An Enhanced Rule-Based Web Scanner Based on Similarity Score

Minsoo LEE[1], Younho LEE[2,*], Hyunsoo YOON[1]
[1]*Department of Computer Science, KAIST, Korea*
[2]*ITM Programme, Dept. Industrial and Systems Engineering, SeoulTech, Korea*
mslee@nslab.kaist.ac.kr, younholee@seoultech.ac.kr, hyoon@nslab.kaist.ac.kr

***Abstract*—This paper proposes an enhanced rule-based web scanner in order to get better accuracy in detecting web vulnerabilities than the existing tools, which have relatively high false alarm rate when the web pages are installed in unconventional directory paths. Using the proposed matching method based on similarity score, the proposed scheme can determine whether two pages have the same vulnerabilities or not. With this method, the proposed scheme is able to figure out the target web pages are vulnerable by comparing them to the web pages that are known to have vulnerabilities. We show the proposed scanner reduces 12% false alarm rate compared to the existing well-known scanner through the performance evaluation via various experiments. The proposed scheme is especially helpful in detecting vulnerabilities of the web applications which come from well-known open-source web applications after small customization, which happens frequently in many small-sized companies.**

Index Terms—intrusion detection, access control, information security, web services, security.

I. Introduction

With the explosive growth of web-based application, a lot of web sites are widely used in our daily life. Many content providers give customers information via web site. To reduce the cost from their web applications, the site managers can use open-source web applications, such as [1]. By employing open-source web applications, they can reduce the cost of developing and installing their web applications. Also, they tend to customize these open-source web applications by modifying small portions of them before their use for various reasons, such as companies' demand and site managers' preference.

To reduce cost, companies may not employ web security manager who is qualified in preventing their web pages from hacker's attacks. In this case, to prevent a web server from hacker's attack, site managers have no choice but to use the tools to the web applications on their web servers, which automatically analyze the vulnerabilities, such as Nikto [2], Wikto [3], N-stalker Security Scanner [4], Acunetix Scanner [5], and other scanners [6-9] due to various reasons such as insufficient security knowledge.

According to [10], these tools are categorized into parameter-based validation method [4-7,11-19] and rule-based testing method [2-3,8-9]. In this paper, we focus on the latter type for efficiency because our main target is open-source web applications. Since they are open and public, their vulnerabilities are easily detected, categorized, and distributed. Moreover, it is likely that the websites based on open-source applications have similar structures.

Unfortunately, the rule-based approach makes a false alarm when the site manager customizes their web application. Because the customized file path or file name can obscure the vulnerability detection. Thus, an enhanced rule-based approach is necessary to prevent such a problem. This problem is non-trivial in that site managers tend to customize open-source web applications before deployment.

We tackle this problem by suggesting an improved rule-based scanner working with similarity score. The similarity score is defined between a target page to be tested and a rule page. It can be calculated by checking the difference of the unique characters in them. The unique characters mean the components of the web pages that are classified into four types, based on the degree of the difficulty in modification. The proposed scanner counts the number of the components that are common in both pages. The counting is executed per each type. The similarity score is proportional to the number of the common components. Also, the score varies on the type of the common components. I.e., the more similarity score is given if the more difficult-to-modify components are common in both pages.

Since the proposed scanner uses the unique characteristic of a web page instead of using only installation paths, it effectively reduces the false alarm rate compared to the existing rule-based methods.

We conduct various experiments in order to prove that the proposed scheme has better accuracy than the existing schemes. The result shows that the proposed scheme outperforms the existing in terms of 12% less false positive rate.

The rest of the paper is organized as follows. In Section II, we review the related work. In Section III, we explain the operation steps of the proposed system and the proposed algorithm to calculate the similarity score between two web pages. Section IV introduces the experiments we conducted to test the accuracy and cost of our approach. We conclude our paper in Section V.

II. Related Work

The past work on the vulnerability analysis in web environment can be generally grouped into two types, which are parameter-based method and rule-based method.

The former relies on generating web requests that include hostile and attack codes. Then, it checks the response of the target hosts against the hostile codes. This approach focuses on the variables used by web application. It has an

*Corresponding author: Younho Lee
This study was supported by the Research Program funded by the Seoul National University of Science and Technology.

Digital Object Identifier 10.4316/AECE.2016.03002

advantage over the other type in terms of detecting unknown vulnerabilities because most possibly-hostile data-set is tested to each variables of web page. However, the disadvantages are that the detection cost is higher than the latter, and false alarms can occur in some situations depending on the server status. The Secubat provides this functionality [20].

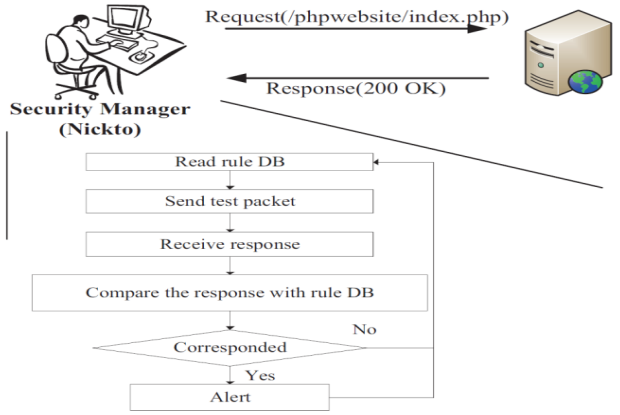

Figure 1. Conventional vulnerability detection procedure in the rule-based approach

The latter uses the database containing pre-defined vulnerabilities, rule name, the contents of request packets and their responses, as in Table I. It tries to request http packets based on these data to the target server that is suspicious to run vulnerable application as in Fig. 1. This approach focuses on checking if vulnerable application exists in the server. Fingerprinting tools such as Nessus and Nmap are included in this category [8-9]. This approach can detect the known vulnerabilities well. In addition, it can inspect the mis-configuration in the server, and its performance is relatively better than former. However, it must have a vulnerability repository. The Wickto and Nikto, web vulnerability scanner, and other commercial products such as AWVS provide vulnerability repositories [2-3].

TABLE I. SAMPLE RULE IN THE NIKTO

Field	Example
Type	Generic
Query Message	/phpwebsite/index.php?module=calendar
Response Message	DB Error:syntax error
Method	GET
Description	phpwebSite 0.9x and below are vulnerable to SQL injection

The rule-based approach is fast because it is pretty simple. Fig. 1 presents how a rule-based web scanner detects vulnerability of a web site by showing an example of Nikto. In this figure, Nikto requests a packet including a file path of vulnerable web application to the web server. Then, if the web server has the same file with the requested one, it sends a response packet including “200 OK”. Otherwise, it sends an http error packet [2]. At this point, Nikto can recognize vulnerability through analyzing the response.

Some previous works pointed out the limitations of black-box testing method for web application [21-23]. Typically, the rule-based method can detect only known vulnerabilities, and it makes a false alarm in some cases. Nevertheless, it is most popular and basic method to find vulnerability.

III. AN ENHANCED RULE-BASED APPROACH

Unlike the rule-based approaches that usually recognize the vulnerability fully depending on fixed file names and installation paths, our system uses a similarity score between two web pages on top of the file-name and installation-path comparison approach. Thus, the proposed scheme can improve the accuracy of vulnerability test. In the following subsections, we detail the proposed algorithm and the way of calculating similarity score.

A. Proposed Scheme

We first introduce the fields in our rule-set as follows: Type, Query Message, Response Message, Method Description, and File name including Instance Contents. The last field is a new field which means an xml-format file that saves a list of feature instances such as 'find_username' and 'sid:hidden'. The feature instances will be addressed in the next subsection.

The proposed scheme is different from the common rule-based approach shown in Fig. 1 in that the first step (Read rule DB) and the second step (Send test packet) changes to the following algorithm, which is used to figure out whether the page file from the rule DB and the target page file to be tested are similar or not. The similarity score is used to represent the degree of the similarity, and the algorithm decides if the scanner sends a test packet based on the similarity score. The following is a pseudo-code description of the proposed algorithm.

In the line #2, the proposed system maintains a list of all page names and paths. It can gain a file list that has the files that are possibly vulnerable after executing line #9. Then, it sends a test packet to these files in the target server and checks the responses. After that, it decides if they are vulnerable or not. In this algorithm, we only consider file names in order to make a list of test candidates. It is a reasonable approach because the web pages in the same web application have relationship with one another such as cross-link relation. Thus, a site manager cannot easily customize the file names of the web pages in a web application.

B. Similarity Score

A web page is usually composed of HTML tags, CSS (Cascading Style Sheets), Scripts, and other objects such as flex and Active X Control. The similarity score is defined as a value that represents the similarity of the parts of visible HTML, program variables, and functions between a rule page and a target page.

A web page has unique noticeable characteristics such as script function name, layout of web page, and image files, depending on the character of the developers who made the page. Thus, these noticeable characters can help to correctly recognize different HTML files. These characters should be considered in calculating the similarity score. The features we have chosen are given in Table II. The reason is provided as follows:

- A different programming style is represented in the web

application such as function name and variable name due to developer.

- A web application composed with several web pages has cross-hyperlinks in their web pages.
- A web application has a unique image file path.

```
1:While visiting the all web page on a target
  server do
2:  save the all web page names and path
3:end while
4:for each of the saved file names in step 2
  do
5:  if the file name is in the rule DB then
6:     rule_file ← request the file contents
                  from rule DB
7:     instance_file ← extract the instance
                   list from the target server
8:     score ← the result of calculating a
               similarity score between
               rule_file and instance_file
9:     if pre-defined threshold < score then
10:         send a test packet
11:    end if
12:  end if
13:end for
```

Figure 2. Proposed scheme

Among the selected features, the names of the functions (feature # 1) and the program variables (feature # 3) are in the role of system, in that they are related to the functions of the web application. On the other hand, link elements (feature # 2) and Img elements (feature # 4) are used for representing the visual parts of a web page. All these features represent the characteristic of a web page in a web application.

We define two types of values α_i, β_i for each feature $\#i$ $(i = 1,2,3,4)$ as in Table III, which reflects the difficulty of modification in a web page. Feature #1s are not easy to modify because they can be used through multiple pages in a web application. Therefore, without full understanding of the whole application, it requires much effort for a site manager to modify the name of functions if the site manager does not have sufficient skill in understanding web applications. This argument also works in the case of feature #2s that are usually used to link from a web page to another web page with relaying some input data, because the change of links might affect the functions of the whole web application. This change is also a difficult work for a site manager who has only a basic knowledge of a web application. Thus, we conclude that these features are relatively more difficult to customize than the other features. Therefore, we assign the values of α_1 and α_2 to two. On the other hand, a programming variable (feature #3) does not tend to be used in multiple pages as the complexity of the web application increases if it is used in multiple web pages. Thus, the low-skill site manager can relatively easily modify its name and the places where it is used. Also, an img element, i.e. feature #4, can be easily modified by a site manager because it is usually independent of other components in a web page. From this argument, we assign α_3 and α_4 as one.

Let us explain the implication of α_i values. If more number of the instances of the high-α_i features are in common between a target page being tested and a rule page, it is more highly probable that these two pages support the same functions than the case where those of low α_i features are in common because it is more difficult to customize the features of high α_i values preserving the functions of the original page.

TABLE II. FEATURES FOR CALCULATING SIMILARITY SCORE

Feature	Comments	Role
Function name description	The function name that is defined in JavaScript or VBScript	System
Program variable	The name property in form tags	
Link element	The link in <a> tags	UI
Img element	The image path in <img> tags	

TABLE III. CLASSIFICATION RULE FEATURES

	More relevant to vulnerability (β_i =1)	Less relevant to vulnerability (β_i =0)
Hard to modify (α_i =2)	Function name of script (feature type #1 (i=1))	Link element (feature type #2 (i=2))
Easy to modify (α_i =1)	Program variable (feature type #3 (i=3))	Img element (feature type #4 (i=4))

β_i reflects the relevance of each feature to vulnerability. Usually the features that are related to user interface (UI) tend to have less relevance to the vulnerability of a web page. Thus, the degree that the same feature instances exist in both pages does not tell about the probability of sharing the same vulnerability. Therefore, we set β_2 and β_4 to zero, whereas the β_1 and β_3 set to one, because the probability of sharing the same vulnerability decreases as more feature instances of system features (#1 and #2) are different in a target page and a rule page. Finally, we have reached the following algorithm to be used for calculating a similarity score with the α_i and β_i values of the features. In the proposed formula, we consider that a feature can have a lot of instances in a web page in general. For example, an HTML document has a lot of tag elements such as <img> tag and the HTML tag has internal attributes. In the case of the "ucp.php" file of phpBB, a HTML document, there are a number of <img> tags such as <img src="./imageset/n_homepage.gif"> and <img src="./imgeset/n_logo.gif">. We verify the following formula with the α_i and β_i values, which are shown in Table III, through Experiment 1 in subsection IV. C.

```
1:Similarity_Score = 0
2:for i = 1 to 4 do
3:   for j = 1 to n_i do
4:      Similarity_Score ← Similarity_Score
                          + f(x_ij)
5:    end for
6:end for
```

In the above algorithm and the below equation, x_{ij} refers jth feature instances of feature type number i in a rule page i

($\in \{1,2,3,4\}$), and n_i is the number of feature instances of type i in a rule page. $f(x_{ij})$ is defined based on whether the same feature instance $f(x_{ij})$ exists in the target page or not. Let's say that it is matched when the same x_{ij} exists in both a rule page and the target page we are checking on. Then, $f(x_{ij})$ can be defined as follows:

$$f(x_{ij}) = \begin{cases} \alpha_i \text{ (Matched case)} \\ -\alpha_i\beta_i \text{ (Unmatched case)} \end{cases} \quad (1)$$

The following Fig. 3 is the pseudo-code description of the algorithm for calculating a similarity score. The result is stored in *score* variable.

```
1:if Name of target file is existed in the
  rule DB then
2:  initialize that the variable score is 0
3:  for i in the set of feature-set do
4:      for j in each instance of rule file do
5:         if j exists in the target file then
6:                  score ← score + α_i
7:         else
8:                  score ← score - α_i β_i
9:         end if
10:     end for
11: end for
12:end if
```

Figure 3. Pseudo-code description of similarity scoring algorithm

C. Normalizing Similarity Score

If the number of feature instances grows, then the fully matched similarity score will be greater because the similarity score depends on the number of feature instances. A suitable normalization of score helps the generalized threshold setting. The normalized similarity score (N_{SS}) is defined as follows:

$$N_{SS} = \frac{(\text{matched score})}{(\text{fully matched score})} - \frac{(\text{unmatched score})}{(\text{fully unmatched score})} \quad (2)$$

The matched score means that is the sum of score when the feature instances are matched between rule file and target file. And the fully matched score is the sum of score when the all feature instances are existed in target file.

In contrast, the unmatched score only considers the unmatched feature instances. In the formula $f(x_{ij})$, if the instance is unmatched, then the result is $-\alpha_i\beta_i$. The unmatched score means the sum of $-\alpha_i\beta_i$s when the feature is not existed in target file. And the fully unmatched score is the sum of score when the all feature instances are unmatched with target file. After normalization, the similarity score has a value between 1 to -1.

IV. Evaluation

We conducted four experiments to assess the performance of the proposed approach, as follows.

A. Experiment 1 - Evaluation of Similarity Score

In our approach, the detection rate of vulnerability totally depends on the accuracy of similarity score. We verify that the similarity score can be used for classifying the different files in this experiment. We select five web pages from five real web sites. Table IV presents the files used in the experiment. There are various types of self-programmed application files and open-source programs in the web pages that we have chosen. We regard each file as a rule file. Then, we compute the similarity scores between the rule file to each of five files, respectively.

Table IV. Target Application List In Experiments

Index	Page name	App. Name	Comments
A	KF6S02Too F00-0.jsp	Korea Univ. Board (http://www.korea.ac.kr)	Self-programmed board
B	Zboard.php	zeroboard (www.xpressengine.com/)	Open source
C	Board.php	Picoboard [25]	Self-programmed board
D	Board.php	GnuBoard [26]	Open source
E	Category/122341	TT Tools & Open source	Open source

Fig. 4 is the result of measuring the similarity score. Each bar means the similarity score. The score between the same files is always a positive number, while it is usually a negative number as Fig. 4 in comparing between two different files. The maximum score varies depending on the number of feature instances in each file. We can conclude that our approach obviously identifies whether two web pages are the same or not regardless of the difference between the file names and/or the file paths installed because it checks the content of a web page.

B. Experiment 2 - Resiliency against installation path changing

The second experiment is designed to evaluate the detection performance of the proposed method and the previous methods on the condition that the installation path changes from the default installation path.

We analyzed the vulnerability on real web site using our system and Wikto, Nikto [2-3]. For legitimate experiment, we made a web site where there are a few web programs including vulnerable web page. The target web sites are composed as follows:

- z-board: to be installed to the default directory (/bbs)
- z-board: to be installed to the customized directory(/zb)
- g-board: to be installed to the default directory (/gnuboard4) [26]
- g-board: to be installed to the customized directory (/board) [26]
- AdminTool: to be installed to the default directory(/) [27]

There are five vulnerabilities in the web site. I.e., each component has vulnerability. The same applications have the same vulnerabilities because they are totally the same program except the places where they are installed. We suppose that we already get knowledge about vulnerabilities and generate the rules based on the knowledge. Table V presents the summery of rules. AdminTool is included in the comparison to verify that both the proposed scheme and the other methods have good detection ability. We insert the above rules to web scanner and analyze the vulnerability in target web site. Table V shows the result of our experiment.

We choose major rule-based tools, Wikto and Nikto in order to compare our tools.

As shown in Table VI, only our approach can identify the zboard2 and gboard2 vulnerabilities. Also, the existing tools may result in false positive in some cases due to this property. For example, if an application that has the same installed path and file name with vulnerable application exists, then existing tools misunderstand normal application. However, the proposed method does not make that kind of the problem.

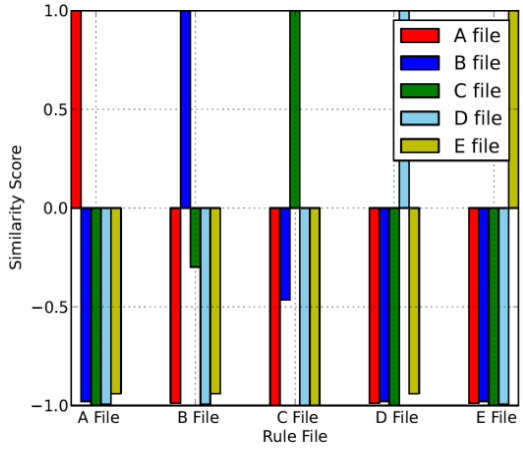

Figure 4. Similarity scores of target pages

TABLE V. SUMMARY OF RULE USED IN EXPERIMENT 2

Name	Path	Query	Response
Zboard	/bbs/zboard.php	Id=wave_default&page=1&desc=1	SQL Syntax error
Gboard	/gnuboard4/bbs/board.php	Bo_table=Vulnerability_test&wr_id=2	200 OK
Admin Tool	/admin.php	None	200 OK

TABLE VI. COMPARISON OF THE PROPOSED WORK WITH PREVIOUS WORK BASED ON EXPERIMENTS

Name	Wikto [3]	Nikto [2]	Ours
Zboard1	O	O	O
Zboard2	X	X	O
Gboard1	O	O	O
Gboard2	X	X	O
Admin tool	O	O	O

C. *Experiment 3 - Detection Rate*

The third experiment evaluates our approach by comparing it with Nikto in real web environment in terms of detection rate. For this experiment, we have collected the 110 web pages using a simple rule, one of the Nikto rule and Google search engine. We used the following rule defined in the rule DB of the Nikto web scanner:

db_tests(“002390”, ”3093”, ”b”, ”/forum/view-topic.php”, ”GET”, ”200”, ””, ””, ””, ””, ”phpBB found”,””,””)

The phpBB is composed of a lot of web program files. The “/forum/viewtopic.php" file is one of the phpBB web application. To check the above rule, the Nikto web scanner sends the HTTP request “/forum/viewtopic.php” to the target server using GET method, then it analyzes the response data. If the response includes the state 200, Nikto recognizes the target web server has vulnerability and alerts the "phpBB found" message to the site manager. On the other hand, the proposed approach checks the similarity score between the target page received and the rule page.

We have found 110 different web servers that provide the “/forum/viewtopic.php” URL in the Internet.

Then, we manually analyze the data set in all of these web pages in order to check if each web page is phpBB. The result is that 94 pages are phpBB and 16 pages are not. After this manual analysis, we executed the proposed method and Nikto with these 110 web pages.

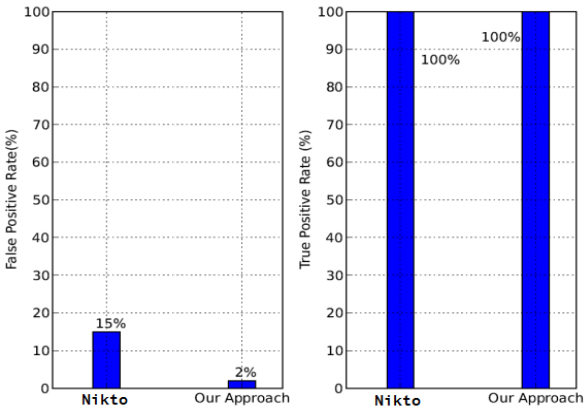

Figure 5. Comparison of false positive rate and true positive rate

Fig. 5 shows the result of this experiment. Nikto has 15% false alarm rate as it concludes all the pages are those in phpBB. This is due to the characteristics of rule-based web scanners, which just checks the filename and the installation path of the target web page, and the responses from given inputs in the rules. It does not check the actual content in it. On the other hand, the proposed scheme only produces 2% false alarm rate. Our method could not detect only the web pages where their HTML structs were customized by a site manager, as the proposed scheme investigates the content of the target web page before applying the rules. In terms of the true positive, both methods work well with 100% accuracy.

D. *Experiment 4 - A cost of vulnerability scan*

In this subsection, we evaluate the performance of our approach comparing with existing method. The proposed approach includes additional steps. It is expected to increase a cost for testing vulnerability. So we estimated whether a cost for testing vulnerability is reasonable.

As mentioned in the Fig. 1, the existing tool sends a request to the target server as much as the number of rules and analyzes each response. On the other hand, our approach collects the name of all pages in target web site after then uses names. So, it needs the time for analyze the site structure and calculating the similarity score.

We define a formula to calculate the cost of performance considering these different characteristics as follows: TCS: 0.036, RTT:0.016, TPW:0.018, and # of rules: 6400.

The number of web pages on web server is important factor in this experiment because a lot of web pages consume the much time in order to analyze the vulnerabilities. In this experiment, we suppose the worst case that the number of vulnerable web page is same with the number of all web pages on target server.

Fig. 6 shows the result of our experiment. The time cost had increased due to the number of web pages. Until approximately 1400 pages, our approach is more efficient than Nikto. Other than that, shows the opposite result. But, the Innowebsoft inc., professional web programming company (http://www.innowebsoft.com) said that general web server has between 30 ~ 60 web pages in their web site. In efficient aspect, our approach is reasonable to analyze small and medium-sized sites more than existing tools.

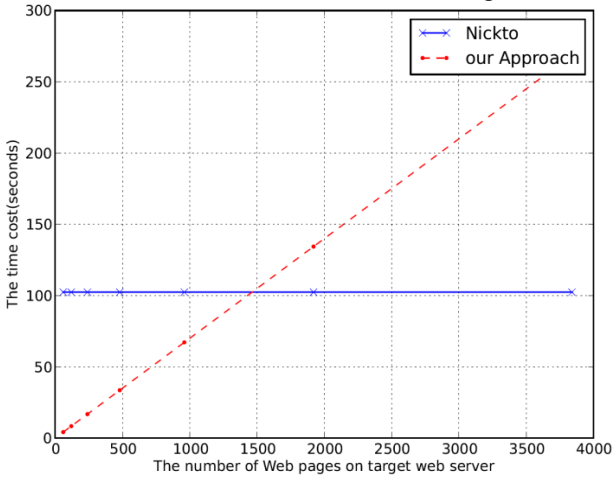

Figure 6. Cost comparison for vulnerability testing

V. Conclusion

In this paper, we proposed an enhanced web scanner in order to identify vulnerability on web site, based on similarity score. Through various experiments, we have shown that our web vulnerability scanner is superior to the existing well-known scanners. Specifically, we have demonstrated that the proposed scheme can detect vulnerabilities even if the web pages are not installed in the conventional paths, unlike the existing rule-based detection tools. The proposed scheme reduces the false-positive rate by more than 12% in a conventional detection case, with even less cost than the existing scanner when the number of web pages on target web server is above a certain threshold, which means that the proposed scheme can work well in practical situation, where the number of web pages is a lot.

References

[1] A. Mockus, R. T. Fielding, and J. Herbsleb, "A Case Study of Open Source Software Development: the Apache Server," Proc. ACM International Conference on Software Engineering, pp. 263-272, 2000. doi:10.1109/icse.2000.870417

[2] Y. C. Ong, and Z. Ismail. "Recent Advances in Information and Communication Technology", pp. 315-324, Springer International Publishing, 2014.

[3] S. Suganya, D. Rajthilak, and G. Gomathi, "Multi-Tier Web Security on Web Applications from Sql Attacks," IOSR Journal of Computer Engineering, vol. 16, no. 2, pp. 1-4, 2014. doi:10.9790/0661-16270104

[4] A. Doupé, M. Cova, and G. Vigna, "Detection of Intrusions and Malware, and Vulnerability Assessment", pp. 111-131, Springer Berlin Heidelberg, 2010.

[5] M. Vieira, N. Antunes, and H. Madeira, "Using Web Security Scanners to Detect Vulnerabilities in Web Services," Proc. IEEE/IFIP International Conference on Dependable Systems & Networks, pp. 566-571, 2009. doi:10.1109/dsn.2009.5270294

[6] K. Ma, R. Sun, and A. Abraham, "Toward a Lightweight Framework for Monitoring Public Clouds," Proc. Fourth IEEE International Conferences on Computational Aspects of Social Networks. pp. 361-365, 2012. doi:10.1109/cason.2012.6412429

[7] P. Davies, and T. Tryfonas, "A Lightweight Web-based Vulnerability Scanner for Small-scale Computer Network Security Assessment," Journal of Network and Computer Applications, vol. 32, no. 1, pp. 78-95, 2009. doi:10.1016/j.jnca.2008.04.007

[8] G. F. Lyon, "Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning", pp. 1-20, Insecure Press, 2009.

[9] S. Jajodia, S. Noel, and B. O'Berry, "Managing Cyber Threats", pp. 247-266, Springer US, 2005.

[10] R. J. Barnett, and B. Irwin, "Towards a Taxonomy of Network Scanning Techniques," Proc. the 2008 ACM annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology, pp. 1-7, 2008. doi:10.1145/1456659.1456660

[11] T. Yu, A. Sung, S. Witawas, and G. Rothermel, "An approach to testing commercial embedded systems," Journal of Systems and Software, vo. 88, no. 2, pp. 207-230, 2014. doi:10.1016/j.jss.2013.10.041

[12] A. A. Alfanookh, "An Automated Universal Server Level Solution for SQL Injection Security Flaw," Proc. International Conference on Electrical, Electronic and Computer Engineering, pp. 131-135, 2004. doi:10.1109/iceec.2004.1374401

[13] J. Chang, K. Venkatasubramanian, A. West, and I. Lee, "Analyzing and Defending against Web-based Malware," ACM Computing Surveys, vol. 45, no. 4, article no. 49, 2013. doi:10.1145/2501654.2501663

[14] N. Khocharre, S. Chalurkar, and B. Meshram, "Web Application Vulnerabilities Detection Techniques Survey," International Journal of Computer Science & Network Security, vol. 13, no. 6, pp. 71-75, 2013. doi:10.1016/b978-1-59749-209-6.00002-3

[15] M. Vieira, N. Antunes, and H. Madeira, "Using Web Security Scanners to Detect Vulnerabilities in Web Services," Proc. IEEE/IFIP Conference on Dependable Systems and Networks, pp. 566-571, Jun. 2009. doi:10.1109/dsn.2009.5270294

[16] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the Art: Automated Black-Box Web Application Vulnerability Testing," Proc. IEEE Symposium on Security and Privacy, pp.332-345, 2010. doi:10.1109/sp.2010.27

[17] N. Antunes, and M. Vieira, "Benchmarking Vulnerability Detection Tools for Web Services," Proc. IEEE International Conference on Web Services, pp. 203-210, 2010. doi:10.1109/icws.2010.76

[18] Z. Duric, "WAPTT-Web Application Penetration Testing Tool," Advances in Electrical and Computer Engineering, vol. 14, no. 1, pp. 93-102, 2014. doi:10.4316/aece.2014.01015

[19] Y. Yun, S. Park, Y. Kim, and J. Ryou, "Information Security Practice and Experience", pp 248-259, Springer Berlin Heidelberg, 2006.

[20] S. Kals, C. Kruegel, and N. Jovanovic, "SecuBat: A Web Vulnerability Scanner", Proc. the 15th International Conference on World Wide Web, pp. 247-256, 2006. doi:10.1145/1135777.1135817

[21] A. Austin, and L. Williams, "One Technique is not Enough: A Comparison of Vulnerability Discovery Techniques," Proc. IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 97-106, 2011. doi:10.1109/esem.2011.18

[22] V. D. Kotov, and V. I. Vasilyev, "Detection of Web Server Attacks using Principles of Immunocomputing," Proc. 2nd World Congress on Nature and Biologically Inspired Computing, pp. 25- 30, 2010. doi:10.1109/nabic.2010.5716269

[23] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, "State of the art: Automated Black-box Web Application Vulnerability Testing," Proc. IEEE Symposium on Security and Privacy (SP), pp. 332-345, 2010. doi:10.1109/sp.2010.27

[24] S. Stefanov, "Building Online Communities with phpBB", pp. 5-15, Packt Publishing, 2005.

[25] M. Choi, H. Ju, H. Cha, S. Kim, and J. Hong, "An Efficient Embedded Web Server for Web-based Network Element Management," Proc. IEEE/IFIP Network Operations and Management Symposium, pp. 187-200, 2000. doi:10.1002/1099-1190

[26] J. Hong, M. Chung, and H. Choo, "Novel Bulletin Board System based on Document Object Model and Client-side Scripting for Improved Interaction," Proc. IEEE International Conferences on Information Networking (ICOIN), pp. 511-516, 2013. doi:10.1109/noms.2000.830384

[27] Å. Blomquist, and M. Arvola, "Personas in Action: Ethnography in an Interaction Design Team," Proc. 2nd ACM Nordic conference on Human-computer interaction, pp. 197-200, 2002. doi:10.1145/572043.57204