International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 08, August-2015

UMLSecCheck: A Tool to Detect Security Vulnerabilities in UML Diagrams

Pallavi T. R[1], Tejavathi T. N[2] and Dr. Anirban Basu[3]
Department of CSE,
East Point College of Engineering and Technology
Bangalore, Karnataka, India
*Project supported by the Government of Karnataka under VGST Scheme (2014-2015)

Abstract: **A vulnerability is defined as any defect in software, which can be used by an attacker to get access to the system. Such vulnerabilities can be the result of inappropriate coding or flaws in design. While vulnerabilities in code have been well documented, little research has been done to detect vulnerabilities in design specified in UML. The presence of vulnerabilities in the design models of the system makes it necessary to have tool that can help developers to avoid or detect them in the design stage.**
This paper discusses a tool called UMLSecCheck developed by the authors to identify vulnerabilities in UML Diagrams. The tool takes XMI data of the UML diagrams produced by using Argo-UML. Then the XMI data is compared against the rule set defined by the user based on the UML diagrams. The outcome of the tool indicates if there is violation of the defined rules for the design models.

Keywords: Vulnerability, UML Models, XMI Data File, Parser, Schema.

I. INTRODUCTION

Software security [1], [2] is an area which is gaining lot of attention. Security lapses can happen due to flaws in design and coding. While security vulnerabilities due to defects in code have been studied and well documented as CWE [3], [4] work on vulnerabilities due to design flaws has not been studied much. An important work in this area has been the book on software design patterns [5].

UML 2.0 diagrams are extensively used in specifying design of Object Oriented software. However, these cannot be manually checked in an efficient way for security flaws. This paper describes the development of a software analysis tool to test UML models for vulnerabilities to indicate inadequacy of security features.

While tools for drawing UML diagrams are available, testing tools for checking security aspects in UML diagrams are not easily available. To fulfill this need, we have developed a tool called UMLSecCheck. The tool has been used to analyze UML diagrams to detect vulnerabilities.

In section II we will discuss about work done in this area. Section III discusses the methodology followed and the architecture of the analysis tool developed. In section IV we discuss working of tool by considering class, state chart and sequence diagrams as input and experimental results are given.

II. LITERATURE SURVEY

There is a need to study the design patterns to know how to apply security patterns to the design models [5].

Test cases can be generated as a result of the model(s) verifier within the trace form of the models which are contradicting the assets (properties) (see [6], [7] for example). M. Dwyer, to make possible for the confirmation engineer to utilize chronological property, has recognized here [8], the group of designing patterns which permit for articulate as chronological properties a set of temporal requirements frequently met in industrial studies. Input/output Symbolic or Labeled Transition Systems have frequently been used to specify test purposes [9] [10].

A few proposals are depending upon the meaning of scenario in support of the test, e.g. in [11], where test case analysis results are coming from UML models as a tree sets. The scenario is taken out by a Breadth First Search upon the tree. Related approaches are developed in the device saying test stories [12], on the basis of a test model defined from basic analysis series prepared of an early state, test data and test story.

One tool called Carisma[13] exists to check UML diagrams based on the stereotypes used in those diagrams. The tool only tests the models if they contain stereotypes in it. So this tool will not identify the weaknesses in the design models if they do not posses any stereotypes.

The uniqueness of the plan by means of these correlated proposals can be reviewed in these points: scientifically, technologically and self-expression. These allow a validation engineer to profit from its fine awareness of the models and to openly utilize the entire objects of the models (objects names).

III. METHODOLOGY FOLLOWED

The methodology the tool is shown in Fig.1 and consists of three major functional modules namely, Parser, Rule validation and Rule Engine.

***UML XMI Data*:** This data is obtained by converting UML diagrams [14] into XMI file format (.xmi extension) using Argo-UML tool.

Parser: Parser will take the input XMI data and produces the schema and one example is shown below for smart card application state chart diagram.

The schema is nothing but the reflection of security rules which are defined based on the test intention. The following test schema intention is to set card status to TERMINATED and to try out all operations to check the model for correctness.

For_each $x from APDU_Set_status

Use any_operation any_number_of _times to_reach

State_respecting(self.state=TERMINATED)

On_instance"card"then

Use $x at_least_once to_reach state_respecting

(self.state=TERMINATED) on_instance"card"

Rule Engine: Security rule sets are designed based on the application, which is given as an input for rule engine.

Rule Validation: At this stage the schema generated from the parser and rules from rule engine are considered as an input to the rule validation. The outcome is rule violation / no violation obtained by comparing schema and rule sets.

This tool detects design flaws based on any of one: the class diagram, state chart diagram or sequence diagrams. Future enhancements are planned for incorporating other diagrams as input.

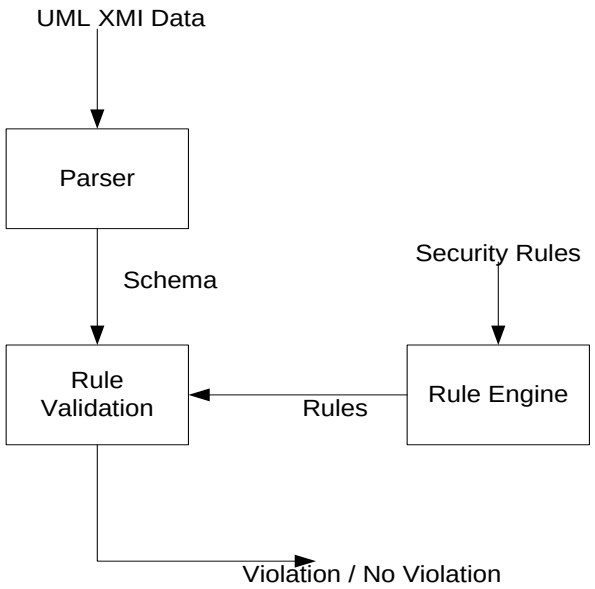

Fig.1. Architecture of UMLSecCheck Tool

The rules with respect to class, state chart and sequence diagrams are discussed below.

Class Analyzer: In this module the class analyzer identifies the class diagram which contains class name and attributes. This module also identifies the number of attributes and contains attribute values like visibility (public or private) which is declared in the class diagram.

Rule1 is designed for class diagrams which identify the class name and attribute name with the visibility and permission type. Rule describes the violation if the attribute visibility is declared as public in the designed. If the attribute visibility is private, then there is no violation.

State Chart Analyzer: In this module it identifies state chart diagram which contains pseudo state, final state and simple state.

Rule2 is designed for state chart diagrams which identifies the set of states and transitions in the input state diagrams and provides security rules to rule engine to get output.

Sequence Analyzer: In this module it identifies the classifier role of the sequence diagram, later which will be exported to XMI data.

Rule3 is designed for sequence diagrams which identifies the classifier role name and function name or interaction message. Rule describes the violation if the sender and receiver has the same interaction message in the design, else there is no violation of rule set.

The front end of the developed tool is shown in Figure.2. It has two screens namely, INPUT and LOG. Input screen allows choosing input UML-XMI Data File and Rule file. When you click on START ANALYSIS button, you can see output on LOG screen.

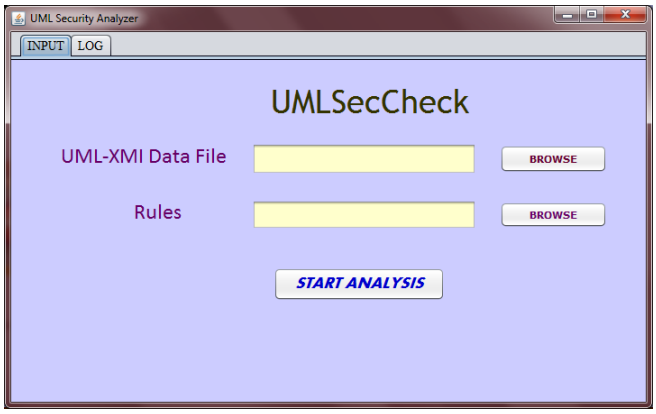

Fig.2. Front-End of the tool

IV. EXPERIMENTAL RESULTS

The tool was used to detect vulnerabilities in all the three diagrams: Class diagram, State chart diagram and Sequence

diagram. The experimental results are discussed as three cases.

CASE 1: A class diagram in the Unified Modeling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. If the attributes of the class declared as public then class rule will be violated. Hence the attributes of the class must be declared as private to achieve security in the system being developed. An example class diagram is shown in Fig.3.and the corresponding test case is given in Table 1.

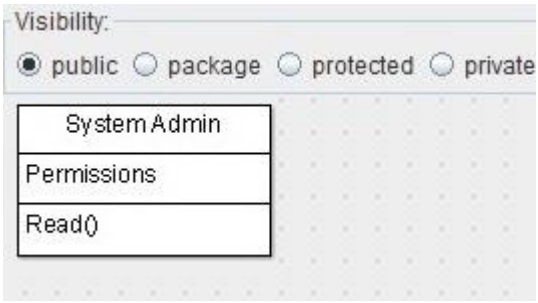

Fig.3. Example Class Diagram

CASE 2: The state chart diagram (state machine diagram) represents the events occurring in the system during its operation by means of states. Any state diagram must consist of an initial state and final state and in the following example state diagram (Fig.4) if there is a transition from "Suspended" state to "Normal" state then state rule will be violated, due to insecurity in this transition, else no violation occurs. The corresponding test case is given in Table 2.

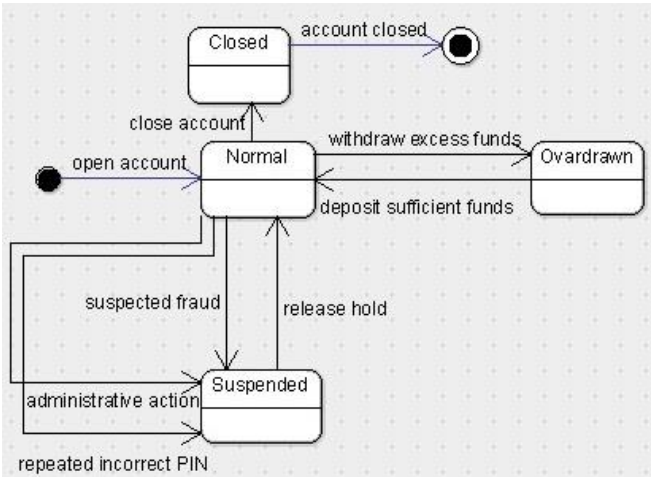

Fig.4. Example State Chart Diagram

CASE 3: The sequence diagram represents the events occurring in the system during its operation by means of interactions (messages). An example sequence diagram is shown in Fig.5. which contains the following, the classifier role names are buyerbank, ledger, Function names are retrieveaccount, getbalance. The sender is Buyerbank, Receiver is ledger and the interaction message between buyerbank and ledger is retreiveaccount. The Rule3 is violated as the function name retrieveaccount matches with the rule sets declared. The corresponding test case is given in Table 3.

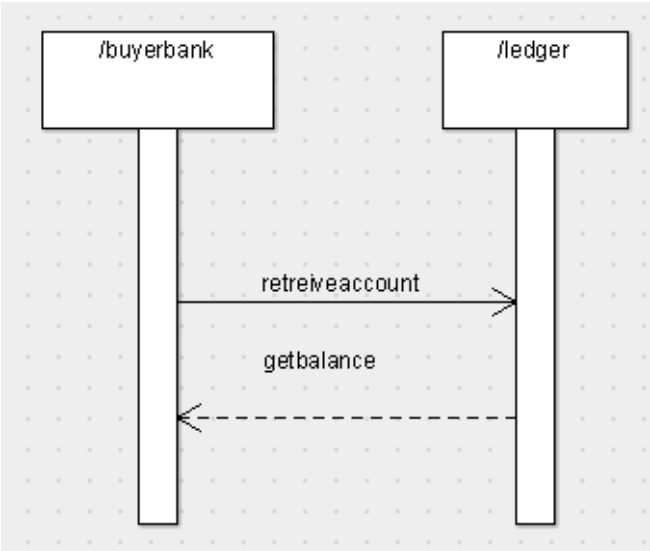

Fig.5. Sequence Diagram

TEST CASES: The experimental results are given for different input UML diagrams as shown in below tables.

Test Case ID	1
Description	Class diagram to be tested(Fig.3)
Input	XMI file obtained for the input class diagram(Fig.3)
Expected Output	Rule1 is Violated because the visibility set as public for the attribute Permissions of the class diagram(fig.3)
Remarks	Successful

Table.1. Test Case for Class Diagram

Test Case ID	2
Description	State chart diagram to be tested (Fig.4)
Input	XMI file obtained for input state chart diagram(Fig.4)
Expected Output	Rule2 is Violated for state transition:Suspended to Normal
Remarks	Successful

Table.2. Test Case for State Chart Diagram

Test Case ID	3
Description	Sequence diagram to be tested (Fig.5)
Input	XMI file obtained for input sequence diagram(Fig.5)
Expected Output	Rule3 is Violated as the function name retrieveaccount matches with the rule.
Remarks	Successful

Table.3. Test Case for Sequence Diagram

V. CONCLUSION

The tool discussed in this paper analyzes input UML diagrams and check for vulnerabilities and gives assurance of security in designed model. The testing of developed tool for input UML diagrams (class, state and sequence) by means of rule sets designed based on the input diagrams is successful. The tool is useful for detecting design errors that may lead to security vulnerabilities. Usually designers make mistakes in the models which could generate software vulnerabilities. The tool is developed by using java. The GUI designed is simple and easy to use by any users. The tool is being enhanced to incorporate other UML diagrams like Use Cases, Activity diagrams etc., and defining different security rules for these.

VI. REFERENCES

[1] G. Mcgraw, Software Security: Building Security In, Addison Wesley, 2006.
[2] A. K.Talukder, M. Chaitanya. Architecting Secure Software Systems, Auerbach Publications, 2009.
[3] http://www.cwe.mitre.org
[4] Priyadarshini R, Ghosh N and Basu A, "SecChech:a tool for detection of vulnerabilities and for measuring insecurity in java programs", International Journal of Software Engineering, Vol. 7, No. 2, pp.67-93, July 2014.
[5] Markus Schumacher, Eduardo Fernandez- Buglioni, Duane Hybertson, Frank Buschmann and Peter Sommerlad, "Security Patterns - Integrating Security and Systems Engineering", John Wiley & Sons Ltd. The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England.
[6] P.E. Amman P. E. Black, and W. Majurski, "Using model checking to generate tests from specifications," Formal Engineering Methods, International Conference on, p46, 1998.
[7] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," in ICSE'99, 21st international conference on Software engineering, LA, California, United States, 1999, pp. 411–420.
[8] C. Jard and T. Jér on, "Tgv: theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems," Int. J Soft. Tools Technol. Transf., vol. 7, no. 4, pp. 297–315, 2005.
[9] Frantzen, J. Tretmans, and T. Willemse, "Test generation based on symbolic specifications," in FATES 2004, Formal Approaches to Software Testing, ser. LNCS,J. Grabowski and B. Nielsen, Eds., vol. 3395. Springer, 2005, pp. 1–15.
[10] D. Clarke, T. Jeron, V. Rusu, and E. Zinovieva, "STG: A symbolic test generation tool," in TACAS'02, Tools and Algorithms for the Construction and Analysis of Systems, ser. LNCS, vol. 2280. Springer, 2002, pp. 151–173.
[11] F. Basanieri, A. Bertolino, and E. Marchetti, "The Cow_Suite approach to planning And deriving test suites in UML projects," in UML'02, 5-th int. conf. on the UML language, ser. LNCS, vol. 2460, London, UK, 2002, pp. 383–397.
[12] Y. Ledru, F. Dadeau, L. Du Bousquet, S. Ville, and E. Rose, "Mastering combinatorial explosion with the TOBIAS-2 test generator," in ASE'07: Procs of the 22nd IEEE/ACM int. conf. on Automated Software Engineering, 2007, pp. 535–536.
[13] http://www-secse.cs.tu-dortmund.de/carisma/
[14] http://www.wikipedia.org/wiki/Unified_Modelling _Language.html.