Finding Accessibility and Interaction Vulnerability of Rational Rose Class Design Using Design Metrics

Soham H. Gandhi, D. R. Anekar, Mahevash A. Shaikh, Ajinkya A. Salunkhe

Abstract— The increasing importance of software measurement has led to the development of new software measures. Many metrics have been proposed related to various constructs like class, coupling, cohesion, inheritance, information hiding and polymorphism. To satisfy security requirements, it is essential to protect our data from unauthorized information disclosure and information alteration. In order to minimize vulnerabilities and achieve target level security, quantification of security is necessary. Unfortunately, quantitative estimation of security in earlier stage of the software development life cycle (SDLC) is largely missing. The design phase of software development provides the foundation for secure software. Reducing vulnerability at this phase minimizes rework in subsequent development phases. Taking security into account from the early stages of a system's development should have a significant impact on decreasing many software vulnerabilities. In order to address this problem, we have developed a methodology which is based on existing research work, which can be able to provide proper prediction of security vulnerabilities with respect to design properties for an object-oriented class design.

Index Terms— Class diagram, Vulnerability, Cohesion, Data encapsulation, Model file, Design metrics, Security measurements.

1 INTRODUCTION

Object-Oriented Analysis and Design (OOAD) of software provide many benefits such as reusability, decomposition of problem into easily understood object(s) and future modification. But the OOAD software development life cycle is not easier than the typical procedural approach. Therefore, it is necessary to provide dependable guidelines. Object-Oriented programming metrics is an aspect to be considered. Metrics are a set of standards against which one can measure the effectiveness of Object-Oriented Analysis techniques in the design of a system. "Vulnerability is an instance of fault in the specification, development or configuration of software such that its execution can violate an implicit or explicit policy" [15].

In order to minimize vulnerabilities and achieve target level security, quantification of security is necessary. Unfortunately, quantitative estimation of security in earlier stages of the software development life cycle (SDLC) is largely missing. The design phase of software development provides the foundation for secure software. Reducing vulnerability at this phase minimizes rework in subsequent development phases. Taking security into account from the early stages of a system's development should have a significant impact on decreasing many software vulnerabilities.

- *D. R. Anekar is currently assistance professor at Sinhgad Academy of Engineering, Pune, India. E-mail: devanekar@gmail.com*
- *Soham H. Gandhi is currently pursuing bachelor degree program in Information Technology in Sinhgad Academy of Engineering, University of Pune, India. E-mail: sohamgandhi91@gmail.com*
- *Mahevash A. Shaikh is currently pursuing bachelor degree program in Information Technology in Sinhgad Academy of Engineering, University of Pune, India. E-mail: shaikhmahevash@gmail.com*
- *Ajinkya A. Salunkhe is currently pursuing bachelor degree program in Information Technology in Sinhgad Academy of Engineering, University of Pune, India. E-mail: ajinkya.f4@gmail.com*

Software metrics are often used to access the ability of software to achieve predefined goal(s) [10]. Software metric is a measure of some property of a piece of software. Software metric is a term that contains many activities, all of which involve some degree of software measurement. Software measurement is the assessment and prediction of well-defined attributes of well-defined entities. Software attributes include, in addition to security, maintainability, performance, reusability and reliability [1]. Security measurements have been defined to assess security at the level of implementation code [3]. This paper proposes a new set of metrics which are capable of assessing the security quality of OO class designs. In our case we use <<secrecy>> stereotype to identify confidential data. Once the metric's results are identified for alternative class diagrams, it is easy to find the most secure one to implement.

1.1 Basic Concept

Process Metrics: Process metrics are known as management metrics and are used to measure the properties of the process which is used to obtain the software. Process metrics include cost metrics, efforts metrics, and advancement metrics and reuse metrics. Process metrics help in predicting the size of the final system and in determining whether a project is running according to schedule.

Products Metrics: Product metrics are also known as quality metrics and are used to measure the properties of the software. Product metrics include product non reliability metrics, functionality metrics, performance metrics, usability metrics, cost metrics, size metrics, complexity metrics and style metrics. Products metrics help in improving the quality of different system components, and in comparisons between existing systems.

1.2 Object Oriented Concept

Cohesion: It is a measure of how strongly related or focused the responsibilities of a single module are. If the methods that serve a given class tend to be similar in many aspects, then the class is said to have high cohesion.

Encapsulation: It is the mechanism that binds together the code and the data it manipulates, and keeps both safe from outside interference and misuse [8]. E.g. When a user selects a command from a menu in an application, the code used to perform the actions of that command is hidden from the user.

2 RELATED WORK

Software metrics can be used to find out the properties of the software that we are developing and predict the needed effort and development period. Many different kinds of metrics have been developed during the past few decades, matching with the different programming paradigms like structural programming and object-oriented programming (OOP). Among these, "LOC (Lines of Code)" is one of the most primitive and oldest metrics. In the beginning of 1990s, Chidamber and Kemerer proposed six new object-oriented metrics to overcome the limitations of the more traditional code-based metrics .They are "weighted methods per class (WMC)", "depth of inheritance tree (DIT)", "number of children (NOC)", "coupling between object classes (CBO)", "response for a class (RFC)",and finally, "lack of cohesion in methods (LCOM)".[9] Their metrics have certainly helped users analyze their code to some extent along with other similar OO metrics. However, as software engineers' focus has shifted to the earlier stages of the life cycle, the shortcomings of OO code metrics like their predecessors have become more apparent. Therefore, a comprehensive approach to developing and applying metrics to artifacts such as designs produced at the early stages of the life cycle is needed.

In the meantime, the Unified Modeling Language (UML) was adopted by the Object Management Group (OMG) in 1997 ending the so-called "OO methods war", and since then has become the de facto specification standard graphical language for specifying, constructing, visualizing, and documenting software systems, business modelling and other non-software systems. UML has been intensively used by software developers since its introduction. Many organizations are using UML as a common language for their project artefacts and have adopted UML as their organization's standard. As the amount of UML models produced within an organization increased, a need for measuring their characteristics has arisen. The overall aim is the developments of software metrics that can be applied to UML models. These metrics are comparable to UML itself in such a way that it plays a role as a standardized metrics suite

One of the earliest studies in this area was the development of software security design principles by Saltzer and Schroeder [12]. These principles were intended as guidance to help develop secure systems, mainly operating systems. Bishop's [13] and McGraw's [6] texts identified several similar security design principles. An existing approach which is used by programmers to assess the level of security of given program code is based on the identification of vulnerabilities [7] [4].

A study conducted by Chowdhury et al. [3] defined a number of security metrics that assess the security of a given program based on code inspections. Measuring the security of the system's architecture has been done by Manadhata et al. [2]. This study focused on the system's 'attack surface'. Similarly, a study that defined design metrics which measure certain software quality attributes was conducted by Bansiya [16]. He identified an approach to improve the Quality Model for Object-Oriented Design (QMOOD) [5].

3 PROPOSED WORK & SYSTEM ARCHITECTURE

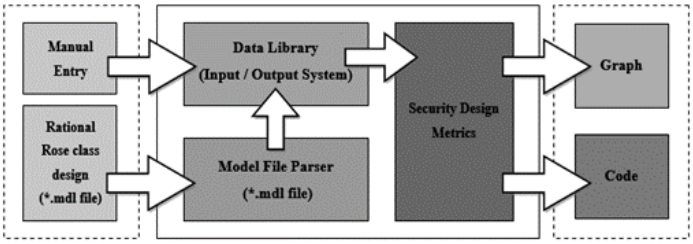

Figure 1: Proposed work and System Architecture

Proposed system take as input from the user (developer) a model file extension with (*.mdl), that is, a rational rose file which contains class diagram. Using the data library which is the input and output system we interact with the model file. In short, the data library loads or unloads the model file. After loading the model file, the Model File Parser tokenizes the words and finds out the class diagram's attributes and operations. After that, Design metrics will be applied on the loaded parsed model file, which will calculate the metrics and display them in a table. After this, a graph will be plotted which can be of any type, in our case we use Radar Graph. Finally, we can generate code for the class diagram that is most secure.

3.1 Manual Entry

This module will be provided where in user will be able to add classes to profile manually. User can enter class name, attribute name, attribute type, attribute stereotype, and attribute access specifier, Operation name, operation parameters and their return type, operation return type and access specifier. This module will also allow user to edit previously added classes. After entering the class, attribute and operation information, all information interacts with data library i.e. input and output system.
All information can be store in class profile. In future one can load or unload the class information.

3.2 Rational Rose Class Design (*.mdl)

Ration Rose Class design is nothing but a class diagram which is drawn in Rational Rose software. Every Rational Rose file

has a *.mdl extension; this file is known as Model file. The proposed system will takes the model file (*.mdl) as input. The model file interacts with the model file parser which extracts the class information. Then that class information interacts with data library that is the input and output system.

Sample structure for Model file (*.mdl) shown in Figure 2.

```
(object Class "EmployeeTwo"
        quid        "50F036AC01D4"
        operations      (list Operations
        (object Operation "GetDetails"
                quid        "50F0378702EE"
                result      "String"
                concurrency     "Sequential"
                opExportControl  "Public"
                uid         0))
        class_attributes    (list class_attribute_list
          (object ClassAttribute "SalaryAmount"
                quid        "50F036EE0148"
                stereotype      "secrecy"
                type        "Double"
                initv       "0")))
```

Figure 2: Sample Model File Internal Structure

3.3 Data Library / Input and Output System

The Data library or the input and output system helps to read files or content from storage media. The Data library module will allow us to save profile to files and later load or unload them. A profile shall contain the names of classes that are read from Rational Rose mode file (*.mdl). Using this data library, we can interact with model file (*.mdl). The Data library interacts with the Model file parser while reading a Rational Rose class diagram from model file.

3.4 MDL file Extractor

This module that will parse a UML diagram is designed using Rational Rose software. The Module file Parser will basically be a parser that will extract all required information from a model file and add all acquired classes' information to the current profile. The Model file parser parses the model file into tokens. Tokens are nothing but individual words and punctuation marks. The Model file parser identifies the class diagram from the model file to apply security metrics.
An overview of proposed Model file extractor or parser is shown in algorithm 1.

Algorithm 1: Class Information MDLExtractor
[Class, Attribute and Operation details] =
MdlEx ("file.mdl")

Input: Model File {*.mdl File}
Output: Class Name
Output: Attribute Name and details
Output: Operation Name and details

1. Initialize input with standard input buffer stream
2. Initialize str , str_adjust as a String
3. Initialize sb as StringBuffer
4. Initialize class_name, Attribute_name, Attribute_type, Attribute_stereotype, Operation_name, object_Parameter_type, object_Parameter, object_Reture_type with new vector
5. Read first line and save to str
6. **While** str not equal to null
   If str contains "(object Class " **then**
   Add next element to Class_name vector
   End if
   If str contains ""(object ClassAttribute" **then**
   Add next element to Attribute_name vector
   str = Read next line
   While str not contains ")"
   sb = sb.append(str.trim())
   str_adjust = sb.toString()
   End while
   Replace tab space and new line character with single space from str_adjust
   str = str_adjust
   If str_adjust contains " type " **then**
   Add next element to Attribute_type vector
   End if
   If str contains "stereotype" **then**
   Add next element to Attribute_stereotype vector
   else
   Add " " to Attribute_stereotype vector
   End if
   End if
   If str contains "(object Operation " **then**
   Add next element to Operation_name vector
   str = Read next line
   If str contains "list Parameter **then**
   While str not contains ")"
   sb = sb.append(str.trim())
   str_adjust = sb.toString()
   End while
   Replace tab space and new line character with single space from str_adjust
   str = str_adjust
   If str_adjust contains "(object Parameter" **then**
   Add next element to object_Parameter vector
   End if
   If str contains " type " **then**
   Add next element to object_Parameter_type vector
   End if
   End if
   If str contains " Result " **then**
   Add next element to object_Reture_type vector
   End if
   End if
   End while

In above algorithm, we have Model file as an input, output is class information like class name, attribute name, and it details as well as operation name and its details.
In this algorithm we have some initialization like buffer, some string variables and most important vector.
Algorithm reads the file line by line till the end of file. When it

finds the token like "(object Class " then next value of token store in separate vector. After this search for "(object ClassAttribute", next value store in proper vector. Likewise algorithm searches for model file keyword. After finding particular keyword next value of keyword get stored.
Using this algorithm we can extract the class information from rational rose model file.

3.5 Security Metrics

This module will calculate security metrics for the current profile and display it in tables. Bandar Alshammari [14] state some seven security matrices. These metrics measure potential information flow properties within a given class based on its design. The metrics have been scaled to all fit with the range 0 to 1. A low value is desired for each. These metrics at this stage are concerned with the properties of individual object-oriented classes.

3.5.1 Accesibility Metrics

Accessibility metrics are used for measuring the access level of attributes and operations or methods in a particular class from access modifiers perspective, like public denoted as '+', private denoted as '-', protected denoted as '#'. These accessibility metrics statically measures the potential flow of information. This category is divided into three kinds of accessibility metrics-CIDA, CCDA, and COA.

3.5.1.1 Classified Instance Data Accessibility (CIDA)

It helps to protect the classified internal representation. It is calculated by dividing the number of classified instance public attribute in a class by number of classified attribute. Higher values indicate higher accessibility to these classified attributes and hence a larger 'attack surface'.

3.5.1.2 Classified Class Data Accessibility (CCDA)

This metric measures the direct accessibility of the classified class attributes of a particular class. This metric aims to protect the classified internal representations of a class. It is calculated by dividing classified methods by the total number of potential interactions with all attributes in that class.

3.5.1.3 Classified Operation Accessibility (COA)

This metric is the ratio of the accessibility of public classified methods of a particular class. It is calculated by dividing the number of classified public methods in a class by number of classified method.

3.5.2 Interaction Metrics

Interaction metrics used to measure the impact of class interaction between method or operation and attribute on the security of that class. This category divided into four kinds of interactions, like classified mutators (setters/constructors), classified accessors (getters) or unclassified methods.

3.5.2.1 Classified Mutator Attribute Interaction (CMAI)

This metric measures the interactions of mutators with classified attributes in a class. It is calculated by dividing number of mutated classified attribute by multiplication of total number of mutators and number of mutated classified attribute.

3.5.2.2 Classified Accessor Attribute Interaction (CAAI)

This metric measures the interactions of accessors with classified attributes in a class. It is calculated by dividing number of accessors classified attribute by multiplication of total number of accessors and number of assessor's classified attribute.

3.5.2.3 Classified Attribute Interaction Weight (CAIW)

This metric is defined to measure the interactions with classified attributes by all methods of a given class. It is calculated by dividing classified methods by the total number of potential interactions with all attributes in that class.

3.5.2.4 Classified Methods Weight (CMW)

This metric is defined to measure the weight of the methods in a class which potentially interact with any classified attributes in a particular class. It is calculated by dividing the number of classified methods by the total number of methods.

3.6 Graph

This module will read values from the table of security metrics for current profile and display it in graphs (Radar or Web Graphs). We can plot any graph like bar graph, histogram, line, scatter, pie chart, but in our case we use Radar graph. We use Radar graph because lines closer to the center mean that a specific class diagram is more secure than the lines away from the center. It easily concludes the result, meaning which diagram is more secure among the alternatives. The Radar graphs have been scaled to fit the range 0 to 1. Lower value indicates more secure class design and higher value indicates less secure class design. We can plot one graph for comparison of one or more class designs, or different graphs.

This radar graph also called the spider graph.

A sample Radar graph that we are using in the proposed system is shown in Figure 3.

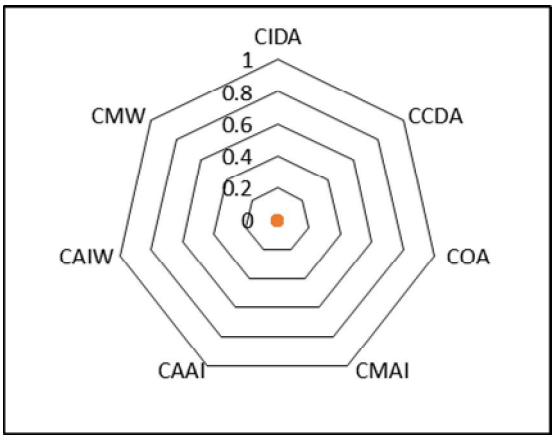

Figure 3: Sample Radar Graph

3.7 Code

We can generate any object oriented code but in our case we choose java to implement code. This module will help user(s) in converting the currently loaded Profile into .java files. We can generate code for the most secure class diagram.

4 CONCLUSION

Fixing security issues in software is an expensive process, more so when errors are discovered after the software product has been dispatched to the end users. In such cases, not only does the software company's reputation suffer, but the cost of the project also goes up. The system described in this paper can prevent all this and also improve efficiency-the software developer need not worry about which of his class diagrams is most secure, the system will assess that for him. These metrics allow designers to discover and fix security vulnerabilities at an early stage, by comparing the security of various class diagrams. Thus, the programmer can focus more on the other quality aspects of coding, while still ensuring security.

Future work will be extension of this paper, we will use some metrics that are reduce the complexity of design as well as code implementation. New system will combining the future of this paper and new metrics. We also are using some code based metrics that are used for calculating complexity of code.

ACKNOWLEDGMENT

We would like to thank Mrs Devata R. Anekar and Mr. Sunil L. Bangare for their valuable guidance and our Institution and other faculty members, without whom this Paper would have been a distant reality. Last but not the least, we would also like to thank to our friends for listening to our ideas, asking questions and providing feedback and suggestions for improving our ideas.

REFERENCES

[1] Sachitano, R. O. Chapman, and J. A. Hamilton, "Security in software architecture: a case study," in *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop*, 2004, pp. 370-376.

[2] P. K. Manadhata, K. M. C. Tan, R. A. Maxion, and J. M. Wing, "An Approach to Measuring A System's Attack Surface," Carnegie Mellon University, Pittsburgh, PA August 2007.

[3] I. Chowdhury, B. Chan, and M. Zulkernine, "Security metrics for source code structures," in *Proceedings of the Fourth International Workshop on Software Engineering for Secure Systems* Leipzig, Germany: ACM, 2008.

[4] M. Howard and D. LeBlanc, *Writing Secure Code*. Redmond, Wash.: Microsoft Press, 2002.

[5] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering,* vol. 28, pp. 417, 2002.

[6] J. Viega and G. McGraw, *Building Secure Software: How To Avoid Security Problems The Right Way*. Boston: Addison-Wesley, 2002.

[7] K. Maruyama, "Secure refactoring: improving the security level of existing code," in *Proceedings of the Second Internation Conference on Software and Data Technologies* Barcelona, Spain, 2007.

[8] E. Balagurusamy, "Object Oriented Programming with C++", 3rd edition, TATA McGraw Hill.

[9] Chidamber, S. R. and Kemerer, C. F. 1994. A Metrics Suite for Object Oriented Design. *IEEE Trans. Softw. Eng.* 20, 6 (Jun. 1994), 476-493.

[10] N. E. Fenton and S. L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing Co., Bostan, MA, USA, 1997.

[11] M. Howard, Attack Surface: Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users.

[12] J. H. Saltzer and M. D. Schroeder, "The protection of information in operating systems," in *Proceedings of the IEEE*, 1975, pp. 1278-1308.

[13] M. Bishop, *Computer Security: Art and Science*. Boston: Addison-Wesley, 2003.

[14] Alshammari, Bandar and Fidge, Colin J. and Corney, Diane (2009) "Security metrics for object-oriented class designs". In: QSIC 2009 Proceedings of: Ninth International Conference on Quality Software, August 24-25, 2009, Jeju, Korea. (In Press).

[15] Y. Shin and L. Williams "IS Complexity Really the Enemy of Software Security?" in the Proc. Of Th 4th ACM Workshop on Quality of Protection, Virginia, USA, Oct. 2008, pp. 47-50.

[16] J. Bansiya, "A Hierarchical Model for Quality Assessment of Object-Oriented Designs," Ph.D. Thesis, University of Alabama in Huntsville, 1997.