



AUTOMATED LOG ANALYSIS USING AI: INTELLIGENT INTRUSION DETECTION SYSTEM

A project report submitted to the School of Informatics and Innovative Systems in partial fulfilment of the requirements for the award of the

BACHELOR OF SCIENCE

COMPUTER SECURITY AND FORENSICS

JARAMOGI OGINGA ODINGA UNIVERSITY OF SCIENCE
AND TECHNOLOGY

Joseph Gitau Mbugua

I132/0886/2013

DEC 2016

Supervisor

Prof. A.J Rodrigues

Declaration

Declaration

This is to certify that the project report entitled **Automated Log Analysis Using Ai: Intelligent Intrusion Detection System** submitted by **Joseph Gitau Mbugua** of registration number **I132/0886/2013** in partial fulfilment for the award of the Degree of Bachelor of Science in Computer Security and Forensics to Jaramogi Oginga Odinga University of Science and Technology .

This project is my original work and has not been presented for an award of a diploma or conferment of a degree in any other university or institution.

JOSEPH GITAU MBUGUA

ADM. No: I132/0886/2013

Signature.....

Date.....

Declaration by supervisor

This project report has been submitted for examination with our approval as the candidates' University Supervisor.

PROF. A. J. RODRIGUES

Signature.....

Date.....

Dedication

This project is dedicated to the devoted team of PyData, SendTex and the university faculty for their part in making this project a success. My lovely daughter and her mother for their support and enduring love plus the trouble that little angel always causes.

Acknowledgement

I am indebted to Jaramogi Oginga Odinga University of Science and Technology for giving me a chance to this course. I wish to thank my classmates and the School of Informatics and Innovative Sciences staff for having contributed in formulation of ideas and for providing a suitable working environment towards the completion of this proposed work. In particular, I would like to thank my supervisors Prof. A. J. Rodrigues for his constant guidance and beneficial input that geared me towards the development and completion of this project.

Abbreviations

IDS – Intrusion Detection System

SQL - Structured Query Language

HTML – Hypertext Mark-up Language

Abstract

The maintenance of web server security, availability, integrity and confidentiality has never been such an overbearing task as is today. With threats coming from hardware failures, software flaws, tentative probing and worst of all malicious attacks. Analysing the server logs to detect suspicious activities is regarded to as key form of defence. However, the sheer size of server logs makes human log analysis challenging. Additionally, the traditional intrusion detection systems rely on methods based on pattern-matching techniques which cannot developers cannot maintain based on the high rates at which new and never seen before attack techniques are launched each and every day.

The aim of this project is to develop an intelligent log based intrusion detection system that can detect known and unknown intrusions automatically. Under a data mining framework, the intrusion detection system is trained with unsupervised learning algorithms specifically the k-means algorithm and the One Class SVM (Support Vector Machine) algorithm. The development of the system was time constrained and limited to machine generated logs due to lack of real access_log files. However, the system's development went smoothly and proved to be up to 85% accurate in detecting anomalous log patterns within the test logs. However, much work still needs to be done to make this a better Intrusion Detection System (IDS) in that a real-time analysis module could be incorporated and a function to retrieve logs from remote location.

Table of Contents

Declaration	ii
Dedication	iii
Acknowledgement	iv
Abbreviations	v
Abstract	vi
Chapter 1: Introduction	1
1.0 Introduction	1
1.1 Motivation	2
1.2 Project Objectives	2
1.2.1 Learning and Detection	2
1.2.2 Generality	2
1.3 Report Organization	3
1.3.1 Literature Review (Background and Related Works)	3
1.3.2 Methodology (Theoretical Aspects)	3
1.3.3 Development	3
1.3.4 Discussion	3
1.3.5 Conclusion	3
Chapter 2: Literature Review	4
2.0 Background	4
2.1 Intrusion	4
2.2 Logs	4
2.3 Intrusion Detection Methods	5
2.3.1 Pattern Matching	6
2.3.2 State-full Pattern Matching	6

2.3.3 Protocol Decode-Based Analysis	6
2.3.4 Heuristic-Based Analysis	6
2.3.5 Anomaly Detection	6
2.4 Data Mining Approaches	7
2.4.1 Supervised learning.....	7
2.4.2 Unsupervised learning	7
2.5 Novelty Detection	9
2.5.1 Probabilistic/GMM approaches	9
2.5.2 Non-parametric approaches	10
2.5.3 Neural network based approaches	11
2.7 Feasibility Study	12
Chapter 3: Methodology	13
3.1 Kmeans	13
3.2 One Class SVM	14
3.3 Text Vectorization and Feature Extraction.....	15
Chapter 4: Development	16
4.1 Introduction.....	16
4.2 Philosophy	16
4.2.1 Experimental Approach	16
4.2.2 Modularity.....	16
4.2.3 Rapid Development	16
4.3 Programming Languages	16
4.4 Modules.....	17
4.4.1 Pre-processing module.....	18
4.4.2 Known Attacks Analysis Module.....	18

4.4.3 Machine Learning and Outlier Detection module	19
4.4.4 Notification module	20
4.4.5 Real-time Detection Module.....	20
4.5 Related packages utilised.....	20
4.5.1 Scikit-learn.....	20
4.6 Chapter Summary	21
Chapter 5: Discussion	22
5.1 Experiment Design.....	22
5.1.1 Performance measures	22
5.2 Experiments	22
5.2.1 Experiment 1 (KMeans Clustering).....	22
5.2.2 Experiment 2 (One class SVM).....	24
Chapter 6: Conclusion.....	26
6.1 Achievements.....	26
6.1.1 Modularity of the System.....	26
6.1.2 Machine Learning and Detection.....	26
6.2 Problems	26
6.2.1 Lack of Data.....	26
6.2.2 Processing power	27
6.3 Limitations	27
6.3.1 Software development	27
6.3.2 Definition of the Norm (Normality definition).....	27
6.3.3 Real-time detection	27
6.4 Future Works.....	28
6.5 Chapter Summary	28

References.....	29
Appendix.....	31
Appendix I: Screenshots.....	31
Appendix II: User Manual.....	34
Environment/Requirements.....	34
Initialization.....	35
Appendix III: Project Management.....	36

Chapter 1: Introduction

1.0 Introduction

In an ever advancing and fast developing field, technology has become cheaper, easier to develop, and deploy. Unfortunately, this has also made probing, and attacking servers cheaper and easier to do. It is therefore vital to ensure that web servers are alert and hence secure against any form of attack.

Server logs have been used to confront failure either hardware or software, record notices, warnings and errors to ensure that system administrators can recover or at least know the cause the event of a system failure. Log recording has also acted as a form of defence against human attacks where predetermined techniques such as SQL injections can be easily identified.

On an average web server receiving traffic of at least 1000 unique visits a day generates a huge log that cannot be analysed manually. Now take the same web server and place it in a large company receiving over 10000 unique visits a day. The sheer size of the log file would turn down most system administrators as it is practically impossible to inspect as a human. Most log based intrusion detection systems on the market are pattern-matching technique based, that is, they compare the log entries to a set of predefined patterns that had been manually updated by security experts. Though this approach is effective in determining attacks of known patterns, the hassle is that for each new attack the system is defenceless and it takes the security experts a lot of time and effort to update the new patterns to the intrusion detection system.

From this point of view, current intrusion detection systems are far from intelligent in that they exclusively rely on human intervention to operate effectively. Therefore, more advanced intrusion detection systems are highly desirable. These systems should be capable of detecting known and unknown intrusions intelligently and automatically, distinguishing normal network activities from those abnormal and possibly malicious ones without or with minimum human intervention.

1.1 Motivation

Recently, some researchers and programmers utilizing data mining algorithms applied to log based intrusion detection systems came up with an effective anomaly detection based intrusion detection system that relied on nothing more other than the inflowing stream of logs to determine what is normal and what is not (possibly an attack). Those algorithms are based on supervised learning. That is to say, they are trained, other than being explicitly programmed, on data sets with labels indicating whether the instances are pre-classified as attacks or not. However, the techniques seemed cumbersome as manually labelling the large volumes of server data mostly over 1GB log files was expensive and difficult. This is what inspired the approach of unsupervised machine learning where no labels are pre-set hence the system is left to determine what is an attack and what is not.

With no requirement for class labels, unsupervised learning algorithms seemed to solve this problem. A broad explanation of approach to intrusion detection systems is that when an intrusion detection system becomes “familiar” with the data through the unsupervised learning algorithms, it is likely to detect “abnormal” data when they come in. Many of which are malicious.

1.2 Project Objectives

This project aims to develop and implement a system that can learn the normal state and nature of a web server where installed and dynamically identify anomalies bringing them to the system administrators attention. The strategy and supporting objectives are described as follows:

1.2.1 Learning and Detection

Based on the unsupervised learning algorithm used, the system will be able to learn from the current states of the server (or when the server was working optimally). Detect the anomalies in the logs and hence alert the system administrator.

1.2.2 Generality

Based on unsupervised learning, detecting abnormal activities shall be executed automatically without too much human intervention.

1.3 Report Organization

The structure listed below is used in the rest of this report.

1.3.1 Literature Review (Background and Related Works)

In chapter 2, a broad background regarding this project is introduced. This includes the seriousness of web server intrusion problem, the format of logs, a review of today's intrusion detection methods, data mining approaches that are to be employed, some novelty detection approaches.

1.3.2 Methodology (Theoretical Aspects)

Theoretical aspects are discussed in chapter 3. Unsupervised and supervised learning algorithms will be briefly introduced. Moreover, the basic assumptions made when designing this log based intrusion detection system and how server attacks are to be detected will also be discussed in this chapter.

1.3.3 Development

Chapter 4 will describe the proposed system development strategy and progress.

1.3.4 Discussion

Chapter 5 demonstrates the performance of the implemented log based intrusion detection system with regards to its unsupervised learning algorithms and intrusion detection methods.

1.3.5 Conclusion

Chapter 6 will give conclusions, assessing the successes and limitations of log based intrusion detection system. Some future directions are also discussed.

Chapter 2: Literature Review

2.0 Background

Designing an intelligent log based intrusion detection system involves with a broad range of knowledge, namely web server security, data mining techniques, learning algorithms and some novelty detection approaches.

In this chapter, initially an introduction will point out the weight of the web server security problems. Afterwards, some conventional intrusion detection methods are briefly discussed before data mining based approaches in the log analysis are introduced. In the next section the topic of novelty detection is covered, which links closely to the detection of intruder intrusions. Finally, some related work will be re-viewed.

2.1 Intrusion

Threats to web servers come typically from the malfunction of hardware or software, or through malicious behaviour by users of software. Promptly resolving incidents is vital, considering the huge costs of data loss and server down-time.

The abundance of computational resources makes lives of computer hackers easier. Without much effort, they can acquire detailed descriptions of system vulnerabilities and exploits to initiate attacks accordingly. According to statistics from CERT[®] Coordination Centre (CERT/CC), the most influential reporting centre for internet security problems, show that there was a dramatic increase of reported network incidents to CERT/CC (Ma, 2003).

2.2 Logs

To protect servers from attacks, a common approach is to record server logs to monitor all those prominent activities. Each time a noticeable event happens in the server, an entry will be appended to a log file, in the form of plain text or binary format. Take web log files as an example. Every “hit” to a web site, including requests for HTML pages as well as images, is logged as one line of text in a log file. This records information about who is visiting, where they

are from and what they are doing with the web server. Below is a sample of an apache log format,

```
-- "%h %u %t \"%r\" %>s %b \"%{Referrer}i\" \"%{User-Agent}i\""
```

This translates to: -

%h – ip address

%u – Authenticated userID if http authenticated

%t – timestamp [day/month/year: hour: minute: second zone]

%r – request line (method_used requested_resource protocol)

%>s – status code

%b size of returned obj

\"%{Referrer}i\" – http header referrer

\"%{User-Agent}i\" – the user agent

Below are examples of apache server access logs

```
120.254.103.132 - - [14/Jan/2016:12:58:17 +0300] "GET /search?=&IntelliIDS HTTP/1.0" 200  
5057 "http://black-adkins.com/about/" "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/5321  
(KHTML, like Gecko) Chrome/14.0.824.0 Safari/5321"
```

```
36.194.62.124 - - [14/Jan/2016:13:24:30 +0300] "GET /productID=3257 HTTP/1.0" 200 4962  
"http://www.hart.info/" "Mozilla/5.0 (Macintosh; PPC Mac OS X 10_8_7) AppleWebKit/5360  
(KHTML, like Gecko) Chrome/14.0.896.0 Safari/5360"
```

An experienced system administrator may take a quick glance at web server logs and realize instantly what has happened. However, it is almost impossible for any normal person to check those logs when the log files have accumulated to thousands if not millions of log entries. Naturally, appropriate methods are needed to remove irrelevant information and extract the most interesting. What is required, therefore, is an intrusion detection system that is intelligent enough to automatically detect those abnormal activities in the logs without too much human inputs.

2.3 Intrusion Detection Methods

There have been several intrusion detection systems that use log analysis on the market. The intrusion detection methods they have been using are categorized as follows:

2.3.1 Pattern Matching

This type of systems examines the contents of network traffic (in real-time intrusion detection systems) or log file (in log based intrusion detection systems) to look for a sequence of bytes as the pattern to match. This approach is rigid but simple to implement and therefore is widely used (Ma, 2003).

2.3.2 State-full Pattern Matching

This performs pattern matching within the context of a whole data stream instead of just looking into current packets.

2.3.3 Protocol Decode-Based Analysis

This makes extensions to the state-full pattern matching method in that it tries to find out the violations against the rules that are defined by the Internet standards.

2.3.4 Heuristic-Based Analysis

Makes decisions based on pre-programmed algorithmic logic. Those algorithms are often the statistical evaluations of the network traffic content.

2.3.5 Anomaly Detection

This approach tries to find out anomalous actions based on the learning of its previous training experience with patterns assumed as normal.

The first four methods are widely used in industry practices. However, most of these pattern-matching based detectors can only deal with already-known intrusions that have been recognized by the security experts. Unfortunately, ill-intentioned hackers are aware of those patterns too. When new attack patterns emerge, very likely they could evade the detection by deliberately avoiding those widely publicized matching patterns. The potential damages caused by those attacks are consequentially substantial.

With regard to attacks that become more cunning, more variant, and hence much more dangerous, it is hard to imagine that human-maintained pattern-matching intrusion detection

systems could be updated quickly enough. Data mining approaches, armed with machine learning algorithms, may come to the rescue.

2.4 Data Mining Approaches

Data Mining is defined as the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner. During the process of data mining, many machine learning algorithms are available for choosing. Depending on whether the class labels are provided for learning, these machine learning algorithms can be classified as either supervised or unsupervised (Li, 2013).

2.4.1 Supervised learning

Trained with data bearing class labels indicating to which subcategories they belong or what real-valued properties they have, a supervised learning algorithm tries to predict the most likely labels for new test data. There are two major subcategories for supervised learning:

Classification is to predict the class membership as one of a finite number of discrete labels.

Regression is to predict the output value as one of a potentially infinite set of real-valued points.

There are many widely used supervised classification techniques. They include but not limited to Support Vector Machines (SVMs), Decision Trees, Neural Networks, Naive Bayes, Nearest Neighbour and Regression models. For example, based on a Naive Bayes classifier, trained with a data set with virus labels on file headers, an automatic email filter that detects malicious Windows executables coming through the email system has been developed in the past.

2.4.2 Unsupervised learning

In unsupervised learning, the data are not labelled, which makes it hard to tell what counts as good. The model generating the output must either be stochastic or must have an unknown and varying input in order to avoid producing the same output every time. From this

point of view, the aim of unsupervised learning could be regarded to as a generative model that gives a high likelihood to the observed data.

From the perspective of machine learning, the searching for clusters is unsupervised learning. To perform clustering is to try to discover the inner nature of the data structure as a whole, and to divide the data into groups of similarity. From the viewpoint of data mining, clustering is the partitioning of a data set into groups so that the points in the group are similar as possible to each other and as different as possible from points in other groups.

There are generally three types of clustering algorithms

Partition-based clustering

Given a predefined number of clusters, find the optimal partitions for each point. Choose the centres so as to minimize the summed distance. The k-means algorithm is a well-known example of this kind of clustering methods. we will present it in detail in the next chapter.

Hierarchical clustering

Hierarchical clustering builds a cluster hierarchy. The hierarchy is a tree of clusters. Every node in the tree contains child clusters while sibling clusters share a common parent node. Depending on how the tree is formed, hierarchical clustering methods fall in two categories, agglomerative and divisive. Agglomerative methods recursively merge points while divisive methods start from a cluster of all data and then gradually split them into smaller clusters.

Probabilistic based clustering

This approach assumes that the data comes from a multivariate and finite mixture model with probability as shown below

$$p(x) = \sum_{k=1}^K \pi_k f_k(x; \theta_k)$$

where π_k is the class component prior probability, $f_k(x; \theta_k)$ is class conditional density function, and θ_k is its model parameters.

2.5 Novelty Detection

Novelty detection refers to the identification of new or unknown data or signal that a machine learning system is not aware of during training. It is one of the fundamental requirements of a good classification or identification system since sometimes the test data contains information about objects that were not known at the time of model training.

Anomaly could be regarded as one kind of novelty. Normally, classifiers are expected to give reliable results when the test data are similar to those used during training. However, the real world is totally different, when abnormal data come in, picking them out is a problem. Compared to conventional 2-class classification problem, an anomaly detection system is trained with only normal patterns and then try to predict those abnormal data based solely on the models built from normal data. There exist a variety of methods of novelty detection that have been shown to perform well on different data sets (Li, 2013).

2.5.1 Probabilistic/GMM approaches

This category of approaches is based on statistical modelling of data and then estimating whether the test data come from the same distribution that generates the training data. First estimate the density function of the training data. By assuming the training data is normal, the probability that the test data belong to that class can be computed. A threshold can then be set to signal the novelty if the probability calculated is lower than that threshold.

For Gaussian Mixture Modelling (GMM) models, the parameters of the model are chosen by maximizing the log likelihood of the training data with respect to the model. This task could be done using re-estimation techniques such as EM algorithm. However, GMM suffers from the curse of dimensionality in the sense that if the dimensionality of the data is high, a very large number of samples are needed to train the model, which makes the computation even harder.

A much simpler way is to just find the distance of test data from the class mean and set a threshold for the variance. If the test data is far away from the mean plus the variance threshold, then it can be claimed to be novel.

2.5.2 Non-parametric approaches

For non-parametric methods, the overall form of the density function is estimated from the data as well as parameters of the model. Therefore, non-parametric methods do not require extensive prior knowledge of the problem and do not have to make assumptions on the form of data distribution, which means that they are more flexible though much more computational demanding.

2.5.2.1 K-nearest neighbour approaches

The k-nearest neighbour algorithm is another technique for estimating the density function of data. This technique does not require a smoothing parameter. Instead, the width parameter is set as a result of the position of the data point in relation to other data points by considering the k-nearest data in the training set to the test data.

For novelty detection the distribution of normal vectors is described by a small number of spherical clusters placed by the k-nearest neighbour technique. Novelty is assessed by measuring the normalised distance of a test sample from the cluster centres.

2.5.2.2 String matching approaches

String matching approaches is biologically inspired by studying how the immune system works. Treating training data as templates, which are represented by a string (vector of features), they could then compute some measure of dissimilarity between training and test data. The self-data is converted to binary format forming a collection S. Then a large number of random strings are generated forming a set R_0 . Strings from R_0 are matched against the strings in S and those that match are eliminated.

Since perfect matching is extremely rare, the matching criterion is relaxed so as to consider only r contiguous matches in the strings. Once R_0 is created, new patterns are converted to binary and matched against R_0 . If a match is found, then new pattern belongs too non-self and is rejected. The major limitation appears to be the computational difficulty of generating the initial repertoire. This method has been applied on the detection of computer virus and claimed some good results.

2.5.3 Neural network based approaches

Quite a number of different architectures of neural networks are applied to novelty detection. A neural network can detect novelty by setting a threshold on the output values of the network. Or it can calculate the Euclidean distance between output patterns and target patterns and throw those with highest distance out as the novelty.

2.7 Feasibility Study

Efficacy of Log Analysis Based IDS => High

Practicality of Log Based IDS => High

Efficacy of AI in Anomaly Detection => High

Efficacy of Unsupervised Learning in Anomaly Detection => High

Top Rated Python Analytical and Statistics Library => SciKit-Learn

Most Favourable Unsupervised Learning Algorithms => K-Means and One Class SVM

Chapter 3: Methodology

In this chapter, a discussion on how an intelligent network log analyzer can be built continues with a more in-depth approach on the theoretical aspects of the algorithms to be incorporated in the system.

In the first two sections, Kmeans and One Class SVM unsupervised learning algorithms are discussed. The following sections will introduce how the logs are vectorized, a short discussion on feature extraction, and the methodology used in the development of the system.

3.1 Kmeans

The KMeans algorithm is cluster based hence one needs to define the number of clusters, the k , hence the name. The clusters are the average locations of all the members of a cluster.

Assume $n =$ data points

then $D = \{x_1, \dots, x_n\}$

Hence to find K clusters

$\{c_1, \dots, c_k\}$

then the algorithm is as describe below

```

initialize  $m_1 \dots m_k$  through random selection as cluster centers

while (no conditions are met, mostly (lack of change in clusters  $c_k$ )

    for  $i = 1, \dots, n$ 

        calculate  $|x_i - m_j|^2$  for each center

        assign  $i$  to the closest center

    end for loop

    re-compute each  $m_j$  as the mean of the data points assigned to it

end while loop

```

The overall formula for KMeans is:

$$\sum_{i=0}^n \min_{\mu_j \in C} (||x_j - \mu_i||^2)$$

3.2 One Class SVM

One class SVM attempts to learn the decision boundaries that achieve the maximum separation between the data points and the origin. The introduction of kernels in One class SVM gives it the ability to learn non-linear decision boundaries as well as account for outliers. One class SVM utilizes an implicit transformation function $\varphi(\cdot)$ that is defined by the kernel chosen. It then learns the decision boundary which separates most of the data from the origin. Data that lie outside the data points are considered as outliers (Gardner, Krieger, Vachtsevanos and Litt, 2006).

By observing that all kernel entries are non-negative (greater than or equal to 0), all the data in the kernel space can be concluded as to belong in the same quadrant.

Assume $g(\cdot)$ is defined as:

$$g(x) = w^T \phi(x) - \rho \quad (1)$$

where w is the perpendicular vector to the decision boundary and ρ is the bias term

Then,

$$f(x) = \text{sgn}(g(x))$$

shows the decision function that one-class SVM uses in order to identify normal points. The function returns a positive value for normal points and negative for outliers:

3.3 Text Vectorization and Feature Extraction

Since apache logs are in human readable text form, vectorization is required to convert them into numerals. The feature extraction and text vectorization techniques used in this system are

Frequency

A frequency matching function that searches through the logs for unique instances of specific row in each column and assigning each a numerical value.

The Bag of Words representation

Count Vectorization from scikit-learn: This tokenizes each unique word in the logs and assigns it a unique numeral value (Scikit-learn.org)

Chapter 4: Development

This chapter focuses on the implementation details of IntelliIDS (Name given to the system). The chapter is segmented into four sections. The first section details the philosophy followed in designing IntelliIDS, then the choice of programming languages, the modularization of the system, and related data mining and machine learning packages from the chosen programming language.

4.1 Introduction

The general purpose of an IDS is to quickly identify attacks in the system. IntelliIDS was developed with this in mind hence it is developed to search for known attacks first and then use machine learning to detect unknown attack patterns

4.2 Philosophy

4.2.1 Experimental Approach

IntelliIDS will be an experimental framework. Given this, the best algorithm will be determined by the highest accuracy scores. This means that a lot of detection methods will be used and the one with the highest scores will be implemented. For faster development, the system will be developed for console usage other than a GUI based approach. However, the system takes up multiple arguments which enhance its usage and allows for custom analysis. For example, the user has the option to choose the type of analysis, algorithm to use, etc.

4.2.2 Modularity

Module based approach will be used for the entire system hence more features can be added easily.

4.2.3 Rapid Development

Existing source code available on Github and public libraries will be used where necessary to hasten the development process.

4.3 Programming Languages

Since IntelliIDS is experiment and exploration oriented, much labour will not be spent

reinventing the wheel by re-writing source code of the algorithms. Majority of the best working supervised and unsupervised machine learning algorithms are already available in the most popular programming languages such as python, R, C, C # etc. Though through research I discovered that the best programming language for machine learning is R (since it is data science based), I choose python as the language of choice because I have intermediate knowledge in python while I have no previous experience in R.

4.4 Modules

Below is the description for each module in IntelliIDS.

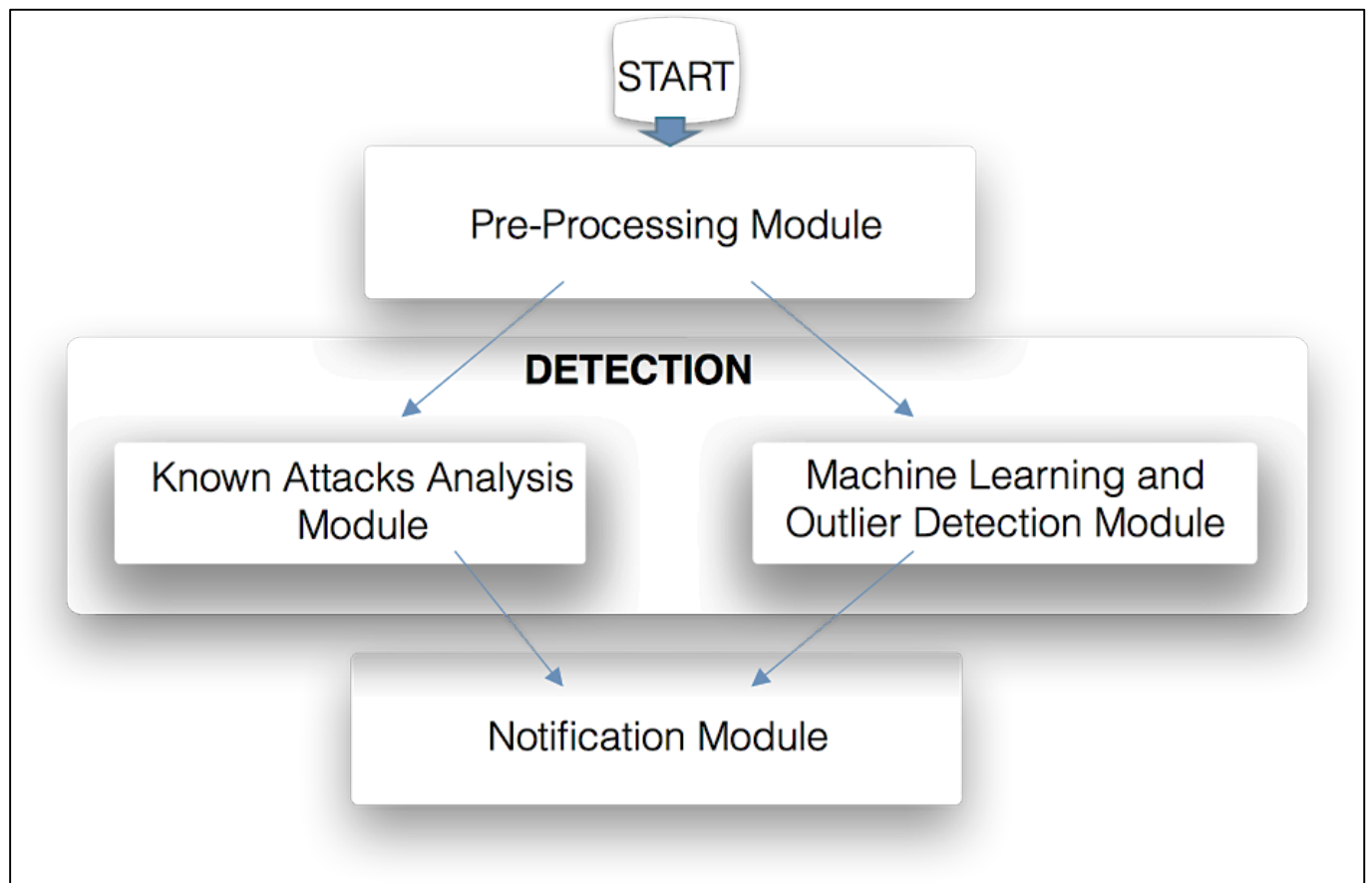


Figure 4.4-fig1 *Modules in the system*

4.4.1 Pre-processing module

Input: Raw Apache logs

Output: Vectorised logs in the following format

1. IP
2. Time (Unix time)
3. Request
4. Status

Implementation:

1. Function to parse logs will be written
2. Function to vectorise the logs will be written.
“This will search through the logs for unique words and assigning them a numerical value”
3. Python’s pandas library will be used to create Data Frames holding the vectorised logs

4.4.2 Known Attacks Analysis Module

Input: Raw Logs

Output: Report on known attacks detected

Function: Two major functions:

1. Scanning through logs for known attack patterns.
2. Exporting results and calling the notification system when one or more patterns are detected

Implementation: Regular expressions will be used to search through the requested URLs for known attack patterns such as;

XSS : Cross-Site Scripting

SQLI : SQL Injection

CSRF : Cross-Site Request Forgery

DOS : Denial of Service

DT : Directory Traversal

SPAM: Spam

ID : Information Disclosure

RFE : Remote File Execution

LFI : Local File Inclusion

This will be achieved through modified code forked from Scalp (a python library).

4.4.3 Machine Learning and Outlier Detection module

Input: Two Inputs

1. Vectorised logs to be checked
2. Vectorised logs to be trained with (Normal dataset)

Output: Detection Results

Function: Two major functions:

1. Using the Normal Dataset to learn
2. Detect outlier activities based on the Normal Dataset

Implementation: Two unsupervised learning algorithms are tried.

1. Kmeans
2. One Class Support Vector Machine (One class SVM)

4.4.4 Notification module

Input: Detection Results

Output: An email to the system administrator containing detected information

Function: sending notifications to system administrators through email

Implementation: Written in Python utilizing the smtplib library.

4.4.5 Real-time Detection Module

(Not yet implemented)

Function: Analyse new log lines as they are generated by Apache

4.5 Related packages utilised

To reduce the development and testing time, external libraries in python are used. They include:

1. Numbly (Offering array capabilities for python)
2. scikit-learn (Data Science based library includes all machine learning algorithm and test data)
3. Pandas (creating manageable data frames)

4.5.1 Scikit-learn

This is an open source machine learning library for Python. It features various classification, regression and clustering algorithms both supervised and unsupervised machine learning algorithms, and is built on top of Python numerical and scientific libraries (NumPy and SciPy) for faster interoperation (Scikit-learn.org).

4.6 Chapter Summary

In this chapter the implementation technique is detailed. The programming language chosen is Python. Modularization the system is detailed in that it contains a pre-processing module, Known-attack analysis module, Machine learning and detection module, notification module and a real-time analysis module (un-implemented). Lastly the related packages to the system's implementation were discussed.

Chapter 5: Discussion

For an intelligent log-based intrusion detection system to be termed as successful as successful, it has to identify both known and unknown attack patterns in the given dataset. In this chapter, a discussion on the performance measures and experiment results are given.

5.1 Experiment Design

5.1.1 Performance measures

The performance measures that deemed effective in this case were the true positive rate which will be referred to as the accuracy.

The **Detection Rate** is the percentage of attacks detected.

$$\text{Detection Rate} = \frac{\text{Number of attack patterns detected}}{\text{Number of attacks in the generated Logs}}$$

5.2 Experiments

A series of experiments were conducted on the two unsupervised learning algorithms chosen to determine which of the two had a higher performance rate.

5.2.1 Experiment 1 (KMeans Clustering)

5.2.1.1 Test1

Given a training sample of 50000 line of logs that are normal logs and a test sample of 1000 log lines of which 80% were attacks and 20% normal

NB: The Data was labeled for reference purposes.

Input Logs format

IP(vectorized)	TIME (Unix time)	Request(Vectorized)	referrer(Vectorized)	Agent(Vectorized)
1231122112	1452162152	-1243	3445	23
01001034004	1453164844	52	45	0
...

The results

Given the parameters the accuracy of the IDS was at 50% which is of little help in the real-world. The parameters were adjusted accordingly by dropping the referrer and the agent paving way for the second test.

5.2.1.2 Test2

With the adjusted parameters, that is,

IP(vectored)	TIME (Unix time)	Request (Vectored)
1231122112	1452162152	-1243
01001034004	1453164844	52
...

The accuracy of the system increased by 10% giving an accuracy level of 60%. However, the results were not satisfactory. This demanded another test with more adjustments to the parameters.

5.2.1.3 Test3

I discovered that the issue was with the time (Unix time) which gave the time in milliseconds hence offering a large dataset that gave the wrong projections to the algorithm. With this in mind, I adjusted the Unix time in the log generator to generate new logs at intervals of 60 second to 300 seconds. Then rounded up the Unix time to 6 digits hence working with a smaller number and a better time range to correlate event.

IP(vectorized)	TIME (Unix time)	Request (Vectorized)
1231122112	145216	-1243
01001034004	145316	52
...

With the new parameters KMeans algorithm delivered an accuracy of 85% which I considered worthwhile. KMeans algorithm module gave consistent results ranging from 80 – 85%

5.2.2 Experiment 2 (One class SVM)

With previous results from the KMeans algorithm. The best working parameters were selected as the initial test.

5.2.2.1 Test1

The same training and testing datasets were used in determining the effectiveness of One Class SVM. The format given as input is as follows

IP(vectorized)	TIME (Unix time)	Request (Vectorized)
1231122112	145216	-1243
01001034004	145316	52
...

With the success of the KMeans algorithm using the same dataset and parameters, the expectation of this algorithm were high. The novelty detection based algorithm did not fail in detection rates as it gave a success rate ranging from 80% to 85%.

A second test was done to determine whether additional information would be relevant to this algorithm and hence improving the accuracy and detection rate.

5.2.2.2 Test2

A fourth column was added to the datasets. One of those that had been removed in 5.2.1.1 Test 1.

IP(vectorized)	TIME (Unix time)	Request (Vectorized)	referrer (Vectorized)
1231122112	145216	-1243	3445
01001034004	145316	52	45
...

With this new parameters, the accuracy of the algorithm was negatively affected the accuracy reduced to a range of 70% - 79%. I reverted back to the old parameters to keep the high accuracy and better detection rate.

The appendix contains screenshot of the accuracy levels, working of the system and a detailed usage of the system.

Chapter 6: Conclusion

The aims of this projects were to build an IDS that utilized machine learning to detect known and unknown attack patterns in apache logs. In this chapter the achievements, problems and limitations of the system are discussed.

6.1 Achievements

The initial objectives of the project were successfully achieved. That is;

1. Modularity of the system
2. Machine Learning and Detection

6.1.1 Modularity of the System

The system has been fully modularized in that additional features can easily be added. Modularization was achieved through classes in python.

6.1.2 Machine Learning and Detection

The system detected most of the known attack patterns using the Known-Attack Analysis module and a descent percentage of the same attacks using the Machine Learning and Detection Module.

6.2 Problems

During the implementation, some problems were encountered which include:

6.2.1 Lack of Data

When working with unsupervised machine learning algorithms the size of the data matters. This is to say that unsupervised learning works best with large data. This was a problem since no one was willing to give access to their entire 'access_log' dump. This became a challenge because the only logs I had access to where from my own websites which have minimal traffic hence a

year's access_log only contained about 3000 log lines.

Solution

The alternative was to generate my own logs. This was achieved through a fork of kiritbasu's Fake-Apache-Log-Generator available here (<https://github.com/kiritbasu/Fake-Apache-Log-Generator>). I modified the code as per the requirements of the project.

6.2.2 Processing power

With generated logs of up to 1GB (over 15 millions log lines), the processing power required was a bit higher than the current machine I was using to develop with. Most of the time, I had to wait for over 40 minutes before getting the results. In a production setting this would not be realistic since.

6.3 Limitations

Since this system was rapidly developed there exists some limitations to its functionality. These include:

6.3.1 Software development

The system developed has a hard coded log pattern it uses to analyze. The current pattern does not allow for per-logged-in-user analysis which would be a big plus since most attacks are carried out from hijacked accounts and/ or rouge accounts.

6.3.2 Definition of the Norm (Normality definition)

The success of the system partially depends on the success of the normality definition. For each analysis a normal state of the analyzed system logs had to have been achieved and recorded in order to determine the outliers based on the current analyzed system logs.

6.3.3 Real-time detection

Due to time limitations, IntelliIDS operates the same way a batch mode data miner would, this is helpful for post analysis, however, the most appealing way intrusion detection systems ought to work is real-time detection. Plans of implementing this are currently underway and will be released at a later date.

6.4 Future Works

As stated in chapter 4: Development, plans are underway to modify the IntelliIDS into a real time scanner. However, many features need to be incorporated in the system to ensure better accuracy and hence higher detection rates. The features that miss in this version that would make this project even better are:

i. Online Learning

Incorporating the ability to access and analyze remote logs from say a hosted server. This will increase the productivity of the system in that one system can be used from a stationary location to analyze and report on anomalies of remote systems in real-time.

ii. Real time analysis

This is a feature that is vital to any IDS, this will ensure that the system analyzes logs as they come in other than a post-analysis based approach that the current version works with.

6.5 Chapter Summary

This is the last chapter and it concluded the thesis with a discussion on the achievements, problems, limitations and future works of the project. Though the accuracy of the system needs to be improved, the project has been an overall success of an IDS that utilizes unsupervised machine learning algorithms for novel/anomaly detection.

References

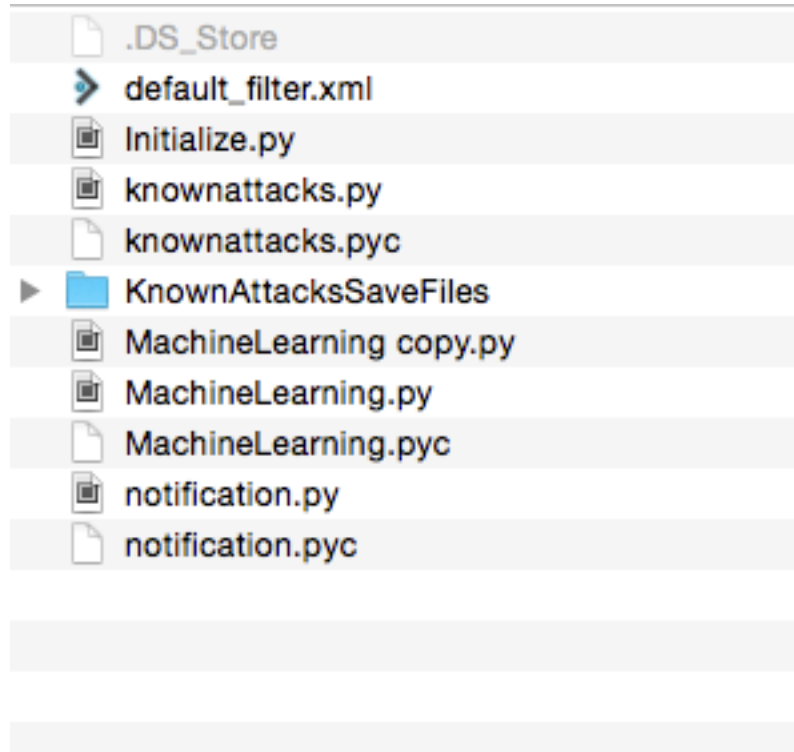
- Alspaugh, S., Chen, B., Lin, J., Ganapathi, A., Hearst, M., & Katz, R. (2014). Analyzing log analysis: An empirical study of user log mining. In *28th Large Installation System Administration Conference (LISA14)* (pp. 62-77).
- Amoli, P. V., Hamalainen, T., David, G., Zolotukhin, M., & Mirzamohammad, M. (2016). Unsupervised Network Intrusion Detection Systems for Zero-Day Fast-Spreading Attacks and Botnets. *JDCTA (International Journal of Digital Content Technology and its Applications, Volume 10 Issue 2)*, 1-13.
- Coates, A., Lee, H., & Ng, A. Y. (2010). An analysis of single-layer networks in unsupervised feature learning. *Ann Arbor, 1001(48109)*, 2.
- Computational Statistics and Predictive Analysis in Machine Learning. (2016). *International Journal Of Science And Research (IJSR)*, 5(1), 1521-1524.
<http://dx.doi.org/10.21275/v5i1.nov152818>
- Gardner, A. B., Krieger, A. M., Vachtsevanos, G., & Litt, B. (2006). One-class novelty detection for seizure analysis from intracranial EEG. *Journal of Machine Learning Research*, 7(Jun), 1025-1044.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). Unsupervised learning. In *The elements of statistical learning* (pp. 485-585). Springer New York.
- Kim, Y., Street, W. N., & Menczer, F. (2000, August). Feature selection in unsupervised learning via evolutionary search. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 365-369). ACM.
- Li, K. L., Huang, H. K., Tian, S. F., & Xu, W. (2003, November). Improving one-class SVM for anomaly detection. In *Machine Learning and Cybernetics, 2003 International Conference on* (Vol. 5, pp. 3077-3081). IEEE.
- Li, W. (2013). Automatic Log Analysis using Machine Learning: Awesome Automatic Log Analysis version 2.0.
- Ma, P. (2003). Log Analysis-Based Intrusion Detection via Unsupervised Learning. *Master of Science, School of Informatics, University of Edinburgh*.
- Manevitz, L. M., & Yousef, M. (2001). One-class SVMs for document classification. *Journal of Machine Learning Research*, 2(Dec), 139-154.
- Matherson, K. (2015). Machine Learning Log File Analysis. *Research Proposal*.

- Muller, K. R., Mika, S., Ratsch, G., Tsuda, K., & Scholkopf, B. (2001). An introduction to kernel-based learning algorithms. *IEEE transactions on neural networks*, 12(2), 181-201.
- S., A. & R., D. (2015). Post-Attack Intrusion Detection using Log Files Analysis. *International Journal Of Computer Applications*, 127(18), 19-21.
<http://dx.doi.org/10.5120/ijca2015906731>
- scikit-learn: machine learning in Python — scikit-learn 0.18.1 documentation*. (2016). *Scikit-learn.org*. Retrieved 2 December 2016, from <http://scikit-learn.org/stable/>
- Svensson, C. (2015). Automatic Log Analysis System Integration: Message Bus Integration in a Machine Learning Environment.
- van der Putten, P. & van Someren, M. (2004). A Bias-Variance Analysis of a Real World Learning Problem: The CoIL Challenge 2000. *Machine Learning*, 57(1/2), 177-195.
<http://dx.doi.org/10.1023/b:mach.0000035476.95130.99>
- Yen, T. F., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W., Juels, A., & Kirda, E. (2013, December). Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In *Proceedings of the 29th Annual Computer Security Applications Conference* (pp. 199-208). ACM.
- Zwietasch, T. (2014). Detecting anomalies in system log files using machine learning techniques.

Appendix

Appendix I: Screenshots

Modules



Module Organization

Initialize.py {If not parameters passed} [calls]
 Knownattacks(), then,
 MachineLearning.KmeansAnalysis , then ,
 MachineLearning.OneClassSVMAnalysis
Notification() is called if any detections

```

=====
IntelliIDS Ver. 0.1
=====

usage: Initialize.py [-h] [--locate LOGFILE] [--train TRAINFILE]
                  [--use {ML,KNOWN,ALL}] [--Mode MODE] [--rows NUM_LINES]

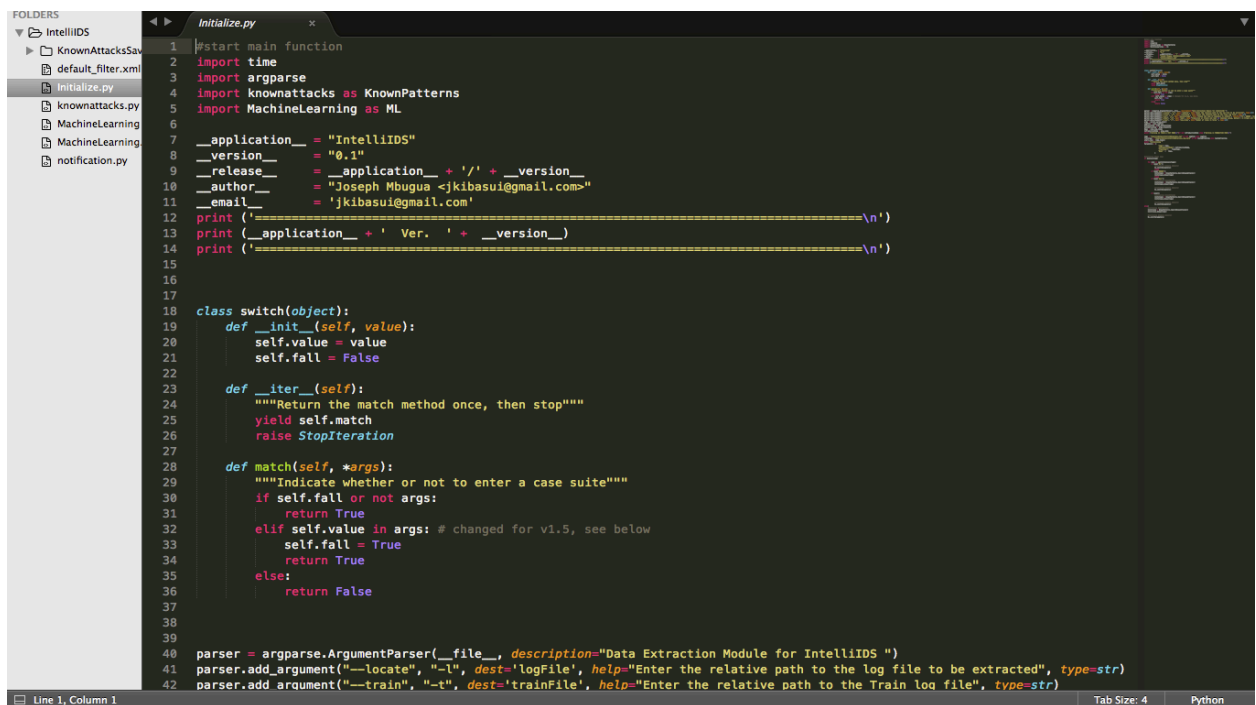
Data Extraction Module for IntelliIDS

optional arguments:
  -h, --help            show this help message and exit
  --locate LOGFILE, -l LOGFILE
                        Enter the relative path to the log file to be
                        extracted
  --train TRAINFILE, -t TRAINFILE
                        Enter the relative path to the Train log file
  --use {ML,KNOWN,ALL}, -u {ML,KNOWN,ALL}
                        Choose the type of analysis to conduct
  --Mode MODE, -m MODE  Production or trains and test (boolean value, Default
                        = 0 [Train and Test])
  --rows NUM_LINES, -r NUM_LINES
                        Number of lines to Parse

```

Enhanced user control of the system. Wide range of parameters for better analysis of logs.

A user has better control of the system in that he can pass the training logs location, the test logs location, the algorithms to use in the analysis, the mode and the number of rows to analyse in the test logs.



```

1 #start main function
2 import time
3 import argparse
4 import knownattacks as KnownPatterns
5 import MachineLearning as ML
6
7 __application__ = "IntelliIDS"
8 __version__ = "0.1"
9 __release__ = __application__ + '/' + __version__
10 __author__ = "Joseph Mbugua <jkibasui@gmail.com>"
11 __email__ = 'jkibasui@gmail.com'
12 print ('=====
13 print (__application__ + ' Ver. ' + __version__)
14 print ('=====
15
16
17
18 class switch(object):
19     def __init__(self, value):
20         self.value = value
21         self.fall = False
22
23     def __iter__(self):
24         """Return the match method once, then stop"""
25         yield self.match
26         raise StopIteration
27
28     def match(self, *args):
29         """Indicate whether or not to enter a case suite"""
30         if self.fall or not args:
31             return True
32         elif self.value in args: # changed for v1.5, see below
33             self.fall = True
34             return True
35         else:
36             return False
37
38
39
40 parser = argparse.ArgumentParser(description="Data Extraction Module for IntelliIDS ")
41 parser.add_argument("--locate", "-l", dest='logFile', help="Enter the relative path to be extracted", type=str)
42 parser.add_argument("--train", "-t", dest='trainFile', help="Enter the relative path to the Train log file", type=str)

```

Clean code and comprehensible. Feature are divided into module and each module divided into function for better utilization of code and code re-use.

```
('coordinates :', array([7511755154, 14566063, 235, 0]), 'label: ', 0)
('coordinates :', array([24518433254, 14566065, 8079, 0]), 'label: '
Kmeans - Prediction Rate: 85.8829028865 Percent
```

Kmeans Accuracy rate 85.8%

```
=====
IntelliIDS Ver. 0.1
=====

Working in TRAIN & TEST MODE

Initializing
Setting parameters
#####
Accessing logs from: /Users/iGitau/IntelliIDS/access_log
Analyzing entire log file
#####
Converting Values...

Starting analysis ...

Checking with KMeans algorithm

=====
Kmeans - Prediction Rate: 90.749656121 Percent

Checking with One Class SVM algorithm
=====
One Class SVM - Prediction Rate: 88.772352132 Percent
```

Kmeans Accuracy [90.7%] : One Class SVM Accuracy [88.8%]

Source Code access:

The code will be made available on Github once the supervisor approves.

Appendix II: User Manual

Environment/Requirements

Python2.7 or higher

Python Packages required:

- sklearn
- argparse
- re
- random
- pandas
- matplotlib
- numpy
- pytz
- Others are pre-packaged within python

All this can be installed by running

```
`sudo pip install -r requirements.txt`
```

from within the IntelliIDS folder.

Generating the logs

You can skip this step if you have you own training and testing logs

Logs can be generated from the bundled script

“apache_logs_faker.py” – Available on Github

Initialization

```
bash:$ python Initialize.py -h
```

-h or --help – calls the help command giving a detailed usage of the command line tool

Output

```
usage: Initialize.py [-h] [--locate LOGFILE] [--train TRAINFILE]
                  [--use {ML,KNOWN,ALL,KMEANS,ONECLASSSSVM}] [--Mode MODE]
                  [--rows NUM_LINES] [--graph SHOWGRAPH]
Welcome to IntelliIDS
optional arguments:
  -h, --help            show this help message and exit
  --locate LOGFILE, -l LOGFILE
                        Enter the relative path to the log file to be extracted
  --train TRAINFILE, -t TRAINFILE
                        Enter the relative path to the Train log file
  --use {ML,KNOWN,ALL,KMEANS,ONECLASSSSVM}, -u
                        {ML,KNOWN,ALL,KMEANS,ONECLASSSSVM}
                        Choose the type of analysis to conduct
  --Mode MODE, -m MODE
                        Production or trains and test (Boolean value, Default
                        = 0 [Train and Test])
  --rows NUM_LINES, -r NUM_LINES
                        Number of lines to Parse
  --graph SHOWGRAPH, -g SHOWGRAPH
                        Represent Data in Graphs (Clustered)
```

PS: Changes in the Notification module are required, that is, add your own email and password :)

Appendix III: Project Management

Selected Project Management Model:

System Development Lifecycle (SDLC) specifically using the Waterfall Model

“SDLC, Software Development Life Cycle is a process used by software industry to design, develop and test high quality software. The SDLC approach aims to produce a high quality software that meets or exceeds the expectations, reaches completion within time and cost estimates”

Stage 1: Planning and Requirement Analysis

Planning

This involved setting the objectives and the approach to pursue the project with.

Determining the systems key modules and functions;

Analysis, Detection, and Notification

Requirements

Extensive logs for training

Logs from a currently working systems (to measure algorithm effectiveness)

An efficient computer. (High end processor to reduce analysis time)

Project milestones

<i>Task</i>	<i>Start</i>	<i>Finish</i>
<i>Project Proposal</i>	<i>10th October 2016</i>	<i>15th October 2016</i>
<i>Proposal Defense</i>	<i>17th October 2016</i>	<i>18th October 2016</i>
<i>System Design</i>	<i>23rd October 2016</i>	<i>25th October 2016</i>
<i>System Development</i>	<i>26th October 2016</i>	<i>22nd November 2016</i>
<i>Testing and Training</i>	<i>23rd November 2016</i>	<i>30th November 2016</i>
<i>Final Presentation</i>	<i>5th December 2016</i>	<i>6th December 2016</i>

Key deliverables

Source codes, Design Drawings, Documentation

System Manual