International Journal of Engineering & Technology, 7 (2.3) (2018) 4-8

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research Paper

SAISAN: An Automated Local File Inclusion Vulnerability Detection Model

Md. Maruf Hassan *,Touhid Bhuyian, M. Khaled Sohel, Md. Hasan Sharif, Saikat Biswas

Software Engineering Department, Daffodil International University, 102 Mirpur Road, Dhaka, Bangladesh,1207
**Corresponding author E-mail: maruf.swe@diu.edu.bd*

Abstract

Communicating and delivering services to the consumers through web applications are now become very popular due to its user friendly interface, global accessibility, and easy manageability. Careless design and development of web applications are the key reasons for security breaches which are very alarming for the users as well as the web administrators. Currently, Local File Inclusion (LFI) vulnerability is found present commonly in several web applications that lead to remote code execution in host server and initiates sensitive information disclosure. Detection of LFI vulnerability is getting very critical concern for the web owner to take effective measures to mitigate the risk. After reviewing literatures, we found insignificant researches conducted on automated detection of LFI vulnerability. This paper has proposed an automated LFI vulnerability detection model, SAISAN for web applications and implemented it through a tool. 265 web applications of four different sectors has been examined and received 88% accuracy from the tool comparing with the manual penetration testing method.

Keywords: *Cyber Security, Web Application Security, Web Application Vulnerability, Automated Vulnerability Detection Tool, Local File Inclusion (LFI).*

1. Introduction

Use of internet reaches over 3.6 billion by this time through different channels and devices [11]. Web application is the key area to modernize the world by automating the processes. Therefore, businesses dedicate their effort in restructuring their processes and delivering the services through web applications to their stakeholders for receiving better outcome. Although web applications make our life easy, it increases the risk of exploitation in case any vulnerability remains in application due to its insecure design and development. The most common vulnerabilities of the web applications are injection, broken authentication and session management, cross-site scripting (XSS), broken access control, security misconfiguration, sensitive data exposure, insufficient attack protection, cross-site request forgery (CSRF), using components with known vulnerabilities, under protected APIs, Local File Inclusion, etc. [12],[13]. The consequence of vulnerability exploitation may lead to service interruption, sensitive data/file disclosure, get full control over the host application, etc. Local File Inclusion (LFI) refers to an inclusion attack through which an attacker can trick the web application by including files/scripts on the web server through exploiting functionality. The effect of successful exploitation of LFI vulnerability includes directory traversal, information disclosure, and remote code execution. In our investigation we found that 48.6% web application contains LFI vulnerability. Thus, detection of LFI vulnerability becomes very crucial for the web administrators to take effective measures. Automatic detection is always preferable than manual way as it reduces the time and efforts. In order to figure out an automated detection solution of different vulnerabilities, a number of researches have proposed several models and in some cases they developed scanner or detection tools based on their proposed models. This study has discovered that most of the proposed model and implemented tools/ techniques have been developed for only XSS and SQLi vulnerabilities. It is observed that very insignificant researches on LFI vulnerabilities and its detection have been taken place. This study has propose a detection model, SAISAN where the solution can detect LFI weaknesses of the web applications. This research also implement SAISAN through a tool where the result of the tool will be compared with manual penetration testing method [14] to figure out the accuracy of the model. This paper is organized in seven sections. Introduction and Literature Review are discussed in section 1 and 2 respectively, LFI exploitation techniques are explained in section 3. Methodology is discussed in section 4 where SAISAN model has been described. Experiment Result and discussion are presented in section 5 and section 6 respectively. The paper is concluded with the outcome of the study, limitation, and future work in section 7.

2. Literature Review

This review observed that a good number of researches have been conducted on web application vulnerability and its detection models. Some researchers performed survey on SQL injection, XSS, broken authentication and session management, Insecure Cryptographic Storage etc. [1][2]. Others conducted case study on different web application vulnerability exploitations in various domains of Bangladesh. A study on three major SQLi techniques implemented on the educational and financial websites of Bangladesh and executes analysis web applications for figuring out the security condition [3],[4]. Another study found on LFI vulnerability and its exploitation techniques based on SQLi and RFI vulnerability in which they examined 153 LFI vulnerable web applications and shown its impact in Bangladesh [5]. Some review related studies have also been conducted on verity of web vulnerability detection tools. Review the security testing on Tunestore using Paros, WebScarab, JBroFuzz, Acunetix, and Fortify vulnerability detection tools and found the accuracy of the tool result compared with manual penetration testing method [6].

A model was proposed and developed for detecting SQLi based on the defined and identified criteria. The model composed a module which will execute a process by employing the Boyer Moore string matching algorithm to make it more efficient and accurate detection [7]. Another research suggested a model, Escrow for detecting large-scale SQLi in an efficient manner. Escrow uses a custom search implementation together with a static code analysis module to find potential target web applications [8]. Based on clustering techniques, a methodology has been developed aiming to identify web application vulnerability. They developed Wasapy vulnerability scanner and compared the result with W3af 1.1, Skipfish 1.9.6b, and Wapiti 2.2.1 focusing on code injection type vulnerability [9]. A method proposed to create test input using attack pattern with applying permutation and combination algorithm for several SQL injection [23].A sample prototype implementation with Open Web Application Security (OWASP) enterprise security application API based on Rapid Application Development (RAD) methodology to minimize web application flaws and prevent from critical malicious attacks [24]. A research presented KameleonFuzz, a blackbox genetic algorithm driven fuzzer targeting Type-1 and 2 XSS. They compared their tool performance with some market available tools and found out the effectiveness of the tool based on true XSS detection skills [10].

In view of the above, it is observed that insignificant researches have been focused on LFI vulnerability and its detection. In this paper, we will propose a model for automated detection of LFI vulnerability, SAISAN from web applications. A tool will be developed based on SAISAN and run it on web applications over four sectors to get the result. The result of the tool will be compared with manual penetration testing result for checking the accuracy.

3. LFI Exploitation Techniques

LFI vulnerability allows attacker to include files or scripts on the web server through exploiting inappropriate use of INCLUDE and REQUIRE function in the code of the web application. There are mainly two types of LFI exploitation techniques that are found in practice. In this study, we will discuss the types of general LFI exploiting technique. The details are explained below:

3.1. $_GET Parameter Based Exploitation

The http $_GET parameter includes different variables containing several files and pages for handling particular operations. It passes the argument through the URL bar. The value of variables is visible to the user and also can be modifiable. For an example, http://www.website.com/downloads.php? file=contact.php is a PHP developed application. To identify the LFI vulnerability of the above URL, attacker will modify the parameter/ value of file as /etc/passwd like e.g. http://www.website.com/downloads.php? file=/etc/passwd. During the exploitation, the use of null byte (%00) will help to bypass firewall restriction. If the execution of the given modification replies following code, it indicates that the site is LFI vulnerable.

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
alex:x:500:500:alex:/home/alex:/bin/bash
…….
Profile:/home/oprofile:/sbin/nologin
sebl:x:500:500:sebl:/home/sebl:/bin/bash mysql:x:27:27:MySQL
```

The above code shows the example of the output of /etc/passwd file for LFI vulnerable site in case of parameter modification. In the reply, system discloses sensitive information like root user, password, SSH login information, etc.

3.2. $_POST Parameter Based Exploitation

Developers usually design the data processing techniques through the HTTP POST methods to impose security feature. Therefore, user will not be able to view the transformation data easily. However, the intruder can view the data from session cookies and can perform LFI exploitation even the $_POST method protection. The exploitation can be performed because of the improper use of different function/ methods. In the cookies, this hidden filename information exists with other data which will be the format as "?file=/etc/passwd".

Developers also store different types of files in different directory to ensure security. As a result, attacker may not be able to find the file that they are looking for. To bypass this type of security, the attacker will perform directory traversal query execution for getting their desired file. The directory traversal [15],[16] command i.e. ?file=../../../../../../../../etc/passwd is used for changing the location directing to the root directory.

4. Methodology

Our investigation followed experimental design methodology to proof the result accuracy of the SAISAN model. We divide our methodology into two parts i.e. experimental environment as our developed detection tool based on SAISAN, and control environment as manual penetration testing method [14]. The SAISAN model based tool is implemented in python programing language.

Fig-1 represents how our proposed model, SAISAN works using HTTP $_GET method for detecting LFI vulnerability. Working process of SAISAN is presented below:

Step 1: The testing phase started with the verification of the inputted URL. The proposed model checks the HTTP 200 web response code for ensuring whether the web application is active or not. If the response is matched, the following steps will be continued; otherwise, exception will be provided.

Step 2: SAISAN will then send a request to the target web application for source code. The given code will be analysed to find out all possible URLs of the above application.

Step 3: The model will observe the parameterized URLs that have the possibility of containing LFI vulnerability. The proposed model only selects the parameterized URLs for consuming the execution time and finding out the possible number of vulnerability.

Step 4: Split the parameters of the selected URLs and add payload along with the parameter.

Step 5: Send the above full query to the target web application using user-agent.

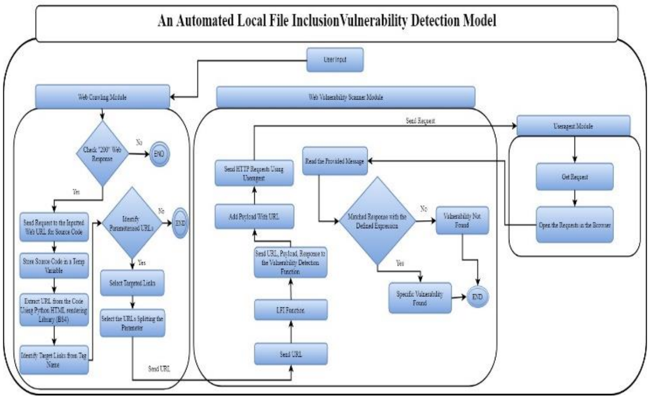

An Automated Local File Inclusion Vulnerability Detection Model
User Input
Web Crawling Module
Check "200" Web Response
No
END
Yes
Send Request to the Inputted Web URL for Source Code
Store Source Code in a Temp Variable
Extract URL from the Code Using Python HTML rendering Library (BS4)
Identify Target Links from Tag Name
Identify Parameterized URLs
No
END
Yes
Select Targeted Links
Select the URLs Splitting the Parameter
Send URL
Web Vulnerability Scanner Module
Send Request
Send HTTP Requests Using Useragent
Add Payload With URL
Send URL, Payload, Response to the Vulnerability Detection Function
LFI Function
Send URL
Read the Provided Message
Matched Response with the Defined Expression
No
Vulnerability Not Found
Yes
Specific Vulnerability Found
END
Useragent Module
Get Request
Open the Requests in the Browser

Fig. 1. SAISAN: Automated LFI Detection Model

Step 6: Response will be received and matched with the predefined expressions to confirm the existence of LFI vulnerability in the web application.

4.1. Implementation of SAISAN Model

The SAISAN model based tool is implemented in python programming language. The tool is developed with three modules with five steps that are sequentially identifying the LFI vulnerability from the inputted URL. The steps are briefly discussed below:

4.1.1. URL Validation

This step verifies web response code of 200 status [17] that ensures the host of provided URL in live state or not. If the response code is matched with 200 status, the program will forward the given URL to the crawling [18] step. Otherwise, an error message with "Host server is not available" will be displayed before quitting the program.

4.1.2. Crawling

This step will send a request to the provided host application for source code of the page. The tool will store the code in a temporary variable and extract the web URLs used in the code with the help of Beautiful-soup 4.0 module [19] for identifying the tag name. After recognizing the parameterized links, the tool will split the parameters of the URLs and send the URLs to the web vulnerability scanning module removing the parameters.

4.1.3. Execution of the URLs

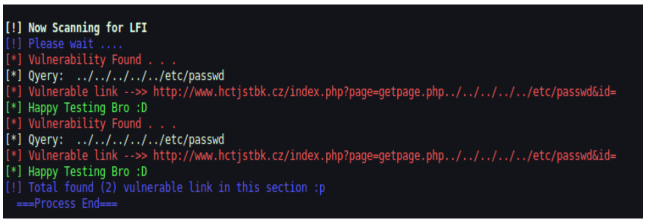
Defined payload will be included with the URLs received form the crawling module to make a valid string. By using user agent [20], the fabricated string will be sent to the target host for a response.

4.1.4. Collect and Matched Response

Once the response is received from the web application, it will be matched with the predefined expressions. The tool will confirm the existence of LFI vulnerability in the application if match found.

4.1.5. Provide Output

This module will provide the outcome in a defined format to the user of the tool. Fig. 2 shows the sample output of the automated LFI detection tool of SAISAN model.



```
[!] Now Scanning for LFI
[!] Please wait ....
[*] Vulnerability Found . . .
[*] Qyery:  ../../../../../etc/passwd
[*] Vulnerable link -->> http://www.hctjstbk.cz/index.php?page=getpage.php../../../../../etc/passwd&id=
[*] Happy Testing Bro :D
[*] Vulnerability Found . . .
[*] Qyery:  ../../../../../etc/passwd
[*] Vulnerable link -->> http://www.hctjstbk.cz/index.php?page=getpage.php../../../../../etc/passwd&id=
[*] Happy Testing Bro :D
[!] Total found (2) vulnerable link in this section :p
  ===Process End===
```

Fig. 2. SAISAN Detecting LFI Vulnerability

5. Experiment Result

In this study, the sample web application selection is platform independent. For choosing the sample site, some google dork has been used. This research has compared our automated detection tool's result with manual penetration testing method [14]. The experiment has conducted on 265 web applications. Small sample technique has been selected as sampling method for this study. The above technique has been constructed using the Eq.1 [21]

$$s = X^2 + NP(1 - P) \div d^2(N - 1) + X^2P(1 - P) \qquad (1)$$

A statistical tool has been G*Power 3.1.9.2 used to figure out the sample size of this study by applying the Eq.1. Linear multiple regression test has been conducted under F tests family. This case the selection predictor is 4.

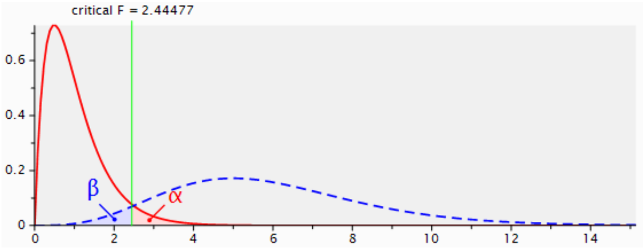


Fig.3. G*Power result for sample size of five predictors using small sample technique

This study has decided to put α err prob value as 0.05 and 1-β err prob as 0.95 in the statistical tool. Result of the tool reflects that minimum of 129 valid samples need to be examined to proof our model. Fig. 3 shows the graph of result for sample size of four predictors using small sample technique [21]. After examining 265 web applications, we finally got 129 LFI vulnerable web applications. That means 48.6% websites were found with LFI Vulnerability among the sample. This analysis received result from both experimental environment and control environment to compare the accuracy of our model. The analysis of the result is described in the following four sections.

5.1. Detection Result Comparison

As it is evident from different literatures, manual penetration testing method always provides cent precent of accuracy [18] and it is selected as the control environment for this research examination. Fig. 4 indicates the result comparison of control and experimental environment using bar chart.

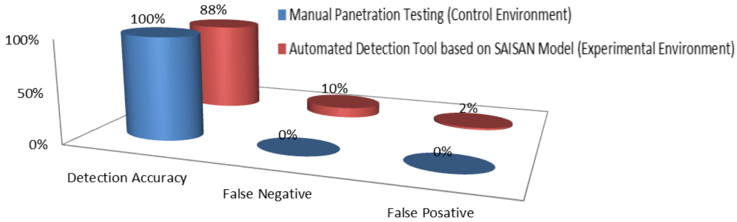


Fig. 4. Result comparison of control and experimental environment

When we ran our tool over the sample, we detected 113 LFI vulnerable sites while manual penetration testing identified 129. That means, SAISAN model received 88% accuracy over control environment. It is also to be noted that our tool observed 2% false positive and 10% false negative result during the study.

5.2. Sector Based Analysis

Table 1 shows the sector wise detection of LFI vulnerable web application. The web

Table 1. Sector wise detection of LFI vulnerable web applications

Sector	Vuln. Web App.	Percentage	C. Percentage
Educational Institutions	55	42.64%	42.64%
E-Commerce	11	8.53%	51.17%
Medical Institute	29	22.48%	73.65%
Govt. Counterpart	34	26.36%	100%
Total	129	100%	

Applications of educational institutions are mostly affected by the LFI vulnerability with the percentage of 42.64%. E-Commerce, Medical Institutes, and Government counterpart sites are having LFI vulnerability with the percentage of 8.53%, 22.48%, and 26.36% respectively.

5.3. Platform Based Analysis

Fig. 5 shows the pie chart of the percentage of vulnerable web applications developed with different programing language platform examined in our study. It is observed that PHP language based web applications are more LFI vulnerable with the percentage of 74% whereas JAVA and ASP.NET have the same weakness with 11% and 15% respectively.

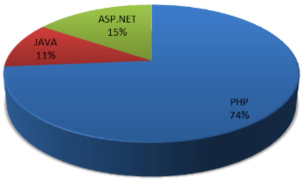
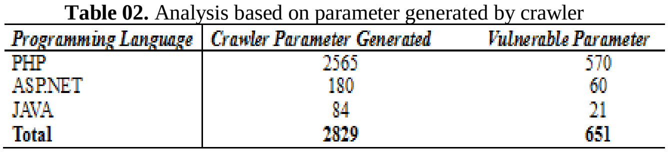


Fig.5. percentage of vulnerable web applications developed with different programing language platform.

Table 02. Analysis based on parameter generated by crawler



Programming Language	Crawler Parameter Generated	Vulnerable Parameter
PHP	2565	570
ASP.NET	180	60
JAVA	84	21
Total	**2829**	**651**

Table 02 represents the analysis based on parameter generated by crawler. It is to be noted that we examined three different types of programming platforms which include PHP ASP.NET and JAVA. To increase the accuracy of our analysis, we designed a web crawler that generally gathers susceptible links for the LFI vulnerability testing process. After gathering all URL parameters from the source page, tool on SAISAN conducts an automated black box testing [22] for every parameter by using LFI detecting model and provides the total number of vulnerable parameters. The result shows that the detection tool based on SAISAN model generated 2565 crawler parameters for PHP developed applications in which 570 parameters were returned with sensitive information from the host application. ASP.NET built web applications disclosed information for 60 parameters out of 180 parameter produced by our tool. 21 parameters returns with sensitive server side information for the JAVA developed web applications in which crawler of the tool generated 84 parameters.

6. Discussion

This study has found 129 LFI vulnerable web applications out of 265 test sites. The implemented automated vulnerable detection tool based on SAISAN model provided result with 88% accuracy comparing with manual penetration testing. SAISAN based tool observed 2% and 10% false positive and false negative result respectively for detecting LFI vulnerability. We analysed our 129 vulnerable web applications in four sectors where we observed the most vulnerable sector is the web applications of education sector with the percentage of 42.64%. This analysis shows that the less vulnerable sector is E-Commerce sector with 8.53%. The examination also conducted based on implemented programming language. The above analysis proved that PHP developed web applications are more vulnerable of LFI vulnerability.

In view of the above, this study can claim that SAISAN model is effective and the tool that was developed based on the model provides maximum accuracy and improved correctness of detecting LFI vulnerability. Thus, this research can say the output of the study satisfies the research objective.

7. Conclusion

In these recent years, web application vulnerabilities have become a critical problem for all types of people who have been connected with the web. This research has presented an automated LFI vulnerability detection model, SAISAN and implemented a tool based on the model, which is developed in Linux platform. An examination has been performed on 265 real world web applications where the tool was successfully able to identify 113 vulnerabilities. Result of the tool has been compared with manual penetration testing outcome and found 88% accuracy. This study has observed that the biggest problem has been recognized as the insecure design of the applications and careless coding practice especially in using data/information retrieving methods. This research is a continuous process, and the research has been working on it to improve the accuracy. SAISAN based tool only works for $_GET method. In next version plan is to add $_POST method for detecting LFI vulnerability from the web application. We will be adding more features in future that will detect other web application vulnerabilities as well.

References

[1] O. B. Al-Khurafi and M. A. Al-Ahmad, "Survey of Web Application Vulnerability Attacks," 2015 4th International Conference on Advanced Computer Science Applications and Technologies (ACSAT), Kuala Lumpur, 2015, pp. 154-158.

[2] P.V. Ami, S. C. Malav, "Top Five Dangerous Security Risks over Web Application," International Journal of Emerging Trends & Technology in Computer Science, 2013, pp.41-43.

[3] D. Alam, M. A. Kabir, T. Bhuiyan and T. Farah, "A Case Study of SQL Injection Vulnerabilities Assessment of .bd Domain Web Applications," 2015 Fourth International Conference on Cyber Security, Cyber Warfare, and Digital Forensic (CyberSec), Jakarta, 2015, pp. 73-77.

[4] T. Farah, D. Alam, M. A. Kabir and T. Bhuiyan, "SQLi penetration testing of financial Web applications: Investigation of Bangladesh region," 2015 World Congress on Internet Security (WorldCIS), Dublin, 2015, pp. 146-151.

[5] A. Begum, M. M. Hassan, T. Bhuiyan and M. H. Sharif, "RFI and SQLi based local file inclusion vulnerabilities in web applications of Bangladesh," 2016 International Workshop on Computational Intelligence (IWCI), Dhaka, 2016, pp. 21-25.

[6] L. Dukes, X. Yuan and F. Akowuah, "A case study on web application security testing with tools and manual testing," 2013 Proceedings of IEEE Southeastcon, Jacksonville, FL, 2013, pp. 1-6.

[7] G. Buja, K. B. A. Jalil, F. B. H. M. Ali and T. F. A. Rahman, "Detection model for SQL injection attack: An approach for preventing a web application from the SQL injection attack," 2014 IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE), Penang, 2014, pp. 60-64.

[8] B. Delamore and R. K. L. Ko, "Escrow: A Large-Scale Web Vulnerability Assessment Tool," 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications, Beijing, 2014, pp. 983-988.

[9] R. Akrout, E. Alata, M. Kaaniche and V Nicomette, "An automated black box approach for web vulnerability identification and attack scenario generation," Journal of the Brazilian Computer Society, 2014, 20(1), 4.

[10] F. Duchene, S. Rawat and J.L. Richier, "KameleonFuzz: Evolutionary Fuzzing for Black-Box XSS Detection," In Proceedings of the 4th ACM conference on Data and application security and privacy, 2014 pp. 37-48.

[11] (October 18, 2017) Internet Users. Available: http://www.internetlivestats.com/internet-users/

[12] (October 18, 2017) Category:OWASP Top Ten Project. Available: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013

[13] (October 18, 2017) CWE/SANS TOP 25 Most Dangerous Software Errors Available: https://www.sans.org/top25-software-errors.

[14] Y. Stefinko, A. Piskozub and R. Banakh, "Manual and automated penetration testing. Benefits and drawbacks. Modern tendency," 2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET), Lviv, 2016, pp. 488-491.

[15] J. Esmet, M. A. Bender, M. Farach-Colton, B.C Kuszmaul,"The TokuFS Streaming File System," InHotStorage, 2012

[16] W. Jannen, J. Yuan, Y. Zhan, A. Akshintala, J. Esmet, Y. Jiao, A. Mittal, P. Pandey, P. Reddy, L. Walsh, M. A. Bender, M. Farach-Colton, R. Johnson, B. C. Kuszmaul, and D. E. Porter, "BetrFS: Write-optimization in a ker- nel file system," Transactions on Storage, Article 18,29 pages, Nov.2015.

[17] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts". 4th International Symposium, RAID 2001 Davis, CA, USA, 2001,pp. 85-103

[18] G. Deepa , P. S. Thilagam, F. A. Khan, A. Praseed, A.R. Pais, And N. Palsetia," Black-box detection of XQuery injection and parameter tampering vulnerabilities in web applications," International Journal of Information Security, 2017, pp. 1-16

[19] Y. L. Chen, H. M. Lee, A. B. Jeng and T. E. Wei, "DroidCIA: A Novel Detection Method of Code Injection Attacks on HTML5-Based Mobile Apps," 2015 IEEE Trustcom/BigDataSE/ISPA, Helsinki, 2015, pp. 1014-1021.

[20] G. Vigna, W. Robertson, Vishal Kher and R. A. Kemmerer, "A stateful intrusion detection system for World-Wide Web servers," 19th Annual Computer Security Applications Conference, 2003. Proceedings., 2003, pp. 34-43.

[21] V.K.Robert, W.M.Daryle, "Morgandeter Mining sample size for research activities", Educational and Psychological Measurement, The NEA Research Bulletin, 1970, Vol. 38,p. 99.

[22] J. S.Kang and H. S.Park, "Web-based automated black-box testing framework for component based robot software," 2012 ACM Conference on Ubiquitous Computing, 2012, pp. 852-859.

[23] N. F. Awang, A. Manaf and S.F. Abidin, "Test Input Generation for Detecting SQL Injection Vulnerability in Web Application," International Journal of Soft Computing, 11(2), pp. 103-106, 2016.

[24] A. B. M. Rasheed, B. Shanmugan, G.N. Samy, N. Maarop, P. Megalingam, K.C. Yeo and S. Azam, "Secure Web Application Development Prototype Using Enterprise Security Programming Interface (ESAPI)," Asian Journal of Information Technology,