Modern Applied Science; Vol. 10, No. 9; 2016
ISSN 1913-1844 E-ISSN 1913-1852
Published by Canadian Center of Science and Education

Presenting a New Method to Classify Alerts Received from Intrusion Detection Systems

Farshid Pourabbas[1] & Adem Karahoca[1]

[1] Computer Engineering Department, Bahcesehir University, Istanbul, Turkey

Correspondence: Adem Karahoca, Computer Engineering Department, Bahcesehir University, Istanbul, Turkey. E-mail: akarahoca@gmail.com/farshidforever@gmail.com

Received: July 27, 2016 Accepted: August 18, 2016 Online Published: August 30, 2016

doi:10.5539/mas.v10n9p191 URL: http://dx.doi.org/10.5539/mas.v10n9p191

Abstract

With the growth of the internet networks today, security of data exchange is considered as an important task. Therefore, the use of security tools is increasing day by day. Intrusion detection systems are among these tools. They are only able to labela message received from a network as‘alert’,but they are unable to describe system status. Some methods have been developed to solve the above problem through correlating the alerts received from intrusion detection systems. By correlating the interrelated alerts, the methods would be able to describe system status. One of the steps of correlation methods of alerts is to classify them. System status can be described better when classification is performed efficiently. Here, we present a method for classifying alerts.

Keywords: intrusion detection systems, alerts, classification, correlation

1. Introduction

Today, with the numerous attacks and sabotages occurringover networks and threatening performance of many customers and its users,security centersattempted to look for solutions tomaintain security over the network more than ever. Various security tools, such as firewalls, intrusion detection systems, etc. are used to improve security level on a network (Kruegel, Valeur & Vigna, 2005).

One of the major problems of intrusion detectionsystems is issuance of many alerts with low-levelabstraction. To solve this problem, we need to have some methods to issue alerts with higher abstraction level while reducing alerts and removing wrong alerts (Kruegel, Valeur & Vigna, 2005).

With respect to the verylarge volume of data passing over the network, importance and confidentiality of the data, necessity to maintain security and protect users’ data in today’s world, there is a pressing need to have a security system to be able to manage network and protect system against possible damages.

In typical systems, the tools such as firewalls, antivirus software, and intrusion detection systems attempt to protect a network and defend against possible attacks. These tools are suitable solutions to reduce the impacts of computer attacks; however, they cannot be considered as an inclusive approach to protect and prevent network form possible damages. One of the tools that gained attentions recently is intrusion detection systems. They are able to detect and issue analert. Two problems concerning intrusion detection systems are 1- A large number ofreceivedalerts, 2- Wrongalerts (Wang, Liu & Jajodia, 2006).

With respect to the above items and the fact that the issuedalerts have low abstraction levels, as a network manager would have no understanding of system status, we need a system to enable us to detect relationships between these alerts. This is realized by collecting alerts from intrusion detection systems and providing network manager with a high-level vision for the attacks taken placeover thenetwork.

‘Correlation of alerts’ means establishing a relationship betweensomealerts and promoting them to higher-level alerts whose management is more convenient for network manager.

As data mining means extraction of useful information out of a large volume of data, it can be used as a method for correlatingalerts. Some of the models proposed for correlatingalerts enjoy data mining techniques; however, all these methods have some drawbacks that make us keep looking for some efficient methods with low computational and memory overhead.

The rest of the article discusses the following items: Section two reviews literature, the proposed method is

explained in section three, conclusions are brought about in section four, and section five discusses the results.

2. Literature

In reference (Zhu & Ghorbani, 2006), correlation probability between two alerts is calculated based on similarity of features of source IP, destination IP, destination port, type of alert, and timestamps using Multi-Layer Perceptron(MLP) neural network and Support Vector Machines (SVM). In this method, when a new alert is received, anultra-alert that includes an alert with maximum correlation probability with the new alert is specified using MLP and SVM. If the detected correlation probability was less than correlation threshold, the new alert is not correlated with any alerts. If the calculated probability exceeded threshold, correlation probability of the new alert is calculated with all the available alerts in the detected ultra-alert.After that, the alert is correlated with the new alert whose difference of probability with the highest probability detected earlier is less than criterion of correlation sensitivity. If there is no alert for correlation, a new alert is placed in a new ultra-alert.

Following figure shows a framework presented for alert correlation in (Sadoddin & Ghorbani, 2009). Unprocessed alerts are received continuously by integration unit. This unit correlates alerts to graph structures based on their connection information with respect to the source and destination of the alerts. Each structural pattern may show attack strategies or maybe the normal pattern created due to positive false alerts. The created patterns may change dynamically as long as they become fixed. The fixed structural patterns are transferred to the next unit to create a set of transactions for the following processes.

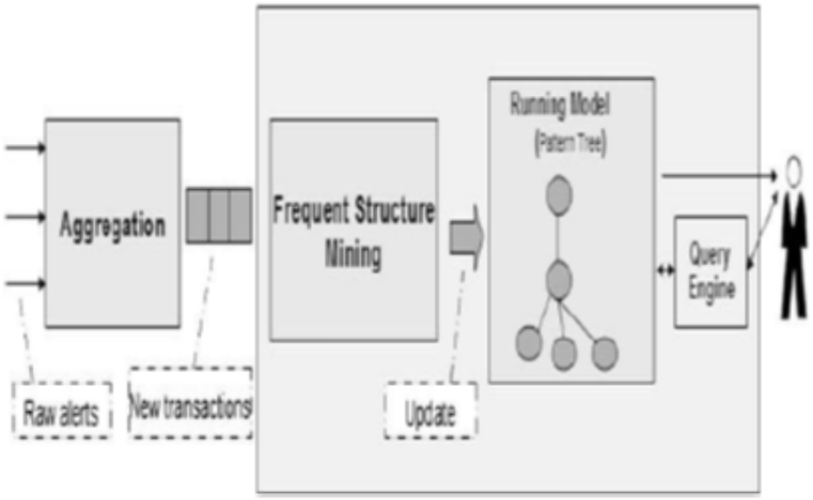

Figure 1. A framework for correlation between alerts (Sadoddin & Ghorbani, 2009)

In this method, features of source IPs, destination IPs, attack classes,and timestamps are used for different alerts. Feature of portis not used in this method asfrequent patterns are shown by data graph structures, which are nodes of network hosts and edges of the alerts issued between hosts. On the other hand, a port is not an unreliable feature source (as each intruder can easily change his/her port) and value of destination port in most attacks is not important.

In the method presented for creating candidate frequent patterns, transactions are created based on the connection information of corresponding alerts. Here, one method is presented for exploringfrequent patterns incrementally and maintaining them in the reduced data structure (FP-tree).

FP-Growth algorithm was used for exploring sequential structures. FP-Growth algorithm uses FP_Tree, which is a compressed data structure for storing frequent candidate patterns

A concept called ‘source’ was used in (Xu & Ning, 2005) to show prerequisite and consequence of an attack. A ‘source’ can be a port, a service, etc. Prerequisite of an attack, input source, and its consequences is called ‘output source’. In this method, the causal relationships between resources were prepared in the form of rules and they are used to create correlations between alerts.

Two alerts are correlated when the output source of either of them include one of the input sources of the other and/or lead to them.

Minor compliance was used in this article. That is, if the result of an alert meets at least one of the prerequisites

of another alert (regarding time relationship), those alerts will be correlated.

3. Proposed Method

Correlation of alerts has several steps as follows. First, alerts are classified after preprocessing. Then an attack scenario is created using the available alerts in a group. An attack scenario is strongly dependent on the earlier knowledge and classification quality.Earlier knowledge is meant the knowledge collected from professionals that can help to create an attack graph (that expresses attack scenario). The richer and more accurate the knowledge is, the presented scenarios will be better. Therefore, we intend to focus on a part to be able to solve the problem using data mining techniques. As a result, we will concentrate on how to classify and correlate alerts.

Our proposed method encompasses the following steps:

1. For all the receivedalerts, we do the following steps.
2. Classification of alerts using the fuzzy method explained below.

First, we calculate output for a pair of alerts using MLP neural network as a correlation engine. We teach the above neural network using training samples.

If the relevant output were bigger than the predefined threshold, we would go through step 3; otherwise, we create a new ultra-alert and put above alert in it.

We connect the received alert to all the available ultra-alertsand we use the second output power of the correlation engine as membership degree of the alert to the present ultra-alerts.

A. Using Neural Network as a Correlation Engine

As explained in the method of Zhu and Ghorbani, a multi-layer neural network can be used as a correlation engine. First, we teach the neural network using the following training samples.The features we used here are:

1- Source IP address, 2- Destination IP address, 3- Destination port number, and 4- To examine if destination IP address of the earlier alert is identical with the source IP address of the current alert

After teaching the above network, it is used as follows. Here, we compare the features extracted from the received alerts and the ending alerts in infra-alerts and give their values to the correlation engine. The network output shows correlation probability of the two alerts. If this value exceeded the predefined threshold (We assumed threshold value equal to 0.5.), we connect it to the ending alert in the above ultra-alert. In this method, one alert may appear in several ultra-alerts. We use output of correlation engine as membership degree of an alert to the relevant ultra-alert.

B. Using Fuzzy Classification to Establish Relationship between Alerts

When output of correlation engine exceeds threshold value for the received alert and final alert in anultra-alert, we put the alert in that ultra-alert and use output of the correlation engine as membership degree of that alert to the ultra-alert.

After examining all alerts, we will have several ultra-alerts that may have common alerts (but with different membership degree).

4. Experiments

It can be proved that this method leads to a better categorization. To do so, we assume that we received alert α_1. Probability of correlation of this alert with the two alerts, which are within two different ultra-alerts, close to one another and it exceeds the threshold (0.5) we defined - for instance, probability of 0.6 for its correlation with the alert in the first ultra-alert and 0.56 for its correlation with the alert in the second ultra-alert. As noticed, such difference is negligible. According to other classification methods, assume that we put this alert in the first ultra-alert, while, in fact, it is related to the second ultra-alert. It is due to the fact that correlation engine is unable to show their correlationfavorably. This might be due to the accuracy of a learning machine (Learning machine's accuracy cannot be hundred percent.) and/or due to lack of appropriate training. Therefore, by losing this alert in the second ultra-alert, we may not be able to extract attack scenario. (Assume a condition in which such mode is repeated several times.)

Now, assuming that we can have this alert in both ultra-alerts, we will be able to compensate defect of attack scenario by having the pertinent alert. We can consider constructingan attack scenario in a way to ignore construction algorithm of their scenario as soon as we observe the irrelevant alerts. It means that placing an alert in such ultra-alert cannot lead to confusion about attack scenario.

We implement all algorithms using MATLABsoftware.

All algorithms are accessible in the "Appendices" part.

We tested our algorithm on 30 sample alerts out of all the alerts of "DARPA 2000" dataset and the result was as follows:

Table 1. Information Exports from DARPA 2000 Dataset

Destination	Protocol	Length	Info
Falcon.eyrie.af.mil	TELNET	60	Telnet Data…
Delta.peach.mil	TCP	60	Telnet Data
Falcon.eyrie.af.mil	TELNET	60	63281>telnet [ACK] Seq=2 Ack=2 Win=33580 Len=0
Falcon.eyrie.af.mil	TCP	60	Telnet Data…
Delta.peach.mil	TCP	60	Telnet Data…
Falcon.eyrie.af.mil	TELNET	60	63281> telnet [ACK] Seq=4 Ack=4 Win=33580 Len=0
Falcon.eyrie.af.mil	TELNET	60	Telnet Data…
Delta.peach.mil	TELNET	60	Telnet Data…
Falcon.eyrie.af.mil	TCP	60	63281> telnet [ACK] Seq=5 Ack=5 Win=33580 Len=0
Falcon.eyrie.af.mil	TELNET	60	Telnet Data…
Delta.peach.mil	TELNET	60	Telnet Data…
Falcon.eyrie.af.mil	TCP	60	63281> telnet [ACK] Seq=6 Avk=6 Win=33580 Len=0
Falcon.eyrie.af.mil	TELNET	60	Telnet Data…
Delta.peach.mil	TELNET	60	Telnet Data…
Falcon.eyrie.af.mil	TCP	60	63281> telnet [ACK] Seq=7 Ack=7 Win=33580 Len=0
Falcon.eyrie.af.mil	TELNET	60	Telnet Data…
Delta.peach.mil	TELNET	60	Telnet Data…
Falcon.eyrie.af.mil	TCP	60	63281> telnet [ACK] Seq=8 Ack=8 Win=33580 Len=0
Falcon.eyrie.af.mil	TELNET	60	Telnet Data…
Delta.peach.mil	TELNET	60	Telnet Data…
Falcon.eyrie.af.mil	TCP	60	63281> telnet [ACK] Seq=9 Ack=9 Win=33580 Len=0
Falcon.eyrie.af.mil	TCP	60	Telnet Data…
Delta.peach.mil	TELNET	60	Telnet Data…
Falcon.eyrie.af.mil	TCP	60	63281> telnet [ACK] Seq=10 Ack=10 Win=33580 Len=0
Falcon.eyrie.af.mil	TELNET	60	Telnet Data…
Delta.peach.mil	TELNET	60	Telnet Data…

Table 2. Cell structure returns

1	0	0.0
2	1	0.9275
3	1	0.7550

Table 3. Data For Learn

1	1	0	1	1	1	1
1	1	0	0	0	0	0.75
1	1	0	0	0.5	0.5	0.85
0.5	1	0	0	0.5	0.5	0.8
0.5	0.5	0	0	0.1	0.3	0
0	1	0	0	0.1	0.2	0
1	0.5	0	1	0.5	0.3	0.65
0	0	0	0	0	0	0
0.5	1	0	0	1	1	0.85
0.5	0.5	0	1	1	1	0.8
1	1	0	1	0	0	0.9
0.5	0.5	0	0	0.5	0	0
0	0	0	1	1	1	0.65
0	0	1	1	1	1	0.9

0	0	1	0	0.5	0	0.8
0	0	1	1	0.5	0.5	0.85
0	0	1	0	0	0	0.8
0.5	0.5	0	0	0.5	1	0

Table 4. CorrelationAl Algorithm Output “correlationAl (dataIDS)

1	0	0
2	1	0.773381779
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	1	0.807486488
12	0	0
13	0	0
14	0	0
15	0	0
16	0	0
17	0	0
18	0	0
19	0	0
20	0	0
21	0	0
22	0	0
23	0	0
24	0	0
25	0	0
26	0	0
27	0	0
28	0	0
29	0	0
30	0	0

Table 5. FuzzyModel Algorithm Output “f = fuzzyModel (dataIDS)

1	0	0
2	1	0.773382
3	2	0.773382
4	3	0.999973
5	4	0.773382
6	5	0.773382
7	6	0.999973
8	7	0.773382
9	8	0.773382
10	9	0.999879
11	10	0.807486
12	11	0.807486
13	12	0.999879
14	13	0.773382
15	14	0.773382
16	15	0.999879
17	16	0.807486

18	17	0.807486
19	18	0.999879
20	19	0.773382
21	20	0.773382
22	21	0.999879
23	22	0.807486
24	23	0.807486
25	24	0.999973
26	25	0.807486
27	26	0.807486
28	27	0.999879
29	28	0.773382
30	29	0.773382

Using neural network and predefined threshold in (Zhu & Ghorbani, 2006), the alerts were classified into several groups. While we placed them in a group using their own method, this result was acceptable because all the alerts were somehow related to each other.

5. Conclusion

Here, we aimed to present a better method for correlating alerts. In our method, first, we use MLP as a correlation engine. This engine specifies probability of correlation of two alerts. Then we classified alerts using an algorithm and present them in the form of anultra-alert. The advantage of this method is that one alert can be placed in several ultra-alerts simultaneously. If one alert is placed in another group by mistake, such advantage will not lead to non-extraction of attack scenario of anultra-alert.

References

Kruegel, C., Valeur, F., & Vigna. G. (2005). Intrusion Detection and Correlation: Challenges and Solutions. *Springer-Verlag.*

Sadoddin, R., & Ghorbani, A. A. (2009). *An incremental frequent structure mining framework for real-time alert correlation.* Master's thesis, University of New Brunswick.

Wang, L., Liu, A., & Jajodia, S. (2006). Using attack graphs for correlating, hypothesizing, and predicting intrusion alerts. *Computer Communications, 29*(15), 2917–2933.

Xu, D., & Ning, P. (2005). *Privacy-preserving alert correlation: A concept hierarchy based approach.* In Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC), 537–546.

Zhu, B., & Ghorbani, A. (1006). Alert correlation for extracting attack strategies. *International Journal of Network Security, 3*(3), 244–258.

Appendices

Appendix 1: correlationAl FUNCTION

```
function hyperAlertList = correlationAl(ListOfAlert)
corrThreshold = 0.5;
corSensity = 0.1;
hyperAlert = zeros(1,3);
hyperAlertList = cell(1,1);
idxHyperAlert = 0;
preAlert = 0;
for m=1:size(ListOfAlert,1)
alert = m;
maxCorr = 0;
if(idxHyperAlert==0)
```

```
idxHyperAlert = idxHyperAlert+1;
hyperAlert(1,1) = alert;
hyperAlertList{1,idxHyperAlert} = hyperAlert;
else
for n=1:size(hyperAlertList,2)
hyperAlertS = hyperAlertList{1,n};
for k=1:size(hyperAlertS,1)
                    probCorr = corrCal(ListOfAlert(hyperAlertS(k,1),:),ListOfAlert(alert,:),preAlert);
if(probCorr>maxCorr)
maxCorr = probCorr;
maxIdxHyperA = n;
maxIdxA = k;
end
end
end
if(maxCorr>corrThreshold)
hyperAlertSelected = hyperAlertList{1,maxIdxHyperA};
flagFind = 0;
for i=1:size(hyperAlertSelected,1)
if(i==maxIdxA)
continue;
end
                        probCorr                                                    =
corrCal(ListOfAlert(hyperAlertSelected(i,1),:),ListOfAlert(alert,:),preAlert);
if((maxCorr - probCorr)<corSensity)
                            l = size(hyperAlertSelected,1);
hyperAlertSelected(l+1,1) = alert;
hyperAlertSelected(l+1,2) = i;
hyperAlertSelected(l+1,3) = probCorr;
hyperAlertList{1,maxIdxHyperA} = hyperAlertSelected;
flagFind = 1;
end
end
if(i==1)
                        l = size(hyperAlertSelected,1);
hyperAlertSelected(l+1,1) = alert;
hyperAlertSelected(l+1,2) = i;
hyperAlertSelected(l+1,3) = probCorr;
hyperAlertList{1,maxIdxHyperA} = hyperAlertSelected;
flagFind = 1;
end
if(flagFind==0)
hyperAlert(1,1) = alert;
```

```
idxHyperAlert = idxHyperAlert+1;
hyperAlertList{1,idxHyperAlert} = hyperAlert;
end
end
end
preAlert = ListOfAlert(m,:);
end
end
```

Appendix 2: correlationAl2 FUNCTION

```
function correlationAl2(listOfAlert)
stackObj = stack;
threshold = 0.1;
      r = randi(size(listOfAlert,1));
stackObj = stackObj.inqueue(listOfAlert(r,1));
graphAttack = listOfAlert(r,1);
idxGraphAttack = 1;
visitedGraph = zeros(size(listOfAlert,1),1);

%acm = calculateACM(listOfAlert);
loadacm;

isEmpty = stackObj.top;
while(isEmpty>0)
stackObj = stackObj.dequeue();
alert =    stackObj.dequeuedElm;
for i=1:size(acmMatrix,2)
forwardCorrStr = acmMatrix(alert,i)/sum(acmMatrix(alert,:));
if(forwardCorrStr>threshold)
if(visited(i,1)==0)
stackObj = stackObj.inqueue(listOfAlert(i,1));
visitedGraph(i,1) = 1;
end
graphAttack(idxGraphAttack,2) = i;
graphAttack(idxGraphAttack,3) = acmMatrix(alert,i);
idxGraphAttack = idxGraphAttack+1;
end
end
isEmpty = stackObj.top;
end
end
```

Appendix 3: readData FUNCTION

```
function readData()
    [data,path] = uigetfile('m2.csv');
data = dataset('xlsfile',sprintf('%s\%s', path,data));
end
```

APPENDIX 4: fuzzyModel FUNCTION

```
function hyperAlertList = fuzzyModel(ListOfAlert)
corrThreshold = 0.5;
%corSensity = 0.1;
hyperAlert = zeros(1,3);
hyperAlertList = cell(1,1);
idxHyperAlert = 0;
preAlert = 0;
for m=1:size(ListOfAlert,1)
alert = m;
%maxCorr = 0;
if(idxHyperAlert==0)
idxHyperAlert = idxHyperAlert+1;
hyperAlert(1,1) = alert;
hyperAlertList{1,idxHyperAlert} = hyperAlert;
else
for n=1:size(hyperAlertList,2)
hyperAlertS = hyperAlertList{1,n};
%for k=1:size(hyperAlertS,1)
                    l = size(hyperAlertS,1);
                    probCorr = corrCal(ListOfAlert(hyperAlertS(l,1),:),ListOfAlert(alert,:),preAlert);
if(probCorr>corrThreshold)
hyperAlertS(l+1,1) = alert;
hyperAlertS(l+1,2) =    hyperAlertS(l,1);
hyperAlertS(l+1,3) = probCorr;
hyperAlertList{1,n} = hyperAlertS;
%                           for i=1:size(hyperAlertS,1)-1
%                                                                                                   probCorr     =
corrCal(ListOfAlert(hyperAlertS(i,1),:),ListOfAlert(alert,:),preAlert);
%                                   hyperAlertS(l+1,i+1,1) = i;
%                                   hyperAlertS(l+1,i+1,2) = probCorr^2;
%                                   hyperAlertList{1,n} = hyperAlertS;
%                           end
else
hyperAlert(1,1) = alert;
idxHyperAlert = idxHyperAlert+1;
hyperAlertList{1,idxHyperAlert} = hyperAlert;
```

```
end
end
end
preAlert = ListOfAlert(m,:);
end
end
```

Appendix 5: featureMatching FUNCTION

```
function [f1,f2,f3,f4] = featureMatching(hyperAlertS,alert,preAlert)
addressIP = xlsread('data\addressIP.xlsx');
hyAlIP = zeros(1,8);
alertIP = zeros(1,8);
    %% Calculation of f1,f2
for k=3:4
for i=1:size(addressIP,1)
if(hyperAlertS(1,k)==addressIP(i,1))
for j=2:5
hyAlIP(j-1,:) = bitget(addressIP(i,j),8:-1:1,'uint8');
end
end
if(alert(1,k)==addressIP(i,1))
for j=2:5
alertIP(j-1,:) = bitget(addressIP(i,j),8:-1:1,'uint8');
end
end
end
%alIP = num2str(alertIP(1,5));
%hyAIP = num2str(hyAlIP(1,5));
for i=1:8
match = 0;
for j=i:8
if(alertIP(4,j)==hyAlIP(4,j))
match = match+1;
else
break;
end
end
matchT(1,i) = match;
end
matchT = sort(matchT,'descend');
if(k==3)
                f1 = (24+matchT(1,1))/32;
else
```

```
                f2 = (24+matchT(1,1))/32;
end
end
%%End of Calculation of f1,f2
   %% Calculate another features
if(hyperAlertS(1,5)==alert(1,5))
            f3 = 1;
else
            f3 = 0;
end

if(preAlert(1,4)==alert(1,3))
            f4 = 1;
else
            f4 = 0;
end
%%End of Calculate another features
end
```

Appendix 6: featureMatchForCls FUNCTION

```
function [f1,f2] = featureMatchForCls(hyperAlertS,alert)
addressIP = xlsread('data\addressIP.xlsx');
hyAlIP = zeros(1,8);
alertIP = zeros(1,8);
    %% Calculation of f1,f2
for k=3:4
for i=1:size(addressIP,1)
if(hyperAlertS(1,k)==addressIP(i,1))
for j=2:5
hyAlIP(j-1,:) = bitget(addressIP(i,j),8:-1:1,'uint8');
end
end
if(alert(1,k)==addressIP(i,1))
for j=2:5
alertIP(j-1,:) = bitget(addressIP(i,j),8:-1:1,'uint8');
end
end
end
%alIP = num2str(alertIP(1,5));
%hyAIP = num2str(hyAlIP(1,5));
for i=1:8
match = 0;
for j=i:8
```

```
if(alertIP(4,j)==hyAlIP(4,j))
match = match+1;
else
break;
end
end
matchT(1,i) = match;
end
matchT = sort(matchT,'descend');
if(k==3)
                f1 = (24+matchT(1,1))/32;
else
                f2 = (24+matchT(1,1))/32;
end
end
%%End of Calculation of f1,f2
```

Appendix 7: learnAl FUNCTION

```
%This function learn a neural network to produce a probability of
%correlation between two alerts.
%Notice that the p and t parameters must be this way: p is a matrix which
%it's rows show the features and it's columns show the elements. t also
% is a matrix which it's rows show the class(Label)s and its columns show
% elements.
function learnAl()
load('dataNet.mat');
    MinAndMax = zeros(4,1);
    MinAndMax = [MinAndMax ones(4,1)];
net = newff(MinAndMax,[4,1],{'tansig','tansig'});
init(net);
    net.trainParam.show = 50;
    net.trainParam.lr = 0.05;
    net.trainParam.epochs = 300;
    net.trainParam.goal = 1e-5;
    p = dataForLearn(:,1:4);
    p = reshape(p,4,18);
    t = dataForLearn(:,7);
    t = reshape(t,1,18);
net = train(net,p,t);
savenetMlpnet;
end
```

Appendix 8: STACK FUNCTION

```
classdef stack
```

```
properties
table = zeros(1,1);
top = 0;
dequeuedElm = 0;
end
methods
function obj = inqueue(obj,value)
                    obj.top = (obj.top)+1;
                    t = obj.top;
obj.table(t,1) = value; %('farshid');
end

function obj = dequeue(obj)
                    t = obj.top;
                    obj.dequeuedElm = obj.table(t,1);
obj.table(t) = [];
                    obj.top = (obj.top)-1;
end
end
end
```

Appendix 9: corrCal FUNCTION

```
function corrProb = corrCal(hyperAlertS,alert,preAlert)
addressIP = xlsread('data\addressIP.xlsx');
hyAlIP = zeros(1,8);
alertIP = zeros(1,8);
    %% Calculation of f1,f2
for k=3:4
for i=1:size(addressIP,1)
if(hyperAlertS(1,k)==addressIP(i,1))
for j=2:5
hyAlIP(j-1,:) = bitget(addressIP(i,j),8:-1:1,'uint8');
end
end
if(alert(1,k)==addressIP(i,1))
for j=2:5
alertIP(j-1,:) = bitget(addressIP(i,j),8:-1:1,'uint8');
end
end
end
%alIP = num2str(alertIP(1,5));
%hyAIP = num2str(hyAlIP(1,5));
```

```
for i=1:8
match = 0;
for j=i:8
if(alertIP(4,j)==hyAlIP(4,j))
match = match+1;
else
break;
end
end
matchT(1,i) = match;
end
matchT = sort(matchT,'descend');
if(k==3)
                f1 = (24+matchT(1,1))/32;
else
                f2 = (24+matchT(1,1))/32;
end
end
%%End of Calculation of f1,f2
   %% Calculate another features
if(hyperAlertS(1,5)==alert(1,5))
          f3 = 1;
else
          f3 = 0;
end

if(preAlert(1,4)==alert(1,3))
          f4 = 1;
else
          f4 = 0;
end
%%End of Calculate another features
      %% Load the Correlation Engine and Calculate probability of correlation
load ('netMlp.mat');
          f = [f1;f2;f3;f4;];
corrProb = sim(net,f);
%%End of Load the Correlation Engine and Calculate probability of correlation
End
```