International Journal of Advanced Information Science and Technology (IJAIST) *ISSN: 2319:2682*
Vol.5, No.5, May 2016 **DOI:10.15693/ijaist.2016.v5.i5.44-48**

Cluster Based Java Input Injection Detection (CJIID) for Efficient Early Vulnerability Detection

Tushar T. Patil
Information Technology
MITCOE, Kothrud
Pune,

Prof. Milind R. Penurkar
Associate Professor, IT
MITCOE, Kothrud
Pune,

Prof. Farhana Shaikh
Assistant Professor,
MITCOE, Kothrud,
Pune, India

***Abstract*—The frequent and basic approach for detecting the web application security threats by using web application scanner tools. These tools are based on methodology of static or dynamic. Still to this date both this approaches are used independently for web application vulnerability scanning and having its limitations and advantages. For protecting the web applications from different attacks, vulnerabilities should remove early. The innovative method called Cluster Based Java Input Injection Detection (CJIID) that can help to detect the vulnerabilities before its deployment has been proposed.This approach can reduce not only the cost of fixing the vulnerabilities but also saves time. The paper presentsthe new framework for detection of different types of web application vulnerabilities efficiently. The proposed method uses both static and dynamic analysis techniques in order to detect vulnerabilities present in web applications. CJIID approach helps in reduction of false positives and false negatives, cost, as well as the time required for fixing the vulnerabilities. The focus is on input injection vulnerabilities as they are most common vulnerabilities found in today's web applications. Injection vulnerabilities are those vulnerabilities that are exploited by entering a malicious input value. The efficiency of this method is further improved by adding the clustering approach of web crawled input URLs. This is a automated technique.**

Keywords—Input Injection, Web vulnerabilities, dynamic analysis.

I. INTRODUCTION

Web applications are highly exposed to attacks. Probability of a vulnerability being exposed, if it exists, is very high. Research by WhiteHat Security [7] confirms that over 80% of Web applications have vulnerabilities and the average site has 56 serious vulnerabilities. The approximate time required to fix a serious vulnerability is 193 days [7]. Jeremiah Grossman [9], CTO of WhiteHat Security, calculated that it takes $4000 (40 man hours * $100 per hour) to fix one vulnerability in a website. Cost to fix vulnerabilities found after deployment of an application is much more expensive than required for fixing it earlier in development cycle. To find vulnerabilities in early stage of development cycle, people use either manual code inspection technique or automatic code analysis technique. Manual code inspections are time consuming and expensive whereas, automatic code analysis is faster and less expensive. There are two common analysis techniques: static analysis and dynamic analysis. Static analysis can achieve high code coverage, but has low confidence on reported vulnerabilitieswhereas dynamic analysis has low code coverage, but high confidence. Dynamic code analysis is typically carried out at the end of development cycle when entire application is hosted in an application server. The risk of performing this analysis so late in the development cycle is reduced time to adjust any design and implementation. The most common web application security weakness is the failure to properly validate input coming from the client or from the environment before using it [10]. This weakness leads to almost all of the major vulnerabilities in web applications, such as cross site scripting, SQL injection, interpreter injection, file system attacks, and buffer overflows. Vulnerabilities exploited by manipulating user input are called as injection vulnerabilities. Open Web Application Security Project (OWASP) [5] top ten list contains most common web application vulnerabilities and flaws found in today's web applications. Injection flaws are at the top position in the recently released list. To exploit injection vulnerabilities, attackers manage to manipulate the input. Thus, it's safe to conclude that improper handling of the user input is the main cause of input injections vulnerabilities [10]. Vulnerabilities in Java programming language continue to be the most frequently exploited target [8].

Now aday's web applications are highly vulnerable to different types of attacks. The probability of such attacks on web applications is very high. From recent study it is clear that 80 % of web applications are vulnerable to different attacks and most of web applications are having the 56 critical vulnerabilities. For fixing such serious vulnerabilities takes total 193 days. There are different solutions that have beenprovided for the detection of such attacks on web applications. Vulnerabilities exploited by manipulating user input are called as injection vulnerabilities such as SQL injection, cross site scripting, XML inject, etc. To detect such vulnerabilities in web applications different vulnerability scanning tools are designed by various authors. These methods are either static or dynamic with their pros and cons. Recently the hybrid approach for vulnerability detection before deployment of java web applications by using both static and dynamic analysis methods together in order to improve the accuracy performance.

II. Literature Survey and Motivation

Injection vulnerabilities are most common web security vulnerabilities. To address the problem of injection vulnerabilities, several solutions have been proposed which make use of static analysis [2], [13], [14], dynamic analysis [12], [17], or both techniques [1], [15], [16]. Static analysis is done by scanning the source or byte code of the application. Dynamic analysis is performed by actually running the web application with vulnerable inputs. The authors of [14], makes use of both analysis techniques to find buffer overflow vulnerability in a code written in C language. The performance of their tool depends on the correctness of the algorithm that is used for initial scanning of the code for marking functions. Yannis Smaragdakis [11] has also made many contributions about static directed testing. Some [1], [12] make use of static analysis to generate a test case for dynamic analysis which helps increase the efficiency and reduce the run time required for dynamic analysis. The work of extending the Findbugs to detect input injection is done in [2], but they only focus on static code analysis.

The approach that is suggestedin CJIID make use of both analysis techniques, before application is deployed, to achieve high confidence on reported vulnerabilities in java web applications. Both analyses are performed before actually deploying the web application. The advantage of this approach is that the vulnerabilities present in the application can be known before deployment which can save a lot of cost required to fix it afterwards.

A. Input Injection Detector Implementation

The functions needed to build the detector. The functions are dividedinto six groups, i.e.

1. Injection dataflow analyzer, is a dataflow analyzer to monitor whether a variable is contaminated or not and has been cleaned or not.

2. Cleaner database manager is a database manager for collecting information about cleaner methods and fields and reading a knowledge database file.

3. Return contaminated database manager, is a database manager for collecting information about return contaminated methods and fields and reading a knowledge database file.

4. Sink database manager, is a database manager for collecting information about sink methods and reading a knowledge database file.

5. Annotation detector is detector for collecting annotation in scanned code.

6. Injection detector uses the dataflow analyzer for finding out the input vulnerabilities in code. This detector is also used for mark method that is being scanned as sink or return contaminated method.

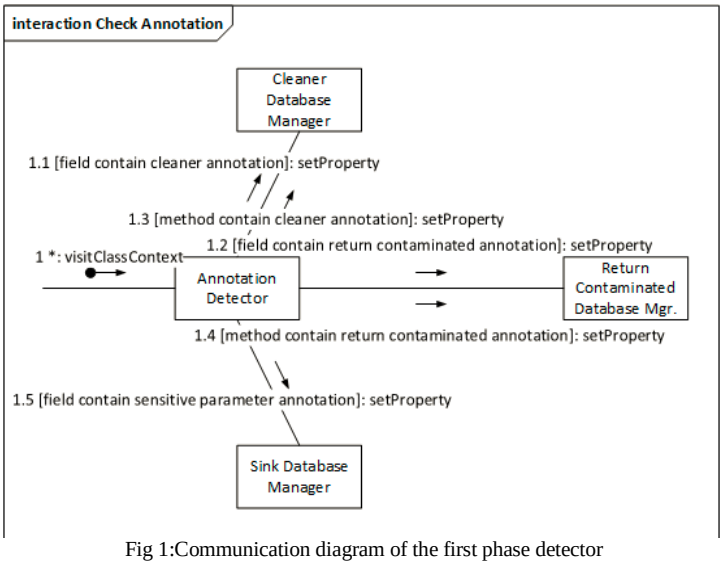

Fig 1:Communication diagram of the first phase detector

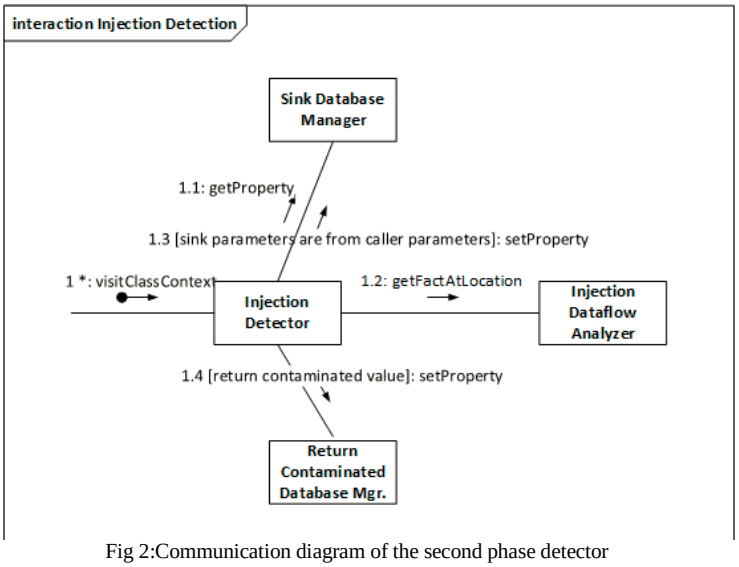

Fig 2:Communication diagram of the second phase detector

B. System Design

In the paper [4], the author has proposed the automatic vulnerability detection system for Java Web program (Vulnerability Detection System for Java). The system is mainly composed of the following modules:

1) Language recognition and code conversion

2) Dependency analysis

3) Taint analysis

4) Vulnerability detection engine

5) Attack mode knowledge base

6) Automata operation library

The Java source code can be converted into equivalent intermediate representation (IR) by lexical and grammar analysis. The well-designed IR preserves the structure information of the code and each element so that all can quickly locate and traverse the source code through IR. By traversing the IR, we can extract the statements dependency and method call dependency of the program.Meanwhile, author collect sensitive point by matching the sensitive API call while traversing the IR. Then author does the reverse taint dependency analysis for each fragile sensitive point based on the generated data dependency graph and method call dependency graph and construct the taint dependency graph from the fragile sensitive point to the pollution source. After that, author usesfinite state automaton to represent the possible value set of the tainted string and evaluate the taint value by using the automata operation library. Finally, it is realized in the program vulnerability detection by taking the intersection between the value of fragile sensitive point and the attack mode.

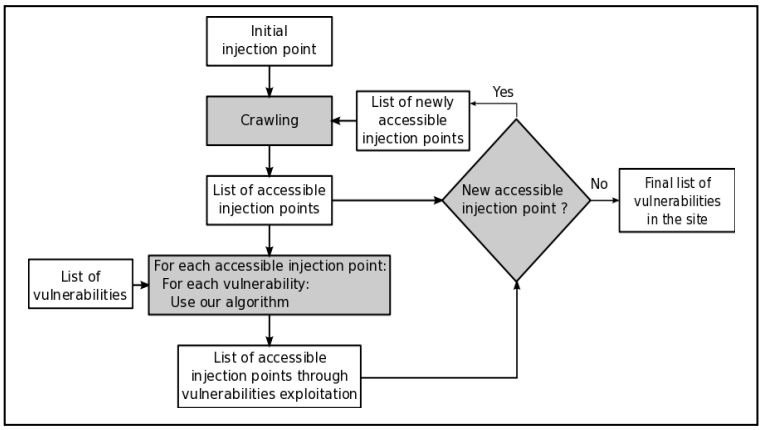

Fig 3: Vulnerability Prediction Model

III. PROPOSED SYSTEM

A. Problem Statement

The idea is to make use of both static as well as dynamic analysis to find the vulnerabilities present in a web application and also use the clustering algorithm for more efficiency. First, the code is scanned using a static analyzer. Then the results are collected, i.e. vulnerabilities that are detected by the static analyzer. To find whether the vulnerabilities reported by static analyzer truly exist, the dynamic analysis is performed. The dynamic analysis of a program will also report some vulnerabilities. The idea is to merge the results of both the analysis and report only those vulnerabilities that are reported during both analyses. In this way high confidence on reported vulnerabilities and low rate of false positives and false negatives can be achieved.

B. Proposed System

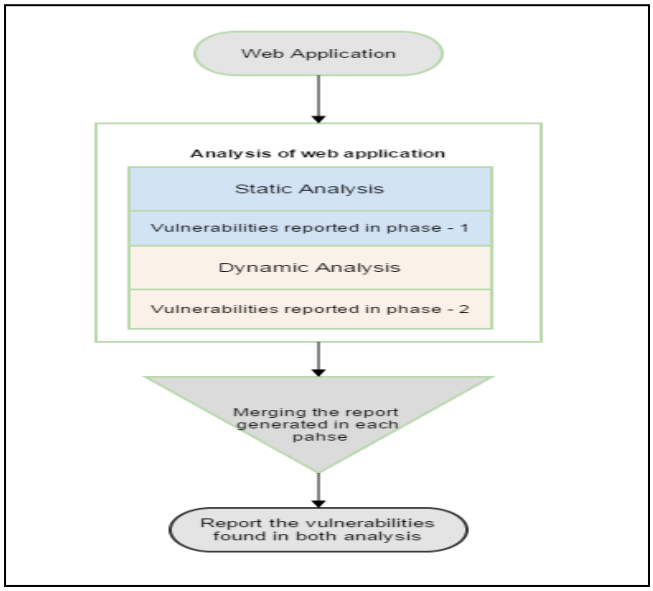

Fig 4: Vulnerability Detection Workflow

Usually static analysis is done before software is deployed and dynamic analysis is performed after software is deployed. It is very time consuming as well as costly to fix the vulnerabilities after a software is deployed. To overcome this problem, a method is proposed in which both static and dynamic analysis can be performed in development phase. Our static analyzer is

built by extending a static analyzer tool called FindBugs [2]. FindBugs works by analyzing the bytecode of a java program. A detector is written which can detect the specified bytecode pattern. A detector is a class which is able to detect one or more vulnerabilities. Every detector class is checked against the bytecode of a Java class to be analyzed. If any detector class finds a matching bytecode pattern, then it will report a vulnerability. On the other hand, dynamic scanners do not have access to source code. They detect vulnerabilities by actually performing attacks in a similar way as an attacker would have done. For example, to detect input validation error, a dynamic analyzer would have run the application using different vulnerable inputs.

C. Implementation

The system is divided in three modules. Module 1 is preparing a crawler to crawl web pages and get all details of the input web application. It takes input URL and crawls through all the pages, take the input and make it ready for analysis.
In Module 2 the JIID approach in eclipse environment will be used to find the input vulnerabilities. A detector is written and is used as a plugin for Findbugs and try to find the input injection vulnerabilities. The name of our plugin is MY_T6_DETECTOR. The following screenshot shows the implementation of the detector over Findbugs plugin.

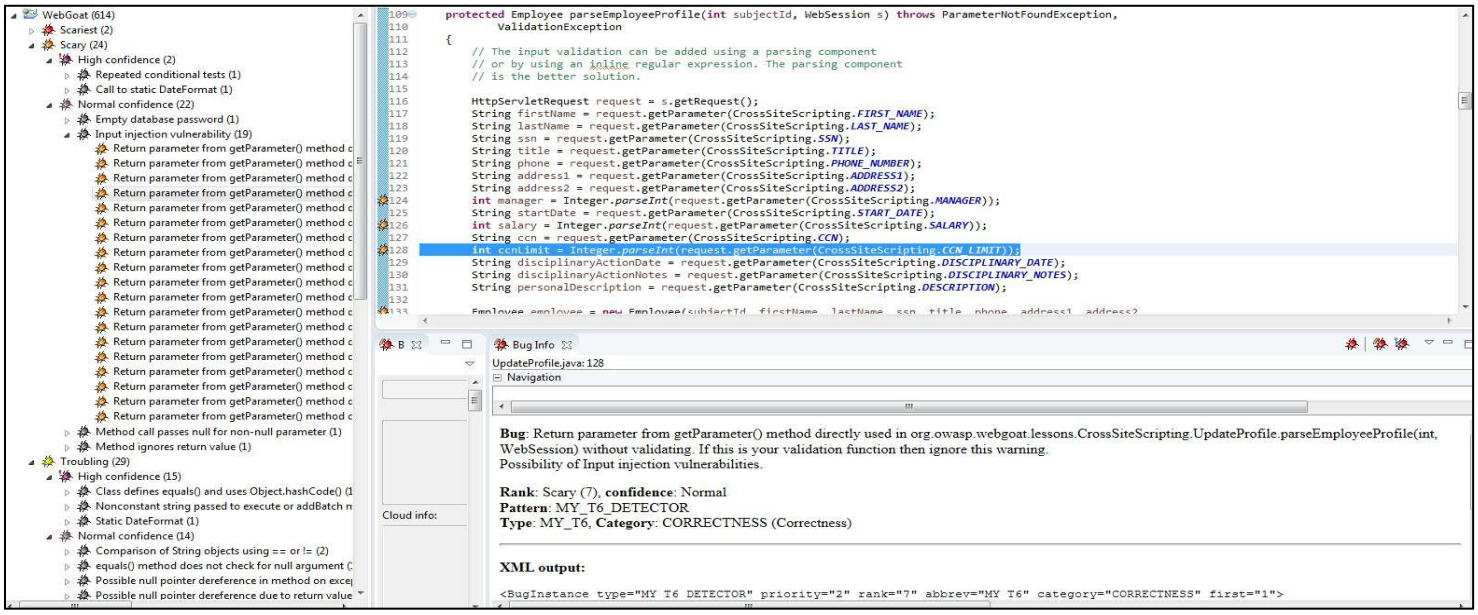

Fig 5: Screenshot for the static detector using Fingbugs plugin

In Module 3, clustering approach is used to find the vulnerabilities and increase the efficiency and true positives.

IV. Experimental Results

The results for static analysis have shown the improvement to detect the input injections in Webgoat 5.4 application (OWASP Project).

TABLE I: Webgoat scanning result for static analysis

Vulnerabilities	Existing Findbugs		Our Detector	
	TP	FP	TP	FP
Input Injections	5	-	19	2
Total	544		614	

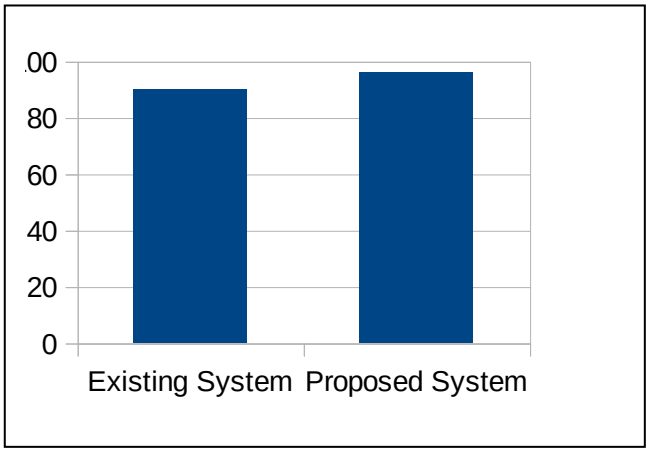

Fig 6: Expected Performance (Time to detect)

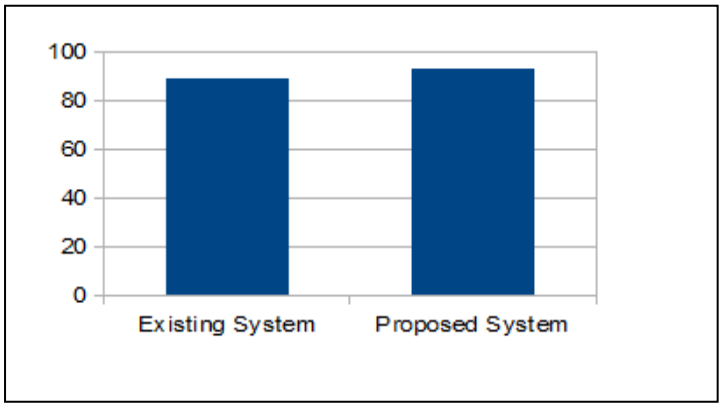

Fig 7: Expected Efficiency (True Positives)

V. Conclusion

To address the problem of injection vulnerabilities and to reduce the total cost and time required to remove the vulnerability, a method of pre-deployment vulnerability detection is proposed that uses both static and dynamic analysis techniques to detect the potential vulnerabilities present in the web application. If static analysis reports some vulnerabilities, then those vulnerabilities are tested using dynamic analysis to make sure that the vulnerabilities are actually present in the web application. The main focus is on Java EE web applications, but in future the plan is to extend this work to detect vulnerabilities in non-Java based web applications.

REFERENCES

[1] Lwin Khin Shar and Lionel C. Briand, "Web Application Vulnerability Prediction using Hybrid Program Analysis and Machine Learning", IEEE Transactions on Dependable and Secure Computing, pp 1545-5971(2014).

[2] Edward S. Pasaribu, Yudistira Asnar, and M. M. Inggriani Liem, "Input injection detection in Java code" In Data and Software Engineering (ICODSE), 2014 International Conference on, pages 1–6. IEEE, November 2014.

[3] A. Aggarwal and P. Jalote, "Integrating static and dynamic analysis for detecting vulnerabilities", In Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International, volume 1, pages 343–350, Sept 2006.

[4] A Static analysis tool. http://findbugs.sourceforge.net/.

[5] Open Web Application Security Project (OWASP). http://www.owasp.org.

[6] David Hoveymer, "The Architecture of FindBugs", http:// mcs.une.edu.au/doc/findbugs/ architecture.pdf

[7] WhiteHat website security statistics report: https://www.whitehatsec.com/assets/WPstatsReport 052013.pdf, 2013.

[8] Cisco Annual security report, http://investor.cisco.com/files/docdownloads/annualmeeting/Cisco-Systems-Annual-Report.pdf, 2014.

[9] JeremiaGrossman, http://jeremiahgrossman.blogspot.in/2009/05/mythbustingsecure-code-is-less.html,2014

[10] OWASP testing guide. https://www.owasp.org/index.php/TestingforInputValidation.

[11] Yannis Smaragdakis and Christoph Csallner, "Combining Static and Dynamic Reasoning for Bug Detection",In Tests and Proofs, volume 4454 of Lecture Notes in Computer Science. 2007.

[12] Christoph Csallner and Yannis Smaragdakis, "DSD-Crasher: A Hybrid Analysis Tool for Bug Finding", In Proceedings of the 2006 International Symposium on Software Testing and Analysis. ACM, 2006.

[13] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda, "Pixy: a static analysis tool for detecting Web application vulnerabilities", In Security and Privacy, 2006 IEEE Symposium on. IEEE, May 2006.

[14] Binbin Qu, Beihai Liang, Sheng Jiang, and Ye Chutian, "Design of automatic vulnerability detection system for Web application program", In Software Engineering and Service Science (ICSESS), 2013 4th IEEE International Conference on. IEEE, May 2013.

[15] Ruoyu Zhang, Shiqiu Huang, Zhengwei Qi, and Haibin Guan," Combining Static and Dynamic Analysis to Discover Software Vulnerabilities", In Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on. IEEE, pp-717-730, June 2011.

[16] G. A. Di Lucca, A. R. Fasolino, M. Mastoianni, and P. Tramontana, "Identifying cross site scripting vulnerabilities in Web applications", In Web Site Evolution, Sixth IEEE International Workshop, IEEE, 2004.

[17] Jos´e Fonseca, Marco Vieira, and Henrique Madeira, "Vulnerability attack injection for web applications",In Dependable Systems & Networks, 2009. IEEE/IFIP International Conference on. IEEE, pp-414-423, June 2009.

[18] Pranahita Bulusu, Hossain Shahriar, and Hisham M. Haddad.,"Classification of Lightweight Directory Access Protocol query injection attacks and mitigation",In Collaboration Technologies and Systems(CTS), 2015 International Conference on.IEEE, June 2015.

[19] A. Dessiatnikoff ,R. Akrout Etl. ,"A clustering approach for web vulnerabilities detection", 17th IEEE Pacic Rim International Symposium on Dependable Computing (PRDC 2011), Dec 2011, Pasadena, CA, United States. IEEE Computer Society, pp.194-203, 2011.

Authors Profile

T.Patilreceived the **B.E.** degree in Computer Science and engineering from the Dr.J J Magdum College, Shivaji University, Kolhapur, India, in 2007.Currently doing **M.E.** in Information Technology in Savitribai Phule Pune University, Pune, India. His research interest includes Security, Input injection in java code.

M.Penurkar received **B.E. (C.S. E.)** degree from S.G.G.S. Institute of Engg. & Tech.(formerly SGGS COET), S.R.T.M.University, Nanded, Maharashtra, India in 2001. He received his **M. Tech. (Comp. Engg.)** from COEP, Savitribai Phule Pune University, Pune, Maharashtra, India in 2007. He is currently doing his Ph.D. (C. S. E.) from VNIT, Nagpur, India. His research interests include Wireless Sensor Networks, Delay Tolerant Networks & Distributed Systems.

F.Shaikh received B.E. (I.T) degree from P.V.G's Institute of Engg. &Tech.(formerlyPVG COET), Savitribai Phule Pune University, Pune, Maharashtra, India in 2009. she received her M. Tech. (Software Engineering) from Jawaharlal Nehru Technological University, Hyderabad, AP, India in 2013. Her research interests include Web Security, IoT, Mobile Ad hoc Networks, Distributed Systems.