SOFTWARE VULNERABILITY PREDICTION USING FEATURE SUBSET SELECTION AND SUPPORT VECTOR MACHINE

*1 Mrs. Kavitha S., *2Ms. Shanthi C.,

*1Assisant Professor, Department of Computer Science Auxilium College (Autonomous), Vellore, TamilNadu, India

*2 M.Phil Research Scholar, Department of Computer Science Auxilium College (Autonomous), Vellore, TamilNadu, India

---***---

Abstract:-*To improve the performance of software engineering processes and imperative to identify and eliminate rework that could have been avoided. While security or its absence is a property of running software many aspects of software requirements, design, implementation and testing contribute to the presence or absence of security in the finished product. Software is continues to function correctly under malicious attack. Verification and validation (V&V) techniques like security testing, code review and formal verification are becoming effective means to reduce the number of post release vulnerabilities in software products. The aim of reduce the dimensionality, removing irrelevant data, increasing learning accuracy and improving result comprehensibility. The feature subset selection algorithm and support vector machine as involves identifying a subset of the most useful features that produces compatible results as the original entire set of features. A feature subset selection algorithm may be evaluated from both the efficiency and effectiveness points of view. A feature subset selection algorithm is used for software vulnerabilities such as verification and validation. The support vector machines are supervised learning models with associated learning algorithms that analyze data and anomaly detection, predict the vulnerabilities in software. The used for classification and regression analysis to result.*

Key Words: *Quality of Software Product, Malicious Attack, Measurement Feature, Testability, anomaly Detection*

I. INTRODUCTION

Software engineering is about the creation of large pieces of software that consist of thousands of lines of code and involve many person months of human effort. One of the attractions of software engineering is that there is no one single best method for doing it, but instead a whole variety of different approaches. Consequently the software engineer needs knowledge of many different techniques and algorithm. This diversity is one of the delights of software engineering and this by presenting the range of current techniques and algorithm.cycle and quality promise. Design-for-testability is a very important issue in software engineering.

In traditional V&V the system provides the context under which the software will be evaluated and V&V activities occur during all phases of the system development lifecycle. The transition to a product line approach to development removes the individual system as the context for evaluation and introduces activities that are not directly related to a specific system. This dissertation describes the extension of V&V from an individual application system to a product line of systems that are developed within an architecture-based software engineering environment. This dissertation describes the extension of V&V from an individual application system to a product line of systems that are developed within an architecture-based software engineering environment. This seeks to ensure that the software is reliable. One of the all-time greats of software engineering. A piece of software that meets its specification is of limited use if it crashes frequently. Verification is concerned with the developers view the internal implementation of the system. Two types of verification are unit testing and system testing. In unit testing, each module of the software is tested in isolation. The inputs to unit testing are:

1. The unit specification
2. The unit code
3. A list of expected test results.

The products of unit testing are the test results. Unit testing verifies that the behavior of the coding conforms to its unit specification. In system testing or integration testing, the modules are linked together and the complete system tested. The inputs to system testing are the system specification and the code for the complete system. The outcome of system testing is the completed, tested software, verifying that the system meets its specification.

A single security problem can cause severe damage to an organization by not only incurring large costs late fixes but by losing invaluable assets and credibility and leading to legal issues. Annual world-wide losses caused from cyber attacks have been reported for. The organizations must prioritize vulnerability detection efforts and prevent vulnerabilities from being injected. One way of identifying the most vulnerable code locations is to use characteristics of the software product itself. Perhaps complex code is more likely to be vulnerable than simple code.

II. Related work

Many factors are believed to increase the vulnerability of software system. The more widely deployed or popular is a software system the more likely it is to be attacked. Early identification of defects has been a widely investigated topic in software engineering research. Early identification of software vulnerabilities can help mitigate these attacks to a large degree by focusing better security verification efforts in these components. Predicting vulnerabilities is complicated by the fact that vulnerabilities are most often, few in number and introduce significant bias by creating a sparse dataset in the population.

To improve the security of software, we must therefore not only look for general problem patterns but also learn specific patterns that apply only to the software at hand. In a investigation of the Mozilla vulnerability history. We surprisingly found that components that had a single vulnerability in the past were generally not likely to have further vulnerabilities. However components that had similar imports or function calls were likely to be vulnerable. Based on this observation we were able to extend Vulture by a simple predictor that correctly predicts about half of all vulnerable components, and about two thirds of all predictions are correct. This allows developers and project managers to focus their efforts where it is needed most "We should look at XPInstall Manager because it is likely to contain yet unknown vulnerabilities."

2.1 General Over view

The mining of textual for many important activities in software engineering tracing of requirements retrieval of components from a repository location of manage text for an area of question etc. Many such activities leave the final word to the analyst have the relevant items been retrieved. Other items that should have been retrieved and analysts become a part of the text mining process. The decisions on the relevance of retrieved elements impact the final outcome of the activity.

Text Mining

In the field of Software Engineering as two distinct and well-defined ways in text mining information retrieval and machine learning methods are applied. The first direction is the exploratory study of existing artifacts of software development. The document hierarchies, code repositories, bug report databases, etc., The purpose of learning new "interesting" information about the underlying patterns. Research of this sort is tolerant to the varying accuracy of text mining methods. The certain subtleties of some datasets might be missed the most general are most likely be discovered in analysis.

Text Mining and Vulnerabilities

A few approaches are related to work as the text mining techniques and treat all or parts of the source code as text. However most work focuses on defect prediction and not on vulnerability prediction the topic of work. Used text features and spam filtering algorithms to predict defects in software analyzed the source code of the changes as text in order to build a predictor to determine whether the introduced changes are buggy. The determined that the use of K nearest neighbors (kNN) technique results in a significant trade-off in terms of precision.

$$\text{Vulnerable} = \begin{cases} 1 \text{ if number of warnings} > 0, \\ 0 \text{ otherwise.} \end{cases}$$

2.2 Machine Learning

Machine learning deals with the issue of how to build programs that improve their performance at some task through experience. Machine learning algorithms have proven to be of great practical value in a variety of application domains. Not surprisingly the field of software engineering turns out to be a fertile ground where many software development and maintenance tasks could be formulated as learning problems and approached in terms of learning algorithms. The deals with the subject of applying machine learning methods to be engineering. The first provide the characteristics and applicability of some frequently utilized machine learning algorithms. Then summarize and analyze the existing work and discuss some general issues. Finally offer some guidelines on applying machine learning methods to software engineering tasks. Machine learning algorithms can out how to perform important tasks by generalizing from examples. This is of- ten feasible and cost-effective manual programming is not. As more data becomes available, more ambitious problems can be tackled.

Representation:

A classifier must be represented in some formal language that the computer can handle. Conversely choosing a representation for a learner is amount to choosing the set of classifiers that it can possibly learn. This set is called the hypothesis space of the learner. If a classifier is not in the hypothesis space, it cannot be learned.

Evaluation

An evaluation function (also called objective function or scoring function) is needed to distinguish good classifiers from bad ones. The evaluation function used internally by the algorithm may differ from the external one that we want the classifier to optimize for ease of optimization (see below) and due to the issues discussed. A method to search among the classifiers in the language for the highest-scoring one. The choice of optimization technique is key to the efficiency of the learner, and also helps determine the classifier produced if the evaluation function has more than one optimum. It is common for new learners to start out using off-the-shelf optimizers are later replaced by custom designed ones

III. PREVIOUS IMPLEMENTATIONS

The identified all faults in the software based on the failures that have surfaced during testing. Additionally the customer reported failures do not complete the identification of all non-security faults as predictors or all security faults and failures as dependent variables. Moreover the testing effort may not have been equal for all components and thus components with fewer failures may appear more reliable or secure. Therefore analyses are based on incomplete data. The Type I (48%) and Type II (43%) error rates are high indicating that the model is not precise if applied at Cisco could lead to effort wasted on low security risk components while some attack-prone components are never found. Additional metrics in a statistical model may help identify attack-prone components with lower Type I and Type II error rates. Furthermore there are few security data making statistical analyses difficult. The model presented one industrial software system and will not necessarily yield the same results on different software systems.

The starting point in study is the source code (including comments) of a software application that consists of a number of Java files. Each Java file is tokenized into a vector of terms (text processing terminology) and the frequency of each term in the file is counted. The frequencies are not normalized to the length of the file. This procedure has been attempted in early experimentation and caused a deterioration of performance. The routine used for tokenization uses a set of delimiters that includes white spaces, Java punctuation characters (such as comma and colon) and both mathematical and logical operators. The routine is implemented is available.

Listing 1. Source code in file HelloWorldApp.java

```
/* The HelloWorldApp class prints ”Hello World!” */
class HelloWorldApp
 {
        Public static void main (String [] args)
        {
        System.out.println (”Hello World!”);
        }
}
```

For instance Listing 1 would be tokenized and transformed into the feature vector of Listing 2, where each monogram is followed by a count. Listing 2. Feature vector for file HelloWorldApp.java args: 1, class: 2, Hello:

2, HelloWorldApp: 2, main: 1, out: 1, println: 1, prints: 1,public: 1, static: 1, String: 1, System: 1,The: 1, void: 1, World: 2 From a computational perspective, creating the feature vectors for one version of a large application took an average of 40 seconds.

IV. SYSTEM IMPLEMETNATION

Many feature subset selection (FSS) algorithms have been proposed but not all of them are appropriate for a given feature selection problem. At the same time is rarely a good way to choose appropriate FSS algorithms for the problem at hand. FSS algorithm automatic recommendation is very important and practically useful. A meta learning based FSS algorithm automatic recommendation method is presented. The proposed method first identifies the data sets that are most similar to the one at hand by the k-nearest neighbor classification algorithm and the distances among these data sets are calculated based on the commonly-used data set characteristics. It ranks all the candidate FSS algorithms according to their performance on these similar data sets and chooses the algorithms with best performance as the appropriate ones. The performance of the candidate FSS algorithms is evaluated by a multi-criteria metric that takes into account not only the classification accuracy over the selected features but also the runtime of feature selection and the number of selected features. The proposed recommendation method is extensively tested on 50 real world data sets with 22 well-known and frequently-used different FSS algorithms for five representative classifiers. The results show the effectiveness of our proposed FSS algorithm recommendation method.

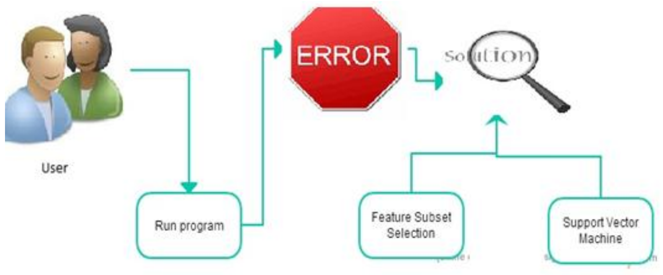

Fig1.1: System Architecture

4.1 Feature Subset Selection

Feature subset selection (FSS) plays an important role in the fields of data mining and machine learning. A good FSS algorithm can effectively remove irrelevant and redundant features and take into account feature interaction. This not only leads up to an insight understanding of the data but also improves the performance of a learner by enhancing the generalization capacity and the interpretability of the learning model Although a large number of FSS algorithms have been proposed. There is no single algorithm which performs uniformly well on all feature selection problems. Experiments have confirmed that there could exist significant differences of performance (e.g., classification accuracy) among different FSS algorithms over a given data set. That means for a given data set some FSS algorithms outperform others.

Assertion Density

The FSS algorithm recommendation method is based on the relationship between the performance of FSS algorithms and the meta-features of data sets. The recommendation can be viewed as a data mining problem. The performance of FSS algorithms and the meta-features are the target function and the input variables respectively. Due to the ubiquity of “Garbage In, Garbage Out” (Lee, Lu, Ling, & Ko) in the field of data mining the selection of the meta-features is crucial for our proposed FSS recommendation method. The meta-features are measures that are extracted from data sets and can be used to uniformly characterize different data sets the underlying properties are reflected. The meta-features should be not only conveniently and efficiently calculated but also related to the performance of machine learning algorithms.

Meta-Knowledge Database Construction

The meta-knowledge database is constructed by the following three steps. Firstly the meta-features are extracted from each historical data set by the module “Meta features extraction”. Then each candidate FSS algorithm is applied on each historical data set. The classification accuracy. The runtime of feature selection and the number of selected features are recorded and the corresponding value of the performance metric EARR is calculated. This is accomplished by the module “Performance metric calculation”. Finally for each data set. The tuple is composed of the meta-features and the values of the performance metric EARR for all the candidate FSS algorithms is obtained and added into the knowledge database.

4.2 FSS Algorithm Recommendation

Based on the introduction of the first part "Meta-knowledge Database Construction" presented the learning target of the meta-knowledge data is a set of EARR values instead of an appropriate FSS algorithm. It has been demonstrated that the researchers usually resort to the instance-based or k-NN (nearest neighbors) methods or their variations for algorithm recommendation. The k-NN based FSS algorithm recommendation procedure is proposed. The recommending FSS algorithms for a new data set firstly the meta-features of this data set are extracted. The distance between the new data set and each historical data set is calculated according to the meta-features. The k nearest data sets are identified and the EARR values of the candidate FSS algorithms on these k data sets are retrieved from the meta-knowledge database. Finally all the candidate FSS algorithms are ranked according to these EARR values the algorithm with the highest EARR achieves the top rank the one with the second highest EARR gets second rank and so forth and the top r algorithms are recommended.

4.3Support Vector Machine

Support vector machines (SVMs) method for binary classification. Traditional training algorithms for SVMs such as chunking and SMO scale super linearly with the number of infeasible for large training sets. Since it has been commonly observed that dataset sizes and development of training algorithms. The survey work on SVM training methods that target this large-scale learning regime. Most of these algorithms use either (1) variants of primal stochastic gradient descent (SGD) or (2) quadratic programming in the dual. For (1) The discuss why SGD generalizes well even though it is poor at optimization and describe algorithms such as Pegasus and FOLOS that extend basic SGD to quickly solve the SVM problem. For (2) the survey recent methods such as dual coordinate-descent and BMRM and proven competitive with the SGD based solvers. the training set size increase and explain SGD-based algorithms are able to satisfy.

4.4 Comparison of Machine Learning Technique and Feature Subset Selection Algorithm

The performed a large scale studies by mining more than 182 open source android applications to check the most vulnerability. The focus on the first release of each application show that it is possible to build a classifier of good quality that predicts whether a file is vulnerable using term frequencies. New projects can be checked against these properties to detect anomalies. The authors validated their approach based on a sample of 182 projects where 90 percent of the top-ranked anomalies uncovered actual defects. The approach is based on bag of words and achieves a mean accuracy of 89 percent mean, recall of 89-90 percent, mean fall-out of 30-35 percent.

Study	Predictors used	Vulnerability	Prediction Technique	Applications	Performance
Shin et al.	Software metrics, code churn, developer activity metrics	MFSA	Logistic regression	Firefox	Precision:15%, Recall:75%
Neuhaus et al.	Imports, function calls	MFSA	Machine Learning Technique	Mozilla	Precision:17%, Recall:45%
Zimmer-mann et al.	Code churn, code complexity, dependency measures, code coverage, organizational metrics, actual dependencies	NVD	Logistic regression, Machine Learning Technique	Windows Vista	Median precision:13-17%, Median Recall:20-40%
Gegick et al.	Non-security failure reports	Cisco security reports	Logistic regression	Cisco software system	Recall:57%, fall-out:48%
Smith et al.	SQL hotspots	Trac issue reports	Logistic regression, Machine Learning Technique	WordPress, Wikka Wiki	Wordpress – avg Precision:12-14%, avg Recall:24%; Wikka Wiki – average Precision:62%

Table 1.1: Machine Learning Technique

Study	Predictors used	Vulnerability	Prediction Technique	Applications	Performance
Hata et al.	Software metrics, code churn, developer activity metrics	MFSA/NVD	Feature Subset Selection Algorithm	Firefox, Android OS Platform,	Firefox – Precision:10%, Recall:89-90%, fall-out:30-35%
Catel et al.	Component dependency graph	NVD	Support Vector Machine	Firefox, Android OS Platform	Firfox-Precision:8%, Recall:83-90%, fall-out:35.5%
Shin et al.	Decision Tree	SCA	Feature Subset Selection Algorithm and Support Vector Machine	Android OS Platform	WordPress Precision:9%, Recall:84.5%, fall out:34%
Williams et al.	Function calls	kNN	Support Vector Machine	Firefox	Precision:8%, Recall:86.3%, fill-out:35%
Mizuno et al.	Software metrics, developer activity metrics	NVD	Feature Subset Selection Algorithm and Support Vector Machine	Firefox, Android OS Platform	Firefox – Precision:10%, Recall:89-92%, fall-out:32-37%

Table 1.2: Feature Subset Selection Algorithm and Support Vector Machine

EVALUATION RESULT:

As main contribution explores the value of a technique backed by text mining and machine learning and applies the technique to a relevant class of applications. The ensuring a potentially high impact in case of success. The approach presented here is applied to the problem of predicting software vulnerabilities. Analyzed 20 "apps" for the Android OS platform and followed their evolution over time. The total analyzed 182 releases spanning. The above -mentioned text mining technique in a series of three experiments of increasing complexity. In the first experiment the focus on the first release of each application. The show that it is possible to build a classifier of good quality that predicts whether a file is vulnerable using term frequencies.

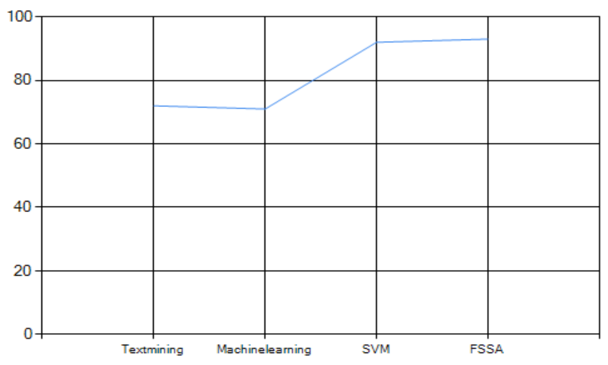

Fig.1.2: Comparison between value for machine learning & feature subset selection algorithm

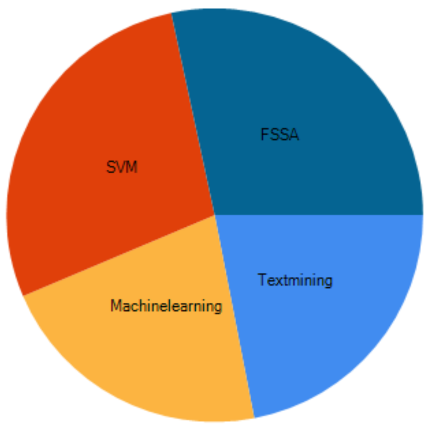

Fig.1.3 Value of Machine Learning and Feature Subset Selection Algorithm

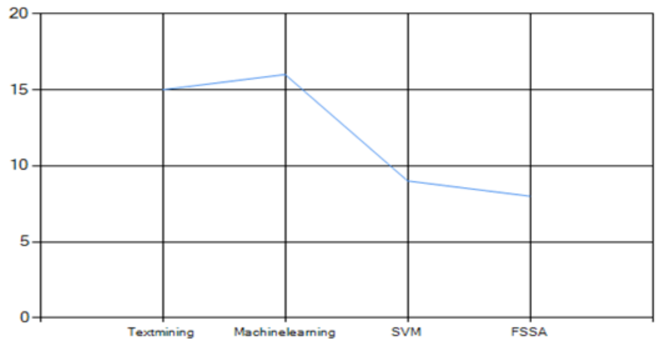

Fig.1.3:Comparison between Times for Machine Learning Technique and Feature Subset Selection Algorithm

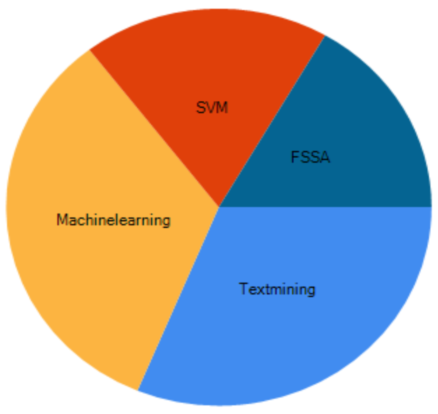

Fig.1.4: Time of Machine Learning and Feature Subset Selection Algorithm

CONCLUSION

The presented empirical evidence that features correlate with vulnerabilities. Based on this empirical evidence as have introduced vulnerable that predicts vulnerable components by looking at their features. It is fast and reasonably accurate. It analyzes a project as complex as Mozilla in about half an hour and correctly identifies half of the vulnerable components. Two thirds of its predictions are correct. The contributions of the present paper are as follows. A technique for mapping past vulnerabilities by mining and combining vulnerability databases with version archives. Empirical evidence that contradicts popular wisdom saying that vulnerable components will generally have more vulnerabilities in the future. Evidence that features correlate with vulnerabilities .A tool that learns from the locations of past vulnerabilities to predict future ones with reasonable accuracy. An approach for identifying vulnerabilities that automatically adapts to specific projects and products. A predictor for vulnerabilities that only needs a set of suitable features and thus can be applied before the component is fully implemented..

Future Work:

The empirically features are good predictors for vulnerabilities. The believe that this is so because features characterize a component's domain. The type of service that it uses or implements and it is really the domain that determines a component's vulnerability. The plan to test this hypothesis by studies across multiple systems in similar domains.

REFERENCES:

1. B. Smith and L. Williams, "Using SQL hotspots in a prioritization heuristic for detecting all types of web application vulnerabilities," in Proc. IEEE Int. Conf. Softw. Testing, Verification Validation, 2011.

2. A. Zeller, T. Zimmermann, and C. Bird, "Failure is a four-letterword: A parody in empirical research," in Proc. Int. Conf. Predictive Models Softw. Eng., 2011.

3. Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," IEEE Trans. Softw. Eng., Nov.–Dec. 2011.

4. S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, "Predicting vulnerable software components," in Proc. ACM Conf. Comput.Commun. Secur., 2007.

5. Ostrand, T.J., Weyuker, E.J., and Bell, R.M., "Predicting the Location and Number of Faults in Large Software Systems", IEEE Trans. Software Eng., 31(4), 2005.

6. Gegick, M., Rotella, P., and Williams, L., "Toward Non-Security Failures as a Predictor of Security Faults and Failures", in Proc. International Symposium on Engineering Secure Software and Systems (ESSoS), Leuven, Belgium, February 04-06, 2009.

7. Adrian Schr¨oter, Thomas Zimmermann, and Andreas Zeller, "Predicting component Failures at Design Time," In Proc. 5th Int'l Symposium on Empirical Software Engineering, New York, NY, USA, September 2006.

8. H. Hata, O. Mizuno, and T. Kikuno, "Fault-prone module detection using large-scale text features based on spam filtering,"Empirical Softw. Eng., 2010.

BIOGRAPHIES

Ms Shanthi. C., M.Phil Research Scholar, Department of Computer Science Auxilium College (Autonomous), Vellore, TamilNadu, India.

Mrs. Kavitha S., M.C.A., M.Phil., Assistant Professor & HOD I/C, Department of Computer Science Auxilium College (Autonomous), Vellore, TamilNadu, India.