Journal of Theoretical and Applied Information Technology
10th March 2016. Vol.85. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

PERMISSION BASED MALWARE ANALYSIS FOR ANDROID APPLICATIONS USING SELF-ORGANIZING MAPS

[1]AHMED BEN AYED
[1]Colorado Technical University, Department of Engineering and Computer Science
E-mail: [1]aben@my.bellevue.edu

ABSTRACT

Android is an open source platform based on Linux kernel; it is one of the first operating systems that use a permission mechanism to control access to resources. The permission mechanism is fine gained and can control what a particular process could and could not perform. Therefore, these permissions should be monitored closely to make sure they are not assigned to the wrong application. This study is not intended to create an anti-malware solution, instead it uses the permissions to classify and categorize android applications. This paper is offering a novel way of using the self-organizing map to study a set of malware application and try to find a pattern of permission requests. This pattern could be used to analyze the application and compare it against the pattern identified earlier using the self-organizing map.

Keywords: *Android Permissions, Machine Learning, Self-Organizing Map, Android Security*

1. INTRODUCTION

This paper will focus on unsupervised learning techniques and in self-organizing maps in particular. The first part of this paper will be a brief about the android operating system and its security model, the second part will be an illustration of a novel theoretical method of analyzing permissions in android applications. The main objective of this study is to investigate which permissions are most popular in Android malware applications. We believe this study could reveal permission pattern usage, and could identify correlations between different requested permissions in malware applications.

2. ANDROID OPERATING SYSTEM

The Android operating system was designed to offer unrestricted use of the device without neglecting the security side of it. It is an open source platform that supports third party applications [1]. The Android security model is built on a very solid foundation, however it still pose drawbacks [2].

2.1 Android Security Model

The Android system is designed as a multi process system where each application is running on its own and has no access to other applications. Each application has its own space and its own resources and has zero interaction with the rest of the processes or applications. Generally speaking, Linux enforce security between application and the system at the process level [3]. The Android system has permission mechanism that determines whether grant access to an application or not. However, the android operating system doesn't assign permissions to each application, but it leaves the approval to the user who grants the permission to the application at the time of installation. This process make the system vulnerable to user's knowledge which is usually limited and not technical oriented. Therefore, Android doesn't have any security measures to determine which permission should be granted and which one should be denied, and it the application needs the permission to function properly or not.

2.2 Android Permissions

The security by permission mechanism is poorly documented and usually misused by third party developers. None of the application has permission to perform any operation that could harm other applications, the user or the operating system. All Applications are required to use a certificate whose private key is kept by the party that created the application. Those certificates are used to identify the party who developed the application; however,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

certificates could be self-signed what makes it untraceable with no possible link to the developer.
Permissions are the primary form of security used by the Android Operating system to assure that no application is granted access to more re-sources than what it actually needs.
According to google, Android 4.4 offers 152 per-missions, 62 of them are either "signature permissions" or "signature Or System permissions" which are available only to application signed with the same certificate as the installed Android operating system. The rest of the permissions are either "Normal Permissions" or "Dangerous Permissions" and they are available to third party developers to use to access the device resources. Normal permissions are granted to any application, they can't harm the device or the user, but they could annoy them [4].

2.3 Malware Attacks on Android Systems

Threats to mobile phones are increasing at an alarming rate, according to [5] over two hundred million malicious program was created in 2010. The below table shows the most used malicious programs that affect smart-phones.

Table 1: Most Used Malware Applications

Malware type	Example	Description
Spyware	Stuxnet	Collect, use, spread sensitive information without the consent of the user. The information could be used for legit activities as advertisement or illegal activities as social engineering.
Worm	XXshenqi.apk	Malware that self-replicates and spreads via mobile network (MMS, SMS) and usually does not require any user interaction.
Botnet	Aurora	Also known as a zombie army, it is composed of many devices that have been set up to forward spam and viruses to other devices.
Toolkit	Phoenix toolkit	An application that is used to lunch a widespread attack on networked mobile devices [5].
Trojan	Gingermaster	Applications that pose as a legit app, it needs the user interaction to be activated.

3. MACHINE LEARNING

Machine learning is a subfield of computer science that research and study learning systems, the application could be applied to many engineering fields. Machine Learning is an algorithm that could study and learn from data without any human interaction [6]. It could be identified as an intersection between statistics, computer science, engineering and optimization [7]. Until late fifties, scholars believed that to estimate an unknown functional dependency, a finite number of parameters are needed, however, during the sixties, scholars were able to discover that using some general properties of a set of functions to which a an unknown dependency belongs could solve the problem. A statistical learning theory could be used to estimate the best approximation of the dependency [8]. There is two types of Machine learning, the super-vised learning, and the unsupervised learning. The supervised approach needs pre-classified data to permit learning [9]. The machine is given inputs and its outputs to let the machine learn how to produce the outputs, then the machine is given an input and expected to predict the output. This could be equivalent to regression analysis in statistics. In unsupervised Learning, the machines analysis the input, and try to come up with a patterns in the data to predict a category without any supervised target. Clustering, Self-organizing maps, and Vector quantization are considered unsupervised learning techniques. In this study we are going to focus on the Self-Organizing Maps.

4. SELF-ORGANIZING MAPS

Self-Organizing maps also known as the Kohonen maps are a class of neural networks invented by Teuvo Kohonen for the purpose of data classification [10]. It is considered a vector quantization method that places the prototype vector on a low dimensional grid in an ordered fashion [11]. The Self Organizing map use unsupervised training, in with the network learn to generate its own classifications without any external help [12]. This assumes the inputted data is broadly defined by common features, and that the network will be able to identify those features across the range of input patterns [13].

Journal of Theoretical and Applied Information Technology
10th March 2016. Vol.85. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.1 Structure of the Self-Organizing Maps

The structure of the self-organizing map typically has 2 dimensions, and depending on the dimensions of the node lattice, each neuron will have a variable number of connected neurons, but in our case (two dimensions) it would be 4 immediately connected neurons. However, each neuron is fully connected to all the source unites in the input layer as shown in figure 1 below.

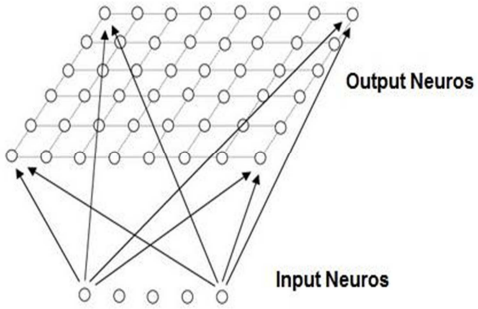

Figure 1: Structure Of The Self-Organizing Map

The structure of the self-organizing map is affected by the neighborhood relations that connect neurons to each other's, which dictates the topology or the structure of the map [11]. While working with a 2-dimensional case, the neurons of the map can be arranged on either a rectangular or a hexagonal lattice as displayed in Figure 3.2 below. However, hexagonal is suggested because of its effective visual display [10].

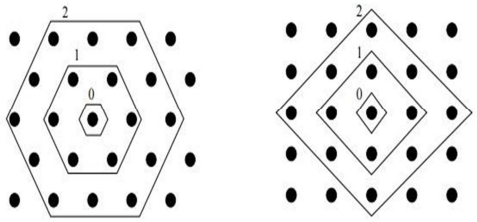

Figure 2: Two Example Of Topological Neighbrhood [11]

As explained by [11] the self-organizing map could be explained as a net that is spread to the data cloud. The algorithm dislocates the weight vectors so they cover the data cloud and the map gets organized. As shown in figure 3.3, when the data is presented to the Self organizing Map, the new weight vectors are weighted averages of the data vectors. As shown in Figure 3, the best matching unit and its neighbors gets updated towards the input sample marked with x.

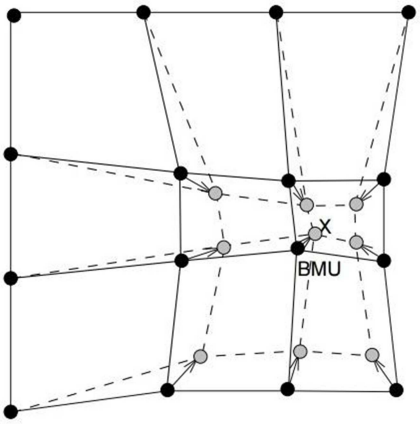

Figure 3: The Solid And Dashed Lines Correspond To Situation Before And After Updating, Respectively [11].

4.2 The Self-Organizing Maps Algorithm

The self-organizing map is achieved by mapping the input vectors with similar values onto neighboring output neurons [14].
The input layer represents input vector data. Usually the network has one layer of n units arranged in lattice; the prototype vector could be represented as:
The self-organizing map is achieved by mapping the input vectors with similar values onto neighboring output neurons [14].
The input layer represents input vector data. Usually the network has one layer of n units arranged in lattice; the prototype vector could be represented as:
The output layer is usually a one or two dimensional map with a possibility of a higher dimension. In our case we will be using the one dimensional map. A weight vector is associated with every neuron; every output neurons is connected to the input neuron with that weight vector.
The first step is going to be initializing the neuron weight as shown in (1):

$$X_i(t) = (X_i(t), X_i(t), \dots X_i(t)) \qquad (1)$$

The output layer is usually a one or two dimensional map with a possibility of a higher dimension. In our case we will be using the one dimensional map. A weight vector is associated with every neuron, every output neurons is connected to the input neuron with that weight vector.
The first step is going to be initializing the neuron weight as shown in (2):

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$W_i(t) = [W_{i1}(t), W_{i2}(t), \dots w_{in}(t) \quad (2)$$
$$for\ i = \{1, 2, 3, \dots, n\}$$

The self-organizing maps use a competitive learning algorithm (Sun, 2000), at each step it choose the winning neuron on the map according to its distance measures. At each output neuron, the Euclidean distance is used as the criterion to compare the input vector and the weight vector. The Euclidean distance could be defined using the formula (3):

$$\left||X(t) - W_i(t)|\right| = \sqrt{\sum_{j=1}^{n}\left(x_j - w_{ij}\right)^2} \quad (3)$$
$$for\ i = \{1, 2, 3 \dots, n)$$

The winning neuron of the competition is selected using the following equation (4):

$$\left||X(t) - W_c(t)|\right| = min_i\{\left||X(t) - W_i(t)|\right|\}$$
(4)
$$for\ i = \{1, 2, 3 \dots, n\}$$

After the winning neuron c has been selected, the weight vector neuron is updated in addition to its neighboring neurons. The weight update function is shown below (5):

$$W_i\,(t+1) = W_i(t) + \alpha(t)h_{ci}(t)[X(t) - W_i(t)](5)$$

For (t) is the learning rate parameter, and h_ci (t) is the neighborhood function defined below (6):

$$h_{ci}(t) = h(\left||r_c - r_i|\right|, t)\alpha(t) \quad (6)$$

The learning rate function (t) is a decreasing function of time. The neighborhood function on the Gaussian form is (7):

$$h_{ci}(t) = \exp\left(-\frac{||r_c - r_i||^2}{2\sigma^2(t)}\right)\alpha(t) \quad (7)$$

In summary the self-organizing maps algorithm could be presented in the pseudo-code below:

```
Do
{   Adjust the neighborhood radius
    Adjust the learning rate
    Randomize the presentations order of the
data
    For each data item
    {   Find the closest matching prototype
        For all cells within the
neighborhood radius
        {   Adjust prototype weights of
each
neuron based on distance  and learning rate
        }
    }
    evaluate the termination criteria
}
While ( Termination criteria is not met )
```

A Pseudo-Code Of The Self Organizing-Maps Algorithm

5. METHEDOLOGY AND DATA COLLECTION

In this work we are going to define each android application as an input pattern. The application is represented in a string of bits as shown below:

Where n is the total number of permissions found in the data set and, permission is either equal to 0 or 1. As an example let's suppose the Android application App_Example is represented as fellow:

$$App: \{permission_{i1}, permission_{i2}, \dots . permission_{in}\}$$

Formatting our dataset as such will make us able to use the self-organizing map to create a 2 dimensional visualization of permissions used in the data set. The algorithm will give us an idea on which permissions are used most, and could give us an idea on which permissions are usually used in combination. The algorithm will cluster the applications using the same permissions into the same region. A Unified Distance Matrix (U-Matrix) is going to be used to display the cluster structure. The representation uses the Euclidean distance explained earlier in section 3.2 to visualize the data in a 2 dimensional space. A sample U-matrix is showing on figure 5 below.

Journal of Theoretical and Applied Information Technology
10th March 2016. Vol.85. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

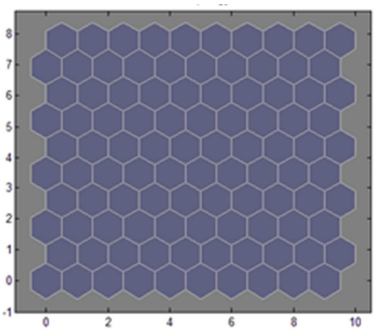

Figure 4: An Empty U-Matrix

Malwares were collected from different android market stores and forums. We successfully collected 1241 applications. To make sure the apps are malicious we run all apps through three commercial anti-viruses, Kaspersky, Avira, and Avast. 100 applications got identified as malicious by all three anti-viruses, only those were used in our study. To extract the permissions from our dataset, a predefined tool "aapt" was used.

6. RESULT AND DISCUSSION

Our malware set contains 77 different permissions, most of the permissions are either Dangerous or Normal permissions, 10% of the permission requested were labeled as System or signature and are prohibited to use by third party developers. Four special permissions were used; those permissions do not behave like dangerous and normal permissions, and should not be used in most applications [15].

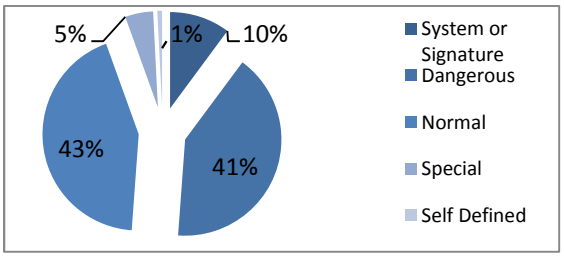

Figure 5: Permission Used In The Dataset

The most 20 used permissions were mostly dangerous permissions, the first and most used permission is the internet permission which is considered a normal permission but widely used by malware application to steal data from infected devices.

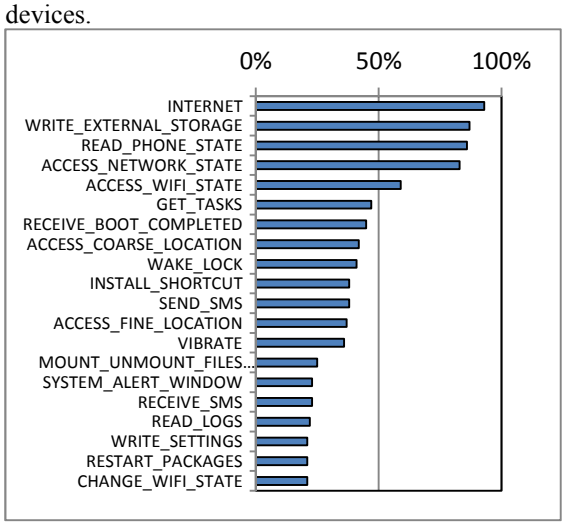

Figure 6: Most Used Permisions

Figure 6 above shows how the first 20 most requested permissions are divided between malware applications. It is found that the most used permissions are considered dangerous, and only 3 normal permissions were requested; the internet, vibrator, and Received_Boot_Completed. Seven of the most requested permissions were chosen to cluster using the self-organizing maps, MatLab 7.0 was used to train and print SOMs.

Figure 7 below shows the SOM for the Internet permission, light shades indicate that the permission was requested in that region; in contrary the dark shades indicate no use of the permission. The light shades cover most of the SOM what means that the internet permission was requested by most of the malware in our dataset (93%). The internet permission allows applications to send HTTP requests to all domains and allows the device to connect to all ports (Felt *et al*, 2010) what makes the permission suitable for malicious applications.

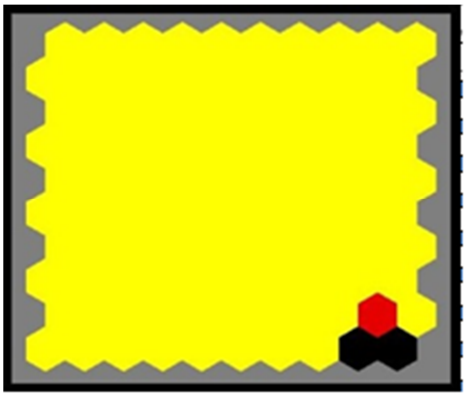

Figure 7: Component Plane Visualization Of The Internet Permission

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Write_External_storage allows the malware to write to external storage as an SD card. This permission is considered dangerous. As shown in figure 8, most malicious apps requested this permission (87%).

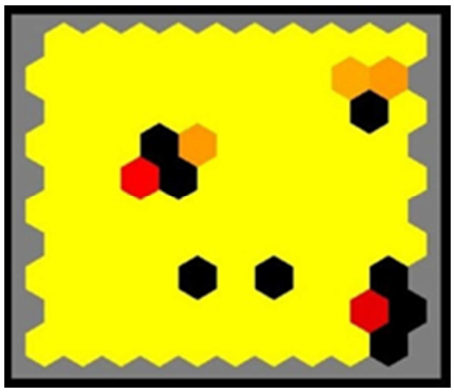

Figure 8: Component Plane Visualization Of The Write_External_Storage Permission

Understanding the permission request pattern in malicious applications is a very complicated task and could help the researcher find an efficient way of detecting that malware. The correlation we found between different permissions are specific to our malware dataset, however, the dataset contains different malware from different regions of the world and different families, and we believe the result could be generalized on the whole Android malware population.

Figure 9 below represent SOM representation for all 77 permissions found in our dataset. When overlooking at the whole output, it is very noticeable that some outputs are very similar to each other what can reveal a correlation between different permissions. If different permissions are correlated, it is most likely to be requested together most of the time.

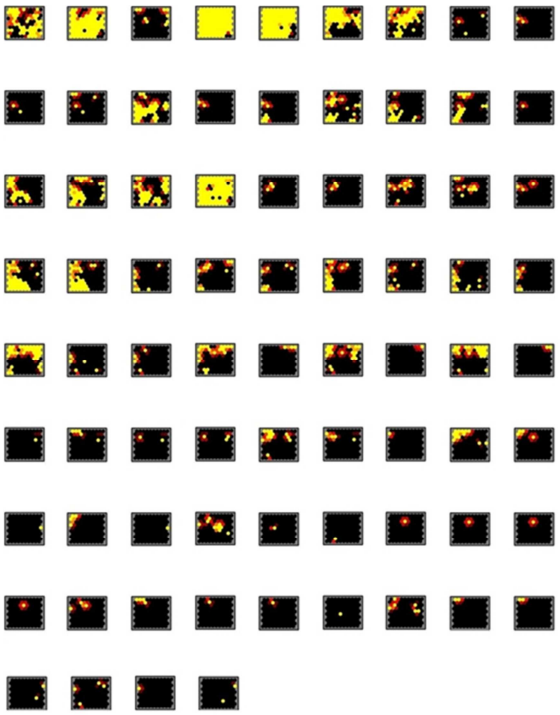

Figure 9: SOM Representations Of All Permissions In The Dataset, The Light Areas Represent The Area Where Permissions Were Requested.

Looking at different outputs, we easily identified 5 different correlations between our set of permissions. Similar component plane visualizations mean the permissions are correlated.

As shown in figure 10 there is a clear correlation between the four different permissions. This implies that those permissions usually get requested together. Any applications having access to those permissions will be able to receive MMS (RECEIVE_MMS), monitor and delete messages without showing it to the user (RECEIVE_WAP_PUSH), Also it change and enforce APN setting to be able to send messages in case APN is disabled and allows its activities to be persistent (PERSISTANT_ACTIVITY). Applications that have capabilities of managing, sending, and reading messages without the knowledge of the user are considered dangerous [2].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

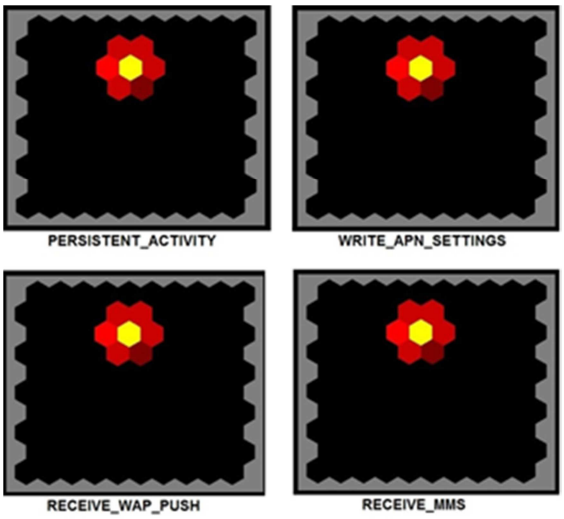

Figure 10: Four Different Permissions Showing Similar Component Plane Visualizations

Figure 11 shows different 3 permissions commonly requested together, the WRITE_SETTING permission is considered a signature permissions and should only be used by system applications, the same applies for the UPDATE_PHONE_STATISTICS, however finding those applications in a 3rd party application should raise a flag.

Figure 11: Three Different Permissions Showing Similar Component Plane Visualizations

The most used permissions in our dataset conclude also that there is a clear correlation between those four permissions. If we look closely at figure 12 below, we conclude that the yellow shades are mostly present at the same spots; the same applies for the black and darker shades.

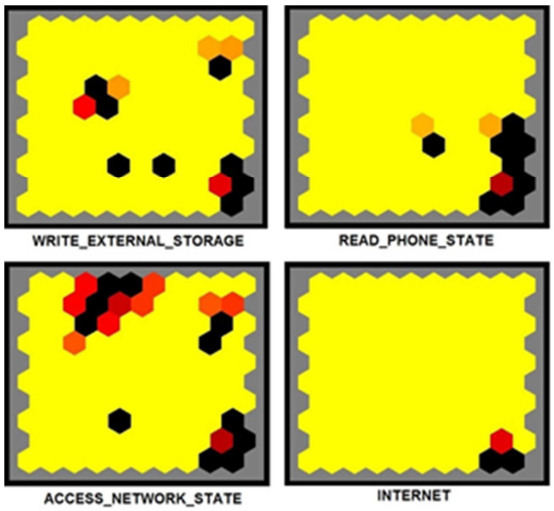

Figure 12: Four Different Permissions Showing Similar Component Plane Visualizations For The Most Used Permissions

7. CONCLUSION

In this paper we present a method of classifying or characterizing android malicious application based on permissions requested, the method could be used to identify malware application or to classify them into categories. This method could be used to analyze different android market-places to determine how dangerous its applications are. Also, it is a suitable method to find correlations between different permissions and identify dangerous combinations. The analysis in this study helped identifying permissions used by malicious applications using a real world dataset. It did as well show the correlation between different permission requested.
Although this research was carefully prepared, we are still aware of its limitations.. The research was conducted on a relatively small dataset of one hundred samples, and the malicious applications used were not classified into different malware families, and were treated as one set of data. Future research could consider separating the dataset into different malware families, and analyze each family separately.

REFRENCES:

[1] Ben Ayed, A., 2015, A literature Review on Android Permission System, International Journal of Advanced Research in Computer Engineering & Technology, Volume 4, Issue 4, pages 1520-1523.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[2] Enck, W., Ongtang, M., McDaniel, P., 2009, Understanding Android Security, Security & Pri-vacy IEEE, Volume 7. Issue 1, Pages 50-57.

[3] Dawson, M., Wright, J., Omar, M., 2015, Mobile Computing and Wireless Networks: Concepts, Methodologies, Tools, and Applications, Infor-mation Science Reference, pages 1103-1123.

[4] Felt, A., Chin, E., Hanna, S., Song, D., Wagner, D., 2011, Android permissions demystified, In Proceedings of ACM Conference on Computer and Communications Security, pages 627-637.

[5] Symantec, Inc., 2011, Internet Security Threat Report, Retrieved from http://www.symantec.com/content/en/us/enterpr ise/other_resources/b-istr_main_report_2011_21239364.en-us.pdf

[6] Kohavi, R., Provost, F., 1998, Glossary Of terms: Special Issue on Applications of Machine Learning and the knowledge discovery process, Kluwer Academic Publishers, pages 271-274.

[7] Jain, A.K., Murty, M.N, Flynn, P.J, 1999, Data Clustering: A review, ACM Computing Surveys, Volume 31, Issue 3, pages 264-323.

[8] Vapnik, V., 1995, The nature of Statistical Learning Theory, Springer.

[9] Hodge, V., Austin, J., 2004, A survey of Outlier Detection Methodologies, Artificial Intelligence Review, Volume 13, Issue 18, Pages 1-43.

[10] Kohonen, T., 1998, The Self Organizing Map, Neurocomputing, Volume 21, Issue 3, pages 1-6.

[11] Vesanto, J., Himberg, J., Alhoniemi, E., Par-hankanges, J., 1999, Self-Organizing Map in Matlab: The SOM Toolbox, Procedings of the matlab DSP conference, pages 35-40.

[12] Thomson., R., Emery, W., 2004, Data Analysis Methods in Physical Oceanography, 2nd Edition, Elsevier.

[13] Bulinaria, J., 2004, Introduction to Neural Networks, retrieved from http://www.cs.bham.ac.uk/~jxb/NN/l1.pdf

[14] Corttrell, M., Fort, J., Pages, G., 1995, Two or three things that we know about the Kohonen algorithm, Proceedings of European Symposium on Artificial Neural networks Brussels, Belgium, pages 235-244.

[15] Android, 2015, System Permissions, Retrieved from http://developer.android.com/guide/topics/secur ity/permissions.html#normal-dangerous