Software Design Level Vulnerability Classification Model

Shabana Rehman *shabana.infosec@gmail.com*
Department of Information System
Salman bin Abdul Aziz University
Al-Kharj, KSA

Khurram Mustafa *kmustafa@jmi.ac.in*
Department of Computer Science
Jamia Millia Islamia
New Delhi, 110025, India

Abstract

Classification of software security vulnerability no doubt facilitates the understanding of security-related information and accelerates vulnerability analysis. The lack of proper classification not only hinders its understanding but also renders the strategy of developing mitigation mechanism for clustered vulnerabilities. Now software developers and researchers are agreed on the fact that requirement and design phase of the software are the phases where security incorporation yields maximum benefits. In this paper we have attempted to design a classifier that can identify and classify design level vulnerabilities. In this classifier, first vulnerability classes are identified on the basis of well established security properties like authentication and authorization. Vulnerability training data is collected from various authentic sources like Common Weakness Enumeration (CWE), Common Vulnerabilities and Exposures (CVE) etc. From these databases only those vulnerabilities were included whose mitigation is possible at the design phase. Then this vulnerability data is pre-processed using various processes like text stemming, stop word removal, cases transformation. After pre-processing, SVM (Support Vector Machine) is used to classify vulnerabilities. Bootstrap validation is used to test and validate the classification process performed by the classifier. After training the classifier, a case study is conducted on NVD (National Vulnerability Database) design level vulnerabilities. Vulnerability analysis is done on the basis of classification result.

Keywords: Security Vulnerabilities, Design Phase, Classification, Machine Leaning, Security Properties

1. INTRODUCTION

Developing secure software remains a significant challenge for today's software developers as they still face difficulty in understanding the reasons of vulnerabilities in the existing software. It is vital to be able to identify software security vulnerabilities in the early phases of SDLC (Software Development Lifecycle) and one of early detection approaches is to consult with the prior known vulnerabilities and corresponding fixes [1]. Identification of candidate security vulnerability pays a substantial benefit when they are deals in early phases like requirement and design phases of the software [2]. Classification of vulnerabilities is fruitful in understanding the vulnerabilities better and classification also helps in mitigating group of vulnerabilities. Identifying and mitigating security vulnerabilities is no doubt a difficult task therefore taxonomy is developed that can classify vulnerabilities into classes and this will help designer to mitigate cluster of vulnerabilities. There are number of approaches of taxonomy development in past, like [3,4,5,6] etc, but no one ever propose any taxonomy that classify design level vulnerabilities on the basis of security properties. We have already proposed a taxonomy in [7], as shown in Table 1.0 (a) in which, priori classification is proposed and vulnerabilities are classified manually. But in this classification there is chance of 'Hawthorne Effect', it also largely depends on the expertise of the classifier. Therefore here we are creating a classifier that can classify a vulnerability data automatically. Machine learning is now a popular tool in the automation task. Researchers have explored the

use of machine learning techniques to automatically associate documents with categories by first using a training set to adapt the classifier to the feature set of the particular document set [8]. Machine learning is a relatively new approach that can be used in classifying vulnerabilities. Therefore here Classifier is proposed, that classifies vulnerabilities on the basis of previously identified vulnerability and can help designer to place vulnerability in the predefined vulnerability classes that are based on the security properties of the software. Therefore mitigation mechanism can be applied for the whole class of vulnerabilities. In this classifier, first data pre-processing is done like text stemming, stop word removal, case transformation then SVM (Support Vector Machine) is used for the final classification with the regression model. Several conclusions are drawn after applying a classification. At last using this classifier NVD (National Vulnerability Database) vulnerabilities are classified and analyzed.

First Level	Second Level	Third level	Fourth Level
Access Control	Access Control at Process Level	Authentication	Missing Authentication procedure
			Insufficient Authentication procedure
			Wrong Authentication procedure
		Authorization	Missing Authorization procedure
			Insufficient Authorization procedure
			Wrong Authorization procedure
		Audit & logging	Missing Audit and logging
			Insufficient Logging or Audit of information
			Wrong Audit or Logging of information
	Access Control at Communication Level	Secured Session Management	Missing Secured Session management
			Insufficient Secured Session Management
			Wrong Secured Session Management
		Secured Information Flow	Missing Encryption of Sensitive Data During Transmission
			Insufficient Encryption of Sensitive Data during Transmission
			Wrong Encryption of Sensitive Data during Transmission
	Exposures leading to Access Violation	Exposures in Error Message	Missing Secured Error Message
			Insufficient Secured Error Message
			Wrong Secured Error Message
		Predictable Algorithm /sequence numbers/file names	Missing Randomness in the Random Sequence Ids
			Insufficient Randomness in the Random Sequence Ids
			Wrong Randomness in the Random Sequence Ids or Wrong Choice of File Name
		User Alertness	Missing User Alerting Information
			Insufficient User Alerting Information
			Wrong User Alerting Information

TABLE 1.0 (a): Taxonomy of Design Level Vulnerabilities

While considering the number of classes in the proposed classifier, we consider only 'access control at process level' and 'access control at communication level' and all the other type of vulnerabilities are considered in the 'Others' class. Because 'exposure leading to access violation'

class covers a large domain of vulnerabilities and needs a separate study, therefore after exploring the domain of this class, we exclude this from the classifier and will consider for the future work.
Rest of the paper is organized as follows, in section 2, related works in the vulnerability classification is discussed, and then in section 3, the development process of vulnerabilities classification model is explained in detail. Classification of vulnerabilities using developed classifier is done in section 4. Conclusion and future work are discussed in section 5.

2. RELATED WORK

There are many classification approaches using machine learning techniques like [9] proposed uses a ontological approach to retrieving vulnerability data and establishing a relationship between them, they also reason about the cause and impact of vulnerabilities. In their ontology vulnerability management (OVM), they have populated all vulnerabilities of NVD (National Vulnerability Database), with additional inference rules, knowledge representation, and data-mining mechanisms. Another relevant work in vulnerability classification area is done by [10], they proposed a CVE categorization framework that transforms the vulnerability dictionary into a classifier that categorizes CVE(Common Vulnerability and Exposure) with respect to diverse taxonomic features and evaluates general trends in the evolution of vulnerabilities. [11], in their paper, entitled “Secure software Design in Practice” presented a SODA (*a Security-Oriented Software Development Framework*), which was the result of a research project where the main goal had been to create a system of practical techniques and tools for creating secure software with a special focus on the design phase of the software. Another approach of categorizing vulnerabilities is that of [12]. In their paper [12], they looked at the possibilities of categorizing vulnerabilities in the CVE using SOM. They presented a way to categorize the vulnerabilities in the CVE repository and proposed a solution for standardization of the vulnerability categories using a data-clustering algorithm. [13], proposed SecureSync, an automatic approach to detect and provide suggested resolutions for recurring software vulnerabilities on multiple systems sharing/using similar code or API libraries. There are many other vulnerability classification approaches like [14,15,16], but all the above mentioned approaches are either to generic in nature or they cannot be used to classify vulnerabilities on the basis of security properties of the software. Therefore in this research work we are proposing a classifier that is developed using machine learning techniques and is very specific to the design phase of the software. In the next section, a development stage of the classifier is explained.

3.0 DESIGN LEVEL VULNERABILITIES CLASSIFICATION MODEL

Software vulnerability databases are essential part of software security knowledgebase. There are a number of vulnerability databases that exchanges software vulnerability information however, its impact on vulnerability analysis is hindered by its lack of categorization and generalization functionalities [10]. To extract useful and relevant information from these databases, lots of manual work is required. For example, if software developer wants to know about the most common and severe vulnerability prevailing in a current software in a particular period of time then he has to study all the vulnerability descriptions published during that period, then he has to classify those vulnerability based on his own criteria and then he has to check the severity rating provided by various experts. This is very unreliable, tedious and protracted task. Using a proposed classification model, researchers and developers can easily classify design level vulnerabilities and identify a mitigation mechanism in the form of design pattern, in the early phases of the SDLC. The classification results with severity rating can further be used to calculate the risk of vulnerability occurrence at the design phase of the software.

Automated text classifier is basically used to classify the text in the predefined classes. An abstract view of the classifier is shown in Fig 3.0 (a). In proposed design level vulnerability classifier, first text is pre-processed using various processes like tokenization, case transformation, stop-word removal and stemming, then SVM (Support Vector Machine) is used to classify the text and finally bootstrap validation is used to test and validate the results. The development process of the classifier is explained in Fig 3.0 (b).

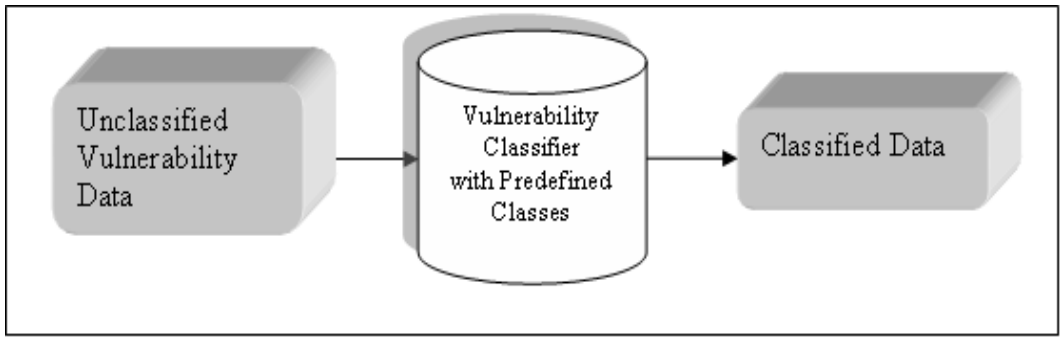

FIGURE 3.0 (a): Abstract Vulnerability Classifier

The vulnerability categorization framework proposed by [10] is similar to this design level classifier. But Chen's framework is a generalized categorization framework that is developed to classify all the vulnerabilities of CVE, on the bases of classification categories of BID, X-force and Secunia. The training data in their framework is also taken from these vulnerabilities databases only.

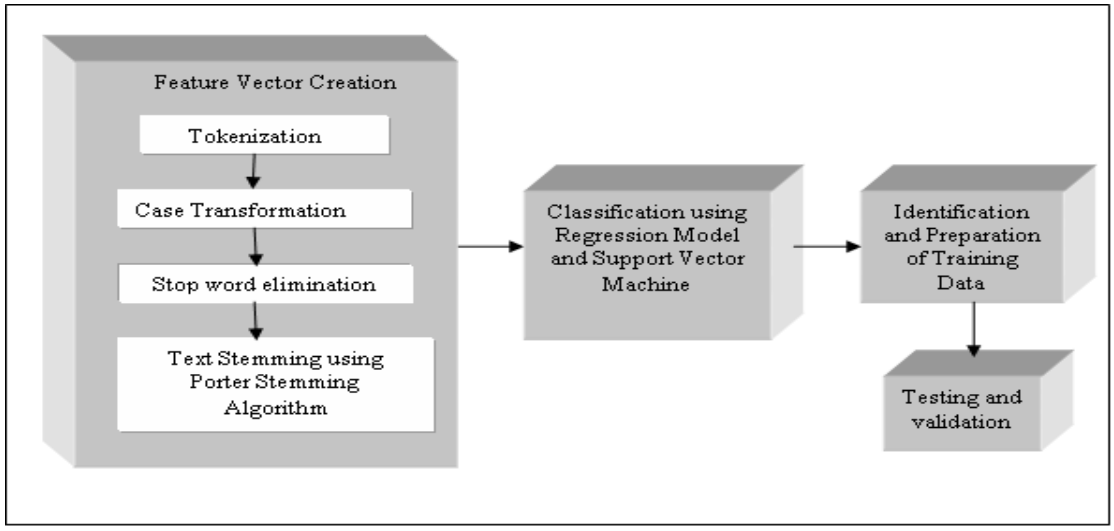

FIGURE 3.0 (b): Design Level Vulnerabilities Classification Process

In our design level vulnerability classifier, only design level vulnerabilities are classified and in training data only those identified vulnerabilities are considered which can be mitigated at the design level of the software. Moreover the classes are defined on the basis of security properties of the software like authentication, authorization etc., which are generally considered while developing the security design patterns of the software. Therefore after classification developers/researchers can priorities prevailing vulnerabilities class before choosing security design pattern.

3.1 Feature Vector Creation

The vulnerabilities in the CVE are defined in the natural language form. Therefore only way to identify a feature vector using the vulnerability description is the frequency of keywords in the description. Therefore feature vector are identified by the keywords used in the description of the vulnerabilities. To make vulnerability description into a structured representation that can be used by machine learning algorithms, first the text will be converted into tokens and after stemming and

stop word removal, and case transformation. There are five steps in the feature creation process, specified as follows:
a). Tokenization
b). Case Transformation
c). Stopword elimination
d). Text stemming of CVE entries
e). Weight Assignment

a).Tokenization
The isolation of word-like units form a text is called tokenization. It is a process in which text stream is to break down into words, phrases and symbols called tokens [17]. These tokens can be further used as input for the information processing. In order to convert text in machine learning form, first the raw text is transformed into a machine readable form, and first step towards it is a tokenization. As shown in Fig. 3.1 (a), the raw text is first feed to the pre-processor, convert the text in the form of tokens then further morphological analysers are used to perform required linguistic analysis.

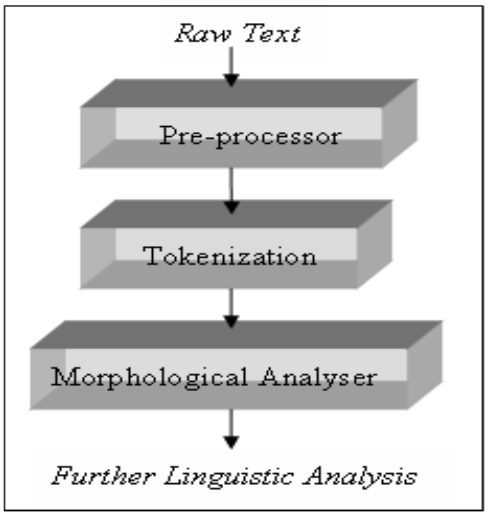

FIGURE 3.1 (a): Text transformations before linguistic analysis

In order to feed vulnerability description in machine learning process, the textual description of vulnerability is first converted in the form of tokens. In Table 3.1 (a), a vulnerabilities description is shown after tokenization.

Vulnerability ID	Vulnerability Description	Vulnerability Description after Tokenization
CVE-2007-0164	Camouflage 1.2.1 embeds password information in the carrier file, which allows remote attackers to bypass authentication requirements and decrypt embedded steganography by replacing certain bytes of the JPEG image with alternate password information.	Camouflage, embeds, password, information, in, the, carrier, file, which, allows, remote, attackers, to, bypass, authentication, requirements, and, decrypt, embedded, steganography, by, replacing, certain, bytes, of, the, JPEG, image, with, alternate, password, information.

TABLE 3.1 (a): Tokenization of Vulnerabilities

b). Case Transformation

When raw text is retrieved for processing from any source, then it contains words in both upper case as well as lower case. The machine learning algorithms reads words in different cases as different words. In order to transform all the words in the same case, Case transformation process is used. Case transformer, transforms all characters in a document to either lower case or upper case, respectively. In our case we have transformed all the words in the document in lower case.

Vulnerability ID	Vulnerability Description	Vulnerability Description after Case Transformation
CVE-2007-0164	Camouflage, embeds, password, information, in, the, carrier, file, which, allows, remote, attackers, to, bypass, authentication, requirements, and, decrypt, embedded, steganography, by, replacing, certain, bytes, of, the, JPEG, image, with, alternate, password, information	camouflage, embeds, password, information, in, the, carrier, file, which, allows, remote, attackers, to, bypass, authentication, requirements, and, decrypt, embedded, steganography, by, replacing, certain, bytes, of, the, jpge, image, with, alternate, password, information

TABLE 3.1 (b): Case Transformation of Vulnerabilities Description

c). Stop word Removal

Stop word elimination is a process of removing those tokens that are considered as only for grammatical function without adding new meaning to sentences they involve [18]. The stop word list generally consists of articles, case particles, conjunctions, pronouns, auxiliary verbs and common prepositions. There is no unique list of stop words which is always used. There are number of lists that are proposed by different researchers. A list of 418 stop word is used by Chen [10]. A similar stop word list is used in information retrieval systems Snowball [19] and Lemur [20].The vulnerability description after stop word removal is shown in Table 3.1 (c)

Vulnerability ID	Vulnerability description after tokenization and case transformation	Vulnerability description after Stopword Removal
CVE-2007-0164	camouflage, embeds, password, information, in, the, carrier, file, which, allows, remote, attackers, to, bypass, authentication, requirements, and, decrypt, embedded, steganography, by, replacing, certain, bytes, of, the, jpge, image, with, alternate, password, information.	camouflage, embeds, password ,information, carrier, file, allows, remote, attackers, bypass, authentication, requirements, decrypt, embedded, steganography, replacing, bytes, jpge, image, alternate, password, information.

TABLE 3.1 (c): Stop word removal from vulnerabilities

d). Text Stemming

Uses of stemming algorithms in modern information retrieval (IR) systems are common these days. Stemming algorithms are helpful for free text retrieval, where search terms can occur in various different forms in the document collection. Stemming makes retrieval of such documents independent from the specific word form used in the query [21]. To extract the information from vulnerability description, stemming algorithm can be used, so that text can be easily transformed into a machine readable form. Porter stemming algorithm is one of the popular algorithms that is generally used in the information retrieval process. Porter's algorithm consists of 5 phases of word reductions, applied sequentially. Within each phase there are various conventions to select rules, such as selecting the rule from each rule group that applies to the longest suffix. In the first phase, this convention is used with the following rule group [22]:

Rule Example

SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

The vulnerability description after applying porter stemming algorithm is shown in Table 3.1 (d).

Vulnerability ID	Vulnerability Description	Vulnerability Description after Text Stemming
CVE-2007-0164	Camouflage 1.2.1 embeds password information in the carrier file, which allows remote attackers to bypass authentication requirements and decrypt embedded steganography by replacing certain bytes of the JPEG image with alternate password information.	camouflag emb password inform carrier file allow remot attack bypass authent requir decrypt embed steganographi replac byte jpeg imag altern password inform

TABLE 3.1 (d): Stemming of words in Vulnerabilities Description

e). Weight Assignment

After text stemming, the next step is a weight assignment of each word of vulnerability description. The simplest approach is to assign the weight to be equal to the number of occurrences of term '*t*' in document *'d'*. This weighting scheme is referred to as *term frequency* and is denoted '$tf_{t,d}$' with the subscripts denoting the term and the document in order [22]. It is normally computed as follows.

$$tf_{t,d} = f_{t,d} / max_k f_{k,d} \quad \text{Eq. 3.1 (e.1)}$$

where

$f_{t,d}$ is the frequency (number of occurrence of the term '*t*' in document '*d'*) and

$max_k f_{k,d}$ (maximum number of occurrences of any term)

Thus, the most frequent term in document 'd' gets a TF as 1, and other terms get fractions as their term frequency for this document. But the disadvantage of using a term frequency is that, in this all the terms are considered equally important. In order to avoid this bias, term frequency and inverse document frequency *(tf-idf)* weighting is used. The IDF for a term is defined as follows [23].

$$idf_{t,d} = \frac{log\ |D|}{|\{d: t \in d\}|} \quad \text{Eq. 3.1 (e.2)}$$

where

- $|D|$: the total number of documents
- $|\{d : t \in d\}|$: number of documents where the term *t* appears (i.e., $\mathrm{tf}(t,d) \neq 0$)

As defined in [22], the *tf-idf* weighting scheme assigns the term '*t*' a weight in document 'd' given by

$$tf\text{-}idf_{t,d} = tf_{t,d} \cdot idf_t$$

In other words, term frequency-inverse term frequency assigns to term *'t'* a weight in document *'d'* is

- high when term occurs many times within a small number of documents ;

- lower when the term occurs fewer times in a document, or occurs in many documents;
- lowest when the term occurs in virtually all the documents.

Words / Row No.	abil	abs enc	Accep t	Access	accoria	account	actio n	…
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…
3	0.0	0.0	0.0	0.069434622 98821593	0.0	0.0	0.0	…
4	0.0	0.0	0.0	0.120849276 3637866	0.0	0.0	0.0	…
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…
7	0.0	0.0	0.0	0.071144256 6706304	0.0	0.0	0.0	…
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	…
9	0.0	0.0	0.0	0.129932379 10966035	0.0	0.0	0.0	…
10	0.0	0.0	0.0	0.079183463 69069837	0.0	0.2787299435 038924	0.0	…
…	…	…	…	…	…	…	…	…

TABLE 3.1 (e): Example set of feature vector with their tf-idf

Now after implementing all the above processing we get each vulnerability description as a *vector*, weight that is calculated on using tf-idf formula. Example set of ten rows is shown in table 3.1 (e). This vector form will be used in the scoring and ranking of vulnerabilities.

3.2 Categorization Using Support Vector Machine

Text categorization is the process of categorizing text documents into one or more predefined categories or classes. Differences in the results of such categorization arise from the feature set chosen to base the association of a given document with a given category [24]. There are a number of statistical classification methods that can be applied to text categorization, such as Naïve Bayesian [25], Bayesian Network [25], Decision Tree [26, 27], Neural Network [28], Linear Regression [29], k-NN [30]. SVM (support vector machine) learning method introduced by [31], are well-founded in terms of computational science. Support vector machines have met with significant success in numerous real-world learning tasks [25]. Compared with alternative machine learning methods including Naive Bayes and neural networks, SVMs achieve significantly better performance in terms of generalization [32, 33]

SVM classification algorithms, proposed by Vapnik [34] to solve two-class problems, are based on finding a separation between hyperplanes defined by classes of data, shown in Figure 3.2 (a).

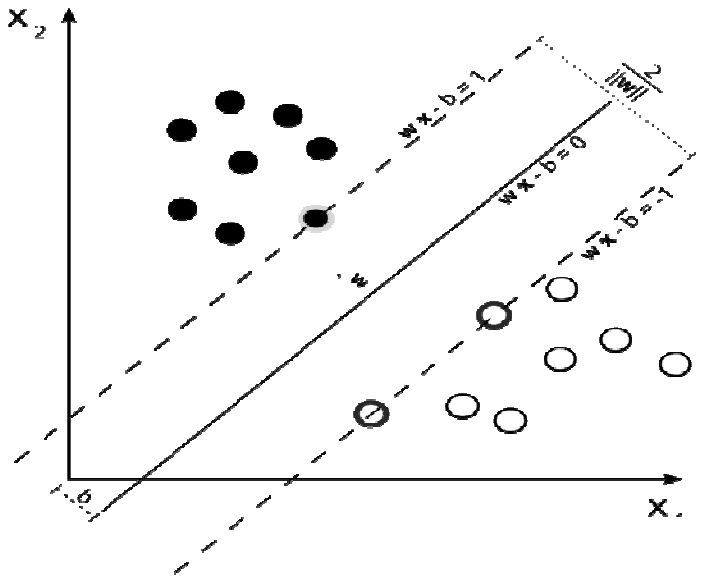

FIGURE 6.2.2: Example of SVM hyper plane pattern

This means that the SVM algorithm can operate even in fairly large feature sets as the goal is to measure the margin of separation of the data rather than matches on features [24]. The SVM is trained using pre-classified documents.

As explained in [34], for a given set of training data $T = \{.xi, yi\}$ $(i = 1,\ldots, m)$, each data point $.xi \in Rd$ with d features and a true label $yi \in Y = \{l1, \ldots, lk\}$. In case of binary classifier, label set is $Y = \{l1 = -1, l2 = +1\}$, it classifies data point in positive and negative by finding separating hyperplane. The separating hyperplane can be expressed as shown in Eq. 3.2 (a).

$$\overline{w} \cdot .\overline{x} + b = 0 \text{ ----------Eq 3.2(a)}$$

where $w \in R^d$ *is a* weight vector is normal to the hyperplane, operator $(\cdot)$ computes the inner-product of vectors w and x and b is the bias. Now we want to choose the '*w*'and *'b'* to maximize the margin, or distance between the parallel hyperplanes that are as far apart as possible while still separating the data. These hyperplanes can be described by the equations

$$w \cdot .x - b = 1 \text{ ----------Eq 3.2 (b)}$$

and

$$w \cdot .x - b = -1 \text{ ----------Eq 3.2 (c)}$$

In the case that the label set $Y = \{l1 = 1, \ldots, lk = k\}$ and $k > 2$, a multiclass SVM learning model is built with two approaches in the proposed framework: *multiclass-to binary reduction* and *multiclass-optimization* methods [10]. In the multiclass-to-binary reduction method, the learning problem in question is reduced to a set of binary classification tasks and a binary classifier is built independently for each label *lk* with the one-against-rest training technique [35]. When more than

two classes are involved then regression model is used. regression model builds a classifier using a regression method which is specified by the inner operator. For each class *i* a regression model is trained after setting the label to (+1) if the label equals *i* and to (-1) if it is not. Then the regression models are combined into a classification model. Here we are using SVM classification model, therefore Regression model is combined into SVM. In order to determine the prediction for an unlabeled example, all models are applied and the class belonging to the regression model which predicts the greatest value is chosen.

3.3 Identification and Preparation of Training Data

There are number of public and private vulnerability databases that classify vulnerabilities on different bases like cause, phase of occurrence, product specific etc. the list is shown in Table 3.3 (a). But they all are too generic in nature. The CWE (Common weakness Enumeration) [36] is a vulnerability database and portal in which each vulnerability is specified with its type, mitigation, phase of introduction and security property it belongs to. Therefore it is a best place from where vulnerability data can be collected on the basis of phase of introduction and the security property it belongs to. Fig 3.3 (a) is screenshot of the CWE window, it is showing a information that each entry of CWE contains. In this we are interested in only those vulnerabilities that can be mitigated in the design phase of the software. But in CWE the vulnerabilities are divided into number of classes and number of examples are given in the description of each class. In order to collect the training data from CWE, we explore the required security class, then collect vulnerability example from each class. Almost all the examples that are used in CWE are from CVE (Common Vulnerability and Exposure).Maximum possible numbers of examples are collected from the CWE for the training set.

S. No.	Database Name	URL
1.	Common Vulnerability and Exposures	http://cve.mitre.org/
2.	Common Weakness Enumeration	http://cwe.mitre.org/
3.	Computer Associates Vulnerability Encyclopedia	http://www3.ca.com/securityadvisor/vulninfo/browse.aspx
4.	Dragonsoft vulnerability database	http://vdb.dragonsoft.com
5.	ISS X-Force	http://xforce.iss.net/xforce/search.php
6.	National Vulnerability Database	http://nvd.nist.gov/
7.	Open Source Vulnerability Database	http://www.osvdb.org/
8.	Public Cooperative Vulnerability Database	https://cirdb.cerias.purdue.edu/coopvdb/public/
9.	Security Focus	http://www.securityfocus.com/vulnerabilities/

TABLE 3.3 (a): Vulnerability Database with their URLs

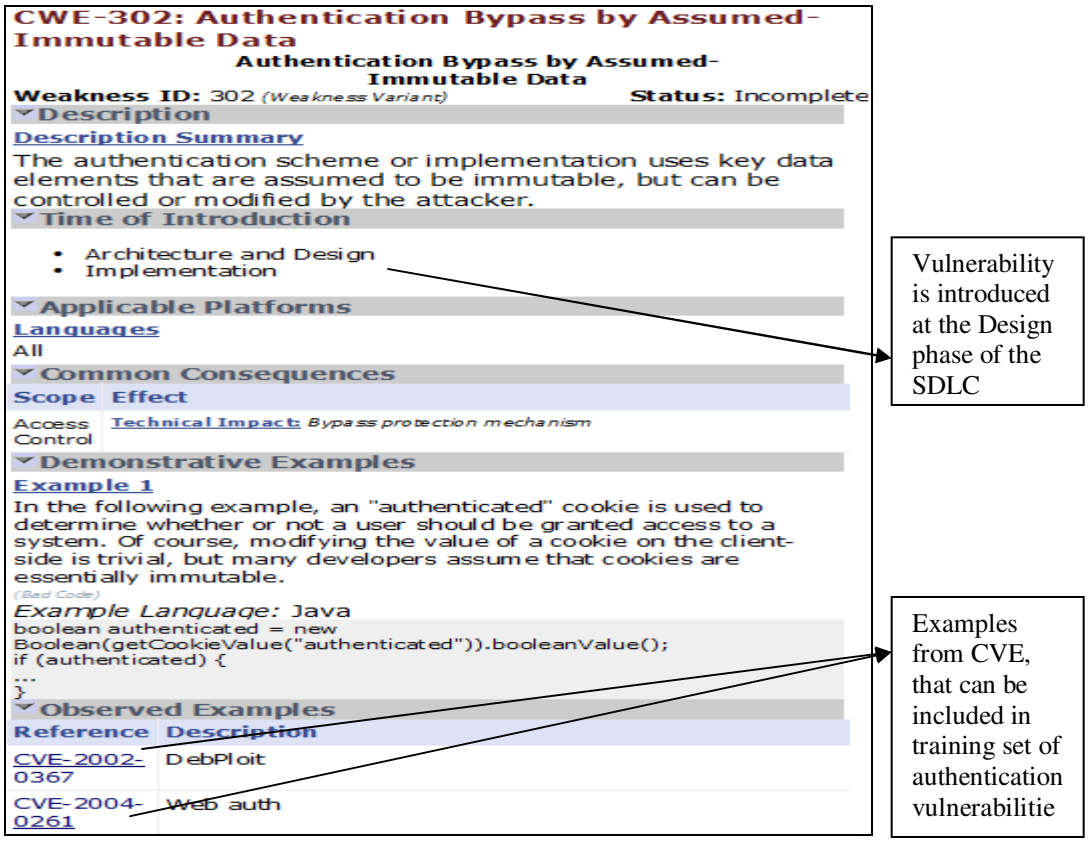

FIGURE 3.3 (a): CWE Vulnerability Class Description Window

After the exhaustive search of CWE vulnerability classes, the number of vulnerability examples that are collected, are shown in Table 3.3 (b). While collecting data from CWE, at most care is taken to include only those vulnerability classes where the time of introduction is specified as "design phase"

S. No.	Vulnerability Class	Number of Training data
1.	Authentication	54
2.	Authorization	50
3.	Audit and Logging	36
4.	Secure Information Flow	31
5.	Secure Session Management	24

TABLE 3.3 (b): Number of training data identified under each class

3.4 Testing and Validation

After the collection of training data, a SVM learning model can be built. But SVM is basically a binary classifier. As we have multiple classes with $|Y|>2$, the learning

problem is decomposed into $|Y|$ binary classification tasks with the multiclass-to-binary reduction method, and subsequently $|Y|$ binary classifiers are built with the one-against-rest training method that essentially transforms the learning task for category l_i of Y into a two-class categorization by treating data points with label l_i in the training data as positive examples and the remaining data points as negative examples [10]. We are using Rapidminer tool to implement the SVM. Regression model is used to classify the data into multiple class. After supplying the data

regression model, bootstrap validation is used to validate the classification. Fig 6.2.4 (a) is showing the screen shot of rapid miner while implementing bootstrap validation.

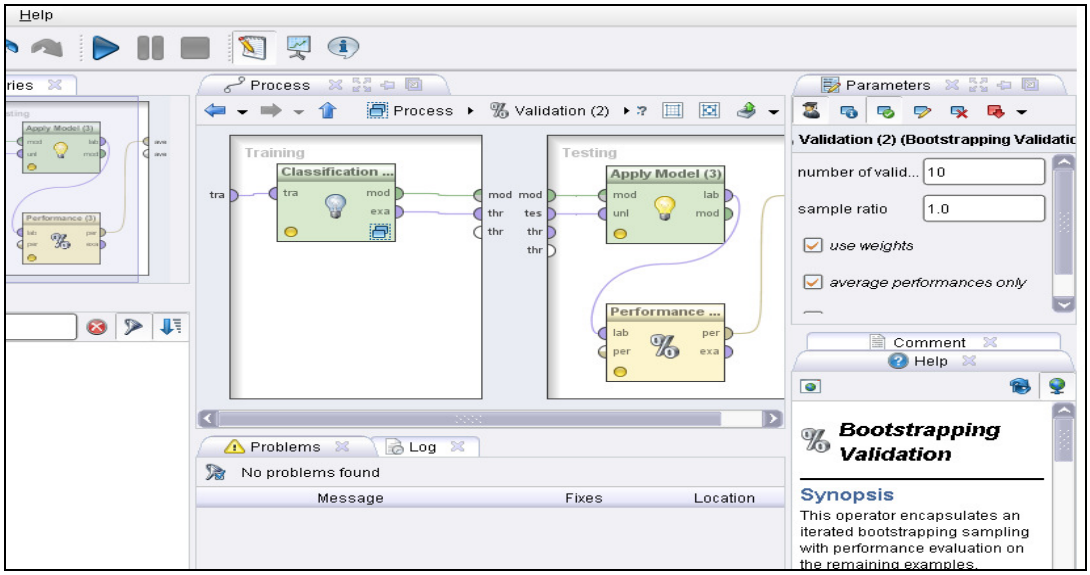

FIGURE 3.4 (a): Screen shot from 'Rapidminer Tool', while implementing bootstrap validation

3.5 Bootstrap Validation

The bootstrap family was introduced by Efron and is first described in [37]. In this method, for given dataset of size n a bootstrap sample is created by sampling n instances uniformly from the data. There are several bootstrap methods. A commonly used one is the 0.632 bootstrap. As explained in by Han and Kimber in their book 'Data Mining: Concepts and Techniques' [38], in this method, suppose we have 'd' tuples. The data set is sampled 'd'/6 times, with replacement, resulting in the bootstrap samples or the training set of d samples. The data tuple that are not included in the training, forms the test set. Now the probability for each tuple to be selected is 1/d, and the probability of not being chosen is (1- 1/d).We have to select d times, so the probability that a tuple will not be chosen during this whole time is (1-1/d)d. If d is large, the probability approaches e^{-1} = 0.368. Thus, 36.8% of tuples will not be selected for training and thereby ends up in the test set, and the remaining 63.2% will form the training set.

The sampling procedure can be recorded k times, where in each iteration, we can use the current test set to obtain the accuracy estimate of the model obtained from the current bootstrap sample. The overall accuracy of the model is then estimated as

$$\text{Acc}(M) = \sum_{i=1}^{k} (0.632 \times \text{Acc}(M_i)_{\text{test-set}} + 0.368 \times \text{Acc}(M_i)_{\text{train\_set}}) \qquad \text{Eq. 3.5 (a)}$$

where $\text{Acc}(M_i)_{\text{test-set}}$ is the accuracy of the model obtained with the bootstrap sample 'i' when it is applied to the test set 'i'. $\text{Acc}(M_i)_{\text{train\_set}}$ is the accuracy of the model obtained with bootstrap sample 'i' when it is applied to the original set of the data tuples. The bootstrap method works well with the small data set.

The whole process that is followed in making the classifier is shown in Figure 3.5 (a). the rapid miner data mining tool is used that have almost all the available data-mining process in the form of operators. First of all training data is feed to the regression model that is integrated with SVM, then 'Apply Model' operator is used to apply the created model and performance operator is used

to measure the performance of the classifier. As an output, the confusion matrix is be obtained that will show the accuracy of the classifier.

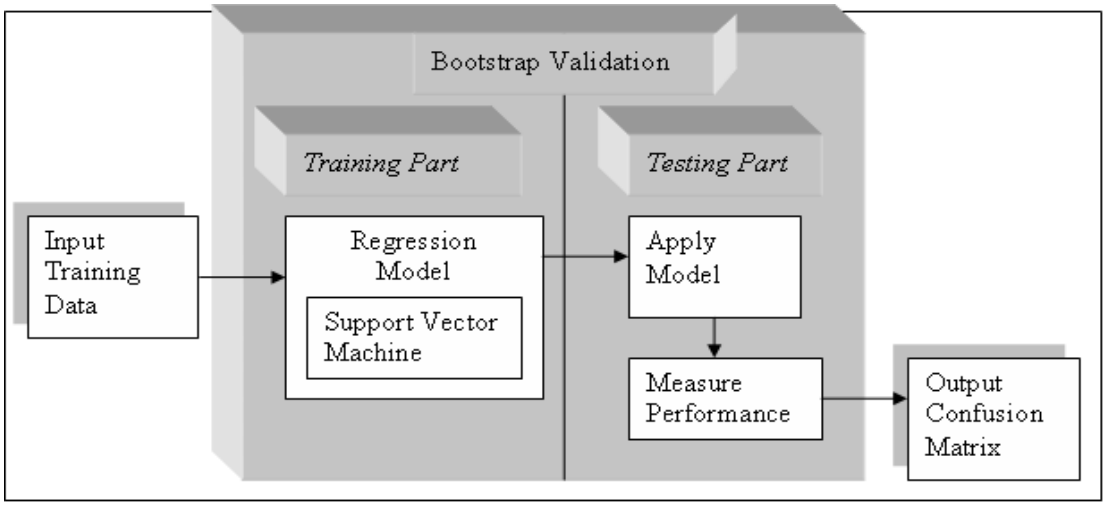

FIGURE 3.5 (a): Classification Model using Bootstrap Validation

The confusion matrix that is obtained after the application of the classifier is shown in Table 3.5 (a) and the 3D- graphical representation of the confusion matrix is shown in Fig 3.5 (b).

True / Pred.	True Authorization	True Others	True Secure-Information-Flow	True Audit and Logging	True Authentication	True Session-management	Class Precision
Pred. Authorization	177	2	0	0	4	0	96.72%
Pred. Others	0	139	0	0	2	0	98.58%
Pred. Secure-Information-Flow	0	0	131	0	0	0	100.00%
Pred. Audit and Logging	0	4	0	128	7	0	92.09%
Pred. Authentication	2	16	0	6	189	0	88.73%
Pred. Session-management	0	4	0	0	3	92	92.93%
Class Recall	98.88%	84.24%	100.00%	95.52%	92.20%	100.00%	

Accuracy: 94.52% +/- 1.85% (mikro: 94.48%)

TABLE 3.5 (a): Confusion Matrix

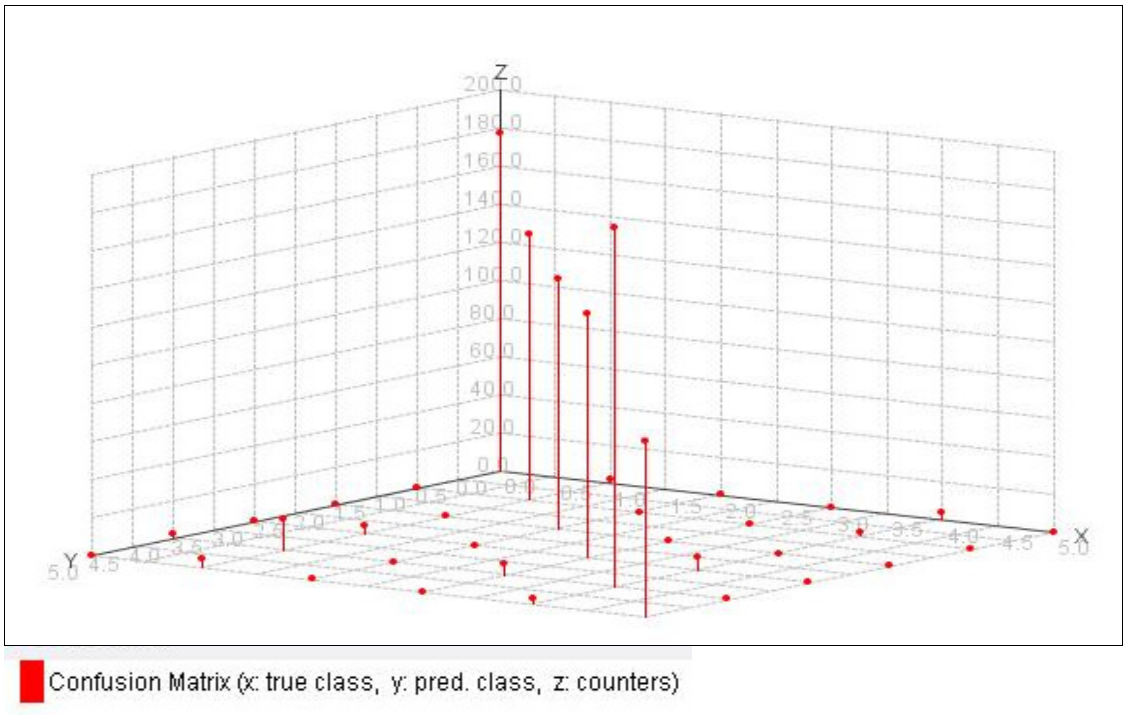

FIGURE. 3.5 (b): Confusion matrix in the form of 3D Graph

All most all the classes have class precision value above 90%. The accuracy rate of about 90% makes the classifier quite accurate (Han and Kamber, 2006). The overall accuracy rate of developed classifier is 94.5 %.

As shown in Table 3.5 (a), the class precision of 'authentication class' is only 88%, because the keywords used in the authentication class are common to other classes also. For example the vulnerability description mainly consist of words like '*unauthenticated user*', '*not allowed authenticated user*', etc, which actually don't indicate the cause as authentication, but classifier gets confused due to the frequent use of theses terms in other classes also, which affect the performance of classifier. But overall accuracy of the classifier is acceptable, which is 94.5%.

4.0 CLASSIFICATION RESULTS

Now using this design level 'Vulnerability Classification Model' the vulnerabilities can be classified into six classes. In NVD (National Vulnerability Database), total 427 vulnerabilities are identified as design level vulnerabilities till February 2009. Now in order to classify these vulnerabilities in our predefined six classes, vulnerabilities first need to be feed in the classifier, then predicted values can be used for further analysis. After feeding 427 design level vulnerabilities in the model, the example set of the predicted values that is obtained is shown in Table 4.0 (a). The screenshot from Rapidminer during the application of the classifier is shown in Fig.4.0 (a) and the final number of classified vulnerabilities is shown in Fig.4.0 (b). From the classification results it is clear that out of 427 vulnerabilities that are classified as design level vulnerabilities, 117 are actually not design level vulnerabilities. From remaining vulnerabilities, Authentication and authorization related vulnerabilities are most prevailing one, constituting about 53% of total vulnerabilities.

C* V.No.	Confidence (Authorization)	Confidence (Others)	Confidence (Secure Information Flow)	Confidence (Audit and logging)	Confidence (Authentication)	Confidence (Session Management)	Prediction
	-0.703822126777	-0.7411796480	-0.05035211037	1.0	-0.995248931575	0.0	Audit and Logging
2	-0.466536388761	1.0	-1.046548455455	-0.54866717509	-0.733920948135	0.0	Others
3	-0.803923996569	-0.5640368166	-0.61342764113	-0.39066602401	1.0	0.0	Authentication
4	-0.927380058636	-0.8005797124	-0.56610700139	0.01646968988	1.0	0.0	Authentication
5	-0.631178417772	-0.72385838954	-0.99080223010	-0.45735700877	1.0	0.0	Authentication
6	-0.674072540417	-1.0328340541	-1.03255019721	0.32822280439	1.0	0.0	Authentication
7	-1.022489071514	-0.9311227632	-0.13558666871	1.0	-0.563459046881	0.0	Audit and Logging
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...

TABLE 4.0 (a): Sample dataset from classification result

Percentage of Authentication is 30%, which makes it most important property to be mitigated at the design phase of the software. Authorization constitute 23.2% of all the vulnerabilities, which makes it second most important security attribute to be

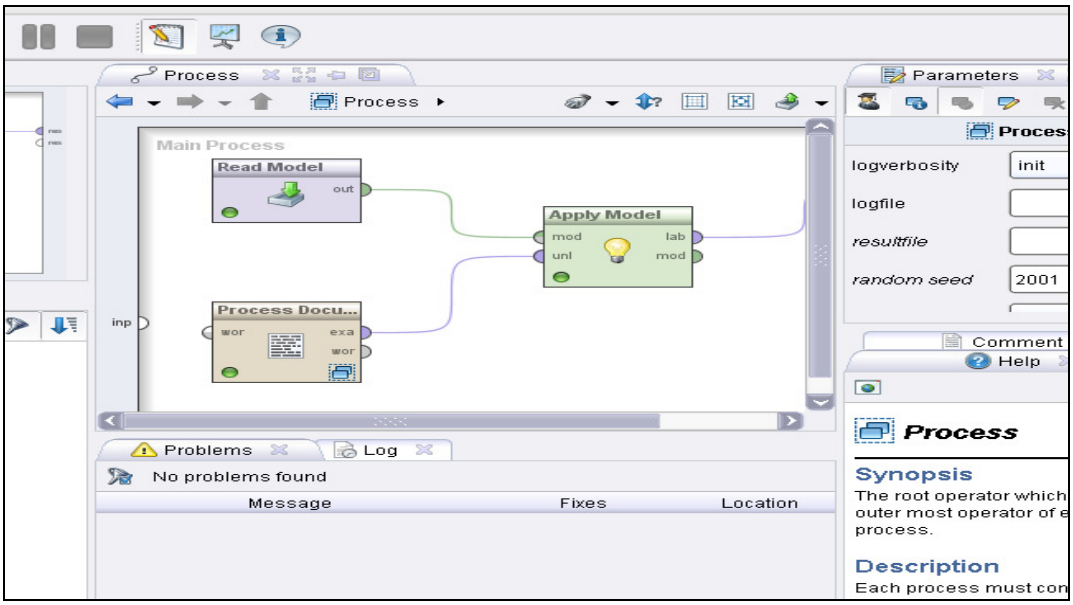

FIGURE 4.0 (a): Screenshot from rapid miner, while implementing the final model

The percentage of 'audit and logging', 'secure information flow' and 'session management' are 18%, 15% and 12% respectively, which makes them almost equally important.

Vulnerability Class	Count	Percentage	Percentage excluding Others
Authentication	96	22.48244	30.96774
Authorization	72	16.86183	23.22581
Audit and Logging	56	13.11475	18.06452
Secure-Information-Flow	47	11.00703	15.16129
Session-management	39	9.133489	12.58065
Others	117	27.40047	0.0
Total	427	100	100

TABLE 4.0 (b): Number of vulnerabilities classified under each class

These vulnerability classification data can be used with the severity rating to calculate the risk of vulnerability occurrence at the design phase.

5.0 CONCLUSION AND FUTURE WORK

As discussed in the previous sections, study of known vulnerabilities is very useful tool for the developer. Our approach is in the direction of identifying, classifying and learning from known vulnerabilities. So that these vulnerabilities can be avoided in the next generation of the software. In available vulnerability databases, there is no information about the vulnerability cause or the SDLC phase in which they can be removed. Using our proposed classification model, developer would be able to classify any vulnerability from any vulnerability database. The classification model will tell the developer whether the vulnerability be mitigated at the design level? If vulnerability is identified as a design level vulnerability then it will classify the identified vulnerability in security feature. After knowing the class of security feature, designer can adapt necessary design patterns that can prevent these vulnerabilities in the new under-developed software. The accuracy of the classified is found to be satisfactory and it can be used to classify future vulnerabilities. Classifying 'exposure leading to access violation' class of vulnerabilities is one of the prompt future works that can be done. The classification results can further be used to calculate the security risk at the design phase of the software. After risk calculation, mitigation mechanisms in the form of design patterns can be identified and thus designer will be able to mitigate security vulnerabilities at the design phase of the software. Another future work that can be done is the creation of tool that can automate the task of vulnerability classifications. After this classification our prompt objective will be to identify, analyze and classify design patterns that can be adapted in order to avoid vulnerabilities in the new software.

REFERENCES

[1] P.T. Devanbu and S. Stubblebine, "Software Engineering for Security: a Roadmap". International Conference on Software Engineering 2000 special volume on the Future of Software Engineering, 2000, pp.227-239.

[2] G. Hoglund and G. McGraw. "Exploiting Software: How to Break Code", New York: Addison-Wesley, 2004

[3] L. Lowis and R. Accorsi. "On a Classification Approach for SOA Vulnerabilities", 33rd Annual IEEE International Computer Software and Applications Conference. 2009, pp 439-444.

[4] V.C. Berghe, J. Riordan and Piessens "A Vulnerability Taxonomy Methodology applied to Web Services", 10th Nordic Workshop on Secure IT Systems, 2005.

[5] N. Moha. "Detection and Correction of Design Defects in Object-Oriented Designs". Doctoral Symposium, 21st International Conference on Object-Oriented Programming, Systems, Languages and Application, 2007.

[6] I.V. Krsul, "Software Vulnerability Analysis". Ph.D. Thesis. Purdue University. USA, 1998.

[7] S. Rehman, and K.Mustafa. "Software Design Level Security Vulnerabilities", International Journal of Software Engineering, 4 (2). 2011.

[8] T. Joachims. "Text categorization with support vector machines: learning with many relevant features". 10th European Conference on Machine Learning. 1998.

[9] J. A. Wang, and M. Guo. "OVM: An Ontology for Vulnerability Management". 7th Annual Cyber Security and Information Intelligence Research Workshop.Tennessee, USA. 2009.

[10] Z. Chen, Y. Zhang, and Z. Chen "A Categorization Framework for Common Computer Vulnerabilities and Exposures". Computer Journal Advance Access, 2009. Available: http://comjnl.oxfordjournals.org/ cgi/content/abstract/bxp040.

[11] P.H. Meland, and J. Jensen. "Secure Software Design in Practice". Third International Conference on Availability, Reliability and Security. 2008.

[12] Y. Li, H.S. Venter, and J.H.P Eloff. "Categorizing vulnerabilities using data clustering techniques", Information and Computer Security Architectures (ICSA) Research Group. 2009.

[13] N.H.Pham, T.T Nguyen, H.A Nguyen,., X.Wang, , A.T. Nguyen, and T.N Nguyen. "Detecting Recurring and Similar Software Vulnerabilities", International Conference of Software Engineering. Cape Town, South Africa. 2010.

[14] D. Byers, S. Ardi, , N. Shahmehri and C. Duma. "Modelling Software Vulnerabilities with Vulnerability Cause Graphs". 22nd IEEE International Conference on Software Maintenance. , 2006.

[15] V. Sridharan, and D.R. Kaeli . "Quantifying Software Vulnerability". Workshop on Radiation effects and fault tolerance in nanometer technologies, Ischia, Italy, 2008.

[16] Y.Wu, R.A. Gandhi, and H. Siy. "Using Semantic Templates to Study Vulnerabilities Recorded in Large Software Repositories". 6th International workshop on software Engineering for secure system, Cape Town, South Africa. 2010.

[17] G. Grefenstette and P. Tapanainen. "What is a Word, What is a Sentence? Problems of Tokenization". 3rd Conference on Computational Lexicography and Text Research . 1994, pp. 79-87.

[18] C. Fox. "Lexical Analysis and Stoplist-Data Structures and Algorithms". New York: Prentice-Hall. 1992.

[19] M. F. Porter. "Snowball: A string processing language for creating stemming algorithms in information retrieval", 2008. Available: http://snowball.tartarus.org.

[20] Lemur Project (2008). The Lemur Toolkit: For Language Modeling and Information Retrieval, 2008. Available: http://www.lemurproject.org.

[21] M. Braschler and B. Ripplinger, "How Effective is Stemming and Decompounding for German Text Retrieval". Information Retrieval, 7, 2003, pp.291–316.

[22] C.D. Manning, P. Raghavan, and H. Schütze. “Introduction to Information Retrieval”, Cambridge University Press. 2008.

[23] A. Rajaraman, and J.D. Ullman, Mining of Massive Datasets. 2010. Available: http://infolab.stanford.edu/~ullman/mmds/ch1.pdf

[24] A. Basu, C. Walters, M. Shepherd. “Support vector machines for text categorization”. 36th Annual Hawaii International Conference,2003

[25] T. Joachims. “A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization”, 14th International Conference on Machine Learning. 1997.

[26] J.R. Quinlan. “Programs for machine learning”. San Francisco: Morgan Kaufmann Publishers.1993.

[27] S. M. Weiss, C. Apte, F.J. Damerau, D.E. Johnson, F.J. Oles, T., Goetz, T. Hampp. “Maximizing text-mining performance”. IEEE Intelligent Systems Magazine, 1999.

[28] E. Wiener, J. O. Pederson, A.S. Weigend. “A neural network approach to topic spotting”, 4th Annual Symposium on Document Analysis and Information Retrieval. 1995.

[29] Y. Yang and , J.O. Pederson. “A comparative study on feature selection in text categorization”. International Conference on Machine Learning. 1997.

[30] Y. Yang. “An evaluation of statistical approaches to text categorization”. Journal of Information Retrieval. 1 (2). 1999.

[31] V. Vapnik,. “The Nature of Statistical Learning Theory”. Berlin: Springer. 1995.

[32] C. Burges. "A tutorial on support vector machines for pattern recognition”. Data Mining and Knowledge Discovery, 2, 1998, pp. 1-47.

[33] J.T.K. Kwok. “Automated Text Categorization Using Support Vector Machine”. International Conference on Neural Information Processing, 1998.

[34] V. Vapnik. “Statistical Learning Theory”. New York: John Wiley and Sons. 1998.

[35] T. Hastie, and R. Tibshirani, “Classification by pair wise coupling. Ann. Statist”, 26, 1998, pp. 451–471.

[36] CWE (Common Weakness Enumeration). Available: http://cwe.mitre.org/

[37] B. Efron. “ Estimating the error rate of a prediction rule: Improvement on cross-validation”. Journal of the American Statistical Association, 78, 1983. pp.316-331.

[38] J. Han, and M. Kamber “Data Mining: Concepts and Techniques”. San Francisco: Morgan Kaufmann Publisher, 2006.