output cluster numbers is shown in Figure 2 (middle).

(d) Next, hierarchical clustering is applied to the same data set as follows:

```
menuDiana(data = smurfregular600.zeros, variables = "SrcIP1, SrcIP2, SrcIP3, SrcIP4, SrcPort, DstIP1, DstIP2, DstIP3, DstIP4, DstPort, Protocol, PacketLength", na.rm = T, diss = F, metric = "euclidean",
```



```
stand = F, save.x = T, save.diss = T, print.type = "Short", save.cluster.p = T, save.num.groups = 4,  
plot.p = F, pltree.p = T)
```

The S-Plus function `menuDiana` with its command-line counterpart `diana` implements hierarchical clustering with Euclidean metric as the distance metric, creates a dendrogram and uses 4 output clusters to cluster its input. The resulting plot is shown in Figure 2 (bottom).

An easy visual comparison of the results of applying the three clustering methods for the Smurf data set is demonstrated by comparing the top left, top right and bottom sections of Figure 2.

8.4. Data collection and processing for the Neptune, IPSweep and Portsweep data sets

The same steps are applied, in an identical manner, to the remaining data sets for Neptune, IPSweep and Portsweep and the results plotted as before. The procedure now is a routine, thus illustrating the power of the framework created by S Language and its environment.

It should be noted that the previous steps described for loading the data sets into S-Plus (or R), performing the required clustering operation, plotting, and further analyzing the results can all be executed by writing a small script. This script can be run in batch mode using either S-Plus or R. Indeed most advanced users of S would prefer using batch mode rather than using the GUI since it is easier to change function parameters and program flow using a text file and rerunning the whole script at once.

Comparing the results of applying the three algorithms to the four data sets reveals many similarities in their clustering approaches. For the Smurf data set, hierarchical clustering (bottom of Figure 2) performed best in terms of separating normal from attack traffic. All normal traffic (first 300 packets) was assigned to clusters 2, 3, and 4. The y-axis in the figure shows the cluster number that is assigned to each input vector. Smurf attack traffic was exclusively assigned to cluster 1. Hence, a clear-cut separation between the two types of traffic is achieved. This is not the case with *k*-means and agglomerative clustering. Using *k*-means and agglomerative clustering all Smurf Attack packets were clustered into clusters 1 and 2 respectively, but some packets from normal traffic were also clustered into clusters 1 and 2 respectively leading to possibilities of false alarms and lower detection rates. Hierarchical clustering on the other hand produced no false alarms and maximum detection rate. From a computational complexity point of view, however, hierarchical clustering algorithm is $O(n^2)$. *k*-means on the other hand is only of order $O(kn)$, where *k* is the number of clusters and *n* is the number of observations. For the remaining data sets, very similar results were achieved by applying the different clustering methods.

The choice of the number of output clusters can affect the clustering results and can provide for better basis for selecting one method over the other. The choice of 4 output clusters for this study was rather arbitrary to simplify the objective of demonstrating the usage of the S language. For a complete study of the best value to use as the number of output clusters, several experiments must be done by varying the number of output clusters and constructing a Receiver Operating Characteristic (ROC) curve to determine the best value to use for the number of output clusters. The ROC curve depicts the relationship between false positive and detection rate for a fixed training/test set combination. It is commonly used to visualize the trade-off between detection and false positive rates.

In this study, both training and testing was done on the same data subset. This is justified by the fact that no information about the data's classification was used in the training phase. This is typical when performing clustering of unlabeled data. Therefore, using the training set for testing is basically the same as using a different data set.

9. Conclusion and Future Work

This study introduces S language and environment as a viable platform for developing and testing various algorithms and methods for intrusion detection. Future work will attempt to automate the usage of S by using scripts and command-line mode to process the entire DARPA data set. This would be done by breaking the data set into smaller data sets suitable for individual processing, aggregating the

results and constructing ROC curves to compare the performance of the three clustering methods and find the optimal number of output clusters to be used.

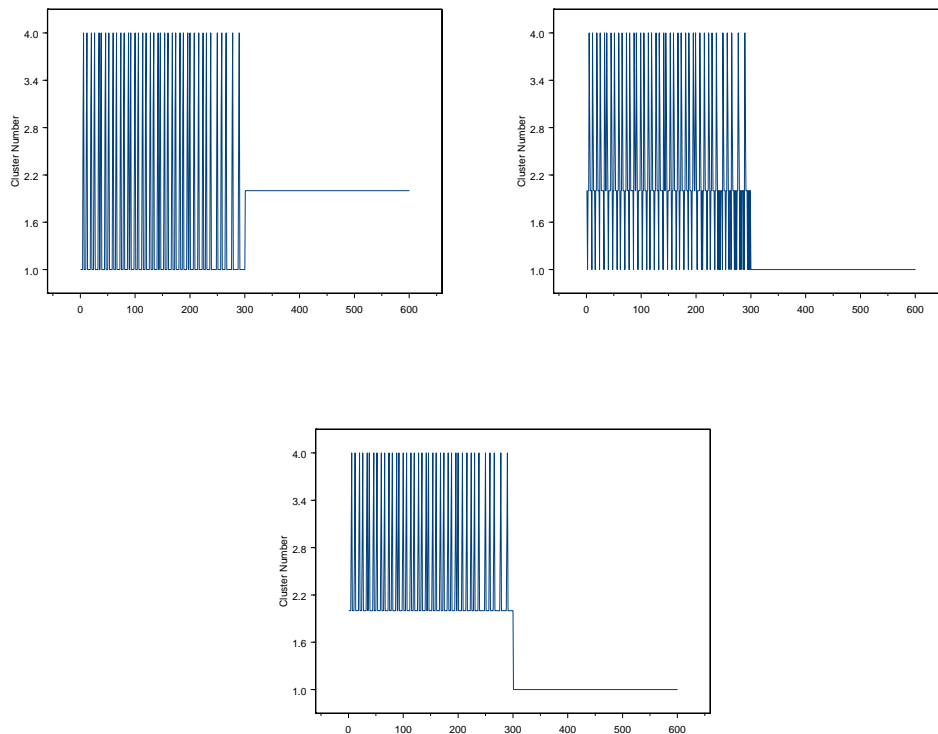


Figure 2: Clustering of Smurf Attack: Agglomerative (Top Left), k -means (Top Right) and Hierarchical (Bottom)

10. References

- [1] S-Plus, Insightful Corporation: <http://www.insightful.com/>
- [2] The R Project: <http://www.r-project.org>
- [3] Venables W. N., Ripley B.D., “*Modern Applied Statistics with S*”. Fourth Edition, Springer-Verlag, 2002
- [4] DARPA Intrusion Detection Evaluation Project: <http://www.ll.mit.edu/IST/ideval/>
- [5] Denning D.E., “An Intrusion Detection Model”. *IEEE Transactions on Software Engineering* SE-13: 222-232, 1987.
- [6] Warrender C, Forest S., Pearlmutter B. “Detecting Intrusions Using System Calls”, *Alternate Data Models*, 1999.
- [7] Portony L., Eskin E., Stolfo J. “Intrusion Detection with Unlabeled Data Using Clustering”, *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*. Philadelphia, PA: November 5-8, 2001
- [8] Knowledge Discovery and Data Mining competition, KDD99 CUP Data Set, 1999. <http://kdd.ics.uci.edu>
- [9] <http://www.scientificweb.com/ncrunch/ncrunch4.pdf>
- [10] Gordon, A.D., “*Classification*”, Second Edition, 1999, London: Chapman & Hall
- [11] *S-Plus: Guide to Statistics*, Volume 2. Insightful Corporation, 2001
- [12] Kaufman, L. and Rousseeuw, P.J., “*Finding Groups in Data: An Introduction to Cluster Analysis*” New York: 1990 John Wiley & Sons, Inc.
- [13] Skoudis E., “*Counter Hack: A Step-by-Step Guide to Computer Attacks and Effective Defenses*”. Prentice Hall Inc., 2002
- [14] Schonlau M., “The Clustergram: A Graph for Visualizing Hierarchical and non-Hierarchical Cluster Analyses”. *The Stata Journal*, 2002, 3, pp 316-32