석사 학위논문

Master's Thesis

개미군집 알고리즘과 의사결정트리를 활용한 생체모방 미지 공격 탐지 시스템

A Bio-inspired Intrusion Detection System for Unknown-attacks combining Ant Clustering Algorithm and Decision Tree

김 경 민 (金 倞 民 Kim, Kyung-min)

전산학부

School of Computing

KAIST

2016

개미군집 알고리즘과 의사결정트리를 활용한 생체모방 미지 공격 탐지 시스템

A Bio-inspired Intrusion Detection System for Unknown-attacks combining Ant Clustering Algorithm and Decision Tree

A Bio-inspired Intrusion Detection System for Unknown-attacks combining Ant Clustering Algorithm and Decision Tree

Advisor : Professor Kim, Kwangjo

by

Kim, Kyung-min
School of Computing
KAIST

A thesis submitted to the faculty of KAIST in partial fulfillment of the requirements for the degree of Master of Science in Engineering in the School of Computing . The study was conducted in accordance with Code of Research Ethics[1].

2015. 12. 14.

Approved by

Professor Kim, Kwangjo

[Advisor]

---

[1]Declaration of Ethical Conduct in Research: I, as a graduate student of KAIST, hereby declare that I have not committed any acts that may damage the credibility of my research. These include, but are not limited to: falsification, thesis written by someone else, distortion of research findings or plagiarism. I affirm that my thesis contains honest conclusions based on my own careful research under the guidance of my thesis advisor.

개미군집 알고리즘과 의사결정트리를 활용한 생체모방 미지 공격 탐지 시스템

김 경 민

위 논문은 한국과학기술원 석사학위논문으로
학위논문심사위원회에서 심사 통과하였음.

2015년 12월 14일

심사위원장 김 광 조 (인)

심사위원 박 홍 식 (인)

심사위원 김 명 호 (인)

MSoC
20143061

김 경 민. Kim, Kyung-min. A Bio-inspired Intrusion Detection System for Unknown-attacks combining Ant Clustering Algorithm and Decision Tree. 개미군집 알고리즘과 의사결정트리를 활용한 생체모방 미지 공격 탐지 시스템. School of Computing . 2016. 23p. Advisor Prof. Kim, Kwangjo. Text in English.

ABSTRACT

Intrusion Detection System (IDS) monitors network traffic and detects users' malicious activities. IDS can be divided by its detection type as Signature-based IDS and Anomaly-based IDS. IDS can be divided as a Network IDS and a Host IDS.

As internet of things era is coming, the amount of network traffic increases explosively. Labeling a traffic manually whether attack or not is difficult under this condition. Also new unknown-attacks are appearing constantly, the detection of unknown-attacks has become the essential part of IDS as well as the detection of known-attacks. Unknown-attack detection is a research area about detecting attacks without any specific prior knowledge of attacks. IDS should have capability to determine an input whether attack or not based on the unlabeled dataset since unknown-attacks are not known to IDS. To solve these difficulties, we need to find a way to learn about normal traffic and attack traffic on the unlabeled dataset.

This paper proposes a novel IDS scheme for unknown-attacks based on the clustering model. The proposed IDS combines two machine learning algorithms, Ant Clustering Algorithm (ACA) and Decision Tree(DT). The IDS learns on the unlabeled dataset by itself and constructs the profile of normal behavior. After construction of the profile, the IDS can detect unknown-attacks. The IDS consists of two main engines: the ACA engine and the DT engine. The IDS builds clusters on unlabeled dataset by using ACA. Based on the clustering result, the ACA engine classifies normal traffic and attack traffic. The DT engine trains detectors based on the result of the ACA engine.

The proposed IDS was experimented on the KDD Cup 1999 Dataset. Evaluation criteria for performance of the proposed IDS are detection rate, false positive rate, and accuracy. The IDS has much higher detection rate and accuracy than Hosseinpour *et al.* [1] which has similar approach with ours.

Keywords : IDS, Unknown-attack Detection, Bio-inspired, Swarm Intelligence

Contents

List of Tables

List of Figures

Chapter 1. Introduction

1.1 Intrusion Detection System

Intrusion Detection System (IDS) monitors network traffic and detects users' abnormal or malicious activities[2]. Two main detection types are exist in IDS: Signature-based IDS and Anomaly-based IDS. Signature-based IDS uses some signatures of known-attacks to detect malicious behaviors. Anomaly-based IDS builds a profile of normal behavior and detects violation of the profile as malicious behavior. IDS can be divided as a Network IDS (NIDS) and a Host IDS (HIDS) by location of IDS sensors. A HIDS is installed on certain host or device and detects attacks from outside of the host or device. A NIDS is installed over a target network. Sensors of the NIDS are installed where a lot of traffic is passed, like gateway or router. Although the term IDS means both of HIDS and NIDS, in this paper, IDS refers to a NIDS.

Firewall looks similar to IDS in detection of attacks. However, IDS and firewall has a big difference. Firewall analyzes packet header and only capable to detect attacks using pre-installed policies or rules based on protocol type, source address, destination address, source port, and destination port. A Packet is rejected if it doesn't match any policy. On the other hand, IDS analyzes the traffic information such as packet payload to determine whether intrusion or not. Therefore, IDS can make a network more secure than firewall.

IDS is composed of the IDS server and the IDS sensor. The IDS server decides whether an attack has occurred or not based on collected information by the IDS sensor. The IDS sensor collects traffic information over a target network. The IDS sensor is installed on a place which large amount of network traffic is passed such as near a gateway or router. The IDS sensor collects many information. For example, the network payload or network flow. After collecting traffic information, the sensor sends the information to the IDS server. An example of IDS is illustrated in figure 1.1.

Generally, datasets and real network traffic data are used to evaluate performance of IDS. Researchers can train their IDS based on the training dataset and test by using the test dataset. They compare prediction results of detection algorithm and analyze the results of actual value of test dataset.

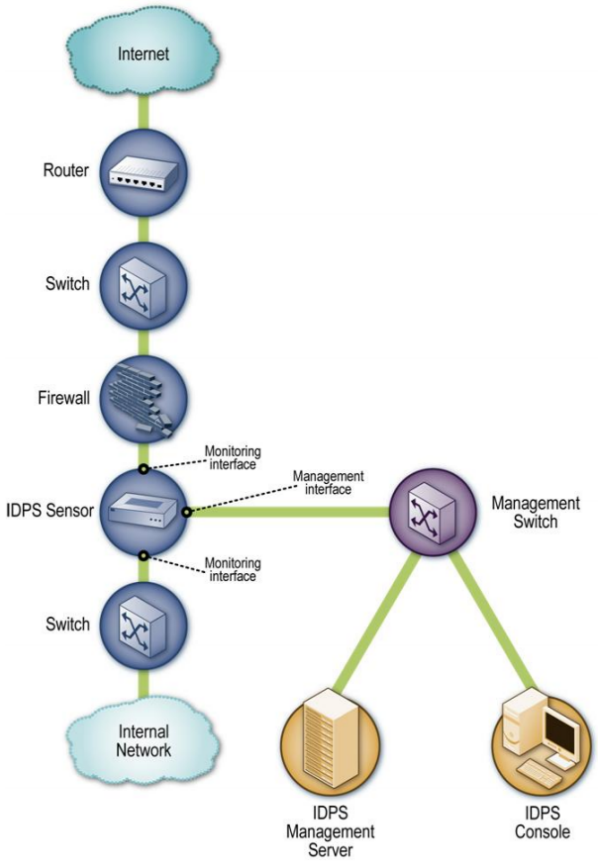

Figure 1.1: Example of IDS architecture

Four categories are exist to evaluate detection result: true positive, true negative, false positive, false negative. In IDS case, true positive is a case that a malicious traffic referred as an attack by IDS. True negative is a case that a normal traffic referred as a normal by IDS. False positive is a case that a normal traffic referred as an attack and false negative is a case that a malicious traffic referred as a normal traffic. Based on these four categories, three important criteria for evaluation performance of IDS can be calculated. First one is detection rate (DR). DR is defined as the number of intrusion instances detected by IDS, same as true positive, divided by the total number of intrusion instances in the test dataset. DR is a criteria which indicates how well IDS detects attacks. High DR means that IDS can detect attacks more than IDS which has low DR. Second one is false positive rate (FPR). FPR is defined as the number of normal instances classified as attack, same as false positive, divided by the total number of normal instances in the test dataset. High FPR means that IDS can misclassify a normal traffic as an attack more frequently than IDS which has low FPR. Final one is accuracy (ACC). ACC is defined as the number of corrected classified instances by IDS, same as sum of true positive and false negative,

divided by the total number of instances in the test dataset. ACC is related to how well IDS classify normal and attacks correctly. Equations for calculation DR, FPR, and ACC are presented in below:

$$DR = \frac{TruePositive}{TruePositive + FalseNegative} \tag{1.1}$$

$$FPR = \frac{FalsePositive}{FlasePositive + TrueNegative} \tag{1.2}$$

$$ACC = \frac{TruePositive + FalseNegative}{TruePositive + FlasePositive + TrueNegative + FalseNegative} \tag{1.3}$$

1.2 Motivation

As internet of things era is coming, many devices are connected on network. We must deal with enormous volumes of network data. Making labels of these enormous data manually whether attack or not is difficult and expensive task. Because of this difficulty, we don't have enough labeled data available. Also, we get labeled dataset only by simulating intrusions. This method has a limitation that we can get just known-attacks and don't reflect unknown-attacks which occurs in near future in the dataset. New unknown-attacks are appearing constantly. The unknown-attacks can give us serious damage. Therefore, we should detects unknown-attacks as soon as possible. However, we cannot get a labeled dataset which contains label of unknown-attacks.

To solve these problems, we need a method for detecting attacks on the unlabeled dataset. The method should not use the supervised learning model, since the supervised learning model have to know about labels of the dataset. The method which uses the unsupervised model doesn't need to know labels. Because the method can learn on the unlabeled dataset by itself, it doesn't need to have any prior knowledge of certain attack. In this paper, we propose a novel IDS which can learn on the unlabeled dataset by itself. We combine two machine learning algorithms to detect unknown-attacks: ant clustering algorithm and decision tree. As combining these two algorithms, the proposed IDS doesn't need any prior knowledge about attacks and can detect unknown-attacks effectively.

1.3 Organization

The rest of this paper is organized as follows: Chapter 2 describes relate work and background about detection types of IDS, unknown-attack detection, dataset for evaluation, clustering algorithms, and decision tree. The methodology of the proposed IDS is described in Chapter 3. Description about the architecture of the proposed IDS is in Chapter 4. Experimental result and some discussion are described in Chapter 5. Finally, the conclusion and future work are discussed in Chapter 6.

Chapter 2. Related Work and Background

2.1 Detection Types of IDS

2.1.1 Signature-based IDS

Signature-based IDS has some signatures of known-attacks and makes rules for detection of known-attacks. When the traffic from the network comes into IDS, signature-based IDS extracts some signatures from the traffic. After extraction, signature-based IDS compares extracted signatures with the signatures which it already has. If some signatures are matched, signature-based IDS decides the traffic as an attack traffic. Signature-based IDS can detect known-attacks well since the IDS is a black-list model. The IDS compares the traffic with black-list. If the traffic contains some signatures in black-list, the IDS must detect an attack.

However, signature-based IDS cannot detect any unknown-attack since signature-based IDS doesn't have any signature of unknown-attacks. Therefore, signature-based IDS classifies an unknown-attack traffic as a normal traffic. Signature-based IDS doesn't capable to detect unknown-attacks.

2.1.2 Anomaly-based IDS

Unlike signature-based IDS, anomaly-based IDS doesn't focus on maintaining black-list of signatures. Anomaly-based IDS choose a white-list model. Anomaly-based IDS makes a profile of normal behavior and detect violation of the profile. Any activity which causes abnormal state of a network is detected by anomaly-based IDS. Therefore, anomaly-based IDS can detect unknown-attacks since any attacks will cause abnormal state of a network even if unknown-attacks.

2.2 Unknown-attack Detection

Unknown-attack detection is a research area about detecting an attack without any specific preliminary knowledge of the attack. Because signature-based IDS needs some signatures of known-attacks, signature-based cannot detect unknown-attacks. On the other hand, anomaly-based IDS can detect unknown-attacks because anomaly-based IDS concentrates on violation of normal behavior profile. Any

form of activities which violates normal behavior profile is detected by anomaly-based IDS even if IDS doesn't have specific knowledge of activities. Therefore, anomaly-based IDS is mainly used to detect unknown-attacks.

Robust modeling of normal behavior profile directly affects to performance of anomaly-based IDS. Many researches about anomaly-based IDS utilize some machine learning and data mining algorithms to accomplish robustness of building normal behavior profile. For example, Support Vector Machine (SVM) and Artificial Immune System (AIS) in supervised learning algorithms and k-Means clustering, DBSCAN, and Ant Cluster Algorithm (ACA) in unsupervised algorithms are used.

2.3 KDD Cup 1999 Dataset

KDD Cup 1999 Dataset is a dataset of DARPA 1998 Intrusion Detection Evaluation Program. The datasets is used in the 1999 KDD intrusion detection contest[3]. The dataset contains normal traffic and four types of attack traffic; DoS, U2R, R2L, Probe. The types of the dataset are described as below.

- Normal : not attack
- DoS : denial-of-service (*e.g.*, syn flood attack)
- U2R : unauthorized access to local superuser/root privileges (*e.g.*, buffer overflow attacks)
- R2L : unauthorized access from a remote machine (*e.g.*, guessing password)
- Probe : surveillance and other probing (*e.g.*, port scanning)

The dataset has 4,898,431 traffic data instances. Each data instances in the dataset has 41 features and its label of traffic type. The traffic distribution of KDD Cup 1999 Dataset is presented in Table 2.1 as below.

Table 2.1: Traffic distribution of KDD Cup 1999 Dataset

Type	# of traffic	Proportion (%)
Normal	972,781	19.86
DoS	3,883,370	79.28
U2R	52	0.00
R2L	1,126	0.02
Probe	41,102	0.84
Total	4,898,431	100

2.4 Clustering Algorithms

Clustering is one of the unsupervised machine learning algorithms. Clustering algorithms don't need labels of dataset. Clustering algorithms collect similar instances and make some clusters on the unlabeled dataset. Some clustering algorithms are used in anomaly-based IDS.

(1) k-Means clustering algorithm partitions the dataset into k clusters. Every data instances are assigned to exactly one cluster which the nearest cluster center. The Euclidean distance is mainly used to calculate distance between a data instance and a cluster center[1]. k-Means clustering requires one parameter: the number of clusters. k-Means clustering is very sensitive to initialization state of the cluster centroids.

(2) Density-Based Spatial Clustering of Applications with Noise (DBSCAN) finds clusters based on the estimated density distribution of the dataset. DBSCAN requires two parameters: maximum radius of the neighborhood and minimum number of samples to form a cluster[1]. DBSCAN is insensitive to initialization state. But DBSCAN is sensitive to data density and data dimension[5].

(3) Ant Clustering Algorithm (ACA) is a heuristic algorithm and one of the swarm intelligence algorithms. The swarm intelligence algorithms are one area of the bio-inspired algorithms. ACA is based on the brood sorting activities of ants. ACA was modeled by Deneubourg *et al.*[6]. Their model is referred as basic model (BM)[7]. In ACA, each data instances of the dataset is randomly scattered in a 2D space. And each ant moves randomly the 2D space and picks up or drops down the data instances based on calculation of probability. Equations for calculation probability of picking up and dropping down are presented in below:

$$P_{pick} = (\frac{k_1}{k_1 + f})^2 \tag{2.1}$$

where f is the perceived fraction of items in the neighborhood of the ant and k_1 is a threshold item.

$$P_{drop} = (\frac{f}{k_2 + f})^2 \tag{2.2}$$

where k_2 is another threshold constant.

ACA has self-organizing characteristic. Self-organization means that accomplish overall process by coordination out of the local interactions between smaller components. Therefore, ACA can makes clusters on the initially disordered dataset by itself. Unlike k-Means clustering, ACA doesn't require predefined number of clusters because of self-organizing characteristic. ACA is not sensitive to initialization state, data density, and data dimension. ACA requires four parameters: the number of ants, size of the 2D space, local area of the ant, and threshold used in picking or dropping decision. High-level description of the ACA is presented in Figure 2.1 [8].

```
//Initialization Phase
Randomly scatter o_i object on the 2D grid
for each agent a_j do
    pandom_select_object (o_i)
    pick_up_object o_i
    place_agenct a_j at randomly selected empty grid location
end for

//Main loop
for t = 1 to t_max do
    random_select_agent (a_j)
    move_agent a_j to new location
    i = carried_object(agent a_j)
    Compute f*(o_i) and p*_drop(o_i)
    If drop = True then
        While pick = False do
            i = random_select object o
            Compute f*(o_i) and p*_drop(o_i)
            Pick_up_object o_i
        end while
end for
```

Figure 2.1: High-level Decription of ACA

Comparison of features between k-Means, DBSCAN, and ACA is presented in Table 2.2 [5].

Table 2.2: Comparison of features between k-Means, DBSCAN, and ACA

	k-Means	DBSCAN	ACA
# of parameters	1	2	4
Insensitivity of initialization	X	O	O
Insensitivity of data density	O	X	O
Insensitivity of data dimension	X	X	O

2.5 Decision Tree

Decision Tree (DT) is one of the supervised machine learning algorithms. DT is a predictive model which using tree-like graph. DT can be trained as a rule-based structure. DT makes a tree based on the training set. DT extracts some rules to classify the training set correctly. Each branch of DT represents a decision rule. Each leaf node represent a set of data instances has same class. DT makes decision rules until every data instances are classified correctly. The goal of DT is to create a model that predicts the value of a target output based on several inputs. Two main types are exist in DT: classification tree and regression tree. When the predicted output is a finite set of classes, classification tree is used. When the predicted output is continuous value, regression tree is used.

DT has some advantages than other supervised machine learning algorithms. First, DT can allow the addition of new possible scenarios. It is appropriate advantage for unknown-attack detection. Second, DT is a white-box model algorithm. Therefore, unlike black-box model, we can analyze why the algorithm predicts a certain output. When using DT as detection algorithm, a security expert can analyze a certain attack and extract some rules after detection of the attack to detect the attack next time.

Chapter 3. Methodology

3.1 Assumption

KDD Cup 1999 Dataset has a drawback in distribution of traffic. Over 80% of data instances is attack traffic in the dataset. This proportion doesn't reflect general network traffic environment. In general network, normal traffic overwhelms attack traffic. The proposed IDS is based on a major assumption to reflect general network environment. The assumption is that there is overwhelmingly much normal traffic than attack traffic in target network. Based on the assumption, we filtered and construct the dataset as 98% of normal traffic and 2% of attack traffic.

3.2 Normalization

The KDD Cup 1999 Dataset has 41 features. The range of variables is diverse feature by feature. This will affects clustering result. As an example, consider two sets of two 2-feature vectors. Let assume that the range of the first feature is from 0 to 10 and the range of the second feature is from 10 to 100.

- {(1,10),(8,10)}

- {(1,10),(1,80)}

Under a Euclidean distance metric, the squared distance between feature vectors in the first set will be 49, while it will be 4,900 for the second set. If cluster width is fixed, the first set will be included in same cluster. But second set will be included in different cluster. Difference between two vectors in each set is same as 7 times from smaller feature to larger feature.

As a possible solution, we converted the data instances to a same distribution. That is, we make same range of every features in the dataset. Then, we can normalize all data instances to a fixed range of our choosing.

3.3 Clustering

To make clusters from the unlabeled dataset, we used ACA. Due to self-organizing characteristic, ACA can makes clusters by itself. Initially, we assigns a cluster to each data instances. In main loop of ACA, when an ant drops down a data instance, the ant looks around and assigns cluster of the data instance as majority cluster around his neighbor. After certain iterations, some clusters are remained with more members than initial state and other clusters are disappeared. Based on the clustering result, we can make labels to each data instances.

3.4 Cluster labeling

Since we are considering to learn on the unlabeled data, we don't have access to labels during training. Therefore, we need to find some other way to decide which clusters contain normal instances and which contain attack instances. Under our assumption about normal traffic constituting an overwhelmingly large portion, over 98%, it is highly probable that clusters containing normal data will have a much larger number of instances associated with them then would clusters containing attacks. Therefore, we label some percentage of the clusters containing the largest number of instances associated with them as 'normal'. The rest of the clusters are labeled as 'attack'[4].

3.5 Detection

After all labels of data instances made by ACA, we can use the dataset as a labeled dataset with the labels by ACA. Therefore, we can train intrusion detector using supervised learning method based on the dataset and labels made by ACA to detect unknown-attacks. Among many supervised machine learning algorithms, we choose DT algorithm to train intrusion detector. The trained detector monitors a network and can detect unknown-attacks.

Chapter 4. Proposed IDS Scheme

The proposed IDS combines an unsupervised machine learning algorithm and a supervised machine learning algorithm to detect unknown-attacks. We use ACA as an unsupervised learning algorithm and DT as a supervised algorithm. As we choose this architecture, the proposed IDS can learn on the unlabeled dataset. Figure 4.1 illustrates the architecture of the proposed IDS.

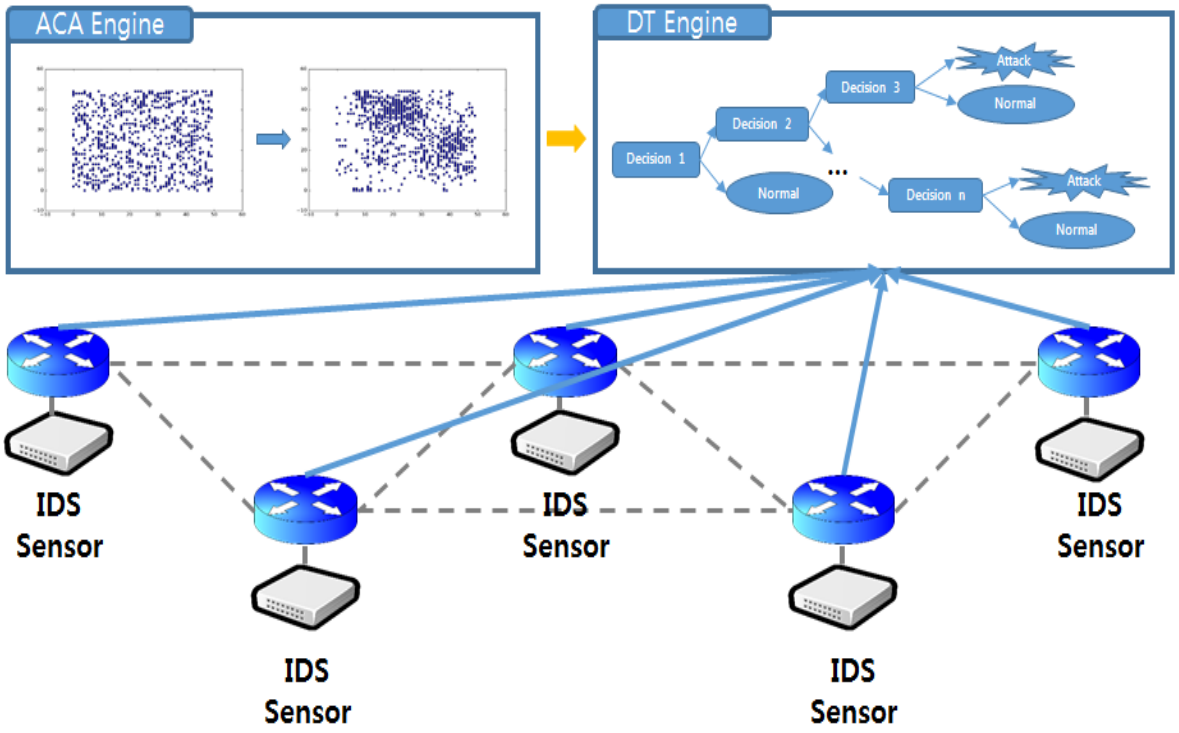

Figure 4.1: Architecture of the proposed IDS

The proposed IDS is composed of two main engines: the ACA engine and the DT engine. The ACA engine plays a role to make clusters on unlabeled dataset and labels to each instances. After making labels, the ACA engine passes the dataset and labels made by the ACA engine to the DT engine. The DT engine receives the dataset with labels from the ACA engine and trains intrusion detector using supervised learning method based on the received dataset with labels. After training done, the trained intrusion detector monitors a network and can detect unknown-attacks.

4.1 ACA Engine

The ACA engine makes clusters based on the unlabeled dataset. Due to self-organizing characteristic of the ACA, the ACA engine can builds clusters by itself. We don't need to consider about the number of clusters. After making clusters, the ACA engine sorts clusters by the number of cluster members. And the ACA engine makes labels to certain percentage of large clusters as 'normal'. Other clusters and outliers are labeled as 'attack' by the ACA engine. The ACA engine passes the dataset and labels made by the engine to DT engine. Figure 4.2 illustrates the process of the ACA engine.

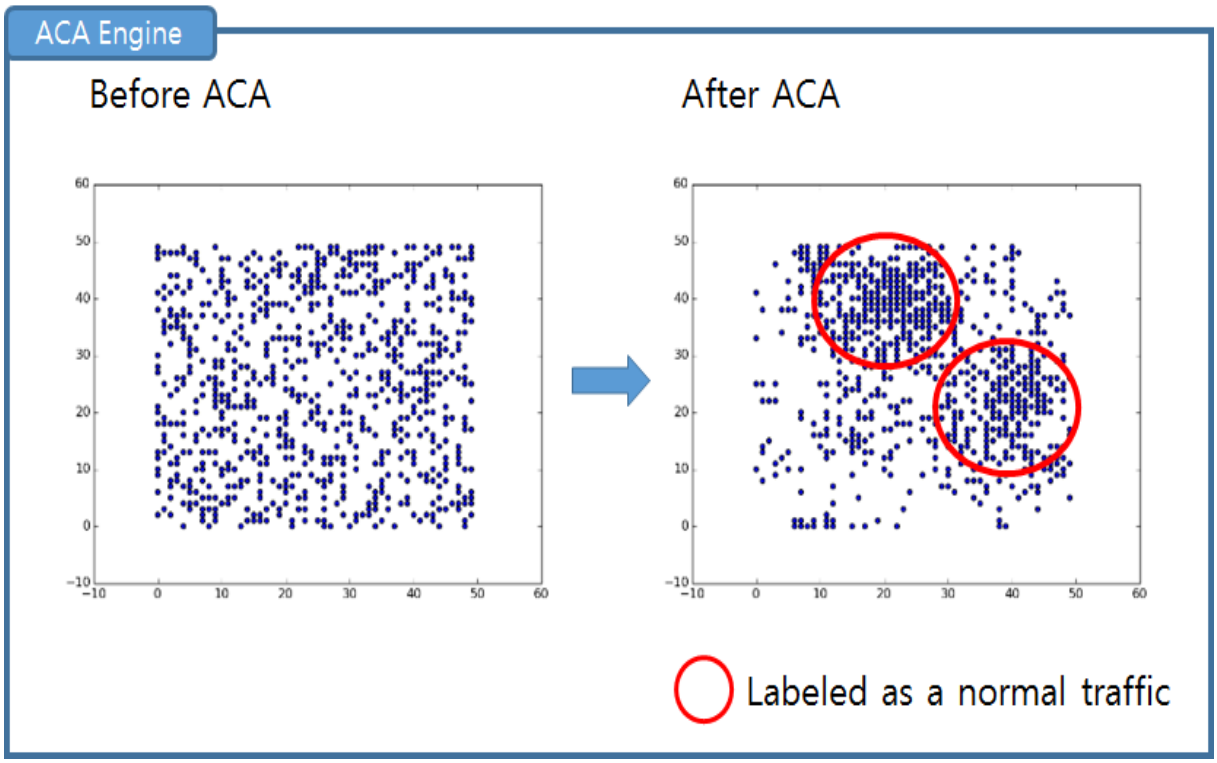

Figure 4.2: Process of the ACA engine

4.2 DT Engine

The DT engine receives the dataset with labels by the ACA engine. In the DT engine's view, the engine has a labeled dataset. Therefore, the DT engine can train the intrusion detector by using supervised learning method. We can analyze certain unknown-attack by security experts to extract some signatures of certain unknown-attack because DT is a white-box model. DT engine can show us reasons

of decisions. Figure 4.3 illustrates the architecture of the DT engine. The trained intrusion detector monitors a network and can detect unknown-attacks.

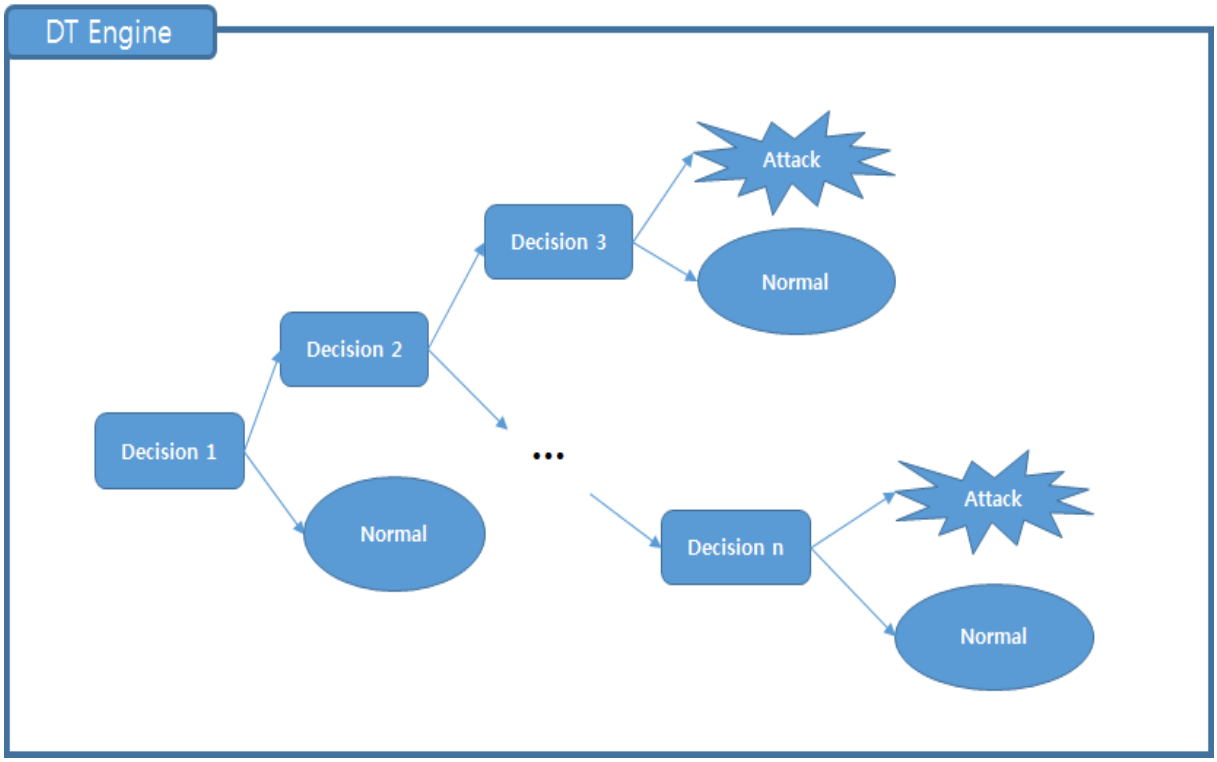

Figure 4.3: Architecture of the DT engine

Chapter 5. Evaluation

5.1 Dataset Decription

Under the our major assumption about the proportion of normal traffic overwhelms attack traffic, we construct the training set and the test set as 98% of normal traffic and 2% of attack traffic. To make 2% of attack traffic, we filtered attack traffic of KDD Cup 1999 Dataset. When we filtered attack traffic, we tried to include all of attack types to prevent biased training result. Also we used 10% version of KDD Cup 1999 Dataset in the experiment.

We partitioned the 10% version KDD dataset into five 20% subsets, each containing approximately 98,000 intances. Some subsets contained instances of biased traffic types. For example, 3rd subset contained only 98,805 DoS traffic instances and 4th subset didn't contain R2L traffic type. We didn't use these subsets as a test dataset because they didn't contained all types of the KDD Cup 1999 Dataset. We use only three of the five subsets, 1st subset, 2nd subset, and 5th subset, as a test dataset which contain all types of the KDD Cup 1999 Dataset. Therefore, we performed three experimental cases.

When a subset was selected as a test set, we had selected rest instances of the 10% version of KDD Cup 1999 Dataset as a training set. And we filtered the training set and the test set to meet our assumption. Our proposed IDS was trained on the filtered training set and evaluated on the filtered test set. Table 5.1, 5.2, and 5.3 show the traffic distributions of three experimental cases.

Table 5.1: Traffic distribution for case 1

	Training set		Test set	
Type	# of traffic	Proportion (%)	# of traffic	Proportion (%)
Normal	41,041	98.00	56,237	98.00
DoS	375	0.90	750	1.30
U2R	43	0.10	9	0.02
R2L	210	0.50	102	0.18
Probe	210	0.50	287	0.50
Total	41,879	100.00	57,385	100.00

Table 5.2: Traffic distribution for case 2

	Training set		Test set	
Type	# of traffic	Proportion (%)	# of traffic	Proportion (%)
Normal	82,290	98.00	14,988	98.00
DoS	1,124	1.34	132	0.87
U2R	32	0.04	20	0.13
R2L	103	0.12	77	0.50
Probe	420	0.50	77	0.50
Total	83,969	100.00	15,294	100.00

Table 5.3: Traffic distribution for case 3

	Training set		Test set	
Type	# of traffic	Proportion (%)	# of traffic	Proportion (%)
Normal	78,010	98.00	19,268	98.00
DoS	761	0.96	277	1.41
U2R	35	0.04	17	0.09
R2L	398	0.50	1	0.00
Probe	398	0.50	98	0.50
Total	79,602	100.00	57,385	100.00

5.2 Evaluation Criteria

Many criteria are used for evaluation performance of IDS such as DR, FPR, ACC, latency, throughput, and *etc.* Since this paper focus on capability of unknown-attacks, we concentrates on the detection capability. Among them, DR, FPR, and ACC is directly related to detecting capability of IDS. Therefore, we evaluate performance of our proposed IDS by comparing DR, FPR, and ACC.

Also, Hosseinpour *et al.*[1] proposed similar approach with our approach to detect unknown-attacks. They proposed two combinations of unsupervised machine learning algorithm and supervised machine learning algorithm. One combination is k-Means clustering and Artificial Immune System (AIS). The other is a combination of DBSCAN and AIS. Since their approach is similar with us, we compared performance of our proposed IDS and their IDS.

5.3 Experimental Result

5.3.1 Clustering Result

Since ACA needs more parameters than k-Means clustering and DBSCAN, we did many experiments in diverse parameter setting. Among them, the best parameter setting case is 1000 ants, 600 X 600 size of 2D grid, 500,000 iterations, 3 X 3 of local area of an ant, and 15 of constant for calculating probability. In this parameters, the ACA engine made 795 clusters based on the training set. Figure 5.1

illustrates initial state of the 2D grid and Figure 5.2 illustrates final state of the 2D grid.

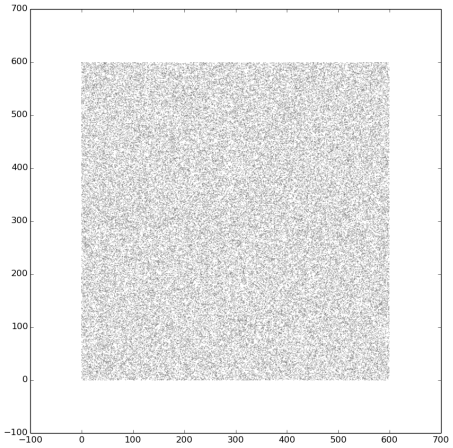

Figure 5.1: Initial state of the 2D grid

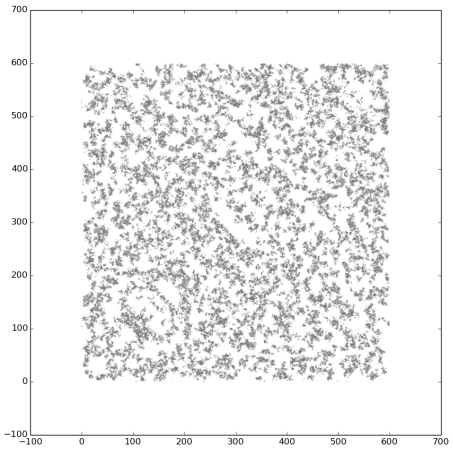

Figure 5.2: Final state of the 2D grid

5.3.2 IDS Performance

Because we performed three experimental cases, we calculated and evaluated performance of our proposed IDS in each experimental cases. Table 5.4 shows the performance of our proposed IDS in three experimental cases and average performance of three cases.

Table 5.4: Performance of our proposed IDS in three experimental cases

	Case 1	Case 2	Case 3	Average
DR(%)	72.65	81.05	95.42	83.04
FPR(%)	0.43	10.29	3.46	4.73
ACC(%)	99.04	89.54	96.52	95.03

Also as Hosseinpour *et al.*[1] proposed similar approach with our approach, we compared performance of our proposed IDS with the IDS of Hosseinpour *et al.*[1]. As we mentioned, compared criteria are DR, FPR, and ACC. Table 5.5 shows comparison of performance between Hosseinpour *et al.*[1] and our proposed IDS.

Table 5.5: Comparison of performance between Hosseinpour *et al.*[1] and the proposed IDS

	[1]	[1]	Proposed IDS
Algorithm	k-Means + AIS	DBSCAN + AIS	ACA + DT
DR(%)	43.1	58.9	83.0
FPR(%)	15.6	0.8	4.7
ACC(%)	60.7	77.1	95.0

As table shows, the proposed IDS has much higher DR than Hosseinpour *et al.*[1]. The proposed IDS has much higher ACC and has low FPR at the same time. This means that the proposed IDS builds more robust profile of normal behavior than the others. Also it means that ACA performed better clustering result than k-Means clustering and DBSCAN.

5.4 Discussion

In this section, we discuss some considerable points. Although the proposed has much higher DR and ACC, FPR still higher than Hosseinpour *et al.* [1]. This means that the ACA engine assigned same cluster to normal traffic and attack traffic sometimes. The reason for this problem is thought since each ant in the ACA engine only look around within their local area, not global area. Because ACA is a heuristic algorithm and has self-organizing characteristic, overall task is done by sum of smaller tasks. Therefore ACA has a local optima problem. Making larger local area of each ant will be a possible solution to mitigate this problem.

Our experimental result is based on the 10% version of KDD Cup 1999 Dataset. It can be thought that the experimental result is biased to the 10% version of KDD Cup 1999 Dataset. Therefore, more experimental cases on various datasets are needed to generalize the proposed IDS. Similarly, KDD Cup 1999 Dataset was announced at 1998. As many things were changed on network environment since

1998, KDD Cup 1999 Dataset no longer contains attack traffic enoughly. Therefore, we need to perform experiments on the latest dataset to reflect contemporary network environment. Kyoto 2006+ Dataset can be a considerable candidate for solution of this problem. Kyoto 2006+ Dataset built on the 3 years of real traffic data from November 2006 to August 2009. It consist of total 24 features [9].

Chapter 6. Conclusion

This paper proposes a novel IDS scheme that can detect unknown-attacks by combining ACA and DT. The proposed IDS can learn on the unlabeled dataset in unsupervised manner. The capability of learning on the unlabeled dataset is appropriate to enormous amount of network traffic environment such as internet of things. The capability can let we be free from making labels whether attack or not manually.

A major assumption is exist in this paper. We assumed that normal traffic overwhelms attacks traffic to reflect general network environment. Therefore, we make the training dataset and the test dataset be composed of 98% of normal traffic and 2% of attack traffic. Based on the assumption, we propose our IDS which choose clustering model.

The proposed IDS combines ACA and DT to detect unknown-attacks. The proposed IDS is composed of two main engine: the ACA engine and the DT engine. Overall flow of the proposed IDS is as follows: firstly, the ACA engine builds clusters on the unlabeled training dataset by self-organizing characteristic. After clustering phase, the ACA engine attaches labels certain percentage of large clusters as 'normal' label and the others as 'attack' label. Thirdly, the ACA engine passes the training dataset with labels made by the ACA engine to the DT engine. Fourthly, the DT engine trains intrusion detector based on the received dataset with labels by supervised manner. Finally, the trained intrusion detector monitors a network and can detect unknown-attacks without any specific knowledge.

Under the major assumption, the proposed IDS has much better performance than Hosseinpour *et al.* [1] which has a similar approach with ours. The proposed IDS has 83.04% of DR, 4.7% of FPR, and 95.03% of ACC. It means that our proposed IDS can builds more robust profile of normal behavior than Hosseinpour *et al.* [1].

However, future research also remained. First, more diverse experiments on various datasets are needed. We just performed experiments on the same dataset. To generalize our proposed IDS, we need to perform more diverse experiments on various datasets. Similarly, the latest dataset is needed since KDD

Cup 1999 Dataset is too old dataset. KDD Cup 1999 Dataset doesn't include contemporary traffic types of a network. Therefore, we need to do experiments on the other dataset which contains contemporary traffic types. Finally, the proposed IDS has better DR and ACC than Hosseinpour *et al.* [1]. But, our proposed IDS still has higher FPR than the combination of DBSCAN and AIS in [1]. Therefore, we need to find some methods to reduce FPR.

References

[1] Farhoud Hosseinpour, Payam Vahdani Amoli, Fahimeh Farahnakian, Juha Plosila, and Timo Hämäläinen, "Artificial Immune System Based Intrusion Detection: Innate Immunity using an Unsupervised Learning Approach", International Journal of Digital Content Technology & its Applications 8.5, 2014.

[2] Karen Scarfone and Peter Mell, "Guide to intrusion detection and prevention systems", NIST Special Publication 800, 2007.

[3] Salvatore Stolfo, Wenke Lee, and Andreas Prodromidis, "Cost-based modeling for fraud and intrusion detection: Result from JAM project", DARPA Information Survivability Conference and Exposition (DISCEX 2000), 2000.

[4] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo, "Intrusion Detection with unlabeled data using clustering", Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001), 2001.

[5] Kyung-min Kim, HakJu Kim, and Kwangjo Kim, "Design of an Intrusion Detection System for Unknown-attacks based on Bio-inspired Algorithms", Computer Security Symposium 2015 (CSS 2015), Nagasaki, Japan, Oct. 21 - 23, 2015.

[6] Jean Louis Deneubourg, Simon Goss, N Franks, Ana Sendova-Franks, Claire Detrain, and L Chrestien, "The Dynamics of Collective Sorting Robot-like Ants and Ants-like Robots", Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats, pp. 356-363, 1991.

[7] O.A. Mohamed Jafar and R. Sivakumar, "Ant-based Clustering Algorithms: A Brief Survey", International Journal of Computer Theory and Engineering, Vol.2, No. 5, pp.787-796, 2010.

[8] Urszula Boryczka, "Ant clustering algorithm", Intelligent Information Systems, pp. 455-458, 1998.

[9] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, Masashi Eto, Daisuke Inoue, and Koji Nakao, "Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation", Proceeding of the First Workshop on Building Analysis Datasets and Gathering Experience Return for Security, pp. 29-36, 2011.

Appendices

A Source Code of ACA Engine

```
import os
import math
import time
import numpy
import pandas
import random
import matplotlib
import numpy.random as nrand


import matplotlib
matplotlib.use('Agg')


import matplotlib.pylab as plt
from sklearn.preprocessing import normalize

import csv
import copy

cluster_labels = []


def write_cluster(fn, datas, plane):
  f = open(fn,'w')
  for i in range(plane.dim[0]):
    for j in range(plane.dim[1]):
      if plane.get_grid()[i][j] != -1:
        tmp = "%d,%d\n"%(i,j)
        f.write(tmp)
  f.close()

def draw_figure(fn):
  fig = plt.figure()
  ax = fig.add_subplot(1,1,1)
  x = []
  y = []
  f = open(fn)
  reader = csv.reader(f)
  for row in reader:
    x.append(int(row[0]))
    y.append(int(row[1]))
  ax.scatter(x,y)
  t = fn + ".png"
  plt.savefig(t)
  plt.close(fig)
```

```
def save_figure(fn, datas, plane):
  fig = plt.figure(figsize = (10,10))
  ax = fig.add_subplot(1,1,1)
  x = []
  y = []
  for i in range(plane.dim[0]):
    for j in range(plane.dim[1]):
      if plane.get_grid()[i][j] != -1:
        x.append(i)
        y.append(j)
  ax.scatter(x,y,s=0.01,color='black')
  plt.savefig(fn)
  plt.close(fig)


class AntColonyOptimization:
  def __init__(self):
    pass

class dataLoader:
  def __init__(self, DIR):
    self.f = open(DIR+"training_filtered.out","r")
    self.datas = []

  def readData(self):
    print "Read!"
    self.reader = csv.reader(self.f)
    self.data_cnt = 0

    pr_type = []
    services = []
    flag = []
    cnt = 0
    for row in self.reader:
      rID = float(row[-1])#ID
      row = row[:41]
      self.data_cnt += 1
      for i in range(len(row)):
        row[i] = float(row[i])

      row.append(rID)
      cluster_labels.append(1)
      row.append(cnt) #cluster_label
      self.datas.append(row)
      cnt += 1

    self.datas_nparray = numpy.array(self.datas)
    print "Read Finish! %d"%(self.data_cnt)


class Grid:
  def __init__(self, height, width, path, loader, rand_test=False):
    self.path = path
    # Store the dimensions of the grid
    self.dim = numpy.array([height, width])
```

```
    self.grid = numpy.full((height, width), -1, dtype=numpy.int)

    self.loader = loader

    if rand_test:
      self.rand_grid(0.25)
    else:
      self.place_instances(self.loader.data_cnt)
      write_cluster("cluster_before.csv",self.loader.datas,self)
      print "random place finish!"
    plt.ion()
    self.max_d = 0.001

  def rand_grid(self, sparse):
    for y in range(self.dim[0]):
      for x in range(self.dim[1]):
        if random.random() <= sparse:
          r = random.randint(0, 1)
          if r == 0:
            self.grid[y][x] = Datum(nrand.normal(5, 0.25, 10))
          elif r == 1:
            self.grid[y][x] = Datum(nrand.normal(-5, 0.25, 10))


  def place_instances(self, data_n):
    for i in range(data_n):
      while 1:
        r = random.randrange(0,self.dim[0])
        c = random.randrange(0,self.dim[1])
        if self.grid[r][c] == -1: #empty cell
          self.grid[r][c] = i
          break



  def matrix_grid_two(self):
    matrix = numpy.empty((self.dim[0], self.dim[1]))
    matrix.fill(0)
    for y in range(self.dim[0]):
      for x in range(self.dim[1]):
        if self.grid[y][x] != -1:
          matrix[y][x] = self.get_grid()[y][x].condense_two()
    return matrix

  def plot_grid(self, name="", save_figure=True):
    plt.matshow(self.matrix_grid(), cmap="RdBu", fignum=0)
    # Option to save images
    if save_figure:
      plt.savefig(self.path + name + '.png')

  def plot_grid_two(self, name="", save_figure=True):
    plt.matshow(self.matrix_grid_two(), cmap="RdBu", fignum=0)
    if save_figure:
      plt.savefig(self.path + name + '.png')
```

```
def get_grid(self):
  return self.grid

def get_diff(self, target, neighbor):
  diff = numpy.abs(self.loader.datas_nparray[target][:41] - self.loader.datas_nparray[
  neighbor][:41])
  return numpy.sum(diff**2)

def get_condense(self, target):
  return numpy.mean(self.loader.datas_nparray[target][:41])

def get_probability(self, d, y, x, n, c):
  # Starting x and y locations
  y_s = y - n
  x_s = x - n
  total = 0.0
  # For each neighbour
  for i in range((n*2)+1):
    xi = (x_s + i) % self.dim[0]
    for j in range((n*2)+1):
      if j != x and i != y:
        yj = (y_s + j) % self.dim[1]
        # Get the neighbour, o
        o = self.grid[xi][yj]
        # Get the similarity of o to x
        if o != -1:
          s = self.get_diff(d,o)
          total += s
  # Normalize the density by the max seen distance to date
  md = total / (math.pow((n*2)+1, 2) - 1)
  if md > self.max_d:
    self.max_d = md
  density = total / (self.max_d * (math.pow((n*2)+1, 2) - 1))
  density = max(min(density, 1), 0)
  t = math.exp(-c * density)
  probability = (1-t)/(1+t)
  return probability

def assign_cluster(self, d, y, x, n, c):
  y_s = y - n
  x_s = x - n
  major = -1
  cluster_con = []
  # For each neighbor
  for i in range((n*2)+1):
    xi = (x_s + i) % self.dim[0]
    for j in range((n*2)+1):
      # If we are looking at a neighbor
      if j != x and i != y:
        yj = (y_s + j) % self.dim[1]
        # Get the neighbor, o
        o = self.grid[xi][yj]
        # If o is not empty cell
        if o != -1:
          cluster_con.append(self.loader.datas[o][42])
```

```
    for i in cluster_con:
      if cluster_con.count(major) < cluster_con.count(i):
        major = i

    cluster_labels[self.loader.datas[d][42]] -= 1
    self.loader.datas[d][42] = major
    cluster_labels[major] += 1

class Ant:
  def __init__(self, y, x, grid):
    self.loc = numpy.array([y, x])
    self.carrying = grid.get_grid()[y][x]
    if self.carrying != -1:
      grid.get_grid()[y][x] = -1
    self.grid = grid

  def move(self, n, c):
    step_size = random.randint(1, 25)
    # Add some vector (-1,+1) * step_size to the ants location
    self.loc += nrand.randint(-1 * step_size, 1 * step_size, 2)
    # Mod the new location by the grid size to prevent overflow
    self.loc = numpy.mod(self.loc, self.grid.dim)
    # Get the object at that location on the grid
    o = self.grid.get_grid()[self.loc[0]][self.loc[1]]
    # If the cell is occupied, move again
    if o != -1:
      # If the ant is not carrying an object
      if self.carrying == -1:
        # Check if the ant picks up the object
        if self.p_pick_up(n, c) >= random.random():
          # Pick up the object and rem from grid
          self.carrying = o
          self.grid.get_grid()[self.loc[0]][self.loc[1]] = -1
        # If not then move
        else:
          self.move(n, c)
      # If carrying an object then just move
      else:
        self.move(n, c)
    # If on an empty cell
    else:
      if self.carrying != -1:
        # Check if the ant drops the object
        if self.p_drop(n, c) >= random.random():
          # Drop the object at the empty location
          self.grid.assign_cluster(self.carrying, self.loc[0], self.loc[1], n, c)
          self.grid.get_grid()[self.loc[0]][self.loc[1]] = self.carrying
          self.carrying = -1

  def p_pick_up(self, n, c):
    ant = self.grid.get_grid()[self.loc[0]][self.loc[1]]
    return 1 - self.grid.get_probability(ant, self.loc[0], self.loc[1], n, c)

  def p_drop(self, n, c):
```

```
    ant = self.carrying
    return self.grid.get_probability(ant, self.loc[0], self.loc[1], n, c)

def make_labels(loader,n_cluster,percent,filedir):
  n_large_clusters = n_cluster * percent
  n_large_clusters = int(n_large_clusters)
  large_clusters = []

  c_cluster_labels = copy.deepcopy(cluster_labels)
  for i in range(n_large_clusters):
    M = -1
    for j in range(len(c_cluster_labels)):
      if c_cluster_labels[j] > M:
        M = j
    large_clusters.append(M)
    c_cluster_labels[M] = -1

  n_normal = 0
  n_abnormal = 0
  n_total = 0

  c_datas = copy.deepcopy(loader.datas)

  for item in c_datas:
    if item[42] in large_clusters:
      item[42] = 0 #normal

      tmp = item[41]
      item[41] = item[42]
      item[42] = tmp

      n_normal += 1
    else:
      item[42] = 1 #abnormal

      tmp = item[41]
      item[41] = item[42]
      item[42] = tmp

      n_abnormal += 1

    n_total += 1

  print "n_normal = %d, n_abnormal = %d, n_total = %d"%(n_normal,n_abnormal,n_total)

  fn = filedir + "aca_result_"
  fn = fn + str(percent)
  f = open(fn,'w')
  c = csv.writer(f)
  for i in c_datas:
    c.writerow(i)
  f.close()
```

```
def optimize(height, width, ants, sims, n, c, outdir, cluster_prop_from, cluster_prop_to
    , freq=500, path="image"):
  """
  Main method for running the algorithm
  """
  # Load data
  loader = dataLoader("../datasets/kdd99/0/")
  loader.readData()
  # Initialize the grid
  grid = Grid(height, width, path, loader)

  # Create the ants
  ant_agents = []
  for i in range(ants):
    ant = Ant(random.randint(0, height - 1), random.randint(0, width - 1), grid)
    ant_agents.append(ant)
  for i in range(sims+1):
    for ant in ant_agents:
      ant.move(n, c)
    if i % freq == 0:
      s = outdir + "img/img"
      s = s + str(i).zfill(6)
      s = s + ".png"
      save_figure(s,loader.datas,grid)

      sc = outdir + "datas/data"
      sc = sc + str(i).zfill(6)
      sc = sc + ".out"
      f = open(sc,"w")
      cf = csv.writer(f)
      for item in loader.datas:
        cf.writerow(item)
      f.close()

      sc = outdir + "datas/cluster_data"
      sc = sc+str(i).zfill(6)
      sc = sc + ".out"
      f = open(sc,"w")
      cf = csv.writer(f)
      cf.writerow(cluster_labels)
      f.close()

      print s+" wrote!"

  n_cluster = 0
  for i in cluster_labels:
    if i > 0 :
      n_cluster += 1

  print "n_clusters = %d"%(n_cluster)

  cur_prop = cluster_prop_from
  while 1:
    make_labels(loader, n_cluster, cur_prop,outdir)
    cur_prop += 0.01
```

```
    if cur_prop > cluster_prop_to:
      break

  print "make_labels done!"

if __name__ == '__main__':
  global cluter_labels
  cluster_labels = []
  optimize(600,600,1000,500000,6,15,"./4_1_600_1000_1_15/",0.0,1.0,freq=25000,path="
    Video 8/")
```

ACA Engine Code - /aca/real_aca.py

B Source Code of DT Engine

```
from sklearn import datasets
from sklearn import tree

import numpy as np
import matplotlib.pyplot as pl

import csv


class SK_DT:
  def __init__(self,file_trn,file_tst,file_out):
    self.ftrn = file_trn
    self.ftst = file_tst
    self.fout = file_out

  def load(self):
    f = open(self.ftrn,'r')
    c = csv.reader(f)

    self.datas_trn = []
    self.labels_trn = []

    cnt = 0

    cnt = 0
    for row in c:
      row = row[:-1]

      for i in range(len(row)):
        row[i] = float(row[i])

      row[-1] = int(row[-1])
      tmp = []
      tmp.append(row[-1])

      row = row[:-1]

      self.datas_trn.append(row)
      self.labels_trn.append(tmp)
      cnt += 1
    f.close()

    #for testing
    cnt = 0
    ft = open(self.ftst,'r')
    ct = csv.reader(ft)

    self.datas_tst = []
    self.labels_tst = []

    for row in ct:
      row = row[:-1]
      for i in range(len(row)):
```

```
        row[i] = float(row[i])

      row[-1] = int(row[-1])
      tmp = []
      tmp.append(row[-1])

      row = row[:-1]

      self.datas_tst.append(row)
      self.labels_tst.append(tmp)
      cnt += 1
    ft.close()


  def train(self):
    #dt
    self.clf = tree.DecisionTreeClassifier()
    self.clf = self.clf.fit(self.datas_trn,self.labels_trn)
    self.out = self.clf.predict(self.datas_tst)

    #printing result
    fw = open(self.fout,'w')
    cw = csv.writer(fw)
    cw.writerow(self.out)

    fw.close()


def demo(fin,fin2,fo,fr,to):
  t = fr
  while 1:
    if t >= 1.0:
      break
    tmp = str(t)

    tmp_tr = fin + tmp
    tmp_out = fo + "dt_res_"
    tmp_out = tmp_out + tmp

    dt = SK_DT(tmp_tr,fin2,tmp_out)
    dt.load()
    dt.train()

    t += 0.01


if __name__ == '__main__':
  demo("./4_2_600_1000_1_15/aca_res/aca_res_","../datasets/kdd99/4/testing_filtered.out"
    ,"./4_2_600_1000_1_15/aca_res/dt_res/",0.0,1.0)
```

DT Engine Code - /aca/sklearn_dt.py

C Source Code of Traffic Filter

```
import csv


normal_list = ['normal.']
dos_list = ['back.','land.','neptune.','pod.','smurf.','teardrop.']
probe_list = ['ipsweep.','nmap.','portsweep.','satan.']
u2r_list = ['buffer_overflow.','loadmodule.','perl.','rootkit.']
r2l_list = ['ftp_write.','guess_passwd.','imap.','multihop.','phf.','spy.','warezclient.
    ','warezmaster.']

class dataset_filter:
  def __init__(self, fn,fo):
    self.fn = fn
    self.fon = fo
    self.f = open(fn,'r')
    self.cnt_n = 0
    self.cnt_d = 0
    self.cnt_p = 0
    self.cnt_u = 0
    self.cnt_r = 0
    self.cnt_a = 0
    self.datas_n = []
    self.datas_d = []
    self.datas_p = []
    self.datas_u = []
    self.datas_r = []

  def counting(self):
    reader = csv.reader(self.f)
    for row in reader:
      #row[41] = row[41][:-1]
      if row[-2] in normal_list:
        self.datas_n.append(row)
        self.cnt_n += 1
      elif row[-2] in dos_list:
        self.datas_d.append(row)
        self.cnt_d += 1
      elif row[-2] in probe_list:
        self.datas_p.append(row)
        self.cnt_p += 1
      elif row[-2] in u2r_list:
        self.datas_u.append(row)
        self.cnt_u += 1
      elif row[-2] in r2l_list:
        self.datas_r.append(row)
        self.cnt_r += 1

    self.cnt_a = self.cnt_d + self.cnt_p + self.cnt_u + self.cnt_r + 0.0

    print "self cnt_n = %d, cnt_d = %d, cnt_p = %d, cnt_u = %d, cnt_r = %d"%(self.cnt_n,
    self.cnt_d,self.cnt_p,self.cnt_u,self.cnt_r)

    self.total = self.cnt_n * (1.0/0.98)
```

```
    self.total_anomalies = self.total - self.cnt_n
    self.total_anomalies = int(round(self.total_anomalies))

  def calc_atnum(self):
    self.to_d = int(round(self.total_anomalies / 4.0))
    self.to_p = int(round(self.total_anomalies / 4.0))
    self.to_r = int(round(self.total_anomalies / 4.0))
    self.to_u = self.total_anomalies - (self.to_d + self.to_p + self.to_r)

    spare = 0
    if self.to_r >= self.cnt_r:
      spare = self.to_r - self.cnt_r
      self.to_r = self.cnt_r

    if self.to_u >= self.cnt_u:
      spare += self.to_u - self.cnt_u
      self.to_u = self.cnt_u

    if self.to_p >= self.cnt_p:
      spare += self.to_p - self.cnt_p
      self.to_p = self.cnt_p

    self.to_d += spare


  def filtering(self):
    fw = open(self.fon,"w")
    cw = csv.writer(fw)

    for i in range(len(self.datas_n)):#normal traffic
      cw.writerow(self.datas_n[i])

    cnt_a = 0
    cnt_u = 0
    cnt_p = 0
    cnt_d = 0
    cnt_r = 0

    for i in range(self.to_p):#probe
      cw.writerow(self.datas_p[i])
      cnt_a += 1
      cnt_p += 1

    for i in range(self.to_u):#u2r
      cw.writerow(self.datas_u[i])
      cnt_a += 1
      cnt_u += 1

    for i in range(self.to_r):#r2l
      cw.writerow(self.datas_r[i])
      cnt_a += 1
      cnt_r += 1

    for i in range(self.to_d):#dos
      cw.writerow(self.datas_d[i])
```

```
        cnt_d += 1


    print "cnt_n = %d, cnt_d = %d, cnt_p = %d, cnt_u = %d, cnt_r = %d"%(self.cnt_n,cnt_d
    ,cnt_p,cnt_u,cnt_r)
    print "total = %f, total_anomal = %d"%(self.total,self.total_anomalies)
    fw.close()


def dofilter(fn,fo):
  df = dataset_filter(fn,fo)
  df.counting()
  df.calc_atnum()
  df.filtering()


if __name__ == '__main__':
  dofilter("./0/training_before.out","./0/training_filtered.out")
  dofilter("./1/training_before.out","./1/training_filtered.out")
  dofilter("./2/training_before.out","./2/training_filtered.out")
  dofilter("./3/training_before.out","./3/training_filtered.out")
  dofilter("./4/training_before.out","./4/training_filtered.out")
```

Traffic Filter Code - /dataset/kdd99/df.py

D Source Code of Normalizer

```
import csv
import math

f = open('../datasets/kdd99/kddcup.data_10_percent_corrected_transed','r')
c = csv.reader(f)

fw = open('../datasets/kdd99/kddcup.data_10_percent_corrected_transed_normalized','w')
cw = csv.writer(fw)

sum = [0.0]*42
datas = []
for row in c:
  for i in range(len(row)-1):
    row[i] = float(row[i])

  datas.append(row)

f.close()
print len(datas[0])
print len(datas)

for i in range(len(datas[0])-1):
  ttmp = []
  for j in range(len(datas)):
    ttmp.append(datas[j][i])

  m = min(ttmp)
  M = max(ttmp)

  for j in range(len(datas)):
    if (M-m) != 0:
      datas[j][i] = (datas[j][i] - m) / (M - m)

for i in range(len(datas)):
  cw.writerow(datas[i])
fw.close()
```

Normalizer Code - /dataset/kdd99/normalizer.py

Summary

A Bio-inspired Intrusion Detection System for Unknown-attacks combining Ant Clustering Algorithm and Decision Tree

침입 탐지 시스템(Intrusion Detection System, IDS)는 특정 네트워크 혹은 시스템을 감시하며 사용자의 악의적인 행동을 탐지하는 시스템이다. IDS는 탐지 방식에 따라 Signature-based IDS와 Anomaly-based IDS로 나눌 수 있다.

사물 인터넷 시대가 다가오며 많은 기기가 네트워크에 연결됨에 따라 네트워크 트래픽이 폭증하고 있다. IDS를 훈련시키기 위한 Dataset을 만들기 위해 사람이 수동적으로 일일히 공격 여부에 관한 Label 만드는 작업은 매우 어려운 작업이다. 한편 현재 지속적으로 새로운 알려지지 않은 미지 공격이 출현하고 있다. 이에 IDS에 있어서 미지 공격을 사전에 탐지하는 능력은 필수적인 기능이 되었다. 미지 공격 탐지란 IDS가 사전에 어떠한 정보도 가지고있지 않은 공격을 탐지하는 연구 분야이다. IDS가 사전 정보를 가지고 있지 않은 공격을 탐지하기 위해서는 공격 여부에 대한 Label이 없는 Dataset에서도 스스로 정상 트래픽과 공격 트래픽을 구분할 능력이 있어야한다. 이러한 문제들을 해결하기 위해 Label이 없는 Dataset에서 스스로 학습할 수 있는 방법을 도출해내야 한다.

본 논문에서는 Clustering-based 모델의 알려지지 않은 공격 탐지를 위한 새로운 IDS를 제안한다. 제안하는 IDS는 개미군집 알고리즘(Ant Clustering Algorithm, ACA)과 의사결정트리(Decision Tree, DT)를 조합하였다. 제안하는 IDS는 Label이 없는 Dataset에서 스스로 학습이 가능하여 정상 트래픽과 공격 트래픽을 구분하기 위한 프로필을 생성하고 그에 기반하여 미지 공격을 탐지할 수 있다. 제안하는 IDS는 ACA 엔진과 DT 엔진으로 이루어져 있다. ACA를 이용하여 Label이 없는 Dataset 상에서 군집을 형성한다. 형성된 군집화 결과를 바탕으로 ACA 엔진은 각 Data instance들의 공격 여부를 구분한다. 그 후 ACA 엔진의 결과를 바탕으로 DT 엔진은 침입 탐지기를 훈련한다.

제안하는 IDS의 성능 평가를 위해 KDD Cup 1999 Dataset을 이용하였다. 성능 비교를 위한 지표로 탐지율(Detection Rate, DR), 오탐율(False Positive Rate, FPR), 정확도(Accuracy, ACC)가 이용되었다. 제안하는 IDS는 본 논문과 유사한 방식을 제안한 Hosseinpour 등의 방식[1]에 비해 월등히 높은 탐지율과 정확도를 보여주었다.

핵심어 : IDS, 미지 공격 탐지, 생체모방, 군집지능

감 사 의 글

이 논문을 완성하기까지 주위의 모든 분들로부터 수많은 도움을 받았습니다. 우선 많이 부족한 저를 학생으로 받아주시고 연구의 하나부터 열까지 모두 가르쳐주시고 또한 연구뿐만 아니라 전인적 지도를 아낌없이 해주셨던 지도교수님이신 김광조 교수님께 글로 표현할 수 없을만큼 많은 감사를 드립니다. 또한 바쁘신 와중에 귀한 시간 내어 학위논문심사위원을 맡아주시며 저의 학위논문이 더욱 가치있게 발전하도록 아낌없는 조언을 주신 박홍식 교수님, 김명호 교수님께도 큰 감사를 드립니다.

그리고 2년간 연구실 생활을 같이 하며 여러 조언을 해주시고 더욱 성장할 수 있게 도움주신 연구실 동기, 선후배인 박준정 형님, 정제성 형님, 안수현 형님, 김학주 형님, 최락용 형님, 홍진아, Aminanto Erza Muhamad 에게도 많은 도움을 받았습니다. 좋은 연구실 동료분들 덕분에 보람차게 2년을 생활할 수 있었던 것 같습니다.

이외에도 고마운 분들이 많습니다. 같이 전산학과에 입학하고 같은 수업을 들으며 막내뻘인 저에게 따뜻하게 대해주신 석사 동기분들께 감사를 드립니다. 특히 같은 조에 속해 많은 정을 나누며 대학원 생활을 잘 할 수 있도록 도움주신 해준이형, 원태형, 기원이형, 지성이형, 은비누나, 정표께 감사드립니다. 또한 2년 내내 지속적으로 관심가져주신 종명이형, 현우형, 준행이형, 미진누나께 감사드립니다.

항상 반갑게 맞아주며 잠시나마 즐겁게 시간을 보낼 수 있게 해준 친구들에게도 감사를 드립니다. 특히 중고등학생 시절부터 이어진 오랜 교감으로 언제 어디서든 편하게 만날 수 있었던 창명, 세진, 인섭, 현해, 범주, 석호, 민수, 승진이에게 감사드립니다. 또한 대전 생활에 대한 정보를 아낌없이 주었던 준홍이, 뜬금없이 찾아가도 잘 만나줬던 재현이에게도 감사드립니다. 그리고 학부생 시절부터 많은 과제와 프로젝트를 함께 수행하며 많은 신세를 진 범진이에게 많은 감사를 드립니다.

끝으로 오늘의 제가 있을 수 있도록 사랑으로 키워 주신 부모님과 친형 철민이형에게 무한한 감사를 드립니다. 부모님과 형의 따뜻한 사랑이 아니었다면 오늘의 제가 없다는 것을 늘 마음에 새기고 있습니다. 저의 이 작은 결실이 모든 분들께 조금이나마 보답이 되기를 바랍니다.

이 력 서

이 름 : 김 경 민

생 년 월 일 : 1991년 4월 25일

주 소 : 서울특별시 강서구 화곡8동 410-42 201호

E-mail 주 소 : saza12345@kaist.ac.kr

학 력

2007. 3. – 2010. 2. 광영고등학교

2010. 3. – 2014. 2. 서강대학교 컴퓨터공학과 (B.S.)

2014. 3. – 2016. 2. 한국과학기술원 전산학부 (M.S.)

경 력

2012. 3. – 2012. 6. 서강대학교 기초공학설계 조교

2012. 9. – 2012. 12. 서강대학교 C프로그래밍 조교

2012. 12. – 2013. 1. 서강대학교 C프로그래밍 조교

2013. 3. – 2013. 6. 서강대학교 기초공학설계 조교

2015. 3. – 2015. 6. 한국과학기술원 정보보호개론 추가조교

2015. 7. – 2015. 8. 한국과학기술원 컴퓨팅보안 일반조교

2015. 9. – 2015. 12. 한국과학기술원 고급정보보호 추가조교

연 구 과 제

2014. 4. – 2016. 2. 생체모방 알고리즘(Bio-inspired Algorithm)을 활용한 통신기술 연구

2014. 7. – 2015. 6. Intrusion Detection System for Critical Infrastructures Using Big Data Analytics

2015. 4. – 2015. 8. 국가재난안전통신망의 장기간 보안성을 보장하는 산업 발전 전략

2015. 4. – 2015. 11. 포스트 양자암호 시스템의 안전성 분석기술 연구

2015. 5. – 2016. 4. 양자컴퓨터 공격에 안전한 새로운 래티스 기반 완전 준동형 서명 방식 설계 및 안전성 분석

2015. 6. – 2016. 3. Authenticated Security for Smart-Grid Using Big-Data Analytics

연 구 업 적

1. 정제성, **김경민**, 김학주, 박준정, 안수현, 이동수, 최락용, 김광조, 김대영, “경량 암호를 이용한 IoT Secure SNAIL 플랫폼 구성(Ⅰ)”,2014 한국정보보호학회 동계학술대회(CISC-W’14), 한양대학교, 서울, 2014.12.06.
2. **김경민**, 김광조, “침입 탐지 시스템의 알려지지 않은 공격 탐지에 대한 최신 연구 비교”,2015 한국정보보호학회 하계학술대회(CISC-S’15), 한국과학기술원, 대전, 2015.06.25-26.
3. **김경민**, 김광조, “개미군집 알고리즘과 인공신경망을 이용한 미지공격의 탐지기법”, 한국정보보호학회 충청지부 학술대회, 서원대학교, 청주, 2015.10.16 - [**우수논문**].
4. **Kyung-min Kim**, HakJu Kim, Kwangjo Kim, “Design of an Intrusion Detection System for Unknown attacks based on Bio-inspired Algorithms”, Computer Security Symposium 2015(CSS 2015), Nagasaki, Japan, 2015.10.21-23.
5. **김경민**, 홍진아, 김광조, “개미군집 알고리즘과 의사결정트리를 활용한 알려지지 않은 공격 탐지 시스템”,2015 한국정보보호학회 동계학술대회(CISC-15’W), 서울여자대학교, 서울, 2015.12.05.
6. **Kyung-min Kim**, Jina Hong, Kwangjo Kim, and Paul D. Yoo, “Evaluation of ACA-based Intrusion Detection Systems for Unknown-attacks”,2016 Symposium on Cryptography and Information Security (SCIS 2016), Kumamoto, Japan, 2016.01.19-22.

7. Muhamad Erza Aminanto, HakJu Kim, **Kyung-min Kim**, and Kwangjo Kim, “Another Fuzzy Anomaly Detection System Based on Ant Clustering Algorithm”,2016 Symposium on Cryptography and Information Security (SCIS 2016), Kumamoto, Japan, 2016.01.19-22.

8. 김광조, 김학주, **김경민**, 최락용, 홍진아, “군집 지능과 기계학습 알고리즘을 이용한 적응형 미지 공격 탐지 시스템의 방식 및 장치”, 특허 [출원중].