ISSN 2349-0292
Impact Factor 2.365
GLOBAL JOURNAL OF ADVANCED ENGINEERING TECHNOLOGIES AND SCIENCES

AUTOMATED VERIFICATION OF ACTIVITY DIAGRAM USING UMLSEC ON XMI DATA

Anand Dongre*, Dr. Prasad Babu B. R, Dr. Chandramoulli H B
* Department of CSE, East Point College of Engineering & Technology Bangalore, Karnataka, India
Project funded by the Government of Karnataka under VGST Scheme (2015-2016)

ABSTRACT
Vulnerability is outlined as any defect within the software package, which results attackers to induce access to confidential data. While vulnerabilities in code are well documented very little analysis has been done to find vulnerabilities in design per UML. The presence of vulnerability within the design model of the system makes it necessary to possess a tool that may facilitate developers to avoid or notice them within the design stage

Here we describe mechanism to find the defect in system. The tool takes UML Diagram as input. And this metadata (XMI DATA) is cross checked against rule. If the requirement fails to hold then error message is returned.

KEYWORDS: Architecture of UMLSec tool, Validation window for selecting the XMI file.

INTRODUCTION
Software program security is an area that is getting lot of interest .Protection lapses can occur because of the flaws in layout and coding .At the same time as protection vulnerabilities because of illness in code were studied and nicely documented as CWE, paintings on vulnerabilities due to layout flaws have now not been studied a good deal .An essential work in this area has been the e-book on software layout patterns

UML diagrams are drastically utilized in specifying design of object oriented software program. But, those cannot be manually checked in an efficient way for the safety flaws. This paper describes the improvement of the software program analysis device to test UML models for the vulnerabilities to suggest insufficiency of security features

While tools for painting UML diagrams are available in the market, testing tool, for validating security aspects in UML diagram are not available. To overcome this problem, we have developed a tool called UMLSec. The main function of the tool is to analyse UML diagram to detect vulnerabilities

In next section (II) considers the methodology followed and the system architecture of the tool. In section III we consider functioning of the tool by considering Activity diagrams as input and speculative results are given.

METHODOLOGY
The methodology of the tool is shown within the fig.1 and carries with it 3 major practical modules particularly parser, rule engine and rule validation

UML XMI Data: this data is obtained by changing UML diagrams [14] into XMI file format (.xmi extension) utilizing Argo-UML tool.

Parser: Here in this part XMI data is the input and using this input parser produces the schema

The schema is however a mirrored image of security rules that are outlined on the basis of test intention.

Rule engine: rule engine includes security rule sets which are outlined based on the application, which acts as an input to rule engine

Rule validation: As the name suggest this part validates for violation in UML diagram by comparing the schema and rule sets, the output is rule violation or no violation

ISSN 2349-0292
Impact Factor 2.365

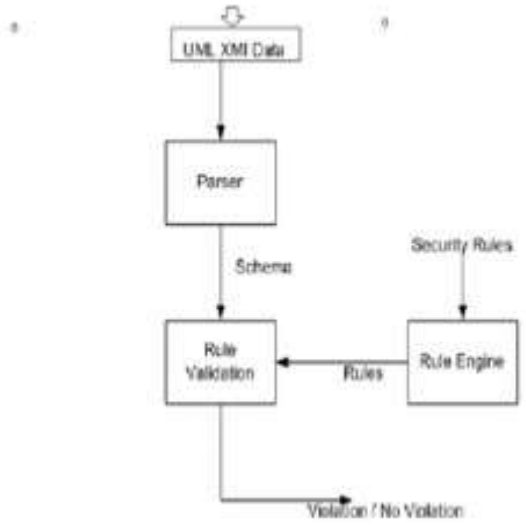

Fig.1. Architecture of UMLSec tool

The purpose of the toll is to detect design issues in any of the following: the class diagram, state chat diagram or sequence diagram, activity diagram, deployment diagram. Future improvements are outlined to include other diagrams as input

The rule for activity diagrams are explained below
Activity Analyser: Activity nodes which can be action, object and control are analysed here and also total number of final states, total no of users which is mentioned in the activity diagram

Rule 1: Rule 1 is plotted for activity diagram which pinpoints the total no of final states and total no of users, if the total no of final states doesn't match the rule. Rule illustrates the violation and there is no violation if the no of final state and no of users match.
The front end of the tool is shown in the below figure.2. The tool is having 3 windows namely:

- CONFIGURATION,
- VALIDATION AND
- LOG.

The configuration window permits for selecting input rule file and the validation window permits for selecting input XMI DATA file. Then user needs initiate (automated analysis)the analysis of activity diagram to get the output, which will be displayed on the LOG screen.

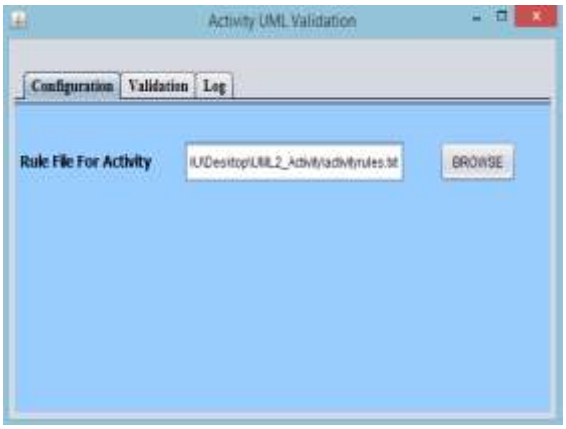

Fig.2. Configuration window for selecting the rule

ISSN 2349-0292
Impact Factor 2.365

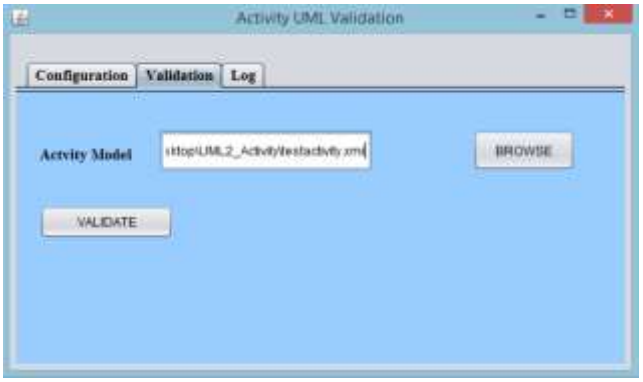

Fig.3. Validation window for selecting the XMI file

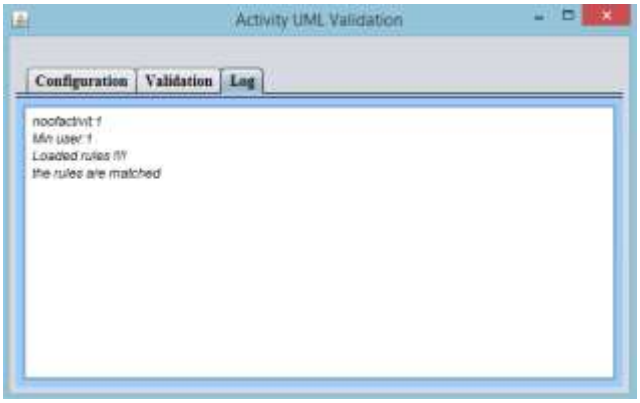

Fig.4. LOG window

EXPERIMENTAL RESULTS

The experimental result of the activity diagram is as follows.
Case1: An Activity diagram is dynamic structure diagram in Unified Modelling Language that illustrates the data flow of the system.

To acquire security in the system being developed one should define attributes of activity diagram and these attributes value should match value present in rule file. An example of activity diagram and corresponding test case are shown in fig.3 and table 1 respectively.

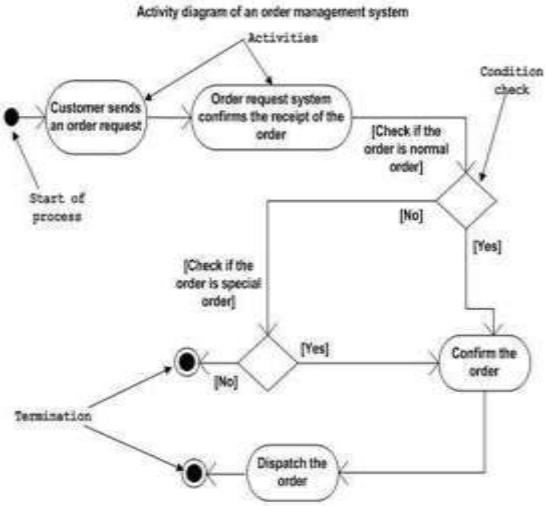

Fig.3.Example for Activity Diagram

ISSN 2349-0292
Impact Factor 2.365

TEST CASES: Test Case 1

Test case ID	1
Description	Activity Diagram to be tested
Input	XMI File obtained for the input activity diagram
Expected output	Rule is violated because there is mismatch between total number of final states and rule set
Remarks	Successful

Table 1: Activity diagram Test Cases

CONCLUSION AND FUTURE WORK

The tool depicted in this paper assesses the UML diagram and research for dangers in the models and ensures the security in model. The testing of UML diagram (Activity diagram) via specific rule set plotted based on the UML diagram is successful. The tool is viable in distinguish the defect in diagram which can bring about the security dangers. Generally engineers make deficiency in diagram that can bring the software defect.

This apparatus can be moved up to cover more muddled graphs like Component, Deployment and so forth by characterizing the diverse security standard for specific outlines. The java dialect is utilized to advance this instrument

ACKNOWLEDGEMENT

This research was supported by EAST POINT COLLEGE OF ENGINNEERING. I am thankful to our guides Dr. B R Prasad Babu and Dr. Chandramoulli H belagal who provided expertise that greatly assisted the research.

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 nondeterministic 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/