Computer and Information Science; Vol. 7, No. 2; 2014
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

A Fraud Detection System Based on Anomaly Intrusion Detection Systems for E-Commerce Applications

Daniel Massa[1] & Raul Valverde[2]

[1] Information Technology and Services, Betsson, Malta

[2] John Molson School of Business, Concordia University, Montreal, Canada

Correspondence: Raul Valverde, John Molson School of Business, Concordia University, Montreal, QC., H3G 1M8, Canada. Tel: 1-514-848-2424 ext. 2968. E-mail: rvalverde@jmsb.concordia.ca

Received: March 25, 2014 Accepted: April 14, 2014 Online Published: April 28, 2014

doi:10.5539/cis.v7n2p117 URL: http://dx.doi.org/10.5539/cis.v7n2p117

Abstract

The concept of exchanging goods and services over the Internet has seen an exponential growth in popularity over the years. The Internet has been a major breakthrough of online transactions, leaping over the hurdles of currencies and geographic locations. However, the anonymous nature of the Internet does not promote an idealistic environment for transactions to occur. The increase in online transactions has been added with an equal increase in the number of attacks against security of online systems.

Auction sites and e-commerce web applications have seen an increase in fraudulent transactions. Some of these fraudulent transactions that are executed in e-commerce applications happen due to successful computer intrusions on these web sites. Although a lot of awareness has been raised about these facts, there has not yet been an effective solution to adequately address the problem of application-based attacks in e-commerce.

This paper proposes a fraud detection system that uses different anomaly detection techniques to predict computer intrusion attacks in e-commerce web applications. The system analyses queries that are generated when requesting server-side code on an e-commerce site, and create models for different features when information is extracted from these queries. Profiles associated with the e-commerce application are automatically derived from a training dataset.

Keywords: Fraud Detection, Fraud Auditing, e-commerce audit, anomaly systems, web attacks prevention

1. Introduction

E-commerce applications are prime targets for criminal attacks. Bolton and Hand (2002) explain that new types of fraud have emerged such as mobile telecommunications fraud and computer intrusion whilst traditional fraud, for instance money laundering, has become easier. Finding the best possible way against fraud is crucial. Different processes have to be implemented in order to protect clients from attackers perpetrating fraud. Fraud Prevention and Fraud Detection are the two (2) classes under which these processes are generally defined.

Fraud Prevention is the process of implementing measures to stop fraud from occurring in the first place (Bolton & Hand, 2002). Prevention is considered the first line of defence, where most fraud is halted at the very beginning. There are different types of Fraud Prevention techniques which can be associated with e-commerce applications, such as Internet security systems for credit card transactions, passwords and tokens to name but a few. However, in practise Fraud Prevention techniques are not perfect and sometimes a compromise must be reached between expense and inconvenience (e.g. to a customer) on one hand, and effectiveness on the other (Bolton & Hand, 2002). Nonetheless, Fraud Prevention can sometimes fail due to vulnerabilities in the system and here is where Fraud Detection is needed.

Fraud Detection is the process of identifying fraud as quickly as possible once it has been perpetrated, with minimal damage conceivable. The processes falling under this class are said to be the second line of defence. When preventive methods fail, Fraud Detection kicks-in. Fraud Detection is an evolving discipline because of the fact that once a detection method becomes known, criminal minds will adapt their strategies and try others methods to circumvent it. In addition, new criminals enter the field, with different capabilities and different mind-sets. If a detection method becomes known by attackers, it does not mean that it is no longer needed. On the contrary, some inexperienced attackers might not be aware of any detection methods that were successful, thus, giving us the edge to detect them.

Although continuous security improvements are installed in auction sites or e-commerce web applications, in practise research has shown that almost every online system has some type of vulnerability which can be exploited to commit fraud (Jaquith, 2002). Even though online identity verification methods are nowadays more sophisticated, illegal activity is still not totally prevented and criminal activity is still successful.

Fraud Detection systems are often derived from statistical tools which are based on the comparison of observed data with expected values. In contrast, an Intrusion Detection System (IDS) is often used to detect Computer Intrusion. The observed data in these statistical tools is generally based on behaviour profiles, including past behaviour. The problem, with statistical tools is the way expected values are derived, since it is determined by the environment context for which the statistical tools are built for. There cannot be a general, all-encompassing-statistical tool; a one-for-all tool.

Statistical Fraud Detection methods are categorised into Supervised and Unsupervised methods. Supervised methods involve techniques which observe training data and construct models based on what has been analysed. In most cases, the observed data includes both fraudulent and legitimate cases. However, adaptability is of a concern with these statistical methods, since they can only be used to detect fraud which has previously occurred.

Unsupervised methods, on the contrary, do not require any training data, but try to find dissimilarities in unlabelled data. Sometimes there are cases when training data is not available, or is very hard to get, thus giving rise to unsupervised methods. One of the difficulties with this type of statistical method is accuracy because they commonly create high volumes of false-positives and false-negatives. Any fraudulent case detected, involves a considerable amount of analysis and resources to identify the real cause and security implications. False-positives are of particular concern since a lot of time and resources are wasted to analyse cases which in reality were genuine.

Intrusion Detection Systems (IDS) are used to detect any kind of attack launched against entire computer networks. IDSs can also be used to detect web-based attacks by configuring them with a number of signatures that support the data of known attacks (Kruegel, Vigna, & Robertson, 2005). The problem with IDSs is that it is very hard to keep signature sets updated with the latest known vulnerabilities. Furthermore, new vulnerabilities must first be discovered before signatures can be applied to protect against them, at which stage it might be too late. In addition, when custom e-commerce applications are developed in-house, new vulnerabilities might be introduced especially when business updates are installed. In practise, it is a very time-intensive and error-prone activity to develop ad-hoc signatures to detect attacks against these applications, apart from the fact that substantial security expertise is also required.

The aim of the research is to develop a Fraud Detection System based on anomaly intrusion detection. The goal is to reduce the number of fraudulent transactions perpetrated through computer intrusion attacks in e-commerce sites. The objective is to use data mining models to detect anomalies as a second line of defence, when preventive methods fail.

2. Background and Literature Review

2.1 Fraud Detection

The biggest problem in the e-commerce industry is fraud. Yufeng et al. (2004) explains that computer intrusion is an activity which leads to fraud and intrusion attacks on e-commerce web applications. Yufeng et al. (2004) states that fraud cases are generally identified from the huge available data sets such as logged data and user behaviour. The data collected through logs and user behaviour, can be a great advantage for fraud detection in order to learn from the recent attacks. Phua et al. (2010) states that in the area of automatic fraud detection there is lack of publicly available real data to conduct experiments on, and lack of published and well-researched methods and techniques. This was concluded in a survey that categorises, compares and summarises most published and technical articles in automated fraud detection within the last ten (10) years.

The problem is that legal and competitive complications within e-commerce make it extremely hard for researchers to obtain real data from companies. Real e-commerce data is very difficult to get access to since it contains personal information that, if made public, would lead to legal issues with data protection laws. In addition, real data could also reveal potential vulnerabilities present in e-commerce site, resulting in a loss of confidence in the service being offered whilst giving rise to further attacks. Furthermore, Phua et al. (2010) concluded that automated fraud detection in the area of e-business and e-commerce is still a big challenge for researchers.

Bolton and Hand (2002) have shown that statistics and artificial intelligence (AI) are effective techniques for fraud detection. Powerful AI techniques, including neural networks and rule-based methods, have proved to be very effective in many applications. However, supervised fraud detection relies heavily on the correctness of the training set used. On the contrary, Bolton and Hand (2002) explain that unsupervised fraud detection is generally used when there are no prior sets of legitimate and fraudulent observations. Such methods include profiling and outlier detection methods (Bolton & Hand, 2002).

Brause, Langsdorf and Hepp (1999) addressed the problem of credit card fraud by combining data mining techniques and neural network algorithms to obtain high fraud coverage with a low false alarm rate. The symbolic features of fraud transactions were rated using a generalisation graph of rules and neural networks of the Radial Basis Function type rated analog features. To better detect credit card fraud, Bhattacharyya et al. (2011) evaluate the use of two data mining approaches known as support vector machines and random forests. Bhattacharyya et al. (2011) conclude that both techniques showed adequate ability to model fraud but random forests demonstrated an overall better performance when evaluated against performance metrics.

2.2 Computer Intrusion

Tan (2002) states that the two most common types of online fraud are conducted in auction sites and web applications which offer general retail merchandising. In general online fraud is achieved through identity theft; a term used to refer to all types of crime in which someone wrongfully obtains and uses another person's personal data in activities that involve fraud or deception, typically for economic gain.

The most common attacks performed on e-commerce applications include Denial of Service Attacks, SQL Injection, Cross-site Scripting (XSS), manipulation of hidden fields, Buffer overflows, malicious software, spam emails, phishing and Brute Force attacks (Almadhoob & Valverde 2014).

Lee, Low and Wong (2002) explains that SQL injection is a subset of Injection attacks, and is used to retrieve sensitive data, such as credit card numbers and personal medical histories, from databases which are accessible through online systems. Mookhey (2010) defines SQL injection as the insertion of SQL meta-characters in user input, such that attackers' queries are executed by the back-end database.

Cross-site Scripting (XSS) is another common attack that targets the end-user and takes advantage of the lack of input and output validations on the web application. It can also be achieved by relying on users' trust in clicking on a URL which serves XSS attacks. Black hats try to embed client side scripting code (such as JavaScript), by supplying it as part of the input. The main goal of this nature is to steal user's session ID from cookies, so as to impersonate the victim, and perform online transactions within the lifetime of the session.

Some payment gateways and online shopping carts also suffer from manipulation of hidden values. The vulnerability is often exposed by having total payable prices of purchased goods stored as a hidden HTML fields in a dynamically generated web page. A user can easily modify the value of this hidden field using proxies, before submitting the request to the server. Repeated attacks of this nature can potentially cripple the viability of the online merchant. Such vulnerabilities are not limited to price fields, but can also be used to tamper with personal data.

Less common attacks on e-commerce applications, but which exist nonetheless, include buffer overflows in which attackers send very large values in the input field or HTP header parameters to make the back-end scripting engine unable to parse the request and display errors which might reveal information about the server or code. This will give further insight on how to perform more refined attacks on the system.

Often identity authentications, within web applications, do not prohibit multiple failed logins and these can be attacked using Brute Force techniques. Similarly, if user credentials are sent as clear text, an attacker sniffing the traffic will easily discover the user's credentials and commit fraud via identity theft. Encrypting can easily be implemented in a web application through Secure Socket Layer (SSL) or Transport Layer Security (TLS). Nonetheless, it is important that password policies are implemented and strong enough, to make it hard for malicious users to crack down user's credentials using dictionary attacks.

2.3 Data Mining and Web Logs

The following section explains how to detect the most critical web application flaws from web application logs. Meyer (2008) explains that a detailed analysis of users' action can be extracted from the web application log files and these can reveal a lot of information about the behaviour of users. However, web server logs do not capture everything and have some limitations since they contain only a fraction of the full HTTP request and response.

HTTP (stands for Hyper Text Transfer Protocol) is the protocol used by web browsers to communicate with the server. The flow in this protocol is one way; the client makes a request to the server and the latter responds. The request contains the location of the web page that the server needs to deliver.

The request created by the client is packaged in chunks defined as request headers and request body, which defined what resources need to be served back to the client. The server parses the request to identify what is requested and returns a response message with a response status that describes if the request was successful or not, response headers containing meta-information about the result data and the content that was requested which is known as the response body.

A lot of information can be extracted from web logs. Feng and Murtagh (2000) propose a solution to extract information related to user behaviour from HTTP web logs. In their proposal, a transaction profile is generated for each user which contains information related to transactions and session from the browsing site. One problem in such a solution is that the extracted data is determined by the standard used to generate logs available and their quality. If traces are not logged and/or not detailed enough, the transaction model will not be able to extract the information (Feng & Murtagh, 2000). In such a system, a data mining framework has to be written for each web application, unless a standard way of generating logs is implemented.

The logs of a web application must follow a specific standard, if we want to apply data mining techniques to any web application. Application logs rarely follow a standard, since developers follow their own or company's standards. However, server logs commonly follow the Common Log Format (CLF) standard, and that is why data mining techniques will be applied on such information.

2.4 Intrusion Detection Systems

Simple secure processes are no longer sufficient with today's complex intrusion detection problems. Nowadays, Intrusion Detection Systems (IDS) are installed on networks, to monitor traffic and system activities for malicious attacks and report any incidents that are found. The primary concern of IDS is to alert an administrator of any intrusion attempts, though some IDSs are configured to stop them in the first place. An IDS is classified under two domains: Misuse Detection and Anomaly Detection.

Penya, Ruiz-Agúndez and Bringas (2011) state that the first efficient methodology was misuse detection, which recognised malicious behaviours based on a knowledge base of rules. Misuse behaviour is when an individual deviates significantly from the established normal usage profiles. A misuse detection system uses patterns of well-known attacks or weak spots of the system to match and identify known intrusion patterns and signatures.

The main disadvantages of a misuse IDS is that it cannot automatically detect any new threats, unless new rules are added to its knowledge base, and as the set grows, the time to analyse the request may increase to a certain extent that it might no longer be viable as a real time system.

Anomaly Detection adopts the same concept as misuse detection. However, instead of using static pre-defined rules to detect intrusions, the set of rules get defined through a learning process. The anomaly detection has two phases; training and detection. The training phase involves the analysis of legitimate traffic in which certain features and characteristics of normal usage are recorded. The goal of this phase is to define how normal accepted traffic looks, and ultimately create the dynamic set of rules (Meyer, 2008). During detection, the system will compare the passing traffic against the rule-set created during the training phase, and any deviations from this rule-set will be labelled as anomalous.

The basic assumption in anomaly detection is that anomalous behaviour differs from normal behaviour and the difference can be expressed quantitatively or qualitatively. Kruegel, Vigna and Robertson (2005), and Corona and Giacinto (2010) present different techniques to extract information from an HTTP request, which later can be applied to profile the behaviour of normal traffic in web applications. Kruegel, Vigna and Robertson (2005), and Corona and Giacinto (2010), in the solutions proposed, take advantage of specific characteristics in the HTTP protocol and web applications in general, to model the legitimate behaviour within the application. Anomaly detection can be easily performed once the profiles have been established and trained. The proposed models can be used to detect fraud perpetrated through computer intrusion in e-commerce web applications.

3. Methods and Realization

3.1 Data Collection

An e-commerce honeypot is deployed on the web to collect data. The goal of the honeypot is to attract fraudsters to attack the e-commerce site and gather information about their attack methods, which lead to fraudulent transactions.

The honeypot used can be classified as a research honeypot. Barfar and Mohammadi (2007) explain that "a research honeypot is used to learn about black hat tactics and techniques, monitoring an attacker while compromising a system". The scope of such a system is to allow intruders to stay and reveal their secrets. There are different types of honeypots, but the one used in this project can be classified as a medium-interaction one. Full services of e-commerce systems are emulated by the honeypot, including vulnerabilities. However, it does not allow attackers to penetrate their attacks to the operating system.

The honeypot system exposes a number of vulnerabilities such as SQL Injection, Cross Site Scripting, buffer overflows and weak authentication policies. The web application is not secured over a Transport Layer Security (TLS) or Secure Socket Layer (SSL) connection, to attract more hackers since the traffic is clear text. Two (2) monitors are installed on the web application honeypot, in order to collect data about the requests being sent to the server. Data collection is a very important task in the project since anomaly detection is based on supervised learning which requires training data. The monitors log all requests sent to the server and their content, and the audit trails generated are later used as training data for the anomaly detection system.

The honeypot will attract both fraudulent and legal transactions. However, to discourage honest users from using the honeypot, the products, listed on the e-commerce site, are highly priced and old fashioned. To protect honest users from any identity theft through the honeypot application, the logs are not available from the web application file system.

By performing data analysis on the data collected from the e-commerce honeypot, the roots of the attacks can be traced, emerging attack techniques detected and mitigation processes against these attacks can be developed. The primary step towards data mining is to convert the audit log to a structured dataset which is explained in the next sections.

Server access logs, referrer logs, agent logs and client-side cookies, are different types of data sources, that are very valuable for anomaly detection and which are generally automatically generated by client-server transactions on a web server (Berendt, Mobasher, & Spiliopoulou, 2002). By default, standard web servers such as Apache generate server logs in the Common Log Format (CLF) standard specification.

The server log will not be the only source of user data since it lacks valuable information related to HTTP request, such as header parameters and POST attribute parameters. In order to collect more information, request that access web pages are logged at the application level, in another log file. By using server-side code, the entire set of HTTP header parameters are logged together with the date and time, and the body of the request. Whereas, in access logs only a fraction of the full data is available, with the custom application logger, other information including POST attribute parameters can be logged.

Before the audit trails from the honeypot can be defined as training data, they will require some pre-processing. The data collected is stored in flat files and cannot be easily used or queried. Nonetheless, the audit trails can also have some irrelevant information which needs cleaning up. The pre-processing of web usage data is performed as displayed in Figure 1.

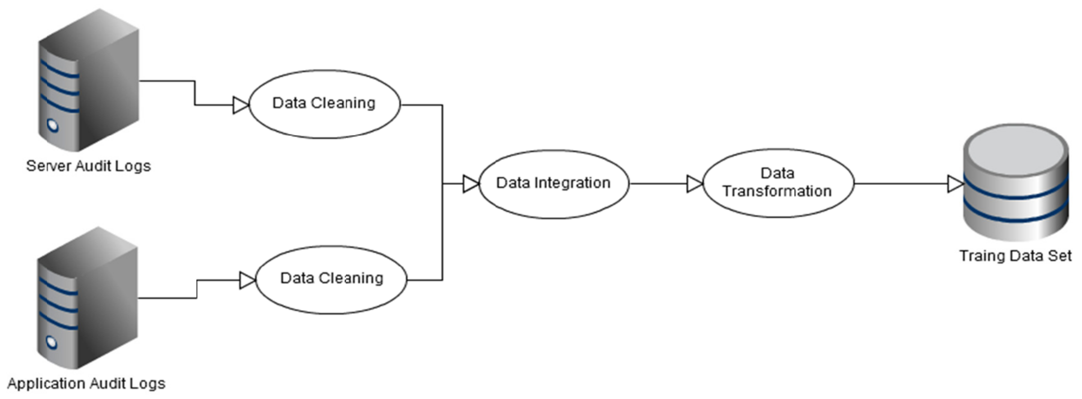

Figure 1. Data preparation workflow

During the data cleaning process, any requests due to spider navigation (such as Google Bots, Yahoo, Bing or Ask) will be removed. It would be insignificant to fill the training dataset with requests generated by spider navigation bots. Nonetheless, any requests which might have some type of log injection attacks, is filtered, removed, and taken note of.

Once the data sources are cleaned, the logs from multiple sources are synchronised and merged together as one source. Once logs from different sources have been merged, a new log is created. The new log has values from both the server audits and from application logs, including different header parameter values. The initial logs are stored in a flat file and so accessing them is not easy and involves parsing. To simplify the process, the merged logs are stored into a relational database for easier access.

3.2 Data Analysis

The main methodology used to analyse the data is data mining. The process of data mining involves the discovering of patterns and rules within our data. To spot fraudulent behaviour, we need to detect regions from the space and define different types of transactions where the data points significantly differ from the rest. This task is normally handled by different models, each analysing different features. In our proof-of-concept, different types of models are built and their results are evaluated to determine the best techniques to use for anomaly detection.

Figure 2 depicts the basic architecture of the anomaly detection system in this project. The architecture is composed of different independent models where each detects anomalies within a set of input requests. Kruegel, Vigna and Robertson (2005) state that a model is a set of procedures used to evaluate a specific feature of a request. The feature under analysis could either be a single query attribute (e.g. the characteristics distribution of the value of an attribute or header parameter), a set of query attributes (e.g. the absence or abnormal presence of attribute or header parameters in a request) or the relationship between a query and the previous requests.

A model is responsible of computing a probability figure for a value in a query or for the query as a whole. The model computes the probability value based on an established statistical profile and any feature, under analysis, resulting in a low probability value, after being passed through the model, is seen as a potential attack.

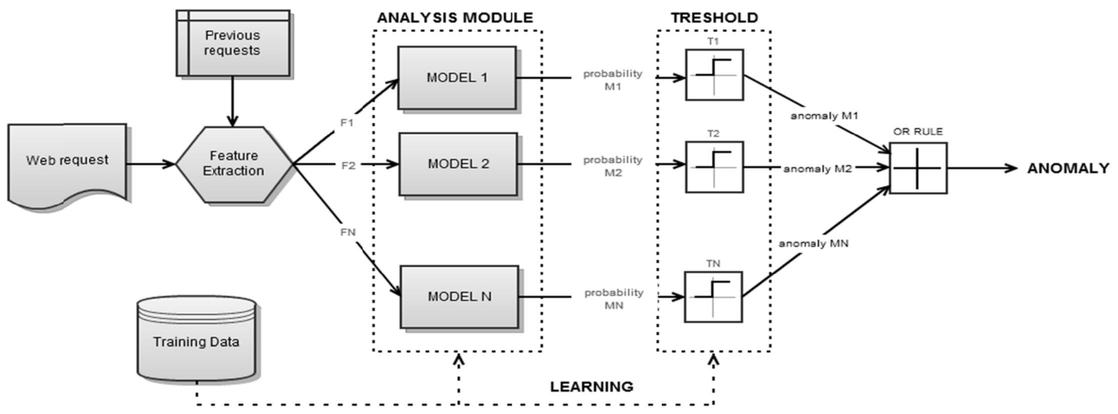

Figure 2. Architecture of proposed anomaly detection system

Each model has two (2) modes of operating; Training Mode and Detection Mode. During training mode, a number of legitimate requests are feed into the model to establish statistical patterns of a specific feature. Each model during training phase, builds a profile of their respective feature under analysis. Once a statistical profile is built and threshold established, the model is able to generate an anomaly probability value of a request based on the training data learnt. Whenever a request reaches the webserver, each model computes an anomaly probability score based on its corresponding features. The decision module applies a suitable threshold to the output of the model to establish whether the feature's value is anomalous or not (Corona & Giacinto, 2010). A request is flagged as an attack if any of the feature models result in an anomalous state when the filter is applied.

Table 1 summarises the features that are employed by the proposed system. For each feature in the table, the principle attack and the model used to detect the attack are described.

3.2.1 Data Mining Fraud Detection Models

This section describes data mining techniques used to analyse features and extract information relevant for the detection of malicious behaviour. Each model operates in two modes: training and detection. The training phase is used to learn statistical patterns and behaviours, while the detection phase calculates probability of malicious use.

(1) Statistical Distribution of Non-Negative Integer Values (Model A)

The role of this model is to learn the statistical distribution pattern in the feature values under study. The main assumption is that the features must always be a positive integer value or zero. If the feature can allow negative values then it is not suitable for this model.

Table 1. Summary of the features profiles used by the proposed system

Feature	Attacks	Model
Length of attribute parameters	Buffer overflow attacks / XSS attacks / SQL injections	Model A (a model for each attribute)
Character distribution of attribute parameters	XSS attacks / SQL injections / Code injections	Model B (a model for each attribute)
Abnormal token value within attribute parameter	Attacks that inject malicious input on attributes with enumerations	Model C (a model for each attribute)
Absence / Abnormal presence of attributes	Any attack that injects malicious input on uncommon attributes	Model D (a model for each page)
Ordering of attribute parameters	Any attack that manually changes the order of attributes	Model E (a model for each page)
Request Method	Information gathering / probing / Cross Site Tracing (XST)	Model D (a model for web application)
HTTP Version	Information gathering / probing	Model D (a model for web application)
Length of HTTP Header attributes	Buffer overflow attacks leveraging on header input	Model A (a model each request header)
Character Distribution in HTTP Header attributes	Code injection attacks leveraging in the header	Model B (a model for each request header)
Frequency Access	Information gathering / automated discovering of vulnerabilities in web application	Model A (a model for web application)
Request Uri	Information gathering / probing / Illegal access of web pages	Model D (a model for web application)

The model is based on the assumption that the higher the value of x_i, the lower the likelihood that it is legitimate. Consider the training set $S \in \{x_1, x_2, .., x_N\}$ where each element is a whole number, i.e. $x \in \mathbb{N}_0$. During learning the value of the mean μ and variance σ^2 is calculated on the sample to approximate the statistical distribution. The values of μ and σ^2 are calculated as follows (Kruegel, Vigna, & Robertson, 2005):

$$\mu = \sum_{i=1}^{n} \frac{x_i}{n} \tag{1}$$

$$\sigma^2 = \sum_{i=1}^{n} \frac{(x_i - \mu)^2}{n} \tag{2}$$

To compute the legitimacy of the value under test, the distance from the mean μ for a random variable X of the statistical distribution is calculated. The calculation uses Chebyshev inequality function. However, only the upper bound of the probability calculation is appropriate since it is assumed that a value which is greater than the mean is said to be malicious. The probability that the value x is legitimate is defined as follows (Corona & Giacinto, 2010):

$$p(x \text{ is legitimate}) = \begin{cases} p\langle |\mathrm{X} - \mu| \geq |x - \mu| \rangle = \frac{\sigma^2}{(x-\mu)^2}, & x > \mu \\ 1, & \text{otherwise} \end{cases} \tag{3}$$

(2) Statistical Distribution of Symbols (Model B)

The discussed model captures the feature's pattern by analysing the statistical distribution of the symbols within the values. The feature under analysis is assumed to have a regular structure (i.e. not random) with printable characters. The algorithm, as proposed by Kruegel, Vigna and Robertson (2005), is based on the frequency values of symbols within the values of the feature. The concept is based on relative frequencies of the alphabetical characters, numbers and special symbols. The value of the feature is assumed to be human readable implying that there will be more alphabetical and numerical characters than the other symbols.

In learning mode, the total number of alphabetical characters, numerical digits and special symbols are each counted for every value in the training set. Unlike Kruegel, Vigna and Robertson (2005) model, the frequency counts are separated into three (3) bins; one for alphabetical characters, one for numerical characters and one for special symbols. Once the model has traversed the entire training set, the mean value μ of each observed feature in each bin is calculated. The relative frequency is used instead of the absolute frequency because the unity of the three (3) bins needs to add up to one, without any preferences to other values with higher absolute frequencies.

To better understand the behaviour of the model, let's consider a feature with the value p@$$w0rd1024. The frequency count for alphabetical characters, numerical characters and special symbols would be 4, 5, and 3 respectively, and the relative distribution value would be 0.33, 0.42, and 0.25. The relative bin value is stored and once the entire training set has been observed, the mean μ is calculated.

During detection, the frequency of the three (3) bins of the value under test is counted. The frequency count is normalised to get the relative frequency. The detection algorithm uses the Pearson χ^2-test (chi-squared test) as a 'goodness-of-fit' (Kruegel, Vigna, & Robertson, 2005), to determine the probability that, the obtained distribution is a sample drawn from the expected statistical distribution of symbols.

Let the set E_i be the expected frequencies for the three (3) bins calculated during the learning phase, and the set O_i be the normalised frequencies of the three (3) bins of the value under investigation. The χ^2 value is calculated as follows:

$$\chi^2 = \sum_{i=0}^{2} \frac{(O_i - E_i)}{E_i} \tag{4}$$

The degree of freedom of the χ^2-test is two (2) since it is the number of bins used minus one. The actual probability value p is derived from the associated value in the χ^2-distribution graph or from a predefined table.

(3) Token Finder (Model C)

Consider a feature value x where $x \in T$ and T is a set of predefined values referred to as enumeration. An enumerable feature is one that has all its values elements of the predefined set T. The goal of this model is to detect if a feature is categorised as enumerable or random, and to learn all the combinations of the enumeration. When a feature is said to be enumerable and a value $x \notin T$, then it is considered an anomalous. On the contrary, when a feature is said to be random, anomalies cannot be detected.

The classification of features should be based on the analysis that the values are bound by an unknown threshold t when enumerable while in the other case these values are mostly random. A feature can be classified as random when the number of different values grows proportional to the total number in the training set. In contrast, if the increase is inversely proportionally then it is assumed that the feature follows an enumeration.

Kruegel, Vigna and Robertson (2005) suggest using statistical correlation to detect if a feature is enumerable or random. Let p be the statistical correlation between the values of the functions f and g, for increasing numbers $1, \ldots, i$ of occurrences of attribute a, where the functions f and g are defined as follows:

$$f(x) = x \tag{5}$$

$$g(x) = \begin{cases} g(x-1)+1, & x^{th}\text{value of } a \text{ is new} \\ g(x-1)-1, & x^{th}\text{value of } a \text{ exists} \\ 0, & x = 0 \end{cases} \tag{6}$$

Once the training set has been observed and learnt, the correlation parameter p is calculated using the following formula:

$$p = \frac{Cover(f,g)}{\sqrt{Var(f) \times Var(g)}} \tag{7}$$

Where $Var(f)$ and $Var(g)$ is the variance of f and g respectively and $Cover(f,g)$ is the covariance of f

and g together.

A negative value of p indicates that f and g are negatively correlated, and thus an enumeration is assumed. On the other hand, when the value p is positive then a positive correlation is indicated meaning that a random feature can be assumed. The set of enumerations observed is stored once the learning phase is complete and an enumeration is assumed. The model assumes that the complete set of enumerations is found in the training set in the case of an enumerable feature.

The detection process is straightforward since when the value of p is positive, the model will always return 1 and when the value of p is negative the model checks that the value of the attribute under test is an element of the enumeration set. If that is the case, the model returns 1 and 0 on the contrary.

(4) Abnormal Presence and Presence of Features (Model D)

The absence or abnormal presence of certain features within a request can be an indica-tive sign of malicious attack. This is normally achieved by manually manipulating the requests using specialised programs. The model discussed in the following section deals with observing that certain features which are regular are kept and alert only when irregular features are present or absent.

The idea behind this model is simple; the model observes the set of features $S \in \{x_1, x_2, .., x_N\}$ and stores whilst traversing the training set. Once the learning phase is complete, a distinct set of features is created. The set S will be used to detect anomalies related to abnormal presence or absence.

For the detection of abnormal presence or absence features, the algorithm observes the set of features S_q of the request under test. It then performs a lookup in the current distinct set and if any feature $x_i \in S_q, x_i \notin S$, the model returns 0, and 1 on the contrary. Also, all the features in S should be present in S_q i.e. $\forall\, x \in S: x \in S_q$. This means that all features observe abnormal characteristics. This implies that the set $S = S_q$ for every request excluding the order of values. If that is not the case, the request is considered to be irregular and 0 is returned.

(5) Irregular Ordering of Features (Model E)

There are instances when a change in order can be an indicative sign of malicious intervention. Attackers who try to exploit web vulnerabilities pay little attention to the order of features (Kruegel, Vigna, & Robertson, 2005). The model discussed hereunder is responsible to detect any irregularities in the order of values of a specific feature.

During the learning phase, the model observes the set of values of a specific feature and records the order whilst traversing the training set. Kruegel, Vigna and Robertson (2005) suggest constructing a directed graph with the orders of values as directed edges. A value x_s precedes another value x_t when x_s and x_t appear together and x_s comes before x_t in the ordered list of values. The order of constraints is defined as the following set

$$O = \left\{(x_i, x_j): x_i \text{precedes } x_j \text{ and } x_{i,} x_j \in \left(S_{q_j} : \ \forall j = 1 \dots n\right)\right\} \tag{8}$$

The set of attribute pairs O is generated through the use of a directed graph. Let G be a directed graph where V, the set of vertices, contains all the distinct values and v_i is associated with the value x_i. For every feature q_j (where $j = 1 \dots n$) that is analysed during the learning phase, the ordered list of its values $x_1, x_2, \dots, x_i$ is observed. For each value pair (x_s, x_t) in this list with $s \neq t$ and $s \geq 1, t \leq i$, a directed edge is inserted into the graph from v_s to v_t.

Upon completion of the learning phase, the graph G would contain all distinct values in V and all order constraints observed from the queries, represented as either an edge between two vertices or a path over a series of directed edges. The only problem is that the graph could potentially have cycles as a result of precedence relationship. To remove these cycles Kruegel, Vigna and Robertson (2005) suggest using Tarjan's algorithm (Tarjan, 1972) to identify all strongly connected components. Once these components have been identified and exist in the graph, all the edges connecting the vertices of the same component are removed.

The model constructs a set of ordered constructs P, to detect any irregularities in the ordering of values, such that all the value pairs (x_j, x_k) with $j \neq k$ and $j \geq 1,\ k \leq i$ are analysed to detect any potential violations. A violation occurs when for any value pair (x_j, x_k), the corresponding pair with swapped elements (x_k, x_j) is an element of O. In such a case, the model returns 0, and 1 on the contrary.

3.2.2 Feature Profiles for the Fraud Detection System

A common observation in web applications is that the *length of a query attribute* does not vary significantly. Generally, query attributes have fixed sized tokens (such as session identifiers or flag variables) or consists of short strings derived from human input (like name and surname input to an HTML form). The variation of lengths within query attributes is not much and sometimes form inputs are also bound with a maximum length.

The intervention of malicious behaviour may considerably change the length of input parameters. For example an attacker would pad the input with extra characters, to perform a buffer overflow attack. A Cross-Site Scripting (XSS) attack might require an increased amount of scripting data to launch the attack. In both cases, the length of the parameter value increases thus differentiating from the norm. The model, as proposed by Kruegel, Vigna and Robertson (2005), allows minor deviations from the normal by genuine users. The model does not allow requests with attributes of length greater than the normal unless very large attribute lengths are not the norm.

Kruegel, Vigna and Robertson (2005) observed that *query attributes*, in general, have a regular structure that contains only printable characters and their values are often human-readable. The model under discussion captures the normal patterns of the query attributes by analysing its character distribution. For example, in an HTML form, when the user is asked for a name and surname, the likely values submitted would be made up of alphanumeric characters. The common trend is that the name and surname of a user do not have special characters.

The algorithm, as proposed by Kruegel, Vigna and Robertson (2005), is based on *frequency values of characters* within the query attribute. The concept is based on relative frequencies of alphabetical characters, numbers and special symbols. Attacks such as XSS use script related tags, full of special symbols, which are embedded into the parameter. Other attacks, such as buffer overflow, pad the string with the same character (normally a whitespace). In both cases, the values tend to deviate from the normal distribution of characters to launch the attack.

Input attribute parameters in web applications often require one out of a small set of values called *enumerations*. Typically these attributes can be flags or indices. The goal of this model is to primarily detect when an attribute parameter can be categorized as enumerable or random. If an attribute is enumerable then anomalies can be detected when any attribute value is not an element of the enumeration set. On the contrary, when an attribute is random, anomaly detections cannot be detected.

As users navigate through a website, client side code automatically sends data to the server via attribute value pairs. The attribute value pairs are either submitted in the URL using GET methods or in the HTML form data using POST methods (the approach is changing with the new REST standards, though the concept is still the same). Usually the attribute names are static and do not change, resulting in a high regularity in the number and name of parameters. Attackers, who try to exploit web vulnerabilities, pay little attention to the completeness of the parameters (Kruegel, Vigna, & Robertson, 2005). This model assumes that the absence or abnormal presence of one or more parameters in a query might indicate malicious behaviour.

During the learning phase, the model observes the set of attribute parameter names making the requests whilst traversing the training set. Once the learning phase is complete, a distinct set of attributes is created for each page requested. This set will be used to detect any *abnormal presence or absence of attribute parameters.*

As explained earlier, since automatic client-side programs construct the request, it is assumed that legitimate invocations of server-side programs often contains the same parameters in the same order. Client-side programs, in general, preserve the relative order of attribute parameters within a request. Malicious attackers pay little attention to the order of attribute parameters when hand-crafting the attack via specialised programs. This model will test the consistency of the order of attribute parameters, and in assuming that *irregular order of parameter indicates malicious behaviour*, will raise alarms when these irregularities are discovered.

The HTTP protocol offers different methods that can be used to perform actions on the web server. Although, the most widely used methods are GET and POST, other methods exist which include HEAD, PUT, DELETE, TRACE, OPTIONS and CONNECT. Some of these HTTP methods are used by developers to test their system. However, if a web server is misconfigured, these methods can potentially pose a security risk (OWASP, 2008). Sometimes attackers create hand-crafted HTTP requests of different method types, to gather information about the web server and to know how to establish attacks. The proposed mode uses the Absence & Presence model to analyse all the method types within the training set of legitimate users. The request type model assumes that the training set contains requests with all method types which are considered legitimate. It also assumes that a request with a method type which is not within the training set is an indication of malicious behaviour.

OWASP (2008) explains that to gather information about the web server, attackers send malformed requests or request non-existent pages to the server. An example used is GET / HTTP/3.0. To detect malicious behaviour via HTTP version, the token finder model is used. Legitimate requests are used as a training set and it is assumed that any legitimate HTTP version is included in the set. *Requests with other HTTP versions*, which are not part of the distinct set of the model, are considered to be malicious behaviour.

Attackers do not only manipulate attribute parameters within the request, but also request headers. The length distribution of request headers is often unchanged since request headers, like attribute parameters, are often fixed sized tokens.

The intervention of malicious behaviour sometimes considerably changes the length of input parameters. The same model used for attribute length, is used to *detect anomalies in header requests*. The model will not allow request header parameters with very large length that deviate from the norm.

It is observed that request headers have a regular structure and often contains only printable characters that can be human-readable. Using the character distribution model, the normal patterns of the request headers are analysed and learnt.

The algorithm, as proposed by Kruegel, Vigna and Robertson (2005), is based on *frequency values of the characters within the request headers*. The concept is based on relative frequencies of the alphabetical characters, numbers and special symbols. Attacks such as XSS use scripts related tags, full of special symbols, which are embedded into the header. It is assumed that malicious intervention will tend to deviate from the normal distribution of characters and thus anomaly can be detected.

Attackers probing any web application will tend to use automatic tools which help in finding web applications vulnerabilities. The frequency at which these automated tools submit requests is generally much faster than a human's standard rate. The goal of the access frequency model is to observe and learn the frequency at which clients submit requests. The idea is to analyse how many requests are submitted in total on each page of the web application and by each individual client (assuming that distinct IPs are distinct clients).

Consider an e-commerce site with a login and review feature. Analysing the frequencies at which these two pages are accessed, one would observe that the authentication page is called less frequently by each individual client since it is only used to login once. However, the total access frequency for the authentication page is very high since it is accessed by many clients. On the other hand, the review page is accessed in bursts by an individual use who is reading reviews about a particular product. However, the total access frequency for the review page can be low since not every client looks at the reviews of a product before buying it.

An irregular burst of invocations may indicate an attacker probing for vulnerabilities. Though the method can indicate irregularities, an attacker can easily trick the model by slowing the rate at which invocations are requested. However, most of the hacking tools available do not support such features. Furthermore, once a vulnerability hole is identified and becomes wide spread, more individuals attack the web applications, thus raising the total frequency to a suspicious level.

The *request Uri-model* works under the principle that the e-commerce application serves a number of different pages and resources to the client. However, the web server also has a set of different programs that are used by system administrators to configure the system properties and to maintain its operations (Gaarudapuram, 2008). The administrative tools include creating logs, configuration of web server settings, authentication and authorisation mechanisms and also creation of users and roles. By default, if the web server is not configured correctly, the tools are accessible over the network with default passwords that are known to everyone.

A user with malicious intent would try to access the administrative pages of the web server if available. If an attacker succeeds in accessing these administrative tools, he would most likely have administrator privileges that allow him to run operating system commands and ultimately take control of the web application. To capture any request which is addressed to unknown pages, this model observes the set of pages that are accessible within the e-commerce application, takes a note of them, and any request to a page which is not part of the e-commerce application, will be an indication of malicious attack. During the learning phase, a distinct set of pages is observed. This set is then used to detect any abnormal requests to pages or resources which do not form part of the e-commerce application.

3.3 Installation and Configuration

To simulate a real world scenario and collect data from real traffic, an e-commerce honeypot is deployed on a remote server. Instead of redesigning a new e-commerce web application, it was decided to use osCommerce (osCommerce, 2011), a free online shop e-commerce solution that offers a wide range of out-of-the-box features and can be setup fairly quickly. OsCommerce is based on the PHP (PHP, 2012) scripting language and the benefit of being script-based is that it can be easily modified to purposely add weaknesses to the system. The OsCommerce uses a MySQL (MySQL, 2012) database to store any information. OsCommerce version 2.2ms.2

was deployed on an external domain.

The idea of using an older version of the osCommerce is that vulnerabilities, for older versions, might be wider spread thus attracting more malicious users. Nonetheless, this version is dependent on older versions of MySQL and PHP. Furthermore, using older versions of MySQL and PHP, results in the addition of vulnerabilities which are already known thus attracting more malicious users.

To promote lack of security within our e-commerce site, it was decided not to secure it over SSL, and thus legitimate users are discouraged because of its lack of security while malicious users are instigated to try and attack it. Once all the configuration of the web application were applied, the admin screens were removed from the remote server, so as not to allow malicious users to raise their privileges to admin users and perform further attacks.

The default template of the web application was altered and styled to give a feeling that the web application is a genuine one and not a honeypot. New items and categories were added to the inventory to promote a real retailer shop. These were added by using the admin screens. To further attract malicious behaviour, the review and search pages were altered to add vulnerabilities to the pages. The development on the review pages, gave rise to Cross Site Scripting (XSS) attacks whilst the changes on the search page provided a back-door for SQL injection attacks. Also password policies were not enforced on the clients thus users could create accounts with simple passwords making brute force attacks more successful.

3.4 Penetration Testing

Though the e-commerce is promoted as best, it is still very unlikely that many malicious users try to attack the site during the time frame of this project. It would be unlikely to have testing data if users do not attack the e-commerce honeypot. To increase the number of malicious requests, data penetration testing is performed on the system, enabling the generation of data which can be used for testing and evaluation.

Penetration testing is normally used to evaluate the security of a web application and there is evidence that has been used successfully in the security evaluation of e-business systems as documented by Stephens & Valverde (2013). However in this case, it is used to simulate malicious behaviour. Normally, penetration testing involves two stages of testing; testing in passive mode to gather information about the e-commerce site and active mode to actually perform the attack. The penetration testing that is used here is based on black-box testing as suggested by OWASP (2008).

3.5 Logging

Server logs and Application logs are the main sources of data that are used as input to the anomaly detection system. In this section, a description of how the logs will be formatted is presented, starting with Server logs and moving to Application Logs.

The Common Log Format (CLF) specification is generally used by standard web servers such as Apache, to log incoming traffic. The CLF specification calls for every HTTP request to be logged on a separate line. A line is composed of several tokens separates by spaces (Meyer, 2008) as shown in Figure 3 below:

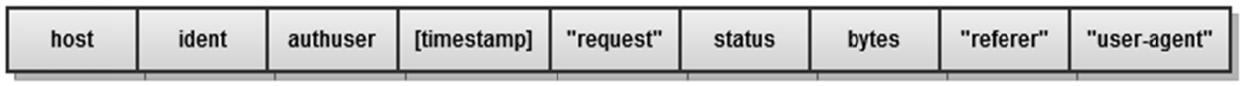

Figure 3. Common Log Format (CLF) specification (Meyer, 2008)

The server, used for the following project, adds the referrer and user agent to the logs. A hyphen / dash (-) indicates that a token does not have a value. The tokens are defined in Table 2:

Table 2. Details of an example of a log entry in CLF

Token	Description	Example
127.0.0.1 - joe [04/Nov/2011:10:11:22 -0500] "GET /catalog/index.php HTTP/1.1" 200 12345 "http://www.decommerceproject.com" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/b		
Host	Fully qualified domain name of client or remote address (IP)	127.0.0.1
Ident	Identity reported by client when IdentityCheck is enabled and client runs identd	-
Authuser	The userid of the client making the request which is determined by HTTP authentication if required	Joe
Timestamp	The time when the server finished processing the request. The time is given in between squared brackets	[28/Jan/2012: 09:02:36 –0500]
Request	The request line requested by the client. This token is given in between inverted commas.	"GET /catalog/index.php HTTP/1.1"
Status	The HTTP response status code returned to the client	200
Bytes	The number of bytes in the object returned to the client excluding all HTTP headers	12345
Referrer	From which website the request was triggered (if any). Basically to identify where people are visiting from. The token value is given in between double quotes	"http://www.decommerceproject.com"
User-agent	The clients' identification of which software is being used to submit the HTTP request. The token value is given in between double quotes so as to escape any space symbols found in this token	"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like Gecko) Chrome/16.0.912.63 Safari/535.7"

Although the server provides a lot of information about requests, the main point of research is to collect more information. An HTTP logger was also implemented at application level using a scripting language (PHP in this case). Since osCommerce is script based, the application logger was easily merged with the framework. The KLogger (Katzgrau, 2008) is used to log data into a text file. The logger was slightly modified so as to support older version of PHP, and to write in files which are rolled over daily.

The HTTP Request Log object, captures the information from the request and formats it into a readable format. The data captured includes the following request headers:

- HTTP_*
- SERVER_PROTOCOL
- REQUEST_TIME
- REQUEST_URI
- QUERY_STRING
- REQUEST_METHOD.

In addition this object also captures the body of the request such as POST attribute parameters. Each request header is printed on a separate line, by first printing the header name, a semi-colon followed by a space and the header value. The last to be printed is the request body, both encoded and decoded, and the same format as the headers is used. An empty line divides a new request from the previous one.

The Application Logger logs all requests which successfully make it to a valid page. However, it does not mean that all requests with response status 200 are logged in the application log. There can be hand-crafted requests which skip the application logger, in which case Server logs will be the only source of information.

3.6 Anomaly Detection System Implementation

The anomaly detection system was developed in Java, and can be executed as a console application. The program accepts different arguments as input parameters, which are used to specify what is needed from the system.

The anomaly detection system has two modes of operations; training and detecting. During training, a training dataset is supplied to the program and through various data mining techniques, behavioural profiles are constructed for different features. The profiles are serialised to xml and saved in a file. The program serialises the profiles to save the state of the profiles once the system has been trained. When in detection mode, the serialised xml files is loaded and detection can commence.

During detection mode, the system de-serialises the xml file to reconstruct the models, and generate a score for each request that is passed. The anomaly score is determined by the value calculated from each model. The models return a value of one (1) or zero (0); the latter indicates a probable anomalous request, while the former indicates a probable genuine request. If any of the models returns a 0, then it is assumed that the request is anomalous.

In reality, the values determined by each model are in decimal format, and are stored in the database. The decimal value is passed through a threshold filter and if it is less than the threshold, it results in a 0 and if above or equal then it returns a 1. However, by default the threshold values of the system are set to 0.5. If the threshold needs to be configured, then this can be done manually, by changing the threshold values within the serialised xml file.

4. Results

The evaluation consists of two phases training and testing. For the training phase, a dataset which was created using random variables was used. This dataset was filtered from any malicious attacks and thus it can be considered to be legitimate traffic. The dataset contains requests to all the pages of the e-commerce site. During training phase the system learns the characteristics from the requests that were supplied. The model values were serialised into an xml file so that the profiles are stored and thus the system can be easily switched to detection mode.

To visualise the anomaly counts that were obtained via the anomaly detection system on the data collected, a number of bar charts were generated. The charts represent the absolute count of the alerts for each model.

4.1 Training Systems

The detection system which is based on supervised learning requires a dataset to be trained. The training data is based on legitimate requests, so as to learn the normal process of requests. In an e-commerce web application, intrusion attacks are a small ratio when compared to legitimate attacks. However, the scope of the honeypot was not to gather legitimate data but to attract malicious attacks. So the data collected from the honeypot cannot be used for training. Due to the factors in place, a dataset of legitimate requests had to be generated that can be used for training. The data of a simulated user using the e-commerce web applications was gathered and formatted. Whilst simulating the user, every single page of the web application and every operation available were requested so that no web page is left out form the training dataset. Once the requests were submitted, the server logs and application logs were supplied to the log parser to filter them and format them. A total of 507 requests were generated by the simulated user.

However, this was not enough as a training dataset. So instead of simulating other users, the logs of the simulated user were replicated with variable data for 99 more users, i.e. a total of 100 users. To replicate this data, users with different information were generated randomly or randomly selected from a list of predefined values, and the user information within each request was replaced by the random generated values. Other information which is not correlated to a user, but to a request, was generated randomly.

The set T of 50,702 requests were manufactured as the training set for the anomaly detection system. The system took approximately five (5) hours to learn the entire dataset T. Approximately, the system took 0.36 of a second to learn each request which shows that the system is quite fast. In addition, the time can be decreased further by updating the code to make it more efficient, and remove code which enables logging.

4.2 Results of Synthesised Datasets

There are four (4) possible outcomes when predicting the purpose of requests, which include the following:

Table 3. Possible outcomes in the classification of requests

True Positive (TP)	Requests that are predicted correctly as Anomalous
False Positive (FP)	Requests that are predicted as Anomalous but in reality are Legitimate
False Negative (FN)	Requests that are predicted as Legitimate but in reality are malicious
True Negative (TN)	Requests that are predicted correctly as Legitimate

First, the detection system was supplied with the training dataset, which did not contain any attacks and thus could safely be labelled as genuine.

4.2.1 Brute Force Attacks

A total of 87 brute force attacks were simulated and labelled as anomalous. Most of these attacks were generated using the application named Brutus (HooBieNet, 2002). However, 10 of the attacks were created manually by submitting them through the web user interface. Four different method types were used to attack, which include GET, HEAD, PUT, and POST. Figure 4 shows the results obtained at different threshold levels, when the attacks were submitted to the detection system. The requests that are not identified as anomalous can be considered as False Negatives (FN). The number of FNs are independent of the threshold applied, this could be attributed to brute force attacks not being able to be filtered by a threshold level.

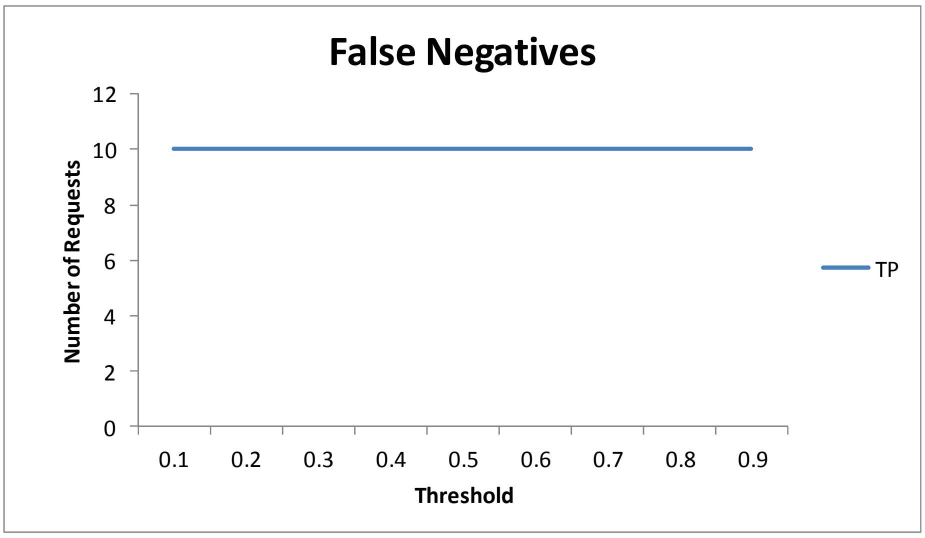

Figure 4. Number of FN requests at different thresholds for Brute Force attacks

Figure 5 displays the models that contributed to the rate of TP during the detection process. After analysing the results obtained, it was discovered that a large subset of the brute force attacks were discovered by the Access Frequency Model since the attacks were launched without any interval in between. Furthermore, the Model responsible with Request type discovered attacks which had been requested using the HEAD and PUT method types. The Attribute Character Distribution Model also predicted some requests as being anomalous since the password field was not conformant with the passwords being used, during the training period.

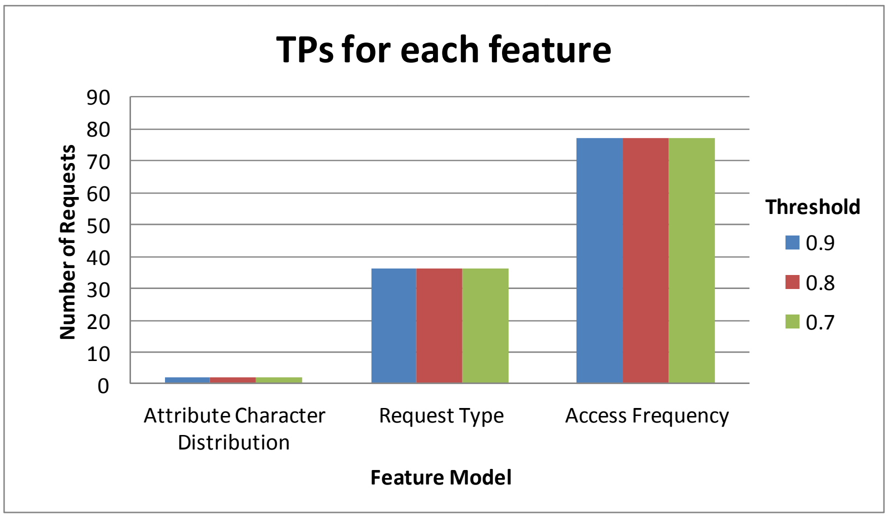

Figure 5. Number of TPs for feature model at different thresholds for Brute Force Attacks

4.2.2 SQL Injection Attacks

A total of 230 SQL Injection attacks were simulated and labelled as anomalous. Most of these attacks were submitted on the advance search page since it is the page which exposes vulnerabilities for SQL Injection attacks. However, a small subset of the attacks was submitted to other pages. A series of different SQL Injection attacks obtained from the cheat sheet on RSnake (2011a) was used to launch the attacks. SQL Power Injector (Larouche, 2007) was used to submit the attacks. The results predicted by the detection system are visible in Figure 6.

The results, as shown in Figure 7, provide us with a clear view of the models which predicted the most anomalous requests. Most of the attacks were identified by the Attribute Character Distribution Model and Attribute Length Model. The Attribute Token model identified attacks which were launched on attributes with predefined values. While performing the penetration testing, the attribute order was intentionally changed in some of the attacks, in order to be discovered by the Attribute Order Model.

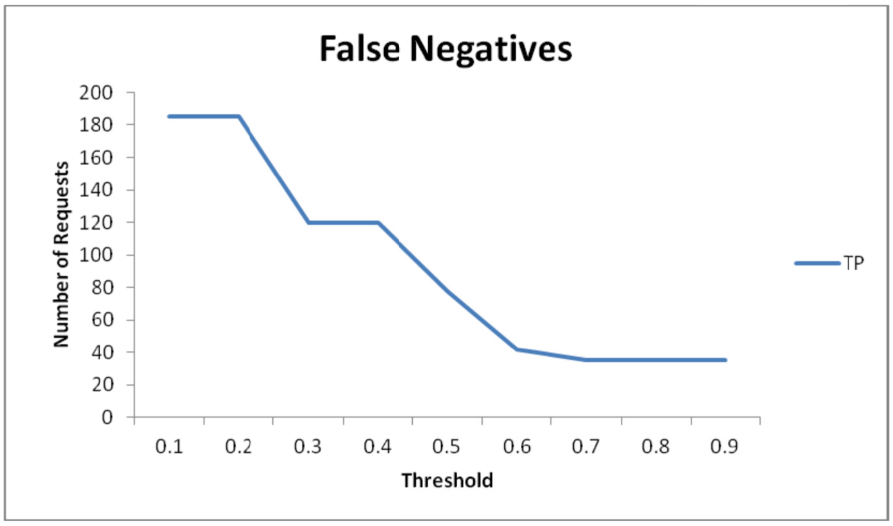

Figure 6. Number of FN requests at different thresholds for SQL Injection attacks

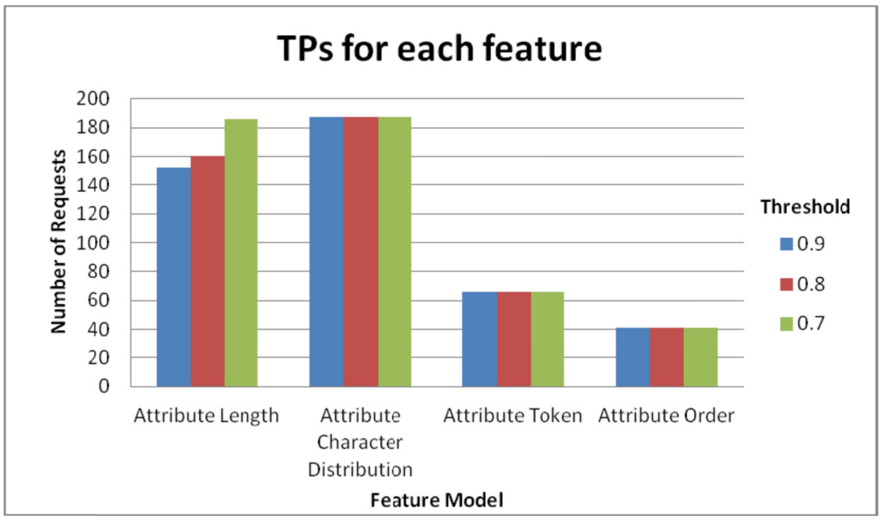

Figure 7. Number of TPs for feature model at different thresholds for SQL Injection Attacks

4.2.3 Cross Site Scripting (XSS)

A total of 205 XSS attacks were simulated and labelled as anomalous. Most of these attacks were submitted on the review page since it is the page which exposes vulnerabilities to XSS attacks. However, a small subset of the attacks was submitted to other pages to verify the accuracy of the detection system. A series of different XSS attacks obtained from the cheat sheet on RSnake (2011b) was used to launch the attacks. The WebScarab (Dawes, 2011) application was used to launch the attacks. The results predicted by the detection system are visible in Figure 8.

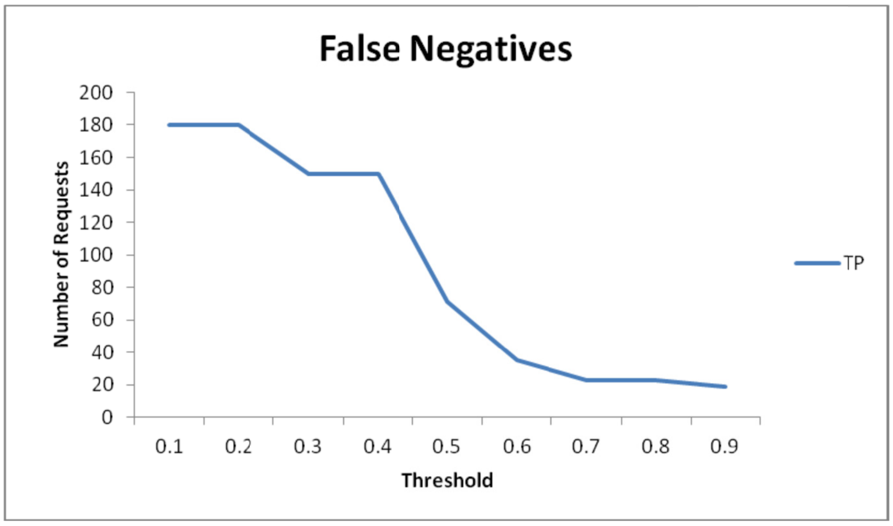

Figure 8. Number of FN requests at different thresholds for XSS attacks

Whilst confirming the initial assumptions, Figure 9 shows that the models which predicted the most XSS attacks included the Attribute Character Distribution Model and Attribute Length Model. The other models only predicted a small fraction of the attack dataset. For example Attribute Token Model identified attacks which were launched on attributes with predefined values.

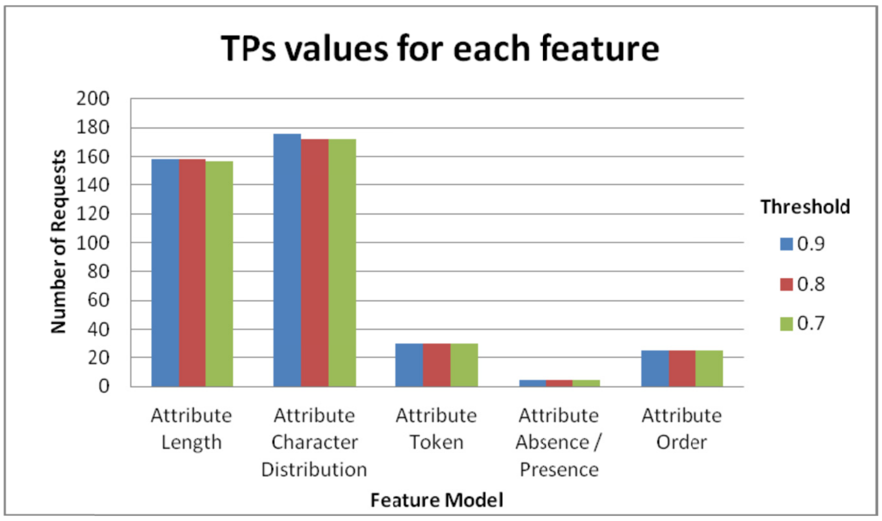

Figure 9. Number of TPs for feature model at different thresholds for XSS Attacks

4.2.4 Dr Ingham's Database of Attacks

A total of 108 attacks were synthesised through a merging process involving attacks from Ingham (2006) database and legitimate requests from the e-commerce application. The attacks have a mixture of Buffer overflow, Input validation error, and URL decoding error. The results predicted by the detection system, for this dataset, are visible in Figure 10.

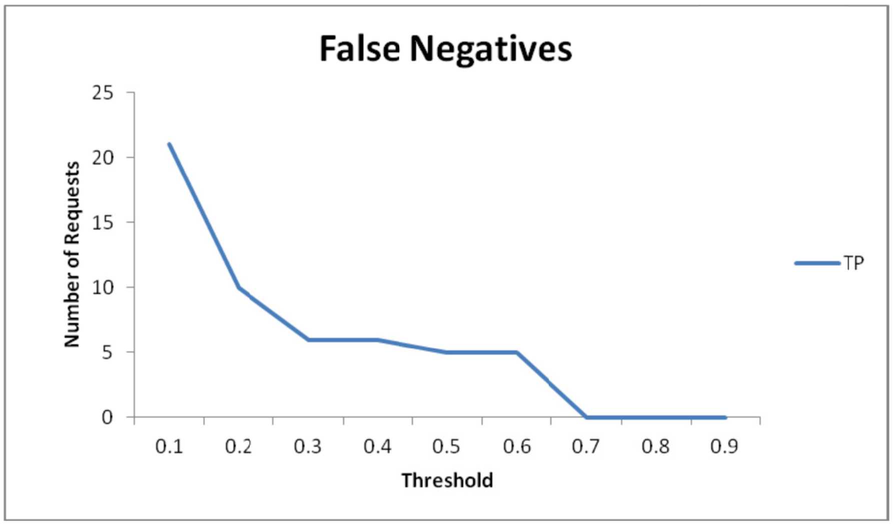

Figure 10. Number of FN requests at different thresholds for Dr Ingham's Attacks

Figure 11 and Figure 12 show the models which predicted the attacks. Buffer overflow attacks were assumed to be predicted by the Attribute Length Model, and this has been confirmed by Figure 11. Also, URL decoding errors were assumed to be predicted by the Request Uri Model, which in fact did as Figure 12 explains.

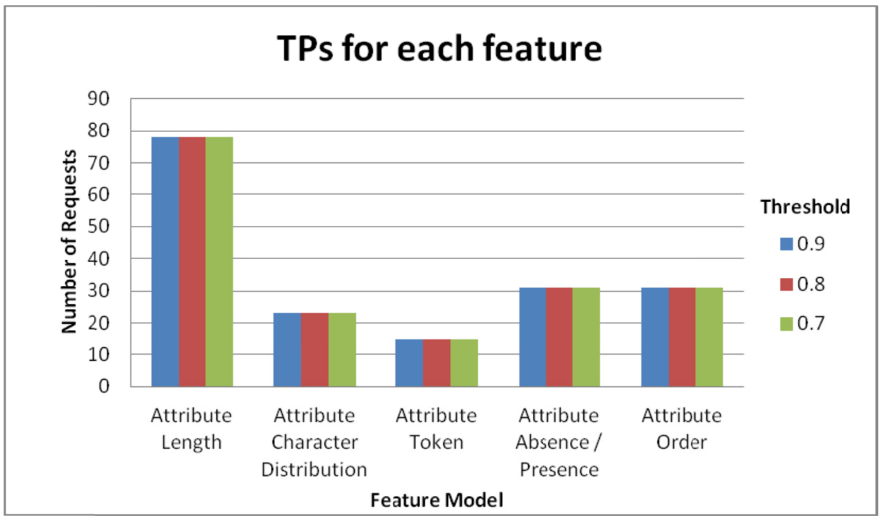

Figure 11. Number of TPs for feature model at different thresholds for Ingham (2006) Attacks

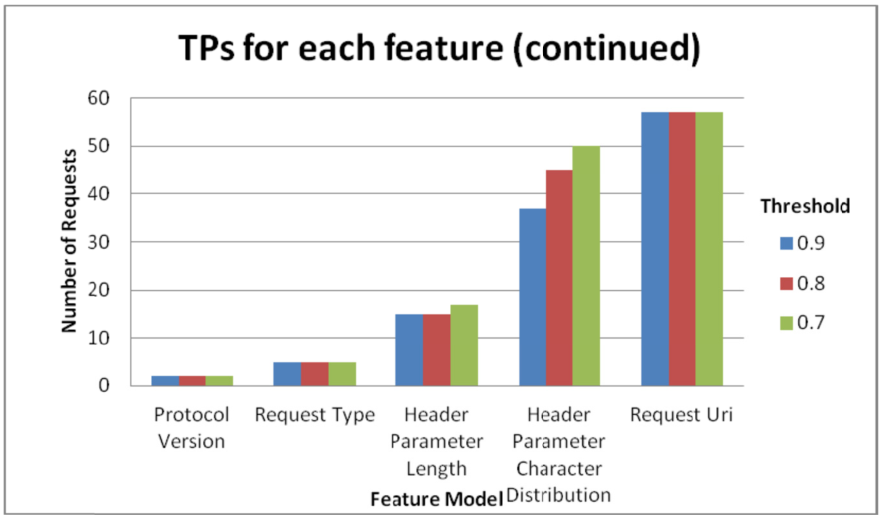

Figure 12. Number of TPs for feature model at different thresholds for Ingham (2006) Attacks (continued ...)

4.3 Systems Performance

Choosing *0.7* as our suitable threshold for the anomaly detection system, the total of false *positives is 2085* and the total of false negatives is *68. A confusion matrix, as stated by* Bhowmik (2011), is used to calculate the performance of the anomaly detection system. Since the sample of legitimate traffic is much greater than the sample of attacks, a random sample of about 4998 requests was selected from the legitimate traffic and is used to calculate the performance of the system. At 0.7 threshold mark, a total of 190 requests classified as anomalous from 4998 requests.

Table 4 shows the confusion matrix that contains information about the actual and predicted classification. Performance is calculated using the data within the matrix. Table 5 shows confusion matrix built on simulated data. The results show that the anomaly detection system has some errors in its prediction, with an accuracy rate of 95%. The equation used to calculate the accuracy rate is found in Table 6. Thought the accuracy rate is an indicative calculation, there are cases that this metric is not an adequate performance metric.

Table 4. Confusion matrix

		Labelled	
		Legal	Fraud
Predicted	Legal	True Negative (TN)	False Positive (FP)
	Fraud	False Negative (FN)	True Positive (TP)

Table 5. Confusion matrix of the synthesised data set used to test the system

		Labelled	
		Legal	Fraud
Predicted	Legal	4808	68
	Fraud	190	562

Table 6. Performance metric

Metric Name	Equation	Value
Accuracy(AC)	$AC = \frac{TN + TP}{TN + FP + FN + FP}$	0.95

4.4 Evaluation of Data from Honeypot

The performance of the anomaly detection system calculated before, gives a broad understanding of the accuracy in the prediction of the HTTP queries on the e-commerce honeypot system. The data collected on the honeypot system ranges from 10th October 2011 to 4th December 2011; 8 weeks of data and a total of 12773 requests. The collected data is divided into week batches as follows:

Table 7. Honeypot's definition of data batches

Batch Code	*From Date*	*To Date*	*No. Requests*
B1	10th October 2011	16th October 2011	4017
B2	17th October 2011	23rd October 2011	1514
B3	24th October 2011	30th October 2011	1551
B4	31st October 2011	6th November 2011	719
B5	7th November 2011	13th November 2011	495
B6	14th November 2011	20th November 2011	2290
B7	21st November 2011	27th November 2011	808
B8	28th November 2011	4th December 2011	1383

Each batch was passed through the detection system and the results were recorded. After analysing the results of each batch, we found that the majority of the requests on the e-commerce honeypot were attacks related to information gathering, mostly trying to access the admin section of the e-commerce web application. However, there were other requests which are considered as False Positives since they are accessing resources which were not included in the training set. Figure 13 contains the number of anomalous requests predicted by the Request Uri Model for each batch.

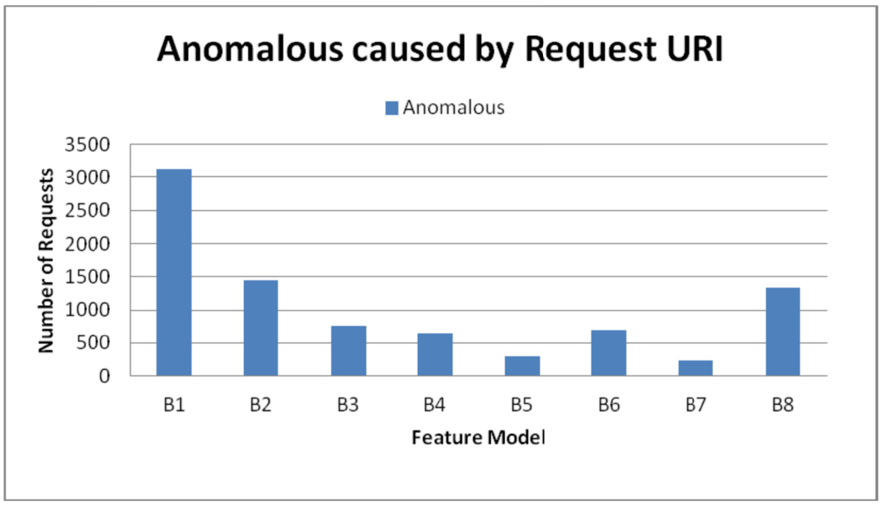

Figure 13. Anomalous requests predicted by Request Uri Model for each batch

The results show that 81% of the requests on the e-commerce honeypot were predicted as anomalous. Due to the lack of accuracy in the detection system, there could be some requests which are False Positives (FP), while others False Negatives (FN). However, the rate of FP and FN is quite small and the number of FP and FN is small as well. FPs are labelled as legal while FN are malicious (see Table 5). If we had to consider them as being almost equal, the two numbers will cross each other, thus proving that 81% is the actual percentage of attacks on the honeypot. This shows that the honeypot was successful, since it attracted more malicious users than legitimate ones. Furthermore, a substantial number of attacks were launched on the e-commerce honeypot system, though most of these attacks were related to information knowledge. Figure 14 shows the percentage of anomalous request in each batch at different threshold values. The number of anomalous requests are independent of the threshold applied, this could be attributed to anomalous requests not being able to be filtered by a threshold level.

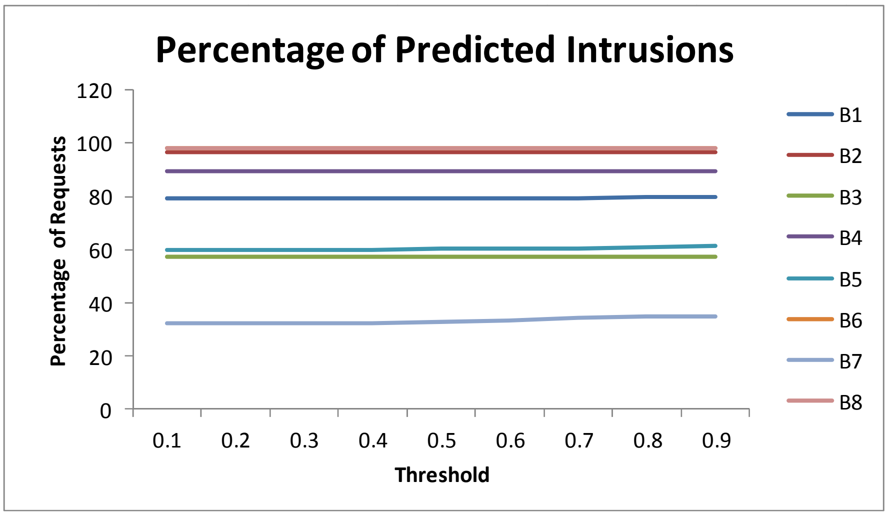

Figure 14. Percentage of anomalous requests for each batch at different thresholds

4. Conclusions

E-commerce fraud has seen an increase in the last few years and action needs to be taken to prevent it. Computer Intrusion is one of the techniques used to perpetrate fraud. Although traditional security techniques prevent malicious users from acquiring confidential information about clients and from committing fraud, there can be various vulnerabilities within an e-commerce web application which allow intrusions by malicious users. Fraud perpetrators take advantage of these vulnerabilities and penetrate the system to steal valuable information that can be used to perform fraud.

E-commerce security within this project is upgraded to another level. The proposed system learns the characteristics of e-commerce web applications by creating profiles of the associated HTTP requests. The system generates different profiles using supervised learning. Once profiles are created, the system can be used to monitor incoming traffic and classify HTTP requests as legitimate or malicious. The profiles generated by the system are based on various features which help distinguish anomalies queries from genuine ones.

The list of features extracted from the HTTP queries include unknown or abnormal ordering of parameters, abnormal length and character distribution of HTTP query parameters, illegal use of HTTP protocol versions and HTTP methods, abnormal length and character distribution of HTTP header parameters, illegal access to specific server resources and access frequency of users.

The benefit of this proposed system is that profiles are learnt with minimal human interventions unlike other Intrusion Detection Systems (IDS). The detection capabilities of the system can be adjusted to suit specific needs of the business, by tuning the threshold values used for the detection process.

To evaluate the system, a labelled dataset of HTTP queries including both malicious and genuine queries was needed. A good database that contained labelled requests could not be obtained and so synthesised data was generated to help us test the system.

The synthesised legitimate dataset was devised by randomly replacing the values of HTTP queries collected from logs (application and server) which were generated by a simulated user. This dataset was used to train the system. Attacks were synthesised by constructing malicious HTTP query using attack values obtained from Ingham (2006) database infused with legitimate HTTP queries obtained from the simulated user. Nonetheless, the dataset of attacks was further increased by performing penetration testing (OWASP, 2008) on the e-commerce web application.

The results of the experiments were presented in this article. From the evaluation it is evident that fraud detection in e-commerce can be reduced using data mining techniques together with statistical analysis. With the given datasets, our approach proofed to be promising though it might need further full-scale tests and configuration flexibilities. The results of the evaluation have been presented as bar-chats to help visualise the performance and accuracy of the proposed system. Moreover, these will help system administrators to easily understand the state of the system and easily identify attacks, if the system had to be taken to a production scale.

From the results it is evident that the system successfully detected a high percentage of malicious attacks, but there were also a number of false positives and false negatives. In certain scenarios, it is very hard to detect if an HTTP query is benign or anomalous from just a simple request. More information is needed in certain cases. However, it can be safely assumed that usually a malicious user will submit several different attacks to successfully perpetrate fraud, unless he knows all the vulnerabilities of the system already. Thus if not all of the HTTP queries are detected as malicious, but the majority are, the system can still be considered useful. By grouping the requests into sessions, there is a possibility that the system gives better results.

The assumptions made in this project, might not be comprehensive since if a new web page is created on the website and the system has not learnt of its existence, then the HTTP request to this page will be considered anomalous, and that might not be the case. Nonetheless, if a web page has been changed and the attribute parameter names changed or the orders changed, then the system must be re-trained with the new parameters since all HTTP queries with new parameters will be considered malicious unless the system is trained.

There is continuous interest in the area of fraud detection in e-commerce. The proposed system is considered a novel approach to the detection of computer intrusion attacks in e-commerce web applications. The techniques used can be further investigated to reduce the number of false positives to a possible minimum. Furthermore, from the results obtained in the evaluation, it is concluded that there is no single technique which works for most attacks. The more features are extracted, the more information is gathered and the better the detection outcome. However, a balance must be reached between accuracy and efficiency.

As a future reference, the system should handle the presence of attacks in the training set. If such a feature is enabled, the administrators of the web application do not need to collect genuine request to be used as training data. Assuming that the majority of the requests in e-commerce are legitimate, the system can be left in training mode for a considerable amount of traffic and switched to detection mode, once valid profiles have been generated.

Another problem is that the system does not allow changes in the web application. The web application might detect changes as a possible threat. There should be a mechanism in place, that whenever changes are deployed on the web application, the system is trained again to generate new profiles, amend old ones and possibly also delete unnecessary ones. Ideally, the system should automatically recognise a change and start training.

The ultimate goal is to allow the system to perform anomaly detection in real time. The system should be able to handle millions of requests per day and ideally the system should not trigger any false alarms.

References

Almadhoob, A., & Valverde, R. (2014). A cybercrime prevention in the kingdom of Bahrain via IT security audit plans. *Journal of Theoretical of Applied Information Technology, 65*(1).

Barfar, A., & Mohammadi, S. (2007). Honeypots: intrusion deception. *ISSA Journal*, 28-31.

Berendt, B., Mobasher, B., & Spiliopoulou, M. (2002) *Web Usage Mining for E-Business Applications.*, ECML/PKDD-2002 Tutorial edn.HHL. Retrieved November 15, 2011, from http://ecmlpkdd.cs.helsinki.fi/pdf/berendt-2.pdf

Bhattacharyya, S., Jha, S., Tharakunnel, K., & Westland, J. C. (2011). Data mining for credit card fraud: A comparative study. *Decision Support Systems, 50*(3), 602-613. http://dx.doi.org/10.1016/j.dss.2010.08.008

Bhowmik, R. (2011). Detecting Auto Insurance Fraud by Data Mining Techniques. *Journal of Emerging Trends in Computing and Information Sciences, 2*(4), 156-162.

Bolton, R. J., & Hand, D. J. (2002). Statistical fraud detection: A review. *Statistical Science*, 235-249.

Brause, R., Langsdorf, T., & Hepp, M. (1999). Neural Data Mining for Credit Card Fraud Detection. *Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence* (p. 103). IEEE Computer Society.

Chang, S. S., & Chiang, M. S. (2005). An e-intelligence approach to e-commerce intrusion detection. *Granular Computing, 2005 IEEE International Conference on* (p. 401).

Corona, I., & Giacinto, G. (2010). Detection of Server-side Web Attacks. In T. Diethe, N. Cristianini, & J. Shawe-Taylor (eds.), *Workshop on Applications of Pattern Analysis* (p. 160). Department of Electrical and Electronic Engineering, University of Cagliari, Italy.

Dawes, R. (2011) *OWASP WebScarab Project*. Retrieved December 16, 2011, from https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

Feng, T., & Murtagh, K. (2000). Towards knowledge discovery from WWW log data. *Information Technology: Coding and Computing, 2000. Proceedings. International Conference on* (p. 302).

Gaarudapuram, S. R. (2008) *Data processing for anomaly detection in web-based applications*, Dissertation (MA), Oregon State University. Retrieved November 17, 2011, from http://hdl.handle.net/1957/8176

HooBieNet. (2002). *Brutus - The Remote Password Cracker*. Retrieved December 18, 2011, from http://www.hoobie.net/brutus/

Ingham, K. (2006) *HTTP-delivered attacks against web servers*. Retrieved December 14, 2011, from http://www.i-pi.com/HTTP-attacks-JoCN-2006/

Jaquith, A. (2002). *The Security of Applications: Not All Are Created Equal*, @Stake, Inc. Retrieved July 27, 2011, from http://www.securitymanagement.com/archive/library/atstake_tech0502.pdf

Katzgrau, K. (2008). *KLogger*. Retrieved September 15, 2011, from http://codefury.net/projects/klogger/

Kruegel, C., Vigna, G., & Robertson, W. (2005). A multi-model approach to the detection of web-based attacks. *Computer Networks, 48*(5), 717-738. http://dx.doi.org/10.1016/j.comnet.2005.01.009

Larouche, F. (2007). *SQL Power Injector Product Information*. Retrieved December 17, 2011, from http://www.sqlpowerinjector.com/

Lee, S. Y., Low, W. L., & Wong, P. Y. (2002). Learning fingerprints for a database intrusion detection system. In *Computer Security—ESORICS 2002* (pp. 264-279). Springer Berlin Heidelberg.

Meyer, R. (2008). Detecting Attacks on Web Applications from Log Files. *Information Security Reading Room*. Retrieved October 25, 2011, from http://www.sans.org/reading_room/whitepapers/logging/detecting-attacks-web-applications-log-files_2074

Mookhey, K. K. (2010). *Common Security Vulnerabilities in e-commerce Systems*. Symantec. Retrieved July 26, from http://www.symantec.com/connect/articles/common-security-vulnerabilities-e-commerce-systems

MySQL. (2012). *MySQL The world's most popular open source database*. Homepage of MySQL. Retrieved January 26, from http://www.mysql.com/

OsCommerce. (2012). *Welcome to osCommerce!* Homepage of osCommerce. Retrieved January 26, 2012, from http://www.oscommerce.com/

OWASP. (2008). *OWASP Testing Guide* (3rd ed.). OWASP Foundation.

Penya, Y. K., Ruiz-Agúndez, I., & Bringas, P. G. (2011). Integral Misuse and Anomaly Detection and Prevention System. In P. Skrobanek (ed.), *Intrusion Detection Systems* (p. 172). InTech.

PHP. (2012). *PHP: Hypertext Preprocessor*. Homepage of PHP. Retrieved January 26, 2012, from http://www.php.net/

Phua, C., Lee, V. C. S., Smith-Miles, K., & Gayler, R. W. (2010). A Comprehensive Survey of Data Mining-based Fraud Detection Research. *Artificial Intelligence Review*. Retrieved July 27, 2011, from http://arxiv.org/abs/1009.6119

RSnake. (2011a). *SQL Injection cheat sheet*. Retrieved December 19, 2011, from http://ha.ckers.org/sqlinjection/

RSnake. (2011b). *XSS (Cross Site Scripting) Cheat Sheet*. Retrieved December 19, 2011, from http://ha.ckers.org/xss.html

Stephens, J., & Valverde, R. (2013). Security of E-Procurement Transactions in Supply Chain Reengineering. *Computer & Information Science, 6*(3).

Tan, H. S. (2002). E-fraud: current trends and international developments. *Journal of Financial Crime, 9*(4), 347-354. http://dx.doi.org/10.1108/eb026034

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM journal on computing, 1*(2), 146-160. http://dx.doi.org/10.1137/0201010

Yufeng, K., Chang-Tien, L., Sirwongwattana, S., & Yo-Ping, H. (2004). 'Survey of fraud detection techniques', *Networking, Sensing and Control, 2004 IEEE International Conference on* (p. 749).