SQLIA TYPES AND TECHNIQUES - A SYSTEMATIC ANALYSIS OF EFFECTIVE PERFORMANCE METRICS FOR SQL INJECTION VULNERABILITY MITIGATION TECHNIQUES

Aduragbemi David Ogundijo, Atiff Abdalla Mahmoud Arabi and Tadiwa Elisha Nyamasvisva
Infrastructure University Kuala Lumpur, MALAYSIA

ABSTRACT

According to Open Web Application Security Project (OWASP), an online community that produces well-researched reports in the field of web application security, Structured Query Language (SQL) injection remains in the top three most common input vulnerability in applications due to the progression from static to dynamic web pages leading to increased database use in web applications. SQL injection vulnerabilities is prevalent in web and mobile applications because of common unsafe coding practices. A successful SQL injection attack poses a significant risk to the database, application, and web server as a whole. In this article, the authors have examined approaches for preventing SQL injection attacks and categorize SQL injection attacks based on the methods used to exploit SQL vulnerabilities. In terms of preventing all forms of SQL injection attacks, the discussed approach appears to be acceptable. This review paper presents a systematic review of the mitigation steps which include reconnaissance, enumeration, and extraction of data. Also discussed are types of injection attacks, some alternative procedures for mitigating SQL attacks and performance metrics for measuring the effectiveness of SQL injection mitigation techniques.

Keywords:
Injection Attacks, Mitigation Techniques, Detection Mechanisms, Vulnerability Exploitation, Anomaly Detection, Tautology, SQLIA

INTRODUCTION

The data management (model), display or front-end tiers (view) and application processing(controller) also called Model-View-Controller (MVC), are conceptually separated in today's web applications, and are based on an n-tier architecture. Instead of rewriting entire programs, developers now just need to add or alter a single layer as needed, making design and maintenance easier. (Al-Ahmad et al., 2014; Aniche et al., 2018; Paolone et al., 2021)

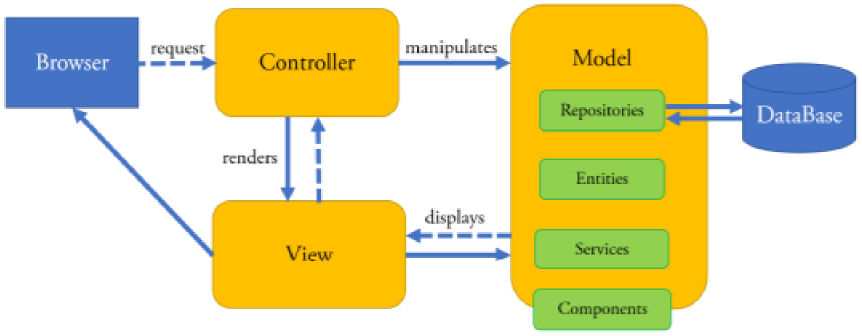

Figure 1. MVC Architecture as illustrated in (Aniche et al., 2018; Paolone et al., 2021)

The model or data management layer consists of a database server that stores and retrieves sensitive information about the application and its users. The database data is frequently used for user authentication, storing records and their relationships, and presenting the data on a dynamically generated web page. Application Programming Interfaces (APIs) such as Open Database Connectivity (ODBC) and Java Database Connectivity are used to link the web application to the database management system (JDBC). Connection to the database servers are established and SQL queries are run using the built-in objects and methods. The SQL query processor receives the queries and executes them. The application server receives the results of the queries. The application server examines the returned data and makes a judgment, after which the data is rendered in the dynamic web page by the browser. User-supplied parameters are frequently included in the query that is sent to the database server for execution. The user-supplied input parameters may or may not be reliable. The query processor will, of course, run the query and deliver the result to the browser for display to user regardless of the query's type. The query, however, might still include malicious code or be logically wrong.

Attackers can inject malicious code into the input parameter as a result of the MVC design. If the code fails to appropriately isolate programme instructions from user data, the adversary may be able to do malicious input. By modifying the SQL query, the attacker can extract confidential information from the database and obtain complete control of the database and database server. The term "SQL injection attack" is used by hackers to describe this type of web application attack. The attack's main benefit is that it uses port 80 (HTTP's default port), which is always open and not blocked or filtered by the firewall. In this article, the SQL injection attack and ways for exploiting it were examined, and the methods were classified based on the approach used to exploit them. The work on preventing SQL injection attacks has been reviewed, and a novel strategy to preventing such attacks has been proposed and presented.

SQL INJECTION

SQL injection attacks are a form of injection attack in which the attacker inserts SQL commands into the input parameters to affect how the server executes SQL queries (Dalai & Jena, 2017). Attackers exploit situations where developers combine SQL statements with user-submitted parameters, injecting SQL instructions within those parameters to change the pre-defined SQL query. As a result, the attacker can use the application processing layer to execute arbitrary SQL instructions and queries on the database server using the application processing layer (Prokhorenko et al., 2016c). A successful SQL injection attack can access sensitive database data, change data (insert/alter/update/delete), run administrative operations, and get the content of a specified file on the database server, as well as run operating system level commands (Damele & Guimares, 2009). Below is an example of a SQL injection attack. Assume a web page is dynamically produced by using the user's parameter in the URL itself, such as:

http://www.testdomain.com/Admit/Candidates.asp?Sid=199

The SQL query that corresponds to the application code is run, such as

SELECT Name, Branch, Department FROM Candidate WHERE MatricNumber = 199

An attacker might take advantage of the fact that the application accepts the parameter "Sid" and sends it to the database server without any validation or escaping. As a result, the arguments can be tampered with to generate malicious SQL queries. For example, if the variable "Sid" is set to "199 or 2=2," the following URL is generated:

http://www.testdomain.com/Admit/Candidates.asp?Sid=199 or 2=2

The SQL statement will now be transformed into

SELECT Name, Department, Location FROM Candidate WHERE MatricNumber = 199 or 2=2

This condition is always true, and the user will receive all of the Name, Department, and Location triplets. By adding arbitrary SQL instructions, the attacker can further exploit this vulnerability. An attacker may, for example, make a request for the following URL:

http://www.testdomain.com/Admit/ Candidates.asp?Sid=199; DROP TABLE Candidate

The semicolon in the above URL stops the server-side SQL query and adds a new one to be executed. The second query is "*DROP TABLE Candidate*" which deletes the table from the database server. An attacker can utilize the "*UNION SELECT*" query to retrieve data from additional tables in a similar fashion. The *UNION SELECT* command allows the results of two distinct *SELECT* queries to be combined.

Many online applications' default security approach treats SQL queries as trusted commands. As a result, attackers can use this flaw to bypass access restrictions, authorisation, and authentication checks. SQL queries can sometimes be used to obtain server operating system commands through stored procedures. In most cases, the database management server includes stored procedures such as the extended stored procedure. "*xp_cmdshell*" is a Microsoft extended stored procedure that is kept in the master database. This process enables you to use T-SQL code to issue operating system commands straight to the Windows command shell. The output of these commands will be returned to the calling function if it is required. As a result, the attacker in the preceding example can set the value of "Sid" to "*199; EXEC master..xp cmdshell dir – –*"; if run, this will return a list of files in the SQL Server process' current directory. The attacker can load and read arbitrary files from the server using *LOAD FILE('xyz.txt')* in MySQL.

STEPS FOR EXPLOITING VULNERABILITIES

Reconnaissance, enumeration, data extraction, and command execution are some of the steps that may be taken to attack the SQL injection vulnerability. The stages are outlined in full below. In this write up Microsoft SQL Server is being used as the main database backend throughout.

Reconnaissance

It is the first and most crucial step in optimizing an application's potential. It's a technique for fingerprinting the technologies used, which helps the attacker conduct a SQL injection attack more successfully. When database server error messages are sent to the client, they may reveal a lot of information about the database server technology that the web application is utilizing. However, if the web application displays a verbose error message supplied by the database, the query "*SELECT @@version*" can be used to acquire precise information about the back-end database server, such as the specific version and patch level. Security and Communication Networks 3 would show up on the screen

Microsoft OLE DB Provider for SQL Server error '80040e0x'Microsoft][ODBC SQL ServerDriver][SQL Server] Conversion failed when converting the varchar
value 'Microsoft SQL Server 2008 -9. 0x.13xx.0x (Intel X86) Nov 15 2008
00:33:37 Copyright (c) 198X-2008 Microsoft-Corporation Express Edition on Windows NT 5.5 (Build 379X: Service Pack 2X)' to data type int. /Candidatesx. aspx, line 213

This demonstrates that the victim's back-end is Microsoft SQL Server 2008. It also contains information about the host operating system and the precise build level. As a result, similar approaches may be used to produce more accurate fingerprints for additional bits of information, such as shown in the following table 1.

Table 1: Reconnaissance: Queries which can be used to exploit detailed information about backend servers

	Query	Description
1	@@version	Ver. For DBMS
2	db name()	Database name
3	@@servername	Server name for MS-SQL installation
4	@@language	language name
5	@@spid	ID of current user's Process

Enumeration

To carry out a successful attack and completely exploit the SQL injection vulnerability, you must first list all of the database's tables and column names. Metadata is information on all of the system and user-defined tables that is stored in a database management system's pre-defined tables. As a result, in order to enumerate the database server's tables/columns, the attacker must first get access to those tables. Table 2 presents the queries to retrieve the database name, table name, and column name.

Table 2: Enumeration queries which can be used to exploit detailed information about Databases, Tables and Columns

	Extract	Query
1	Databases	select name from master..sysdatabases
2	Tables	SELECT name FROM Databasename..sysobjects WHERE xtype='U'
3	Columns	SELECT name FROM Databasename..syscolumns WHERE id = (SELECT id FROM Databasename..sysobjects WHERE name = 'Tablename')

Extraction of data

After determining the column, table, and database names, the following step is data extraction from the tables. The "*UNION SELECT*" query is used to extract the data. The number of columns in the injected query must match the number of columns in the preexisting *SELECT* query in the *UNION SELECT* statement. We may use an *ORDER BY* statement to get the precise number of columns in an existing query. The query must be run again and again until it runs without errors and the number of columns is revealed by the last successfully completed query. The number of columns may also be determined by progressively increasing the number of columns in the "*UNION SELECT*" expression until the query executes successfully, as shown below.

http://www.testdomain.com/Admit/Candidates.asp?Sid=199+union+select+1--
http://www.testdomain.com/Admit/Candidates.asp?Sid=199+union+select+1,2--
http://www.testdomain.com/Admit/Candidates.asp?Sid=199+union+select+1,2,3--

The *UNION* operator combines two *SELECT* queries into one and displays the result. Consequently, you may use the *UNION SELECT* query to acquire the data you need from the database server. The command will then be executed in the next stage. This phase comprises exploiting the injection flaw to execute system commands. To perform system commands, the current user must have elevated privileges. To run system commands in MS-SQL, use *xp cmdshell*, such as *exec master.xp cmdshell 'ipconfig'*.

TYPES OF SQL INJECTION ATTACKS

As numerous research (Al-Khashab et al., 2011; Buehrer et al., 2005; Liu et al., 2009; Yeole & Meshram, 2011) have shown, there are various forms of SQL injection attacks. These attack types were normally called after the techniques used to exploit the injection vulnerability. Tautology, addition of comments, type mismatch, query piggy bank, union query, stored procedure function and inference techniques are discussed in Table 3 below.

Table 3: Types of SQL Injection Attacks, Sample Codes and Explanations

Attack Type	Sample Code	Brief Explanation
1.Tautology	*"Select from admin where user id = ' ' and password = ' ' or 'a' Equals 'a' "*	A tautology is a logical assertion that is TRUE regardless of the interpretation (cite,xxxx). The same principle is utilized in SQL queries in the conditional statement, in the *WHERE* clause, to make it always TRUE and return all data. To conduct the injection attack, this is frequently placed in the susceptible parameter. Tautology is mostly used to get around the login authentication process. Blind SQL injection vulnerability is also confirmed using tautology.
2.Addition of comments	*"SELECT ∗ from admin where userid= 'xxx';* *-- and password ='yyy';"*	SQL, like other programming languages, allows you to include a comment line in your code. The code can be commented by adding a double hyphen in MS-SQL or a # in MySQL. The code is not executed because of the comment line. The attackers take advantage of this by inserting a comment in the susceptible parameter, which disables the rest of the code that follows the vulnerable parameter. The following is a simple example of how to use a comment line. The above code can bypass the login authentication by giving only valid user id.
3.Type Mismatch	*"http://www.testdomain.com/Admit/Candidates.asp?Sid=system user* *The error output is like [Microsoft][ODBC SQL Server Driver][SQL Server] error: xxx, Conversion failed when converting the varchar value 'sa' to data type integer"*	The "Type mismatch in expression" error indicates that Access cannot match an input value to the data type it expects for the value. For example, if you give Access a text string when it is expecting a number, you receive a data type mismatch error. In case of type mismatch in the query, SQL provides a verbose error message, for instance, from the above error message, we can clearly know that the current user is 'sa'; hence, the attacker takes advantage of this and provides type mismatch queries like giving characters to a numeric type and vice versa and can easily extract a lot of information.
4.Query Piggyback		A query that is stacked or piggybacked query is one that executes a series of SQL queries in a single connection to the database server. When opposed to merely injecting code into the original query, the ability to terminate the old query and attach a whole new one while leveraging the fact that the database server would execute both of them gives the attacker greater freedom and options. The stacked query is supported by the majority of database management systems. For *ALTER*, *DELETE*, and other operations, stacked queries

		can be constructed and performed. This can have a significant influence on the back-end database.
5.Union Query	*“http://www.testdomain.com/Admit/UNION SELECT Candidates.asp?Sid=199 FROM LOGIN, USERID, AND PASSWORD;”*	This is a query that joins two or more tables together. The union operator takes the results of two *SELECT* queries and merges them into a single result. As a consequence, after enumerating the table and column names, the vulnerable parameter may be used to inject the *UNION SELECT* statement, which will combine the results with the original query and obtain the data. The following is an example of how to use *UNION SELECT*. The userid and password pair will be combined with the original query and shown to the client in the above request. The query may be tweaked further to loop through all of the rows in the login database.
6.Stored Procedure Functions	*“SELECT password hash FROM logins.sys.sqlhttp://www.testdomain.com/Admit/Candidates.asp?Sid=199+union+select+master.varbintohexstr(password hash)+ dbo.fn where +name=‘sa'+from+sys.sql+logins”*	A stored procedure in a database management system is a collection of SQL statements that are concatenated to form a process that is saved in the data dictionary. Stored procedures are available in compiled form, allowing many programs to share them. The usage of stored procedures can help with productivity, data integrity, and data access control. These stored procedures can be used by the attacker to have a significant influence on the SQL injection attack. The stored procedure exec master is an example of how to use it. ‘*ipconfig*' in xp cmdshell xp cmdshell is an MSSQL extended stored procedure that allows administrators to perform operating system level commands and obtain the necessary results. SQL injection can also be aided by the usage of system defined functions. The sql logins view in SQL Server 2005 stores hashes. The query may be used to get the system hash. The method fn *varbintohexstr()* transforms the password hash saved in varbinary form to hex so that it can be viewed in a browser, and then it is decrypted into plain text using tools like "Cain and Abel."
7.Inference	*“http://www.testdomain.com/ Admit/Candidates.asp?Sid=199 and SUBSTRING(user name(),1,1)=‘c' SUBSTRING(user name(),1,1)=‘c' SUBSTRING(user name — "*	The act or process of drawing logical conclusions is known as inference (cite, xxxx). We use inference to extract information from time to time; for example, “if we receive this output, then this may be occurring at the back-end.” By observing the answer to a given query, inference methods can extract at least one item of data. The key is observation, since when the query is true, the answer will have a different signature than when it is false. The states of False and True are determined by the response on the page after each request is received; that is, if the response includes the phrase "no records exist," the state was False; otherwise, the state was True. Similarly, starting with the letter "a" and going through the alphabet, we can deduce all subsequent characters of the USER name by repeating the technique.

ALTERNATIVE METHODS

Input filters are frequently used in web applications to defend against common threats such as SQL injection. Attackers may employ encoding techniques to get around such filters. Case variation, URL encoding, CHAR function, dynamic query execution, null bytes, layering striped expressions, exploiting truncation, and other techniques are used to achieve the approach. The attacker gets through the defense measures by employing the methods listed above. The following are some examples of how to use alternative approaches.

Several strategies for avoiding SQL injection attacks have been developed. One of the most recent security trends is the focus on the security of smart devices that use the Android operating system. Some recent papers (Azfar et al., 2014, 2015, 2016a, 2016c, 2016b, 2017) demonstrate approaches for maintaining security in an Android context. Security in web applications, on the other hand, cannot be overlooked due to its widespread use. These strategies are presented and described in Table 4: Strategies for Avoiding SQL Injection Attacks (Refer to the table 4)

Table 4: Strategies for Avoiding SQL Injection Attacks

	Strategy	Description	References
1	Static Evaluation.	Some techniques depend solely on static examination of source code. These approaches examine the program and utilize heuristics or information flow analysis to find code that is vulnerable to SQL injection attacks. Before being included into the query, each and every user input is scrutinized. These approaches can yield false positives due to the inaccuracy of the static analysis that is being performed. Furthermore, because the approach depends on declassification criteria to turn untrustworthy data into more reliable data, it may result in false negatives. To identify whether an application may create questions that include tautologies, Wassermann and Su offer an approach that combines static analysis and automated reasoning techniques. The sorts of SQL injection attacks that this method may identify are restricted.	(Gould et al., 2004; Lam et al., n.d.; Livshits & Lam, 2005; Wassermann & Su, 2004; Xie & Aiken, 2006)
2	Runtime Monitoring and Static Analysis	AMNESIA (Analysis and Monitoring for Neutralizing SQL Injection Attack) is an approach that combines static analysis with runtime monitoring. They create legitimate queries in the static part, which the application can generate automatically. In the dynamic section, the dynamically built runtime questions are monitored and confirmed for compatibility with the static part's queries.	(Halfond & Orso, 2005b, 2005a, 2006)

3	Context-Oriented Approach	Prokhorenko's context-oriented approach provides a unique way for protecting web applications against many sorts of attacks. This paper provides a single general solution for many forms of web application injection attacks. The authors have chosen a different approach to the vulnerability's underlying cause. The typical attack features are examined in this paper, and a context-oriented model for web application protection is built as a result. The presence of a backdoor in the code, on the other hand, may be undetected by the model. The method may not be able to work as intended if code obfuscation, code hiding, and other techniques are used. A general and extendable PHP-oriented protection framework is provided by Prokhorenko et al. The suggested framework is mostly dependent on the application developer's knowledge of his or her intentions. It monitors the execution in real time and detects deviations from the planned behavior, assisting in the prevention of potentially harmful activities. This approach is just for detecting attacks in the PHP environment of 6 Security and Communication Networks. If the application is built with a technology other than PHP, this technique will fail to defend against assaults.	(Prokhorenko et al., 2016a, 2016b)
4	Validation of input.	The incorrect separation of code and input data is the source of many injection problems. As a result, different approaches based on input validation have been presented. Controlling the flow of user input through the secure gateway is done using Security Policy Descriptor Language (SPDL) By imposing user input limitations, the defined policy analyzes and modifies each request/response. PowerForms all utilize a similar approach. These signature-based methods may have insufficient input validation processes, resulting in false positives. Because these methods are reliant on humans, determining the data that needs to be filtered and the policy that should be implemented takes a lot of time.	(Brabrand et al., 2000; Kareem et al., 2021; Khalaf et al., 2021; D. J. Scott, 2005; D. Scott & Sharp, 2002)
5	Randomization of Instruction Sets.	Each term and operator in all SQL statements in the programme code is assigned a random token using a technique called SQLrand. The query is double-checked before being submitted to the database to ensure that all operators and keywords include the token. The assaults would be readily identified because the attacker's operators and keywords do not contain that token. This method is cumbersome because it requires randomising both the underlying SQL parser in the database and the SQL statements in the computer code. When the random tag is applied to the entire SQL statement and each phrase, the query becomes arbitrarily long. It's also vulnerable to brute-force attacks if you use this method.	(Boyd & Keromytis, 2004)

6	Anomaly Detection or Learning-Based Methods	To learn all of the required query structure statically or dynamically a variety of learning-based methods has been suggested. The integrity or precision of the learning algorithms determines how successful detection is.	(Asmawi et al., 2008; Halfond & Orso, 2005a; Jia Yew et al., 2014; Lee et al., 2002; Valeur et al., 2005)

PERFORMANCE METRICS

The Table 5 describes performance metrics which can be utilized to evaluate the effectiveness of the suggested model were the False Acceptance Rate (FAR), Genuine Acceptance Rate (GAR), False Rejection Rate (FRR), Receiver Operating Characteristics (ROC) curve, and Area Under ROC curve (AUC). For the full description refer to Table 5: Performance Evaluation Metrics for Model Effectiveness.

Table 5: Performance Evaluation Metrics for Model Effectiveness

	Performance Metric	Description	Reference
1	Rate of False Acceptance (FAR).	The frequency of attack vectors that can get past the attack detection mechanism is measured in FAR. This statistic is intended to assess how well the suggested strategy performs in the attack detection mode. When the application server's security system fails to intercept a malicious web request, the query with SQL injection code is transmitted to the database server for execution, resulting in a false acceptance.	(Arora & Kumar, 2021; Chakladar et al., 2021; Chandra et al., 2021; J. Chen et al., 2021; Hammad & Wang, 2019; Tomar & Singh, 2021; Wan et al., 2021)
2	Genuine Acceptance Rate (GAR).	The frequency of acceptance in relation to the legitimate web requests provided for execution is referred to as GAR. These data are intended to assess the proposed approach's performance when utilized in attack verification mode. When a legitimate web request is categorized as a regular (nonattack) pattern, it is said to be genuine.	(Hammad & Wang, 2019)
3	Rate of False Rejection (FRR).	The frequency of rejections in relation to the number of valid web requests that should be forwarded for execution is referred to as the FRR. These data are used to evaluate how well the proposed method performs in the verification mode. When a legitimate web request is misclassified as malicious, a false rejection occurs.	(Arora & Kumar, 2021; Chakladar et al., 2021; D. Chen et al., 2021; J. Chen et al., 2021; Hammad & Wang, 2019; Wan et al., 2021)

4	Operational Characteristics of the Receiver (ROC).	The ROC curve shows the relationship between GAR (Genuine Acceptance Rate) and FAR as the threshold value changes. Linear, logarithmic, and semilogarithmic scales are used to plot the curve.	(Chandra et al., 2021; Hammad & Wang, 2019; Tomar & Singh, 2021; Wan et al., 2021)
5	The Surface of the ROC Curve (AUC).	The Area Under the Curve (AUC) is the percentage of coverage under the ROC curve. The more coverage the system has, the more accurate it is. In an ideal world, GAR = 1 at FRR = 0 for a system with 100 percent accuracy, resulting in AUC = 100 percent.	(Natole, 2020; Yang et al., 2021; Yuan et al., n.d.)
6	Error Rates Are Equal (EER).	The EER is the point on a ROC curve when the FAR and FRR are equal. As a result, a lower EER implies higher performance. Table 1 and Figure 8 show that the suggested technique produces good results. The suggested solution is then compared to current strategies in terms of their ability to fight against different forms of SQL injection attacks. The results demonstrate that the proposed model outperforms its competitors. The results of comparisons with known techniques are summarized in Table 2. It is obvious that the suggested technique is resistant to all forms of SQL injection attacks.	(Chandra et al., 2021; Hammad & Wang, 2019; Tomar & Singh, 2021)

CONCLUSION

SQL injection attacks remain at the top long-term running threat to the web and it resources. Presented is a review of are the steps which include reconnaissance, enumeration, and extraction of data. Also discussed are types of injection attacks including tautologies, addition of comments, type mismatch, query piggy backing, union query, inference, and stored procedures. some alternative procedures for mitigating SQL attacks were also discussed which included static evaluation and static analysis, runtime monitoring, content-oriented approach, validation of input, randomization of instruction sets, anomaly detection and learning based methods. Performance metrics for measuring the effectiveness of SQL injection mitigation techniques were presented last which included, rate of false acceptance (FAR), genuine acceptance rate (GAR), rate of false rejection (FRR), Operational characteristics of the receiver (ROC)surface of ROC curve (AUC) and equal error rates. To check for fraudulent input, the paper suggests successively extracting the intended user input from the dynamic query string.

AUTHOR BIOGRAPHY

Aduragbemi David Ogundijo is student of the postgraduate programme PhD (Information Technology) at Infrastructure University Kuala Lumpur (IUKL) Faculty of Engineering, Science and Technology. He holds a BSc in IT (Software Engineering), MSc in Information Systems. His research interests are mainly in mitigating SQL Injection Attacks Email: aduragbemi.ogundijo@gmail.com.

Atiff Abdalla Mahmoud Arabi is student of the postgraduate programme PhD (Information Technology) at Infrastructure University Kuala Lumpur (IUKL) Faculty of Engineering, Science and Technology. He obtained his BIT and Masters in IT in Networking from IUKL. His research interests

include Zero Trust, Biometrics Authentication, and Prevention of Network-Based Academic Dishonesty. Email: atiff2009@gmail.com

Tadiwa Elisha Nyamasvisva, **PhD** is a member at the Faculty of Engineering and Science Technology in IUKL. His research interests are in Computer Algorithm Development, Data Analysis, Networking and Network Security, and IT in Education. Email: tadiwa.elisha@iukl.edu.my

REFERENCES

Al-Ahmad, H., Atan, R., Azim, A., Ghani, A., & Murad, M. A. (2014). SOFTWARE MAINTAINABILITY ASSESSMENT BASED ON COLLABORATIVE CMMI MODEL. *Infrastructure University Kuala Lumpur Research Journal*, 2(1).

Al-Khashab, E., Al-Anzi, F. S., & Salman, A. A. (2011). PSIAQOP: Preventing SQL Injection Attacks based on query optimization process. Proceedings of the 2nd Kuwait Conference on E-Services and e-Systems, KCESS'11. https://doi.org/10.1145/2107556.2107566

Aniche, M., Bavota, G., Treude, C., Gerosa, M. A., & van Deursen, A. (2018). Code smells for Model-View-Controller architectures. Empirical Software Engineering, 23(4), 2121–2157. https://doi.org/10.1007/s10664-017-9540-2

Arora, M., & Kumar, M. (2021). AutoFER: PCA and PSO based automatic facial emotion recognition. *Multimedia Tools and Applications,* 80(2), 3039–3049. https://doi.org/10.1007/s11042-020-09726-4

Asmawi, A., Sidek, Z. M., & Razak, S. A. (2008). System architecture for SQL injection and insider misuse detection system for DBMS. Proceedings - International Symposium on Information Technology 2008, ITSim, 3, 4–9. https://doi.org/10.1109/ITSIM.2008.4631942

Azfar, A., Choo, K. K. R., & Liu, L. (2014). A study of ten popular Android mobile VoIP applications: Are the communications encrypted? Proceedings of the Annual Hawaii International Conference on System Sciences, 4858–4867. https://doi.org/10.1109/HICSS.2014.596

Azfar, A., Choo, K. K. R., & Liu, L. (2015). Forensic taxonomy of popular Android mHealth apps. 2015 Americas Conference on Information Systems, AMCIS 2015, August, 13–15.

Azfar, A., Choo, K. K. R., & Liu, L. (2016a). An Android Communication App Forensic Taxonomy. *Journal of Forensic Sciences, 61*(5), 1337–1350. https://doi.org/10.1111/1556-4029.13164

Azfar, A., Choo, K. K. R., & Liu, L. (2016b). An android social app forensics adversary model. Proceedings of the Annual Hawaii International Conference on System Sciences, 2016-March, 5597–5606. https://doi.org/10.1109/HICSS.2016.693

Azfar, A., Choo, K. K. R., & Liu, L. (2016c). Android mobile VoIP apps: a survey and examination of their security and privacy. *Electronic Commerce Research, 16*(1), 73–111. https://doi.org/10.1007/s10660-015-9208-1

Azfar, A., Choo, K. K. R., & Liu, L. (2017). Forensic taxonomy of android productivity apps. Multimedia Tools and Applications, 76(3), 3313–3341. https://doi.org/10.1007/s11042-016-3718-2

Brabrand, C., Møller, A., Christensen, R. M., & Schwartzbach, M. I. (2000). PowerForms: Declarative Client-Side Form Field Validation. BRICS Report Series, 7(43), 1–20. https://doi.org/10.7146/brics.v7i43.20210

Buehrer, G., Weide, B. W., & Sivilotti, P. A. G. (2005). Using parse tree validation to prevent SQL injection attacks. SEM 2005 - Proceedings of the 5th International Workshop on Software Engineering and Middleware, September, 106–113. https://doi.org/10.1145/1108473.1108496

Chakladar, D. das, Kumar, P., Roy, P. P., Dogra, D. P., Scheme, E., & Chang, V. (2021). A multimodal-Siamese Neural Network (mSNN) for person verification using signatures and EEG. Information Fusion, 71, 17–27. https://doi.org/10.1016/j.inffus.2021.01.004

Chandra, S., Singh, K. K., Kumar, S., Ganesh, K. V. K. S., Sravya, L., & Kumar, B. P. (2021). A novel approach to validate online signature using machine learning based on dynamic features. Neural Computing and Applications, 33(19), 12347–12366. https://doi.org/10.1007/s00521-021-05838-6

Chen, D., Yan, Q., Wu, C., & Zhao, J. (2021). SQL Injection Attack Detection and Prevention Techniques Using Deep Learning. *Journal of Physics*: *Conference Series,* 1757(1). https://doi.org/10.1088/1742-6596/1757/1/012055

Chen, J., Cai, L., Tu, Y., Dong, R., An, D., & Zhang, B. (2021). An Identity Authentication Method Based on Multi-modal Feature Fusion. *Journal of Physics: Conference Series,* 1883(1). https://doi.org/10.1088/1742-6596/1883/1/012060

Dalai, A. K., & Jena, S. K. (2017). Neutralizing SQL injection attack using server side code modification in web applications. *Security and Communication Networks*, 2017. https://doi.org/10.1155/2017/3825373

Gould, C., Su, Z., & Devanbu, P. (2004). JDBC checker: A static analysis tool for SQL/JDBC applications. Proceedings - International Conference on Software Engineering, 26(June 2004), 697–698. https://doi.org/10.1109/icse.2004.1317494

Halfond, W. G. J., & Orso, A. (2005a). AMNESIA: Analysis and monitoring for NEutralizing SQL-injection attacks. 20th IEEE/ACM International Conference on Automated Software Engineering, ASE 2005, 174–183. https://doi.org/10.1145/1101908.1101935

Halfond, W. G. J., & Orso, A. (2005b). Combining static analysis and runtime monitoring to counter SQL-injection attacks. 1–7. https://doi.org/10.1145/1083246.1083250

Halfond, W. G. J., & Orso, A. (2006). Preventing SQL injection attacks using AMNESIA. Proceedings - International Conference on Software Engineering, 2006, 795–798. https://doi.org/10.1145/1134285.1134416

Hammad, M., & Wang, K. (2019). Parallel score fusion of ECG and fingerprint for human authentication based on convolution neural network. *Computers and Security, 81*, 107–122. https://doi.org/10.1016/j.cose.2018.11.003

Jia Yew, T., bin Samsudin, K., Izura Udzir, N., & Jahari bin Hashim, S. (2014). BUFFER OVERFLOW ATTACK MITIGATION VIA TRUSTED PLATFORM MODULE (TPM). Infrastructure University Kuala Lumpur Research Journal, 2(1).

Kareem, F. Q., Ameen, S. Y., Salih, A. A., Ahmed, D. M., Kak, S. F., Yasin, H. M., Ibrahim, I. M., Ahmed, A. M., Rashid, Z. N., & Omar, N. (2021). SQL Injection Attacks Prevention System Technology: Review. *Asian Journal of Research in Computer Science,* 13–32. https://doi.org/10.9734/ajrcos/2021/v10i330242

Khalaf, O. I., Sokiyna, M., Alotaibi, Y., Alsufyani, A., & Alghamdi, S. (2021). Web attack detection using the input validationmethod: Dpda theory. *Computers, Materials and Continua*, *68*(3), 3167–3184. https://doi.org/10.32604/cmc.2021.016099

Lam, M. S., Whaley, J., Livshits, V. B., Martin, M. C., & Carbin, M. (n.d.). Context-Sensitive Program Analysis as Database Queries.

Lee, S. Y., Low, W. L., & Wong, P. Y. (2002). Learning Fingerprints for a. Architecture, 264–279.

Liu, A., Yuan, Y., Wijesekera, D., & Stavrou, A. (2009). SQLProb: A proxy-based architecture towards preventing SQL injection attacks. Proceedings of the ACM Symposium on Applied Computing, 2054–2061. https://doi.org/10.1145/1529282.1529737

Livshits, V. B., & Lam, M. S. (2005). Finding Security Errors in Java Programs with Static Analysis. Proc. Usenix Security Symposium, 271–286.

Natole, M. J. (2020). Fast Optimization Algorithms For AUC.

Paolone, G., Paesani, R., Marinelli, M., & Felice, P. di. (2021). Empirical Assessment of the Quality of MVC Web Applications Returned by xGenerator. https://doi.org/10.3390/computers

Prokhorenko, V., Choo, K. K. R., & Ashman, H. (2016a). Context-oriented web application protection model. Applied Mathematics and Computation, 285, 59–78. https://doi.org/10.1016/j.amc.2016.03.026

Prokhorenko, V., Choo, K. K. R., & Ashman, H. (2016b). Intent-Based Extensible Real-Time PHP Supervision Framework. IEEE Transactions on Information Forensics and Security, 11(10), 2215–2226. https://doi.org/10.1109/TIFS.2016.2569063

Scott, D. J. (2005). Abstracting application-level security policy for ubiquitous computing. http://www.cl.cam.ac.uk/

Scott, D., & Sharp, R. (2002). Developing secure web applications. IEEE Internet Computing, 6(6), 38–45. https://doi.org/10.1109/MIC.2002.1067735

Tomar, P., & Singh, R. C. (2021). Cascade-based Multimodal Biometric Recognition System with Fingerprint and Face. Macromolecular Symposia, 397(1). https://doi.org/10.1002/masy.202000271

Valeur, F., Mutz, D., & Vigna, G. (2005). A learning-based approach to the detection of SQL attacks. Lecture Notes in Computer Science, 3548(Detection of Intrusions and Malware, and Vulnerability Assessment: Second International Conference, DIMVA 2005. Proceedings), 123–140. https://doi.org/10.1007/11506881_8

Wan, J., Chen, Y., & Bai, B. (2021). Joint feature extraction and classification in a unified framework for cost-sensitive face recognition. Pattern Recognition, 115. https://doi.org/10.1016/j.patcog.2021.107927

Wassermann, G., & Su, Z. (2004). An analysis framework for security in Web applications. SAVCBS 2004 Specification and Verification of Component-Based Systems, 70.

Xie, Y., & Aiken, A. (2006). Static detection of security vulnerabilities in scripting languages. 15th USENIX Security Symposium, 179–192.

Yang, Z., Xu, Q., Bao, S., He, Y., Cao, X., & Huang, Q. (2021). When All We Need is a Piece of the Pie: A Generic Framework for Optimizing Two-way Partial AUC framework Image Processing View project Saliency Detection with Comprehensive Information View project When All We Need is a Piece of the Pie: A Generic Framework for Optimizing Two-way Partial AUC. https://www.researchgate.net/publication/354047024

Yeole, A. S., & Meshram, B. B. (2011). Analysis of different technique for detection of SQL injection. International Conference and Workshop on Emerging Trends in Technology 2011, ICWET 2011 - Conference Proceedings, Icwet, 963–966. https://doi.org/10.1145/1980022.1980229

Yuan, Z., Guo, Z., Xu, Y., Ying, Y., & Yang, T. (n.d.). Federated Deep AUC Maximization for Heterogeneous Data with a Constant Communication Complexity. Retrieved January 12, 2022, from www.libauc.org