Mobile Information Systems 8 (2012) 275–286
DOI 10.3233/MIS-2012-0143
IOS Press

Detection of cross site scripting attack in wireless networks using n-Gram and SVM[1]

Jun-Ho Choi, Chang Choi, Byeong-Kyu Ko and Pan-Koo Kim*
Department of Computer Engineering, Chosun University, Gwangju, Korea

Abstract. Large parts of attacks targeting the web are aiming at the weak point of web application. Even though SQL injection, which is the form of XSS (Cross Site Scripting) attacks, is not a threat to the system to operate the web site, it is very critical to the places that deal with the important information because sensitive information can be obtained and falsified. In this paper, the method to detect themalicious SQL injection script code which is the typical XSS attack using n-Gram indexing and SVM (Support Vector Machine) is proposed. In order to test the proposed method, the test was conducted after classifying each data set as normal code and malicious code, and the malicious script code was detected by applying index term generated by n-Gram and data set generated by code dictionary to SVM classifier. As a result, when the malicious script code detection was conducted using n-Gram index term and SVM, the superior performance could be identified in detecting malicious script and the more improved results than existing methods could be seen in the malicious script code detection recall.

Keywords: Malicious code, SQL injection attack, n-Gram, SVM

1. Introduction

With the development of wireless network and internet, many parts of offline services have been converted into online services and currently most parts of online service are occupied by web services. Due to the merit of being available in anytime and anywhere, the importance of the web has been increased more and more everyday and the attacks aiming it has also been increased. Large parts of attacks targeting the web are aiming at the weak point of web application and SQL injection attack which is the form of XSS (Cross Site Scripting) attacks is not a threat to the system that uses or operates the web applications compared to other attacks, but it is very critical to the places that deal with the important information because sensitive information can be obtained and falsified. Various techniques have been studied in different fields to detect and prevent this critical SQL injection attack, including typically web framework, static and dynamic analysis and method using machine learning [7].

The web framework in a wireless network provides the filtering methods for input values but it is only filtering the special characters entered, so there are many obfuscate techniques using XSS. The static analysis analyzes the types of user input so it is more effective than simple filtering method, but it has the disadvantage that the attacks matching with the input type can't be detected. The dynamic analysis can find the weak point without modifying the web application but it can't find all weak points. The static and dynamic analysis method which complements the disadvantages of both static analysis and dynamic

*Corresponding author: Department of Computer Engineering, Chosun University, Gwangju, Korea. Pan-Koo Kim, Tel.: +82 62 230 7636; Fax: +82 62 230 7636; E-mail: pkkim@chosun.ac.kr.
[1]This paper extends our previous work [16] published on Internationalconference of NBiS in 2011.

analysis is effective in detecting SQL injection attacks, but it has disadvantage that it is complicate in using because it is used by mixing both static analysis and dynamic analysis. The method using machine learning has the advantage of being able to detect unknown attacks but the misuse detection (False-Negative and False-Positive) can occur [17,18].

In this paper, it was determined whether the malicious script code could be detected or not using SVM (Support Vector Machine), which provides efficiency and accuracy in the process of a binary classification and to increase the accuracy of binary pattern classification, the index term applying n-Gram and code dictionary were generated. In the generated code dictionary the binary values obtained from matching with lexical grammar in the malicious script code were applied to the input node of SVM.

This paper is organized as follows: In Section 2, the related studies for the malicious script code classification was explained, in Section 3, XSS attack script pattern learning using SVM was described, and in Section 4, the study results using SVM were explained. Finally, in Section 5, the conclusion and future study direction were suggested.

2. Related work

2.1. SQL injection attack

A large number of applications that use database including the web application makes SQL queries using the user input values. For user login, the web application makes SQL queries for user account and password to check whether the user enters the effective account and password. At this time, the normal action can be interfered by sending fabricated username and password to change the normal SQL queries through SQL input attack technique [21].

In situations that the queries for database are generated dynamically, the input values of the user are treated as a meaningful statement in database, but the input values are treated as a simple string in the web application. Such attack method is called SQL injection attack.

Table 1
An example of the weak point of SQL input attack

```
$user_name = $_POST[‘username’]
$passwd = $_POST[‘password’]
Mysql_query(“SELECT COUNT(*) FROM Users
            WHERE username = ‘$user_name’ AND password = ‘$passwd’ ”);
```

Table 1 is an example of the weak point of SQL input attack that can be seen in the web application. In this example, $user_name and $Passwd are where the user enters. At this time, if the attacker enters the statement in $user_name as below, the SQL statement like Table 2 is created.

Table 2
A generated SQL statement by SQL input attack

```
SELECT COUNT(*) FROM Users
WHERE username = “or 1 = 1 –’ AND password = ‘’
```

“ ‘or 1 = 1 –”

In this case, the WHERE clause will be always true for "OR $1 = 1$". That is, the attacker will obfuscate the user authentication using SQL injection attack. These vulnerabilities of SQL injection attack of the above Web application have been found in many websites in reality. If unsafe input information from the outside is used in the sensitive tasks such as execution of the script or SQL statement, the vulnerability of the Web application is exposed.

SQL injection attack converts SQL query syntax fixed into the existing web application into a new malicious SQL query syntax using malicious data values and then it requests and handles the data in database abnormally [12]. To prevent such SQL injection attacks, the web developers are basically using filtering method of input data value, but there are lots of obfuscate methods so it is difficult to prevent SQL injection only with simple filtering method. Therefore, it is necessary to have more advanced SQL injection attack detection and prevention method than simple filtering method.

2.2. SQL injection attack detection method

2.2.1. Method using static and dynamic analysis

The mixed method of static and dynamic analysis analyzes SQL query statically in the Web application and compares and analyzes the dynamic SQL query generated from the outside.

SQLCheck defines SQL injection attack and proposes *sound and complete algorithm* based on Context-free grammars and Complier parsing techniques [27]. There are advantages of SQLCheck in that there is no misuses detection (false-negative and false-positive) and the source code can be modified and applied to the web application directly. AMNESIA finds hotspots that executes SQL queries in the web application and then it creates every possible SQL queries [11]. The static SQL query generated and dynamic SQL query received from the user are classified and analyzed using JSA Library.

Buehrer uses a parse tree that is used a lot in lexical analysis so it has advantage that SQL query syntax can be analyzed correctly [1]. However, since SQL query syntax is different for each DBMS, it has the disadvantage of being dependent on DBMS. Wei proposed the method to detect the SQL injection attack by comparing and analyzing the static SQL query which is the stored procedure in the Web application and dynamic SQL query generated dynamically by making *control flow graph* [26]. Different from detecting and preventing method of SQL injection attack at the CGI-tier, this method can generaly detect and prevent the SQL injection attack at DBMS and stored procedure.

2.2.2. Method using SQL query profiling

Park et al. [15] detected the SQL injection attack by comparing and analyzing the dynamic SQL query generated dynamically, profiling the SQL query of the web application using "Pairwise sequence alignment of amino acid code formulated" method. This method can detect SQL injection attacks without modifying the web application, but it is inconvenient in that whenever the web application is changed, it should be profiling again.

2.2.3. Method using machine learning

Valeur proposed the Intrusion Detection System using machine learning [10]. This method learns the SQL query that generates from the web application, creates the detection model, and identifies the SQL injection attack by checking the SQL query generated in real time is the same as training model. This method can effectively detect and prevent the unknown attacks such as Zero-day attack. However, if insufficient training data set is used, the misuses detection (false-negative and false-positive) can occur.

WAVES finds the weak point in the web application through web crawler and it forms the attack code based on the pattern list and attack techniques [12]. The weak point of SQL injection attack is found using formed attack code.

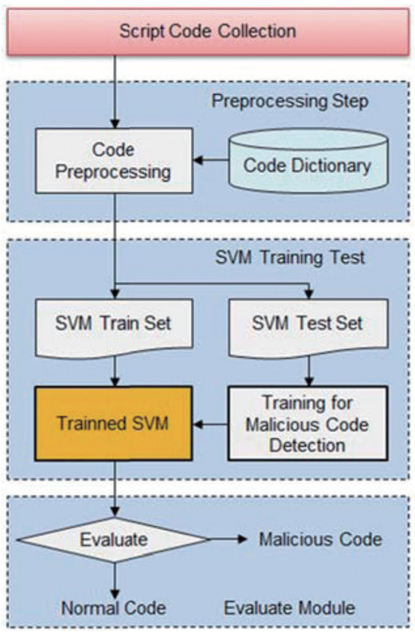

Fig. 1. A malicious script code detecting process.

3. XSS attack script pattern training using SVM

In case of existing malicious script code detection, the filtering is performed through pattern matching of specific words, but if the modified type of malicious script code is transmitted, this can't perform the filtering effectively and as the number of pattern is increased, it has a problem of increasing the time for filtering the malicious script code proportionally.

To solve such problems, in this paper the data set obtained through matching the malicious code pattern generated using n-Gram and code dictionary were applied to SVM classifier and the detection of malicious script was performed efficiently. N-Gram indexing and the malicious script filtering using SVM perform the following steps.

3.1. Entire configuration of mobile malicious script code detection method

To experiment the method proposed in this paper, after collecting the normal script code and malicious script code, the code dictionary and n-Gram index term were generated from collected code. Code dictionary was created manually by applying the frequency of words and the index term was created automatically by applying n-Gram for vocabulary in the script. In here, the training data set and test data set were formed by matching the generated index term and code dictionary.

The training data set generated from preprocessing step performs the training process of SVM classifier, and the test data set determines whether it is the normal script code or malicious script code by entering test pattern for malicious code. The entire configuration module for the malicious script code detecting using n-Gram index term and SVM is shown in Fig. 1.

3.2. Malicious script code pattern generation

The code dictionary is generated by extracting m number of high-frequency words, examining the frequency of lexical grammar in the code. In the generated code dictionary, the lexical grammar related

Table 3
An example of code dictionary

Code indexing	1	2	3	4	...	n
Code dictionary	SELECT	UPDATE	INSERT	Login	...	DELETE
Code indexing	201	202	203	204	...	m
Code dictionary	password	AND	username	account	...	Param1

Table 4
An example of code dictionary using n-Gram

Query	Unigrams	Bigrams	Trigrams
A	SELECT	SELECT_LastName	SELECT_LastName_FROM
	LastName	LastName_FROM	LastName_FROM_users
	FROM	FROM_users	FROM_users_WHERE
	users	users_WHERE	users_WHERE_UserID
	WHERE	WHERE_UserID	WHERE_UserID_1
	UserID	UserID_1	
	1		
B	SELECT	SELECT_FROM	SELECT_FROM_users
	FROM	FROM_users	FROM_users_WHERE
	users	users_WHERE	users_WHERE_login
	WHERE	WHERE_login	WHERE_login_'victor'
	login	login_'victor'	login_'victor'_AND
	'victor'	'victor'_AND	'victor'_AND_password
	AND	AND_password	AND_password_'123'
	password	password_'123'	
	'123'		

to malicious code is set. The index of code dictionary is composed of m number of words that match with the number of SVM input nodes and it is used for matching the generated index terms and words defined in code dictionary. Table 3 shows an example of generated code dictionary.

3.3. *N-Gram index term generation*

N-Gram is applied to the first collected codes to generate the training data set or test data set for collected codes, and the index term is generated by applying bigram to 4-gram using collected codes as shown in Table 4. The training data set or test data set is formed by matching index terms generated from this and code dictionary generated from Table 3.

– Query A: SELECT LastName FROM users WHERE UserID = 1
– Query B: SELECT * FROM users WHERE login = 'victor' AND password = '123'

3.4. *SVM dataset generation*

Figure 2 is the step that applies the index term generated by applying n-Gram and code dictionary. If the index term applying n-Gram and dictionary word are matched, it will have the value 1 (matching) as shown in Table 5 but if it can't find the matching words in code dictionary, it will have the value 0 (mismatching).

Table 5 shows the result obtained from matching the index term generated by applying n-Gram with code dictionary. After forming the training data set or test data set using matching result values as shown in Table 5, it is used as input vector of SVM. At this time the training step is performed using SVM classifier or the malicious script code is filtering using test data set.

Table 5
A matching result of code dictionary and n-Gramindexing

Indexing of code dictionary	1	2	3	4	5	......	200	......
Code set (1)	1	0	1	1	0	......	1	......
Code set (2)	1	1	0	1	0	......	0	......
Code set (n-1)					....			
Code set (n)	0	1	1	0	1	......	0	......

Table 6
A SVM training data set in experiment

Data SetCode type	Training set (Total 500)
Normal code	Normal code (250)
Malicious code	SQL injection related malicious code (250)

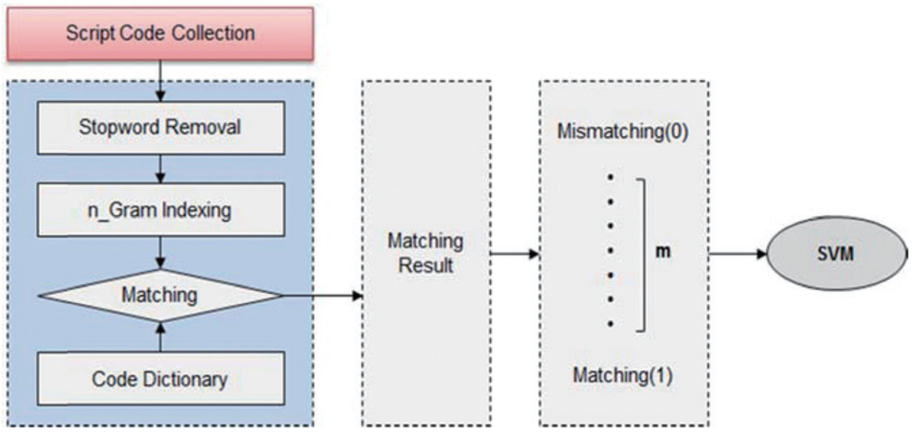

Fig. 2. A preprocessing for applying n-Gram indexing and SVM.

4. Experiments and results of XSS attack detection using SVM

4.1. Experiment environment of XSS attack detection

For XSS attack detection experiment, SQL injection attack code data among XSS attacks was collected, and with the collected sample, the training data set and test data set for SVM application were formed through preprocessing step explained in Section 3.

Each data set is composed of normal code and malicious code, and SQL injection related code from collected data set is classified as a malicious code and the rest is classified as normal code. Table 6 shows the SVM training data set for SQL injection malicious code filtering, and the number of training data set consists of 500. Among 500 training data, the normal code and malicious script code consists of 250, respectively.

The entire configuration for XSS attack detection proposed in this paper is shown in Fig. 3 and the experiment was performed using libSVM library [6] for spam mail filtering and the experiment results according to libSVM parameters were compared.

4.2. Results

The most efficient classification method of mobile malicious script code was studied and the performance for each classification method was measured. Total of 500 script codes were collected to perform

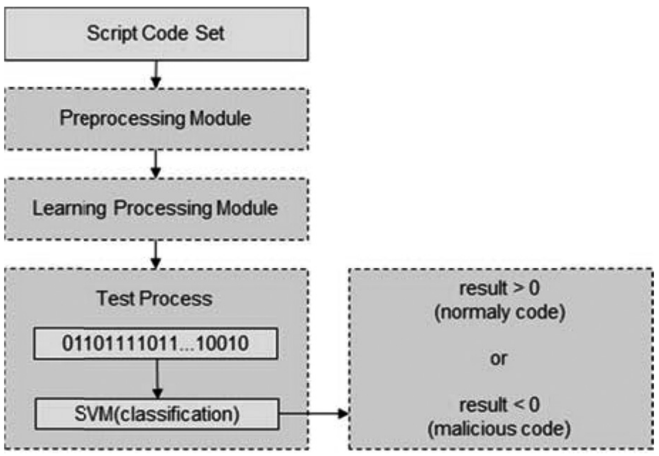

Fig. 3. A processing of malicious script code detecting using SVM.

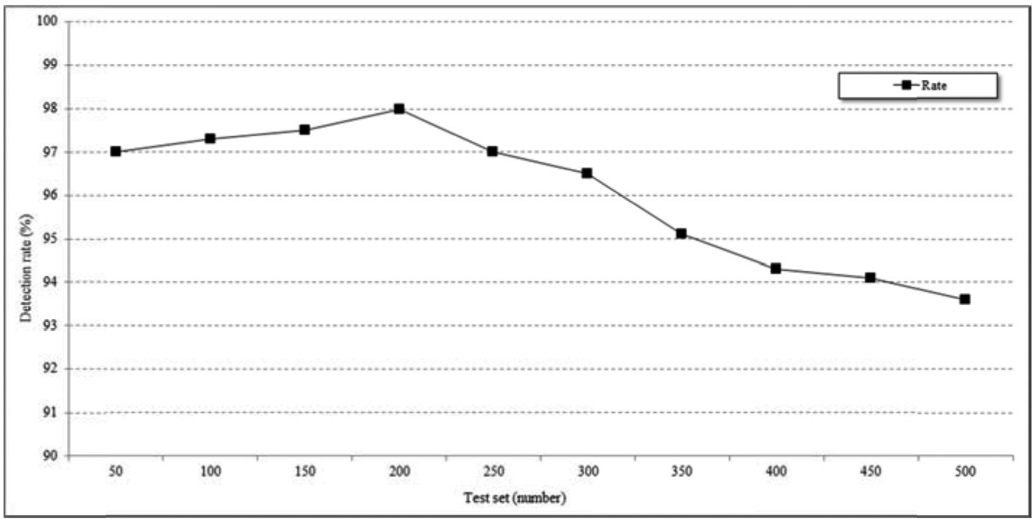

Fig. 4. A result of applying dot kernel.

the test, including 250 script codes were set as a training data set for SVM learning and 250 script codes were used as a test data set. In here, the normal script code and malicious script code were composed as 50% ratio, respectively.

The types of kernel used in test environment were Dot, Polynomial and Radial, and these three kernel functions were used to perform test. The performance of malicious script code detection according to kernel function is as follows.

4.2.1. Results of applying dot kernel

Dot kernel was used as a first test kernel function and m number of nodes which were matched with input node as the input node of SVM. The detection rate of malicious script code according to test data set increase is shown in Fig. 4.

The important characteristics according to the result are that as the number of test set was over 300, the normal malicious script code detection ratio decreased, while the false positive ratio and false negative

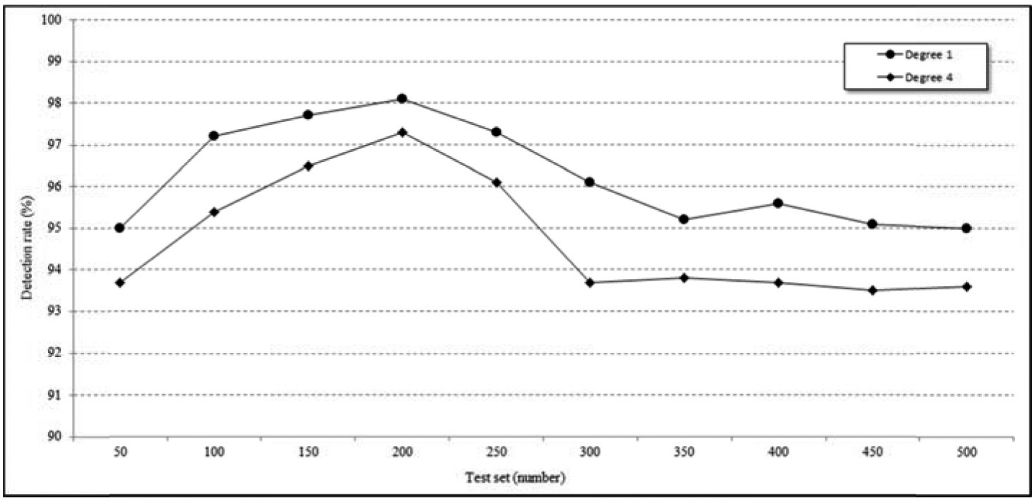

Fig. 5. A result of applying polynomial kernel.

ratio were increased. It is because there are many cases that the grammar of SQL syntax and diversity of variables in some script code are defined differently from the trained result through SVM.

4.2.2. Results of applying polynomial kernel

Polynomial kernel includes the parameter values and the test can be performed by adjusting the parameter values as shown in Eq. (1). The parameter has integer value and the test was performed by adjusting the degree value of parameter.

$$k(x,y) = (x * y + 1)^d \tag{1}$$

In the test using Polynomial kernel, it showed the similar false-positive ratio and false-negative ratio to Dot kernel method, but the malicious script code detection ratio showed more superior test results and the entire test result using Polynomial kernel is shown in Fig. 5. As shown in the test results of Fig. 5, when the degree value of parameter was set to 1, it had slightly better results than set to 4, and when the degree value was set to more than 4, the result value was impossible to derive due to the increase of false-positive and false negative ratios.

4.2.3. Results of applying radial kernel

As a last applied kernel function, Radial was used and the kernel function is defined as shown in Eq. (2). The parameter has real number value and the test was performed by adjusting the gamma value of parameter.

$$K(x,y) = \exp\left(-\frac{|x-y|^2}{\delta^2}\right) \tag{2}$$

In the test using Radial kernel, it showed the similar false-positive and false-negative ratio to the test result using Dot or Polynomial kernel method, but it showed the most excellent performance in the result of entire malicious script code detection.

The most important characteristics in the method using Radial kernel, the false-negative ratio was the best when the gamma value was set to 0.2 but it had a problem of increasing false-positive ratio relatively.

Table 7
Experimental resultsbyapplying each kernel function and arameter

Kernel definition	Feature	Parameter	Test set (500)			
			FP	FN	Detection recall	Detection precision
Dot Kernel	127	N/A	1.3	3.5	95.8	96.8
Polynomial	127	degree1	1.3	3.2	93.1	96.8
Kernel		degree 4	1.2	3.6	94.8	97.0
Radial	127	gamma0.2	2.1	1.1	96.4	97.7
Kernel		gamma0.6	1.9	1.1	96.4	97.5

– FP: False positive, FN: False negative.
– Detection precision = malicious code/classified malicious code.
– Detectionrecall = malicious code/total malicious code.

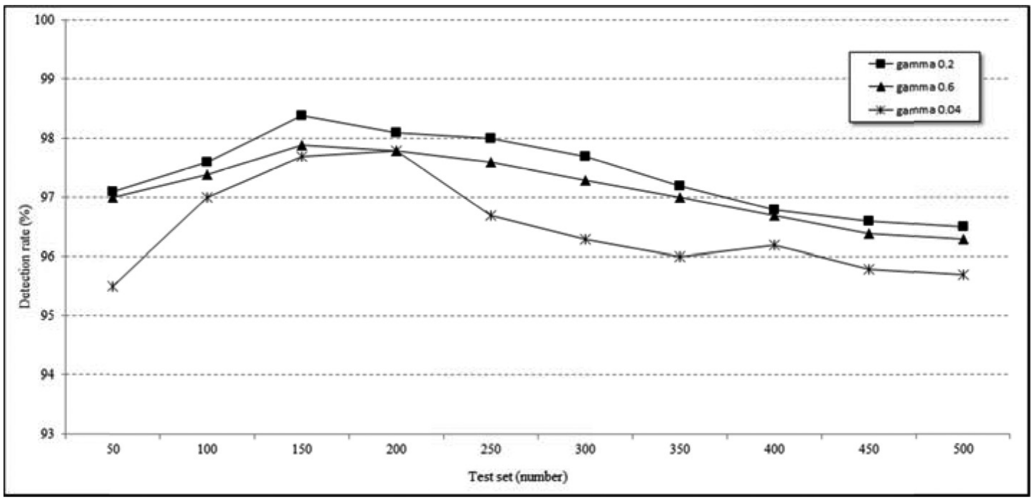

Fig. 6. A result of applying radial kernel.

In addition, when the gamma value was set to 0.01, the malicious script code detection ratio had the similar result to the malicious script code detection ratio using Dot or Polynomial kernel method, but it showed slightly less performance by comparing with different gamma values. The test results according to each kernel function and parameter are shown in Table 7.

The performance evaluation of malicious script code detection is expressed as detection precision and detection recall. As a result, the method applying Radial kernel as shown in Table 7 showed more excellent performance than Dot or Polynomial kernel in the test results of malicious script codedetection, especially when gamma was set to 0.2, the most excellent performance of malicious script code detection was shown.

4.3. *Performance comparison of malicious script code detection*

In order to compare the performance between the method proposed in this paper and existing detection methods of malicious script, Naïve Bayesian approach and keyword pattern were used as test filters for malicious script code detection. The method proposed in this paper collected 500 script codes and by having 65% and 35% composition of normal script and malicious script code ratio, the malicious script code detection test was performed.

Table 8
A result of malicious script code detection test

Filter used	Parameter	False Positive (%)	False negative (%)	Detection recall (%)	Detection precision (%)
SVM	gamma0.2	1.7	0.9	96.3.9	97.2
Keyword patterns	None	None	None	60.2	94.8
Naïve Bayesian	None	4.4	6.9	94.4	92.6

The malicious script code detection precision and malicious script code detection recall were used as the performance evaluation method for proposed method and the existing methods, and the method of applying n-Gram index term and SVM showed better performance than the method using keyword pattern in the malicious script detection precision and it had similar results to the Naïve Bayesian method. In addition, the malicious script code detection recall showed more excellent filtering performance than methods using Naïve Bayesian and keyword pattern and even though there is a slight difference according to the applied kernel function to the malicious script code vusing proposed method, it took average of about 1.5 seconds.

When the malicious script code detection was conducted using n-Gram index term and SVM as above, the superior performance could be identified in detecting malicious script code and the more improved results than existing methods could be seen in the malicious script code detection recall.

5. Conclusion

In this paper, the method to detect the SQL injection malicious script code which is the typical XSS attack using n-Gram indexing and SVM (Support Vector Machine) was proposed. In order to test the proposed method, the test was conducted after classifying each data set as normal code and malicious code, and the malicious script code was detected by applying index term generated by n-Gram and data set generated by code dictionary to SVM classifier. For kernel functions used in SVM classifier, Dot, Polynomial and Radial methods were used and the test results for each kernel method were explained in Table 7.

When the radial method was used as a kernel function of SVM classifier, it showed the most excellent performance, and comparing the existing studies such as Naive Bayesian methods showed the improved performance in the malicious script code detection. However, the codes containing a lot of SQL syntax has the problem of decreasing detection performance as the ratio of vectors that is out of the trained results values by SVM classifier increases. The solution for this can be resolved through pattern definition of SQL syntax in the preprocessing step. In this study, the malicious script code detection was performed limited to SQL injection attack which is the representative malicious script. However, it is necessary to detect various types of script-based malicious codes in the future.

Acknowledgments

This research was financially supported by the Ministry of Education, Science Technology (MEST) and National Research Foundation of Korea(NRF) through the Human Resource Training Project for Regional Innovation.

References

[1] G. Buehere, B.W. Weide and P.A. Sivilotti, Using Parse Validation to Prevent SQL Injection Attacks, InProceedings of the 5th international Workshop on software Engineering and Middleware, 2005, pp. 105–133.

[2] B.-Y. Zhang, J.-P. Yin, J.-B. Hao, D.-X. Zhang and S.-L. Wang, Using SupportVector Machine to Detect Unknown Computer Viruses, *International Journal ofComputational Intelligence Research* **2**(1) (2006), 100–104.

[3] B. Zhang, J. Yin and J. Hao, Intelligent Detection Computer Viruses Based onMultiple Classifiers, *Ubiquitous Intelligence and Computing* **4611** (2007), 1181–1190.

[4] C. Fung, Collaborative Intrusion Detection Networks and Insider Attacks, *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*(*JoWUA*) **2**(1) (2011), 63–74.

[5] C.-L. Chen, Design of a secure RFID authentication scheme preceding market transaction, *Journal ofMobile Information Systems* **7**(3) (2011), 201–216.

[6] C.C. Chang and C.J. Lin, LIBSVM: a library for support vector machines. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm, 2001.

[7] Cisco Systems, What Is the Difference: Viruses, Worms, Trojans, and Bots, 2010.

[8] D. Moon, B. Park, Y. Chung and J.-W. Park, Recovery of flash memories for reliable mobile storages, *Journal of Mobile Information Systems* **6**(2) (2010), 177–191.

[9] F. Palmieri, U. Fiore and A. Castiglione, Automatic security assessment for next generation wireless mobile networks, *Journal of Mobile Information Systems* **7**(3) (2011), 217–239.

[10] F. Valeur, D. Mutz and G. Vigna, A Learning-Based Approach to the Detection of SQL Attacks, Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment, 2005, pp. 123–140.

[11] W.G. Halfond and A. Orso, AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks, Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering, 2005, pp. 174–183.

[12] Y. Huang, S. Huang, T. Lin and C. Tasi, Web application security assessment by fault injection and behavior monitoring, Proceedings of the 12th international Conference on World Wide Web, 2003, pp. 148–159.

[13] I. You, Y. Hori and K. Sakurai, Enhancing SVO Logic for Mobile IPv6 Security Protocols, Journal of Wireless Mobilie Networks, *Ubiquitous Computing and Dependable Applications* **2**(3) (2011), 26–52.

[14] A. Jacob and M. Gokhale, Language classification using n-grams accelerated by fpga-based bloom, 2007.

[15] J. Park and B. Noh, SQL injection Attack Detection: Profiling of Web Application Parameter Using the Sequence Pairwise Alignment, *Information Security Applications LNCS* **4298** (2007), 74–82.

[16] J. Choi, H. Kim, C. Choi and P. Kim, Efficient Malicious Code Detection Using N-Gram Analysis and SVM, *International Conference of Network-Based Information Systems* (*NBiS*) (2011), 618–621.

[17] J. Kolter and M. Maloof, Learning to Detect and Classify Malicious Executables inthe Wild, *Journal of Machine Learning Research* **7** (2006), 2721–2744.

[18] M.G. Schultz, E. Eskin, E. Zadok and S.J. Stolfo, Data Mining Methods for Detectionof New Malicious Executables, Proc. of the 2001 IEEE Symposium on Security andPrivacy, CA: IEEE Computer Society Press, Oakland, 2001, pp. 38–49.

[19] M.P. Jayamsakthi Shanmugam, Cross site scripting-latestdevelopments and solutions: A survey, *International Journal of OpenProblems in Computer Science and Mathematics* (*IJOPCM*) **1**(2) (2008), 8–28.

[20] P. Nobles, S. Ali and H. Chivers, Improved Estimation of Trilateration Distances for Indoor Wireless Intrusion Detection, Journal of Wireless Mobile Networks, *Ubiquitous Computing, and Dependable Applications* (*JoWUA*) **2**(1) (2011), 93–102.

[21] W. Robertson, G. Vigna, C. Kruegel and R. Kemmerer, Using generalization andcharacterization techniques in the anomaly-based detection of web attacks. In: NDSS '06: Proc. 13th ISOC Symposium on Network and Distributed Systems Security, 2006.

[22] S. Caballé, F. Xhafa and L. Barolli, Using mobile devices to support online collaborative learning, *Journal of Mobile Information Systems* **6**(1) (2010), 27–47.

[23] M. Shafiq, S. Khayam and M. Farooq, Embedded Malware Detection Using Markov n-Grams, *Lecture Notes in Computer Science* **5137** (2008), 88–107.

[24] T.A. Zia and A.Y. Zomaya, A Lightweight Security Framework for Wireless Sensor Networks, Journal of Wireless Mobilie Networks, *Ubiquitous Computing and Dependable Applications* **2**(3) (2011), 53–73.

[25] V. Hamine and P. Helman, A Theoretical and Experimental Evaluation ofAugmented Bayesian Classifiers, American Association for Artificial Intelligence, Retrieved March 20, 2006.

[26] K. Wei, M. Muthuprasanna and S. Kothari, preventing SQL injections attacks in stored procedures, Software Engineering, 2006, pp. 18–21.

[27] Z. Su and G. Wassermann, The Essence of command Injection Attacks in Web Applications, In Conference Record of the 33er ACM SIGPLAN- SIGACT Symposium on Principles of Programming Languages, 2006, pp. 372–382.

Jun-Ho Choi received a doctoral degree in the Department of Computer Engineering at Chosun University of Korea in 2004. Currently, he is working as a lecturer at the same university. His research interests include multimedia processing, semantic information processing, ontology and semantic web.

Chang Choi is a Ph.D. Candidate in the Department of Computer Engineering at Chosun University of Korea. Currently, he is working as a lecturer at the same university and toward the Ph. D degree. His research interests include semantic information processing, semantic web and multimedia data processing.

Byeong-Kyu Ko is a student for the doctoral degree in the Department of Computer Engineering at Chosun University of Korea. He is received a master degree at the same university in 2012. His research interests include web documents classification, semantic information processing and semantic web.

Pan-KooKim is received the BS degree in computer engineering from Chosun University of Korea and the MS and PhD degrees in computer engineering from Seoul National University of Korea in 1994. He is a full professor in the Department of Computer Engineering at Chosun University. His specific research interests include semantic web techniques, semantic information processing and retrieval, multimedia processing and semantic web.