multithreaded cross-platform programs, optimizing algorithms and architectures for scalable computing are current tasks.

The purpose of this work is to address the thread race problem in multithreaded computing of resource-intensive tasks with parallel access to shared data using appropriate synchronization mechanisms, such as mutexes.

The thread race problem is considered using the example of solving a typical task of finding the sum of elements of a super-large array in multithreaded mode.

The algorithm for solving this task is implemented in C++ in the Microsoft Visual Studio 2022 IDE using the std::vector class – a dynamic array from the STL (Standard Template Library). The standard std::thread class is used to create and manage threads in the program. Each thread is responsible for a certain segment of the array for which it calculates the sum of elements in a specially developed function with execution timing. The standard std::mutex class is used to solve the thread race problem in critical areas of code.

As a result, a multithreaded algorithm has been developed to implement a typical task of processing super-large data arrays with protection of the critical area to prevent the thread race problem using mutexes; the performance of the developed algorithm was investigated with a significant (10^8–10^9 elements) amount of processed data and a variable (1–14) number of computing threads; a concept was developed for further application of effective approaches to data protection in concurrent programs implemented on multiprocessor and multi-core systems.

It was found that solving the thread race problem in the considered task on a modern PC with an Intel Core i7-12700H processor slows down the program execution by approximately 10 times when increasing the array size in the aforementioned range. Increasing the number of used threads from 2 to 14 in this case slows down the application implementation by approximately 4 times, regardless of the amount of data processed.

The use of the obtained developments, together with other advanced software tools to increase computational performance, will contribute to the development of effective computer models of technological processes and systems.

UDC 004.65

RESEARCH ON THE DETECTION OF SQL INJECTION ATTACKS BASED ON THE STRING MATCH APPROACH

A.Kopp (andrii.kopp@khpi.edu.ua)
National Technical University «Kharkiv Polytechnic Institute» (Ukraine)

Abstract. *This paper focuses on the problem of detecting and preventing SQL (Structured Query Language) injection attacks. Recent studies in this area are analyzed and the most common type of SQL injection attack is examined. The string match approach already known from related work on SQL injection detection is used to answer the question – which special characters can be used to detect and prevent SQL injection attacks when analyzing HTTP (Hyper Text Transfer Protocol) traffic. Several null hypotheses are tested and the subset of characters with the strongest impact is used to build the regular expression for malicious SQL code detection.*

Problem statement. One of the Top 10 Web Application Security Risks recognized by OWASP (Open Web Application Security Project) [1] is SQL (Structured Query Language) Injection Attacks, often known as SQLIA. Intruders utilize SQLIA, malicious behavior, or policy violations, to attack online applications in order to get access to protected information without authorization or even take over the database server [2]. Authors of [3] claim that SQLIA is a type of cyber-attack that takes use of breaches in web applications that relate to database-stored data. As a result, hackers may purposefully alter SQL queries to compromise database security. It is made feasible by taking advantage of weak security measures used by the online application or DBMS (Database Management System) and SQL syntax flaws [3].

One of the most dangerous and regular online application threats is an injection, which happens when an intrusive party modifies, deletes, or queries data from a DBMS server. Database integrity, availability, and security may be compromised by SQLIA. Therefore, detecting and preventing SQL injection attacks is a pertinent yet established study area [4]. The diverse cyber-attack patterns and strategies employed by hackers make it difficult to identify SQLIA despite the frequency of attacks and the wealth of studies in this field [5].

Paper [6] asserts that attackers typically take advantage of the WHERE clause, which governs the dataset results requested from the database. The SQL commands SELECT (to query tables), UPDATE (to edit records), and DELETE (to delete records) all support the use of a WHERE condition. This kind of attack, an example of which is shown in Fig. 1 [6], is typically referred to as a tautological SQLIA or boolean-based SQL injection [7].

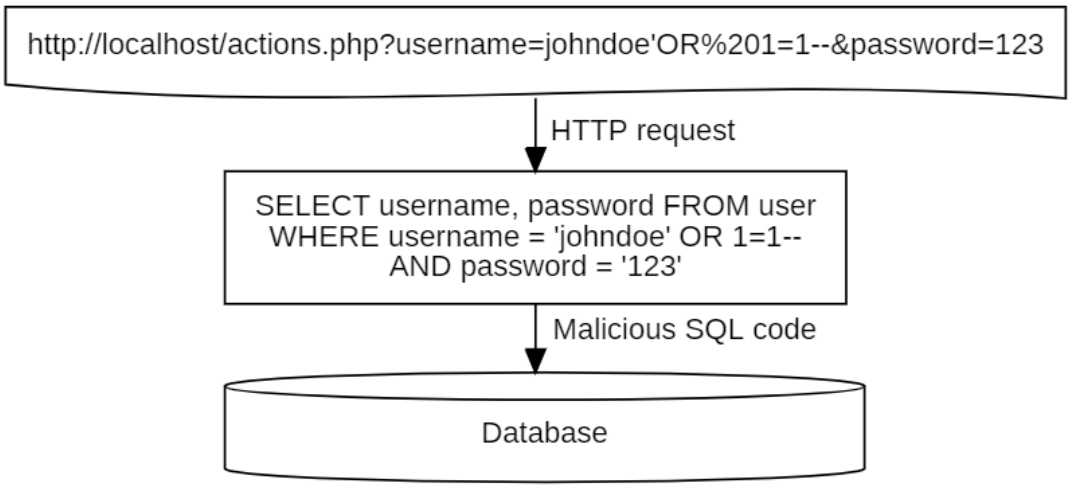

Figure 1 – Tautological SQLIA and query string illustration [6]

As demonstrated in Fig. 1, such a malicious SQL query injects the second, always TRUE condition "1 = 1" via the logical OR operator, forcing the web application to deliver a result. The database server will not process the remaining portion of this SQL query, which comes after the comment symbols "--". As a result, the updated SQL query produces a new outcome that is unrelated to the accuracy of the user name or password.

As a result of this, authors of [7] take into account particular symbols used to create SQLIA code in order to identify such cyber-attacks (Table 1).

Table 1 – Symbols typically used to create SQLIA code

Type	Characters
String	'
Comment	--, #, /*, */
Wildcard	%
Terminator	;
Concatenate	+, ||
Assignment	=
Comparison	>, >=, <, <=, ==, <>, !=

Thus, this study aims to answer the following research question – which special characters could signalize the presence of SQL injection attacks in the HTTP (Hyper Text Transfer Protocol) traffic?

Solved tasks. Therefore, to answer this research question, it is necessary to test the following null hypothesis:

– for string characters – H0,1: There is no correlation between string characters and SQLIA;

– for comment characters – H0,2: There is no correlation between the presence of comment characters and SQLIA;

– for wildcard characters – H0,3: There is no correlation between the presence of wildcard characters and SQLIA;

– for terminator characters – H0,4: There is no correlation between the presence of terminator characters and SQLIA;

– for concatenate characters – H0,5: There is no correlation between the presence of concatenate characters and SQLIA;

– for assignment characters – H0,6: There is no correlation between the presence of assignment characters and SQLIA;

– for comparison characters – H0,7: There is no correlation between the presence of comparison characters and SQLIA.

Research results. To perform the necessary experiments, let us use "HttpParamsDataset" that represents values which can be found as values of parameters in HTTP requests [8]. This dataset includes 31067 records, which are divided into two categories [8]:

– benign records (19304 items designated as normal);

– suspicious records (11763 items tagged as anomalies).

Out of 31067 records, 10852 records are labeled as "sqli" – these are SQL injection attacks.

Therefore, this dataset [8] was processed to detect the presence of different special characters in the HTTP request parameters analyzed. The final dataset, which was used to test the hypotheses formulated, includes the following fields (Fig. 2)

– query – the HTTP request string;

– string – the number of string characters occurred in the HTTP request string;

– comment – the number of comment characters occurred in the HTTP request string;

– wildcard – the number of wildcard characters occurred in the HTTP request string;

– terminator – the number of terminator characters occurred in the HTTP request string;

– concatenate – the number of concatenate characters occurred in the HTTP request string;

– assignment – the number of assignment characters occurred in the HTTP request string;

– comparison – the number of comparison characters occurred in the HTTP request string;

– test – indicates whether the HTTP string is suspicious for SQLIA (1) or not (0).

	query	string	comment	wildcard	terminator	concatenate	assignment	comparison	test
436	1' where 6406=6406;select count(*) from rdb$fields as t1,rdb$types as t2,rdb$collations as t3,rdb$functions as t4--	1	1	0	1	0	1	0	1
437	1) and 8514=(select count(*) from domain.domains as t1,domain.columns as t2,domain.tables as t3) and (4666=4666	0	0	0	0	0	1	0	1
438	-3136%') or 3400=6002	1	0	1	0	0	1	0	1
439	1) where 7956=7956 or sleep(5)#	0	1	0	0	0	1	0	1
440	-7387'))) order by 1--	1	1	0	0	0	0	0	1

Figure 2 – Fragment of the created dataset

Using the logistic regression classification algorithm [9], there was found the "concatenate" variable has a p-value (0.347) greater than 0.05 (the alpha significance level). Therefore, it is clear that there is no correlation between the presence of concatenate characters and SQLIA and accept the null hypothesis H0,5. However, the p-value of the other variables is 0 (< 0.05), which allows us to reject the remaining hypotheses H0,1 – H0,4 and H0,6 – H0,7.

Conclusion. This paper addressed the problem of SQLIA detection. The different types of SQLIAs were discussed and the string match approach was considered for detecting SQLIAs. In addition, the meaningful features for SQLIA detection were defined. Therefore, the future machine learning models for SQLIA detection could be based on the selected features except "concatenate".

References

[1] OWASP, “OWASP Top Ten,” *Owasp.org*, 2021. https://owasp.org/www-project-top-ten/

[2] T. Singh and B. Aksanli, “Real-time Traffic Monitoring and SQL Injection Attack Detection for Edge Networks,” *Proceedings of the 15th ACM International Symposium on QoS and Security for Wireless and Mobile Networks*, Nov. 2019, doi: https://doi.org/10.1145/3345837.3355952.

[3] S. Bhardwaj and M. Dave, “SQL Injection Attack Detection, Evidence Collection, and Notifying System Using Standard Intrusion Detection System in Network Forensics,” *Lecture Notes on Data Engineering and Communications Technologies*, pp. 681–692, 2021, doi: https://doi.org/10.1007/978-981-33-4968-1_53.

[4] M. Alghawazi, D. Alghazzawi, and S. Alarifi, “Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review,” *Journal of Cybersecurity and Privacy*, vol. 2, no. 4, pp. 764–777, Sep. 2022, doi: https://doi.org/10.3390/jcp2040039.

[5] D. Lu, J. Fei, and L. Liu, “A Semantic Learning-Based SQL Injection Attack Detection Technology,” *Electronics*, vol. 12, no. 6, p. 1344, Mar. 2023, doi: https://doi.org/10.3390/electronics12061344.

[6] “Applied Machine Learning predictive analytics to SQL Injection Attack detection and prevention,” *ieeexplore.ieee.org*. https://ieeexplore.ieee.org/document/7987433 (accessed Oct. 11, 2023).

[7] O. C. Abikoye, A. Abubakar, A. H. Dokoro, O. N. Akande, and A. A. Kayode, “A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm,” *EURASIP Journal on Information Security*, vol. 2020, no. 1, Aug. 2020, doi: https://doi.org/10.1186/s13635-020-00113-y.

[8] “HttpParamsDataset,” *www.kaggle.com*. https://www.kaggle.com/datasets/evg3n1j/http-paramsdataset (accessed Oct. 11, 2023).

[9] A. Roberts and J. M. Roberts, *Multiple Regression*. SAGE Publications, 2020.

УДК 004.932

КВАНТУВАННЯ ТРАНСФОРМАНТ ДВОВИМІРНИХ ОРТОГОНАЛЬНИХ ПЕРЕТВОРЕНЬ ПРИ УЩІЛЬНЕННІ ЗОБРАЖЕНЬ

Майданюк В. П., Матвійчук О. В. (maidaniuk2000@gmail.com, 888sasha@gmail.com)
Вінницький національний технічний університет (Україна)

Розглянуто особливості квантування трансформант двовимірних ортогональних перетворень при ущільненні зображень. Показано, що збільшення коефіцієнта ущільнення може бути досягнуто через векторне квантування трансформант дискретного ортогонального перетворення (Уолша-Адамара, ДКП та інших). Ідеальними для вирішення завдань векторного квантування є нейронні мережі, що самоорганізуються, запропоновані фінським вченим Т. Кохоненом (Self-Organizing Feature Map – SOFM).

Зростання складності систем обробки даних сприяє безперервному збільшенню потоку інформації, який прямує до центрального процесору, пристроїв відображення чи в канали зв’язку. Досить часто така інформація відрізняється значною надлишковістю, що в свою чергу веде до нераціонального використання обладнання. З метою полегшення роботи операційних пристроїв, зниження об’єму пам’яті і мінімізації смуги частот систем передачі, початкову інформацію у таких випадках бажано попередньо обробляти, для чого виконується ущільнення цієї інформації і одночасно перетворення у форму, зручну для подальшого використання в цифрових блоках.

Найбільшу складність викликає ущільнення зображень, оскільки необхідно обробляти масиви даних великих розмірів з високою швидкодією. Ущільнення зображень полягає в мінімізації кількості інформаційних елементів, які потрібні для представлення зображення. Відновлення зображення у попередню форму супроводжується, як правило, деякими спотвореннями.

Кодування на основі перетворень радикально відрізняється від класичних методів кодування, таких як імпульсно-кодова модуляція, кодування з передбаченням або з інтерполяцією, які