Journal of Engineering and Applied Technology
Vol. 1, No. 2, August 2020, pp. 85-96

E-ISSN: 2716-2265
P-ISSN: 2716-2257

Hybrid method integrating SQL-IF and Naïve Bayes for SQL injection attack avoidance

Faisal Yudo Hernawan[1*], Indra Hidayatulloh[2], Ipam Fuaddina Adam[1]

[1]Study Program of Informatics, Faculty of Industrial and Informatics Technology, Telkom Purwokerto Institute of Technology, Indonesia
[2]Department of Electronics and Informatics Engineering Education, Faculty of Engineering, Universitas Negeri Yogyakarta, Yogyakarta, Indonesia

*E-mail: faisalyudo@gmail.com

* corresponding author

ABSTRACT

Web applications are the objects most targeted by attackers. The technique most often used to attack web applications is SQL injection. This attack is categorized as dangerous because it can be used to illegally retrieve, modify, delete data, and even take over databases and web applications. To prevent SQL injection attacks from being executed by the database, a system that can identify attack patterns and can learn to detect new patterns from various attack patterns that have occurred is required. This study aims to build a system that acts as a proxy to prevent SQL injection attacks using the Hybrid Method which is a combination of SQL Injection Free Secure (SQL-IF) and Naïve Bayes methods. Tests were carried out to determine the level of accuracy, the effect of constants (K) on SQL-IF, and the number of datasets on Naïve Bayes on the accuracy and efficiency (average load time) of web pages. The test results showed that the Hybrid Method can improve the accuracy of SQL injection attack prevention. Smaller K values and larger dataset will produce better accuracy. The Hybrid Method produces a longer average web page load time than using only the SQL-IF or Naïve Bayes methods.

ARTICLE INFO

Article history
Received:
4 November 2020
Revised:
13 January 2021
Accepted:
21 January 2021

Keywords
SQL injection
SQL injection free secure
Naïve baye
Web application database

1. Introduction

Web applications are applications that are stored and executed by a web server. There are two types of web applications, namely static web (without database) and dynamic (requiring database) for dynamically changing data. The more popular web applications are, the greater the potential for being targeted by attackers [1]. This is because popular web applications usually store important information such as personal data, passwords, credit cards, and other data that can be used by attackers. The 2017 Breach Investigation Report (DBIR) shows that the number of attacks on web applications is in the first rank or the most occurring [2].

Based on data from the Open Web Application Security Project (OWASP) 2017, the SQL injection technique ranks first most often used by attackers to break into a web illegally [3]. SQL injection is a type of injection attack to a web application where the attacker can execute malicious SQL queries which result in the attacker being able to view, modify, delete, and retrieve data, and even take over the database [4]. SQL injection has six types of attacks, namely tautology, piggy-backed queries, union queries, logically incorrect queries, inferences, and stored procedure injection [5]. SQL injection attacks exploit vulnerabilities from inadequate input validation on existing forms or parameters in web applications [6]. According to Veracode in the report of State of Software Security Volume 6, Classic ASP programming language; ColdFusion; and PHP; occupies the top three in terms of vulnerability to SQL injection attacks [7].

Several studies used pattern matching to detect and prevent SQL injection attacks. Prabakar et al. [8] used a pattern matching algorithm where the injection parameters are matched with the attack pattern. Saleh et al. [9] developed a web application vulnerability detection method that includes SQL injection; XSS; buffer overflow; and CSRF using the Boyer-Moore String Matching Algorithm. The algorithm scans character by character injection parameters from right to left to find matches with attack patterns that have been registered in the system. Ellysa et al. [10] used the SQL Injection Free Secure (SQL-IF) method by checking the SQL keyword, unique characters, and Boolean characters in the input data. Research [10] has not been tested if the injection parameters are encoded.

In addition, several other studies used machine learning to detect and prevent SQL injection attacks. Kim and Lee [11] used internal query trees from the log database as input for the SVM classification method to detect SQL injection attacks at the database level. Joshi and Getha [12] combined the Naïve Bayes method with Role-Based Access Control mechanisms to classify whether input data was an attack or not. The limitation of the study [12] is that it has not been able to detect blind SQL injection attacks.

This study proposes a combination of pattern-matching and machine learning methods to prevent SQL injection attacks. The proposed method is the Hybrid Method which is an integration of the SQL Injection Free Secure (SQL-IF) and Naïve Bayes methods. In addition, the SQL-IF method is proposed to use a constant (K) as the attack detection threshold whose value can be adjusted to solve the problem of false positives. In this study, a system that acts as a proxy to prevent SQL injection attacks using the Hybrid Method was built.

This paper is organized into four parts. The first part is an introduction, containing the background to the problem and related previous studies. The second part contains the methodology, which describes the methods used and the stages of the study. Next, the third section shows the results and discussion of this study and ends with the fourth chapter, namely the conclusion.

2. Method

The stages carried out in this study are described as follows.

2.1. Data Collection

At this stage, a simulation laboratory is conducted to obtain attack log data. Simulation is carried out by creating a Linux server with a web server and installed database. Then web applications such as WordPress CMS, Joomla, and web applications that have SQL injection vulnerabilities are installed on the webserver. After that, penetration testing of web applications was carried out by several respondents who had hacking skills. The testing was black-box testing. At this stage, pentester performs web application security testing, especially SQL injection vulnerabilities. All attacks sent to the server are collected for attack log data and stored in the database. Furthermore, the attack log data is selected and grouped into attacks data and normal data. The data used in this study were 250 attack log data consisting of 147 attacks data and 103 normal data. Sample data are shown in Table 1 and Table 2.

Table 1. Sample of attacks data

No	Input
1	/search.php?search='+and+order+by+1+--+-+
2	username='+or+1+=+1&password=aa
3	/movie.php?id_movie=13'order+by -- -+
4	/search.php?search='+anu+ord+hex
…	…
147	/genres.php?id_genre=3%'))) AND 2015=8028 AND ((('%'='

Table 2. Sample of normal data

No	Input
1	/wordpress/wp-login.php
2	log=faisal&pwd=faisal123?&wpsubmit=Log+In&redirect_to=http://103.250.83.29/wordpress /wp-admin/&testcookie=1
3	/wordpress/wp-admin/
4	/wordpress/wp-admin/load-scripts.php?c=1&load[]=jquerycore,jquery-migrate,utils&ver=4.9.1
…	…
103	/search.php?search=order

2.2. Dataset Creation

At this stage, the attacks data and normal data are converted into numeric tokens by calculating the number of keywords considered triggers for SQL injection that exist in each attacks data and normal data. These keywords are grouped into several categories, namely SQL Comments, SQL Operators, Logical SQL, and SQL Keywords. In addition, it also identifies the User-Agent used when carrying out attacks. The list of used keywords is shown in Table 3.

The calculation and identification results are represented in a dataset table consisting of five (5) features/attributes, namely Comments; Operator; Logical; Keyword; and User-Agent, to which a label is added, namely Status. This dataset will be used as training data to prevent SQL injection attacks using the Naïve Bayes method. An example of the dataset is shown in Table 4.

Table 3. List of keywords

Category	Keywords
Comment SQL	--, #, -+, ++, --, -'', /*, */, /**/
Operator SQL	<, >, ==, != <=, >=, <<, >>, ||, &&
Logical Operator SQL	OR, AND, NOR, XOR
Keyword SQL	UNION, SELECT, ORDER, CONCAT, GROUP, INFORMATION_SCHEMA.TABLES, TABLE_NAME, TABLE_SCHEMA, GROUP_CONCAT, COLUMN_NAME, INFORMATION_SCHEMA.COLUMNS, LIMIT, COUNT, CHAR, BY, SLEEP, BENCHMARK, LIKE, WAITFOR, LOAD_FILE, DECLARE, INSERT, UPDATE, FROM, DATABASE, WHERE, EXEC, HEX, DELAY, DESC, FALSE, COUNT, EXPORT_SET, ORD
User-Agent	SQLMAP, HAVIJ, WVS, NESSUS, W3AF, DIRBUSTER, NIKTO, OPENVAS

Table 4. Example of dataset

No	Comment	Operator	Logical	Keyword	User Agent	Status
1	0	0	1	1	0	Normal
2	0	0	1	1	0	Normal
3	0	0	1	1	0	Normal
4	0	0	1	1	0	Attack
5	0	0	1	1	0	Attack
...	...	...	...	...	...	...
250	0	0	1	0	0	Attack

2.3. Implementation of the Free Secure SQL Injection Method

SQL Injection Free Secure (SQL-IF) is a method specially developed to detect SQL injection attacks. The steps taken in detecting are checking special characters, keywords, and Boolean [13]. In this study, the SQL-IF method is used to detect whether the input data in the form of an HTTP request and User-Agent sent by the client is an attack or not an attack (normal).

After the input data is received, the next step is to check the User-Agent. If a client is detected using a list of keywords in the User-Agent category, access will be blocked because it is considered to be carrying out a SQL injection attack using tools. If not, then the URL of the HTTP request using the GET method or input parameter data sent using the POST method will be matched with the keyword list in the SQL Comment, SQL Operator, SQL Logical Operator, and Keyword category. In addition, the calculation of each match in each category is also carried out and the value is stored in four (4) attributes, namely Comment, Operator, Logical, and Keyword.

SQL injection attacks are detected when the input data and keyword list in the above category results in the number of matches more than the predefined constant (K) value. For example, the value of K is three (3), if the sum of the four attributes results in a value exceeding the value of K, it will be considered an attack. The steps for preventing SQL injection attacks using the SQL-IF method are shown in Algorithm 1.

Algorithm 1. SQL-IF Method

Input: HTTP request dan User-Agent Output: Attack or normal
1. Receive input. 2. Check User-Agent compatibility. If it matches, it is blocked. If not, move on to stage 3. 3. Count the number of comments on the input based on the SQL Comment keyword list. 4. Count the number of Operators on the input based on the SQL Operator keyword list. 5. Calculate the Logical number on the input based on the SQL Logical Operator keyword list. 6. Count the number of keywords in the input based on the SQL keyword list. 7. Calculate the total value of all attributes. 8. Compare the value of the constant (K). 9. If it is greater than the value of K then it is considered an attack, if it is smaller then it is considered normal.

2.4.Implementation of the Naïve Bayes Method

Naïve Bayes is a simple probabilistic classification method that assumes all features/attributes are independent or not interdependent [12]. It can be used to predict the future based on previous experience [14] and does not require large training data [15]. In this study, Naïve Bayes is to classify input data into attacks and not attacks (normal).

The dataset is trained before being used for classification. The input data is in the form of an HTTP request sent by the client using either the GET or POST method and the User-Agent is converted into a numeric token and broken down into five attributes as in the dataset. Four (4) attributes on the dataset (training data) and input data (test data), namely Comment; Operator; Logical; and Keyword has continuous data, whereas User-Agent contains discrete data.

After that, a probability calculation is carried out based on the training data that has been provided. If the value of the calculation result is higher in the attack class then a label is given, namely the status as an attack. Conversely, if the calculated value is higher in the normal class then a label is given, namely the status as normal. The naïve Bayes formula used in classifying using the Gaussian Naïve Bayes is in Equation (1). The steps to prevent SQL injection attacks using the Naïve Bayes method in this study are shown in Algorithm 2.

$$P(X_i = x_i | Y = y_j) = \frac{1}{\sqrt{2\pi\sigma^2}} exp^{\frac{-(xi-\mu ij)^2}{2\sigma^2 ij}} \qquad (1)$$

P : opportunity
Xi: attribute to i
xi: the value of the i^{th} attribute
Y: the class we are looking for
yj: the subclass we are looking for
μ: the average of all attributes
σ: the standard deviation used to express variation across attributes

Algorithm 2. Naïve Bayes method

Input: HTTP request and User-Agent Output: Attack or normal
1. Receive input. 2. Count the number of comments in the input based on the SQL Comment keyword list. 3. Count the number of operators in the input based on the SQL Operator keyword list. 4. Calculate the Logical number on the input based on the SQL Logical Operator keyword list. 5. Count the number of keywords in the input based on a list of SQL Keyword keywords. 6. Check the User-Agent with a list of keywords in the User-Agent category. If there is a match then it has a value of 1, if it is not a match, it has a value of 0. 7. Convert into numeric tokens. 8. Calculate the probability of attack and normal status on the dataset. 9. Calculate the mean and standard deviation of each attribute that has continuous data (Comment, Operator, Logical, and Keyword) for the attack and normal classes. 10. Calculate the probability of each attribute with continuous data for the attack class and normal using the Gaussian Naïve Bayes formula. 11. Since User-Agent is not continuous data, use the User-Agent value from step 6 as the probability value for the User-Agent attribute for the attack and normal classes. 12. Calculate the probability value for each class. 13. Determine the status as an attack or normal based on the class with the highest probability value.

2.5. Implementation of Hybrid Method

At this stage, the Hybrid Method is implemented, which is an integration of the SQL-IF and Naïve Bayes methods. The integration in this study is the use of two methods in sequence (SQL-IF and Naïve Bayes) to check SQL injection attacks. The Hybrid Method steps are shown in Fig. 1.

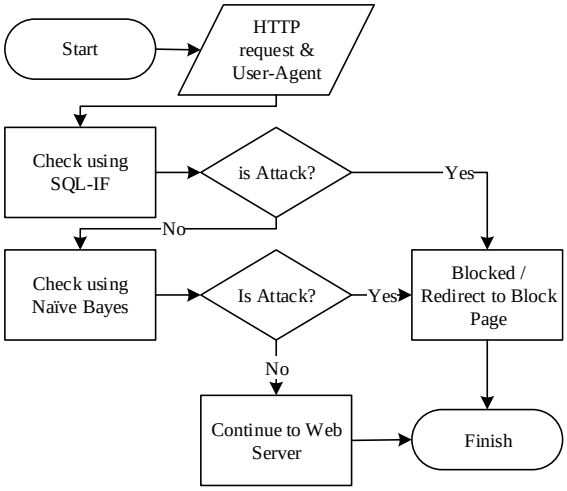

Fig. 1. Flowchart hybrid method

Input data in the form of an HTTP request and User-Agent from the client will be checked using the SQL-IF method first. If detected as an attack, it will be redirected to the block page and recorded as attack log data in the database. If not, the input data is forwarded to be checked using the Naïve Bayes method. If an attack is detected, it will be redirected to the block page. If not, the input data is

forwarded to the webserver to be processed according to the request and returned to the client. The Hybrid Method is implemented into a system built using the Python programming language, MySQL database, and the Socat library as a proxy so that SQL injection attacks can be prevented before entering the web server and database server. The system architecture is shown in Fig. 2.

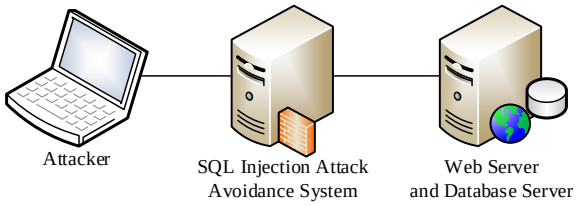

Fig. 2. System architecture

2.6. Testing

At this stage, testing is carried out to determine the accuracy and efficiency of SQL injection attack prevention using the SQL-IF method, Naïve Bayes, Hybrid Method. Testing is carried out by sending attacks either manually or using SQLMAP software. Attacks were carried out 100 times for each method with different input data. Accuracy testing is carried out with the scenarios shown in Table 5.

Table 5. Accuracy Testing Scenarios

Test Code	Constant	Dataset	Accuracy
Uji_A1	K = 5	125	Accuracy Percentage
Uji_A2	K = 3	250	Accuracy Percentage

The calculation of accuracy is conducted with the following formula.

$$\text{Accuracy} = \frac{\text{Num of Success Attacks}}{\text{Num of Attacks}} \text{x } 100\% \quad (2)$$

An example of testing the accuracy of the SQL-IF method manually (not using tools) is in Fig. 3.

```
C:\Python27\python.exe "C:/Users/Paijotelo/Dropbox/14102019_Faisa
########################################
###        Input From User        ####
########################################
[+] Payload    : /search.php?search='union select 1,2,3,4,5 -- -+
[+] User-Agent : Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0
########################################
####         Detections            ###
########################################
[+] Comment Detection     :  -- -+  ( 2 )
[+] Operator Detection    :   ( 0 )
[+] Logical Detection     :   ( 0 )
[+] Keyword Detection     :  UNION SELECT  ( 2 )
[+] User-Agent Detection  :   ( 0 )
########################################
[+] Value of Constants    :  3
[+] Value of Detections   :  4
[+] Check User-Agent  : Scanner Not Detected !
########################################
####          Conclustion           ##
########################################
[+] Status : SQL Injection Detected !
```

Fig. 3. SQL-IF accuracy manual testing

In Fig. 3, when the attacker enters the payload 'union select 1,2,3,4,5 - ++ then the system redirects to the block page and records it as an attack in the attack log in the database. This is because the payload contains SQL keywords that are commonly used to perform SQL injection, namely UNION and SELECT. In addition, the payload also contains comments commonly used in SQL such as - and - +, and single quotes that are often used to launch SQL injection attacks. After calculating the total value of all attributes and compared it with the constant value (K), it is found that the total value of all attributes is smaller than the K value so that the status is declared as an attack. An example of testing the accuracy of the Naïve Bayes method manually (not using tools) is shown in Fig. 4.

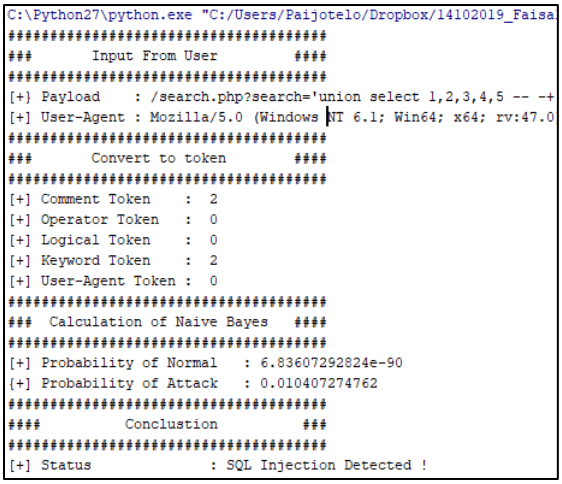

Fig. 4. Naïve Bayes accuracy manual testing

In Fig. 4, the input data of payload 'union select 1,2,3,4,5 - ++ is converted into a token and the attribute value is calculated based on a list of keywords that are often used to perform SQL injection attacks. After that, the calculation using the naïve Bayes method shows that the probability value of the attack is greater than the normal probability so that the status is declared an attack. An example of testing the accuracy of the SQL-IF method using a tool, namely the SQLMAP software is shown in Fig. 5, 6, and 7. Fig. 5 shows that the SQLMAP software detects the presence of a WAF (Web Application Firewall) / IPS (Intrusion Prevention System) / IDS (Intrusion Detection System) which protects the web application from SQL injection attacks launched using SQLMAP.

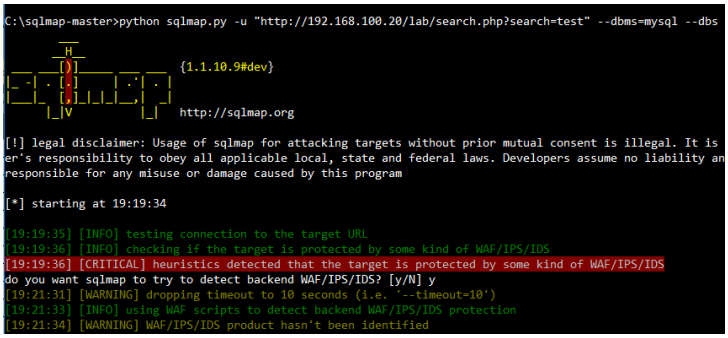

Fig. 5. Testing using SQLMAP

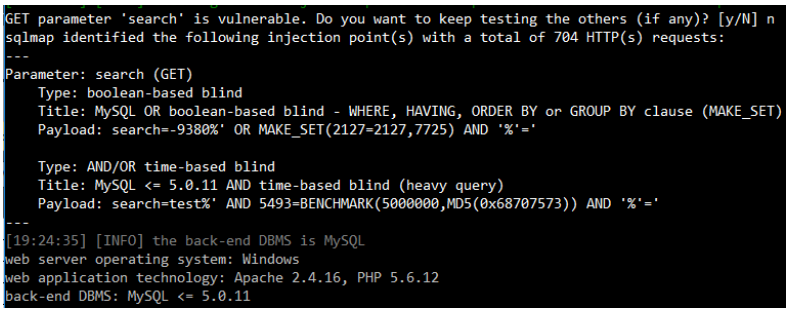

Fig. 6. SQLMAP identifies the target of attacks

Fig. 6 shows that the SQLMAP software can identify vulnerable targets, proven by identifying the web server OS, web application technology, and DBMS.

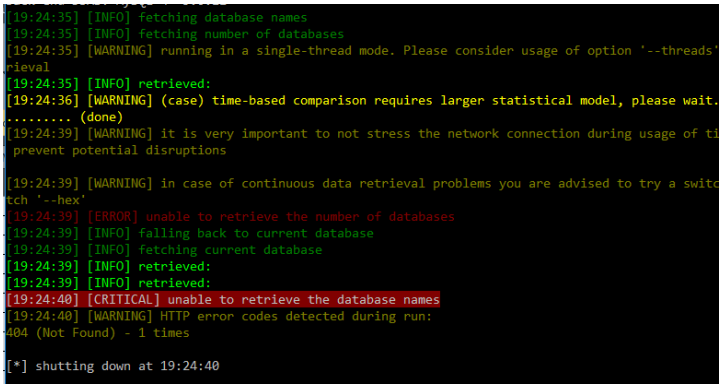

Fig. 7. Attack using failed SQLMAP

Fig. 7 shows that SQLMAP failed to carry out subsequent attacks such as getting database name, table name, column name, and dump data because it could be stopped by both SQL-IF and Naïve Bayes methods. Efficiency testing is carried out by knowing how fast the load time of web pages on localhost and VPS is accessed using a browser. The efficiency testing scenario is shown in Table 6.

Table 6. Efficiency testing scenarios

Test Code	Testing Place	Efficiency
Uji_E1	Localhost	Average load time
Uji_E2	VPS	Average load time

3. Results and Discussion

After all the stages of the research starting from data collection to the completion of testing, the results of testing the accuracy and efficiency of SQL injection attack prevention using the SQL-IF, Naïve Bayes, and Hybrid Methods are obtained. The results of the accuracy-test are shown in Table 7 and Fig. 8.

Table 7. Accuracy testing results

Test Code	Accuracy		
	SQL-IF	Naïve Bayes	Hybrid
TC_A1	72%	59%	72%
TC_A2	83%	88%	90%

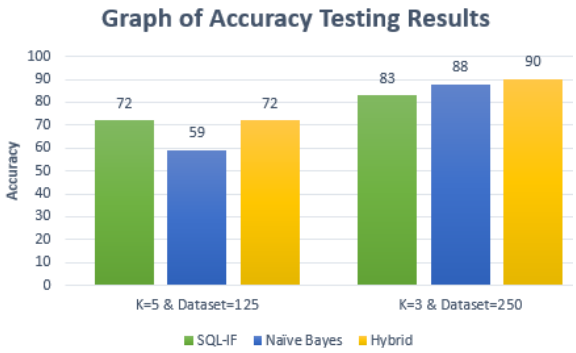

Fig. 8. Graph of accuracy test results

Based on the results of the accuracy test, it is revealed that the system can prevent SQL injection attacks carried out manually or using tools, such as the sqlmap software. In the first accuracy test, with a constant value (K) of 5, the SQL-IF method can prevent SQL injection attacks with an accuracy value of 72%; with 125 datasets, Naïve Bayes only has the lowest accuracy value, namely 59%; while the Hybrid Method produces an accuracy value of 72% (the same high as the SQL-IF accuracy value).

In the second accuracy test, decreasing the K value from 5 to 3 increases the accuracy value of the SQL-IF method to 83%; increasing the number of datasets from 125 to 250 increases the accuracy value of Naïve Bayes to 88% (higher than the accuracy value of SQL-IF); while the Hybrid Method has the highest accuracy value, namely 90%. In addition, it is known that a small constant value and a large number of datasets can produce higher accuracy values. Efficiency tests carried out on localhost and VPS produce the average load time value of web pages as shown in Table 8, Fig. 9, and Fig. 10.

From the results of the first efficiency test (on localhost), the average value of web page load time without the SQL injection prevention method is 3.164 s. The application of the method causes the average load time of web pages to be longer, namely SQL-IF is 4.2435 s, Naïve Bayes is 4895 s, while the Hybrid Method is 5.3645 s.

Table 8. Efficiency test results

Test Code	Average of Load Time (s)			
	Without Method	SQL-IF	Naïve Bayes	Hybrid
TC_E1	3.164	4.2435	4.895	5.3645
TC_E2	6.3745	7.261	7.862	8.556

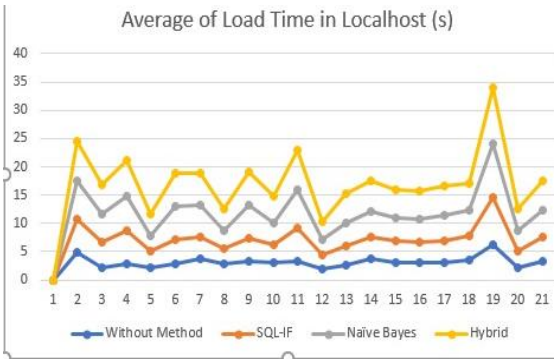

Fig. 9. Graph of average load time on localhost

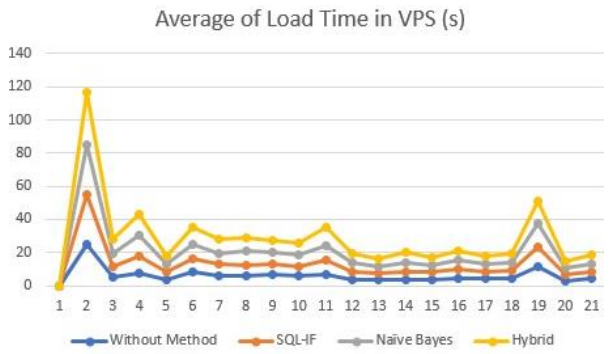

Fig. 10. Graph of average load time on VPS

The results of the second efficiency test (at VPS) show that the average load time value without the SQL injection prevention method is 6.3745 s. The application of the method produces a longer average load time for web pages, namely SQL-IF is 7,261 s, Naïve Bayes is 7,862 s, while the Hybrid Method is 8,556 s.

Based on the results of the first and second efficiency tests, it is noted that the difference in the average load time value of web pages without a method and using a method considered to be short, namely ± 1s. The Naïve Bayes method's average load time value for web pages is longer than SQL-IF. The Hybrid Method produces the longest average load time for web pages. This happens because there are more checks and arithmetic operations, resulting in decreased access speeds. In addition, it is known that the average load time of web pages on VPS is longer than on localhost. This can be due to the fluctuating bandwidth characteristics that affect the access speed [16], [17], [18].

4. Conclusion

Hybrid Method which is an integration of SQL Injection Free Secure (SQL-IF) and Naïve Bayes methods can increase the accuracy of SQL injection attack prevention with an accuracy value of 90%. This accuracy value is higher than the accuracy value of the SQL-IF method which has a value of 83% and the Naïve Bayes method with a value of 88%. In addition, there is a relationship between the constant value (K) in SQL-IF and the number of datasets on Naïve Bayes to an increase in the accuracy value. The smaller the K value and the more datasets used can produce better accuracy values. In terms of efficiency, the application of the Hybrid Method causes a decrease in access speed with an average load time of 5.3645 s web pages on localhost. This value is lower than the SQL-IF method which has an average web page load time value of 4.2435 s and the Naïve Bayes method with a value of 4,895 s.

References

[1] V. Prokhorenko, K.-K. R. Choo, and H. Ashman, "Web application protection techniques: A taxonomy," *Journal of Network and Computer Applications*, vol. 60, pp. 95-112. 2016.

[2] Verizon, "2017 Data Breach Investigations Report 10th Edition," Verizon, 2017. [Online]. Available: https://www.verizonenterprise.com/resources/reports/2017_dbir_en_xg.pdf

[3] OWASP, "OWASP Top 10 – 2017 rc 1," OWASP, 2017. [Online]. Available: https://www.owasp.org/images/3/3c/OWASP_Top_10__2017_Release_Candidate1_English.pdf
[4] J. Clarke, SQL Injection Attacks and Defense. Waltham, MA, USA: Elsevier, 2012.
[5] K. P. Rao, D. A. B.Sasankar dan D. V. Chavan, "Analysis of Detection and Prevention Techniques Against SQL Injection Vulnerabilities," *IJCST*, vol. 4, no.1, pp. 50-55. 2013.
[6] C. Basta, A. Elfatatry, and S. Darwish, "Detection of SQL Injection Using a Genetic Fuzzy Classifier System," *IJACSA*, vol.7, no.6, pp.129-137. 2016.
[7] Veracode, "State of Software Security Focus on Application Development Supplement to Volume 6," Veracode, 2015. [Online]. Available: https://www.veracode.com/sites/default/files/Resources/Reports/state-of-software-security-focusonapplicationdevelopment.pdf?mkt_tok=3RkMMJWWfF9wsRovua3NZKXonjHpfsX%2F6u8uUaag38431UFwdcjKPmjr1YIARcJiI%2BSLDwEYGJlv6SgFTbnFMbprzbgPUhA%3D
[8] M.A. Prabakar, M. KarthiKeyan, and K.Marimuthu, "Wavelength-switched passively coupled single-mode optical network," in *Proc. ICECCN*, Tirunelveli, India, 2013, pp. 585–590.
[9] A.Z.N. Saleh, N.A. Rozali, and A.G. Buja, "A Method for Web Application Vulnerabilities Detection by Using Boyer-Moore String Matching Algorithm," *Procedia Computer Science*, vol.72, pp.112-121. 2015.
[10] R. Ellysa, M. Husni, and B.A. Pratomo, "Pendeteksi Serangan SQL Injection Menggunakan Algoritma SQL Injection Free Secure pada Aplikasi Web," *Jurnal Teknik Pomits*, vol.2, No.1, pp.1-6. 2013.
[11] M.-Y. Kim and D. H. Lee, "Data mining-based SQL injection attack detection using internal query trees," *Expert Systems with Applications*, vol.41, No.11, pp.5416-5430. 2014.
[12] A. Joshi and V. Geetha, "SQL Injection Detection using Machine Learning," in *Proc. ICCICCT*, Kanyakumari, India, 2014, pp. 1111–1115.
[13] K. Natarajan and S. Sabramani, "Generation of SQL-injection free secure algorithm to detect and prevent SQL-injection attacks," *Procedia Technology*, vol.4, pp.790-796. 2012.
[14] A. Saleh, "Implementasi Metode Klasifikasi Naïve Bayes Dalam Memprediksi Besarnya Penggunaan Listrik Rumah Tangga," *Citec Journal*, vol.2, No.3, pp.207-217. 2015.
[15] S.A. Pattekari and A. Parveen, "Prediction System for Heart Disease using Naïve Bayes," *International Journal of Advanced Computer and Mathematical Sciences*, vol.3, No.3, pp.290-294. 2012.
[16] E.A. Permanasari, I. Hidayah, and I.A. Bustoni, "Forecasting Model for Hotspot Bandwidth Management at Department of Electrical Engineering and Information Technology UGM," *Int. J. Appl. Math. Stat*, vol.53, No.4, pp.227-234. 2015.
[17] I. Hidayatulloh and I.A. Bustoni, "Sarima-Egarch Model to Reduce Heterscedasticity Effects in Network Traffic Forecasting," *JATIT*, vol.95, No.3, pp.554-560. 2017.
[18] I.A. Bustoni et al., "Fuzzy Logic Tsukamoto for SARIMA On Automation of Bandwidth Allocation," IJACSA, vol.8, No.1, pp.392-397. 2017.