SECURITY TESTING OF WEB APPLICATIONS FOR DETECTING AND REMOVING SECOND-ORDER SQL INJECTION VULNERABILITIES

By

NAJLA'A ATEEQ MOHAMMED DRAIB

Thesis Submitted to the School of Graduate Studies, Universiti Putra Malaysia, in Fulfilment of the Requirements for the Degree of Doctor of Philosophy

November 2022

FSKTM 2022 27

DEDICATION

بِسْمِ ٱللَّٰهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

This work is dedicated to my husband, Ahmed Ali Alsulaimani, for his continuous support, patience, understanding, love, and tolerance. Without his support, this work would not have been possible.
And
to my beloved family, my parents, my kids, my brothers and my sister for their endless love, prayer, and support.

Abstract of thesis presented to the Senate of Universiti Putra Malaysia in fulfilment of the requirement for the degree of Doctor of Philosophy

SECURITY TESTING OF WEB APPLICATIONS FOR DETECTING AND REMOVING SECOND-ORDER SQL INJECTION VULNERABILITIES

By

NAJLA'A ATEEQ MOHAMMED DRAIB

November 2022

Chairman : Professor Abu Bakar Md Sultan, PhD
Faculty : Computer Science and Information Technology

Structured query language injection vulnerability (SQLIV) is one of the most prevalent and severe web application vulnerabilities. It is usually exploited by SQL injection attacks (SQLIA) for the purpose of gaining unauthorised access to the back-end databases by altering the original SQL statements through input data manipulation. A successful attack can hinder integrity, privacy, and information availability in the database. As a particular type of SQL injection (SQLI), the second-order SQLIA tends to be more severe and difficult to detect. It has a more significant impact on the back-end database than the first-order SQLIA, simply because its respective SQL injection is seeded first into the application's persistent storage, which is usually deemed a trusted source, before its actual exploitation. In order to protect a web application from a malicious user, test procedures for identifying and removing SQLIVs must be implemented earlier in the software development life cycle (SDLC) of web applications, specifically before bringing it onto production and possibly becoming available to a malicious attack. Critically, several efforts have been devoted to detecting SQLIVs and preventing their exploitation, and the majority focused on approaches that address the detection of first-order SQL injection vulnerabilities. However, the mechanisms needed to detect first-order SQLIV, which may lead to SQLIA on the application level, may not afford to catch second-order SQLIV. This is specifically because the malicious inputs supplied by the attacker can be concatenated with the SQL statement at the database level. Moreover, the existing techniques only reported the detected vulnerabilities, and they left their removal as a burden on the programmer. As far as the literature shows, none of the current automated methods exhibited the ability to deal with this phenomenon. Hence, the actual fixing process of any vulnerabilities is left for the human developer to handle. However, manual removal of such vulnerabilities is tedious, error-prone, and costly. Second-order injections are also difficult to prevent as the point of injection differs from the point of attack, and therefore more care should be taken to detect and prevent them. Both attack points should be validated carefully (i.e., point of injection and point of attack). In order to address the weaknesses above and the identified research gaps, this study invents a white-box testing technique for automated detection and removal of the second-order SQLIVs in web applications using source code static analysis. Static analysis is devoted to identifying candidate pairs of vulnerable paths to second-order SQLI. It statically detects when the data comes from tainted sources, when they are stored in the back-end database, and when they are retrieved later in another point to build a new SQL statement without proper sanitisation. This technique also applies the removing algorithm, which uses escaping method to remove the detected vulnerabilities. The prototype tool, called Second-order SQL injection Protector (SoSQLiP), was developed and implemented to test the proposed technique. The test was conducted using eleven PHP Web applications: ten applications available on the internet and that other researchers have used and one application that the researcher developed. The results were empirically evaluated with an existing tool to determine the effectiveness of the automatic detection of second-order SQLIVs. Promising results have been obtained from both of these evaluations. The experiments show that the proposed technique has a detection rate of 100% and a vulnerability removal rate of 100%. The proposed technique has shown a better vulnerability detection rate than the state-of-the-art tool (i.e., SQLMAP). However, future studies should expand the scope of the research to include more types of vulnerabilities, such as second-order XSS vulnerabilities.

Abstrak tesis yang dikemukakan kepada Senat Universiti Putra Malaysia sebagai memenuhi keperluan untuk ijazah Doktor Falsafah

PENGUJIAN KESELAMATAN APLIKASI WEB BAGI MENGESAN DAN MENGHAPUS KERENTANAN SUNTIKAN SQL PERINGKAT KEDUA

Oleh

NAJLA'A ATEEQ MOHAMMED DRAIB

November 2022

Pengerusi : Profesor Abu Bakar Md Sultan, PhD
Fakulti : Sains Komputer dan Teknologi Maklumat

Kerentanan suntikan bahasa pertanyaan berstruktur (SQLIV) adalah salah satu kerentanan aplikasi web yang sangat lazim dan teruk. Ia biasanya dieksploitasi oleh serangan suntikan SQL (SQLIA) untuk tujuan mendapatkan capaian yang tidak dibenarkan ke pangkalan data bahagian belakang dengan mengubah arahan-arahan SQL asal melalui cara manipulasi data input. Serangan yang berjaya boleh menghalang integriti, privasi dan ketersediaan maklumat dalam pangkalan data. Merujuk kepada jenis suntikan SQL tertentu (SQLI), SQLIA peringkat kedua cenderung lebih teruk dan sukar dikesan. Ia mempunyai kesan yang lebih ketara pada pangkalan data bahagian belakang berbanding SQLIA peringkat pertama kerana suntikan SQL tersebut dimasukkan terlebih dahulu ke dalam storan kekal aplikasi, yang biasanya dianggap sebagai sumber yang dipercayai, sebelum eksploitasi sebenar. Untuk melindungi aplikasi web daripada pengguna yang berniat jahat, prosedur ujian untuk mengenal pasti dan mengalih keluar SQLIV wajib dilaksanakan terlebih dahulu dalam kitaran hayat pembangunan perisian (SDLC) aplikasi web, terutamanya sebelum membawanya ke fasa produksi dan kemungkinan terdedah kepada serangan berniat jahat. Secara kritikalnya, beberapa usaha telah diusahakan untuk mengesan SQLIV dan mencegah eksploitasinya dan kebanyakannya memberi tumpuan kepada pendekatan pengesanan kerentanan suntikan SQL peringkat pertama. Bagaimanapun, mekanisme yang diperlukan untuk mengesan SQLIV peringkat pertama yang mungkin membawa kepada SQLIA pada tahap aplikasi mungkin tidak mampu untuk menangkap SQLIV peringkat kedua. Terutamanya adalah kerana input berniat jahat yang dibekalkan oleh penyerang boleh digabungkan dengan pernyataan SQL di aras pangkalan data. Tambahan lagi, teknik yang sedia ada cuma melaporkan pengesanan kerentanan, dan mengabaikan penyingkirannya sebagai beban pada pengaturcara. Setakat yang ditunjukkan oleh kajian terdahulu, tiada kaedah automatik semasa yang berupaya untuk menangani fenomena ini. Oleh itu, proses pembaikan sebarang kerentanan ditinggalkan untuk dikendalikan oleh pembangun perisian. Bagaimanapun, penyingkiran kerentanan secara manual tersebut amat rumit, terdedah kepada kesilapan, dan mahal. Suntikan SQLIV peringkat kedua juga sukar untuk dicegah kerana titik suntikan adalah berbeza dari titik serangan, dan oleh itu lebih banyak usaha diperlukan untuk mengesan dan mencegahnya. Kedua-dua titik serangan perlu disahkan dengan teliti (iaitu, titik suntikan dan titik serangan). Bagi menangani kelemahan di atas dan jurang penyelidikan yang dikenalpasti, kajian ini mencipta teknik ujian kotak putih untuk pengesanan automatik dan penyingkiran SQLIV peringkat kedua dalam aplikasi web menggunakan analisis statik kod sumber. Analisis statik dikhaskan untuk mengenal pasti pasangan calon laluan rentan ke SQLI peringkat kedua. Ianya mengesan secara statik apabila data berasal dari sumber yang tercemar, bila ianya disimpan dalam pangkalan data bahagian belakang, dan apabila ia dicapai semula dalam bentuk yang lain untuk membina pernyataan SQL baharu tanpa pembersihan yang betul. Teknik ini juga menggunakan algoritma penyingkiran yang menggunakan kaedah pengelakan untuk menghapus kerentanan yang dikesan. Alat prototaip, yang dipanggil Pelindung suntikan SQL Peringkat Kedua (SoSQLiP), telah dibangunkan dan dilaksanakan untuk menguji teknik yang dicadangkan. Ujian ini dijalankan menggunakan sebelas aplikasi Web PHP: sepuluh aplikasi yang terdapat di internet dan yang telah digunakan oleh penyelidik lain, dan satu aplikasi yang dibangunkan oleh penyelidik. Hasilnya telah dinilai secara empirikal dengan tool sedia ada untuk menilai sejauh mana efektifnya pengesanan otomatik SQLIV peringkat kedua. Keputusan-keputusan yang memberansangkan telah diperolehi daripada kedua-dua penilaian. Experimentasi menunjukkan teknik yang dicadangkan mencapai tahap pengesanan 100% and kadar penyingkiran kerentanan juga 100%. Teknik yang dicadangkan juga menunjukkan pengesanan keretanan lebih baik berbanding tool canggih (contoh., SQLMAP). Walau bagaimanapun, kajian masa hadapan harus memperluaskan skop penyelidikan dengan mengambil kira lebih banyak jenis kerentanan, seperti kerentanan XSS peringkat kedua.

ACKNOWLEDGEMENTS

In the name of Allah, the most gracious, the most merciful.

All praise and glory to Almighty Allah (S.W.T.) for His greatness and for giving me the strength, health, opportunity, and endurance to carry out this work.

I would like to express my unrestrained appreciation to my supervisor Professor Dr. Abu Bakar Md Sultan, for his patience, generous guidance, and encouragement in carrying out the research work. My appreciation also goes to my co-supervisor, Professor Dr. Hazura Zulzalil, for her constant support, advice, and expertise, and to my late co-supervisor Professor Dr. Abul Azim bin Abd Ghani for his guidance in the right direction and for his inspiration which kept me moving forward in my research.

A big thank you goes to Dhamar University, Yemen, for providing me with a scholarship to pursue a PhD at Universiti Putra Malaysia. Their financial and moral support made my study possible.

I also acknowledge the effort of the entire staff of Universiti Putra Malaysia in the Faculty of Computer Science and Information Technology, School of Graduate Studies, to understand the difficulties I faced and gave me the chance to complete my PhD project, and for all support given. I also would like to express my special gratitude to my labmate, Isatu Hydara, whom I travelled with, shared knowledge and encouraged each other in this long journey.

I would not have been able to succeed in this study without the endless support of my family and friends, for whom I have the utmost love and appreciation. Particular appreciation goes to my husband for his wonderful and endless love and support.

A million thanks to all the friends I met during my study at UPM for making this place a second home away from home. Finally, I thank ALLAH (S.W.T), for HE always directs my path, answers my prayers, accepts my supplications, and guides me to be a successful and helpful member of the Muslim Ummah.

This thesis was submitted to the Senate of Universiti Putra Malaysia and has been accepted as fulfilment of the requirement for the degree of Doctor of Philosophy. The members of the Supervisory Committee were as follows:

Abu Bakar bin Md Sultan, PhD
Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Chairman)

Abdul Azim bin Abd Ghani, PhD
Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

Hazura binti Zulzalil, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

ZALILAH MOHD SHARIFF, PhD
Professor and Dean
School of Graduate Studies
Universiti Putra Malaysia

Date: 11 May 2023

Declaration by the Graduate Student

I hereby confirm that:

- this thesis is my original work;
- quotations, illustrations and citations have been duly referenced;
- this thesis has not been submitted previously or concurrently for any other degree at any institutions;
- intellectual property from the thesis and copyright of thesis are fully-owned by Universiti Putra Malaysia, as according to the Universiti Putra Malaysia (Research) Rules 2012;
- written permission must be obtained from supervisor and the office of Deputy Vice-Chancellor (Research and innovation) before thesis is published (in the form of written, printed or in electronic form) including books, journals, modules, proceedings, popular writings, seminar papers, manuscripts, posters, reports, lecture notes, learning modules or any other materials as stated in the Universiti Putra Malaysia (Research) Rules 2012;
- there is no plagiarism or data falsification/fabrication in the thesis, and scholarly integrity is upheld as according to the Universiti Putra Malaysia (Graduate Studies) Rules 2003 (Revision 2012-2013) and the Universiti Putra Malaysia (Research) Rules 2012. The thesis has undergone plagiarism detection software

Signature: ________________________ Date: ______________

Name and Matric No: Najla'a Ateeq Mohammed Draib

Declaration by Members of the Supervisory Committee

This is to confirm that:

- the research conducted and the writing of this thesis was under our supervision;
- supervision responsibilities as stated in the Universiti Putra Malaysia (Graduate Studies) Rules 2003 (Revision 2012-2013) are adhered to.

Signature:	
Name of Chairman of Supervisory Committee:	Professor Dr. Abu Bakar bin Md Sultan
Signature:	
Name of Member of Supervisory Committee:	Professor Dr. Abdul Azim bin Abd Ghani
Signature:	
Name of Member of Supervisory Committee:	Associate Professor Dr. Hazura binti Zulzalil

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SQL	Structured Query Language
SQLIV	SQL Injection Vulnerabilities
SQLIA	SQL Injection Attack
SDLC	Software Development Life Cycle
SoSQLiP	Second-order SQL injection Protector
OWASP	Open Web Application Security Project
AST	Abstract Syntax Tree
CFG	Control Flow Graph
WVS	Web application Vulnerability Scanners
SSA	Static Security Analysis
MST	Main Symbol Table
sqlStm	SQL Statement
SoSqli	Second-order SQL Injection

CHAPTER 1

INTRODUCTION

1.1 Research background

Web applications are nowadays the backbone of the modern internet. Their popularity and acceptance are growing rapidly due to the high level of convenience, accessibility and ubiquitous they offer. These online applications are reliable and efficient solutions to business challenges. Aside from delivering information and services and serving as a great communication medium, web applications store and process vast amounts of potentially sensitive data for a large number of users. The processed data is usually stored or retrieved into and from the back-end database. Timely, web application users should move with the back-end database via user interfaces for many tasks: extracting information, making queries, and updating data, among others. Hence, the attacker's unauthorised access to sensitive data can threaten data confidentiality, integrity, and application availability.

Typically, web applications are designed with hard time restrictions; and therefore, they are often deployed with varying degrees of unexpected security vulnerabilities that hackers exploit through different types of attacks (Kaur & Kaur, 2016; Kieyzun et al., 2009; Medeiros et al., 2016).

Due to their high global exposure and the presence of vulnerabilities besides the critical assets that web applications usually store, web applications are considered attractive and ideal targets for security attackers who continue to hunt for vulnerabilities that allow them to pervade an organisation. For example, a vulnerable web application could pave the way for unauthorised access to underlying systems, access to the back-end database, and/or simply cause a denial of service (Olivo et al., 2015).

In the same vein, Structured Query Language Injection Vulnerabilities (SQLIVs) have been consistently top-ranked among web application vulnerabilities for the past few years (Acunetix, 2020; OWASP, 2021; SANS/CWE, 2019; TRUSTWAVE, 2018).

SQLIVs refer to potential software security flaws associated with database-driven web applications that can be exploited by means of SQL Injection Attacks (SQLIAs). Typically, SQLIV takes place in code when user-supplied data (i.e., URL parameters or HTML form input) is allowed to propagate from an input source to a security-critical operation (e.g., database queries) without proper sanitisation. The vulnerability is caused by code fragments where unsensitised input is interpreted as SQL code instead of being treated as data (Johari & Sharma, 2012; Shar & Tan, 2013; Verma & Kaur, 2015). SQLIA is a notorious hacking technique in which the attacker exploits SQLIV of a web application connected to a database to inject SQL code fragments into vulnerable input parameters (e.g., HTTP requests). The malicious code masquerades as user input and is embedded in the SQL query.

The consequences of an SQLIA can be devastating. A successful attack can hinder the privacy, integrity and availability of information in the database. The attacker can use this attack to bypass the authentication process (loss of authentication), extract data from the back-end database (loss of confidentiality), and/or modify existing data (loss of integrity) (Alwan & Younis, 2017; Faker et al., 2017; Hu et al., 2020; Johari & Sharma, 2012; Mishra et al., 2014). According to OWASP Top Ten 2017, the overall security risk to an organisation can be determined based on a number of factors, including the likelihood associated with the threat agent, attack vectors, security vulnerabilities and the business impact on the organisation. Table 1.1 shows attack vectors, security weaknesses, and the impact of SQL injection, as reported by OWASP (2017).

Table 1.1 : OWASP Report on SQL Injection

Threat agent	**Exploitability**	**Security Weakness**		**Impacts**	
		Prevalence	**Detectability**	**Technical Impact**	**Business Impact**
Application-Specific	Easy	Common	Average	Severe	Depending on web application needs and data

Hackers have developed many types of SQLIAs to exploit SQLIVs. Several studies classified these attacks, based on the injection technique, into seven basic types: tautologies, illegal/logically incorrect queries, union query, piggy-backed, stored procedure, alternate encoding, and inference-based attacks (Faker et al., 2017; Johari & Sharma, 2012; Kaur & Kaur, 2016; Singh, 2017).

Practically, SQL injection can be introduced into vulnerable web applications using two main mechanisms based on the injection order: first-order SQL injection and second-order SQL injection (Faker et al., 2017; Halfond et al., 2008; Kim & Lee, 2014; Liu & Wang, 2018).

First-order SQL injection is the primary type of SQL injection attack. In such an attack, the attacker inserts SQL commands into a vulnerable input field that flows directly from an entry point (e.g., $_GET) to a sensitive sink (e.g., mysqli_query). The successful injection results are delivered immediately upon user-input submission. However, if the malicious code has been an argument of an escape method, it can be blocked, but it is stored in the database and can cause second-order SQLIA later. In practice, first-order SQL injection attacks can be launched using any one of the aforementioned attack types by injecting malicious input through user input, cookies, or server vulnerabilities (Faker et al., 2017).

Second-order SQL injection, also called stored or persistent SQLIA is a particular type of SQLIA that is more serious, more difficult to be detected, and has strong concealment (Choudhury et al., 2016; Liu & Wang, 2018; Muraleedharan, 2015; Ping, 2017). This technique can be applied successfully to all kinds of injections mentioned above

(Muraleedharan, 2015). In such attacks, the attacker first seeds SQL commands into the database and then uses that input at a later stage in a sensitive sink for launching the attack. Unlike first-order SQLIA, the malicious code in second-order SQLIA is not initiated immediately. However, it is first stored in the application's back-end database and then retrieved and activated by the victim/attacker (Dahse & Holz, 2014). The security violations of such attacks can be disastrous; they may include identity theft, loss of confidential or sensitive data, taking control of data, and destroying the back-end database (Sharma & Jain, 2014).

A Second-order SQL injection attack is developed on a first-order SQL injection attack. To illustrate, Figure 1.1 presents a typical architecture of the second-order SQLIA.

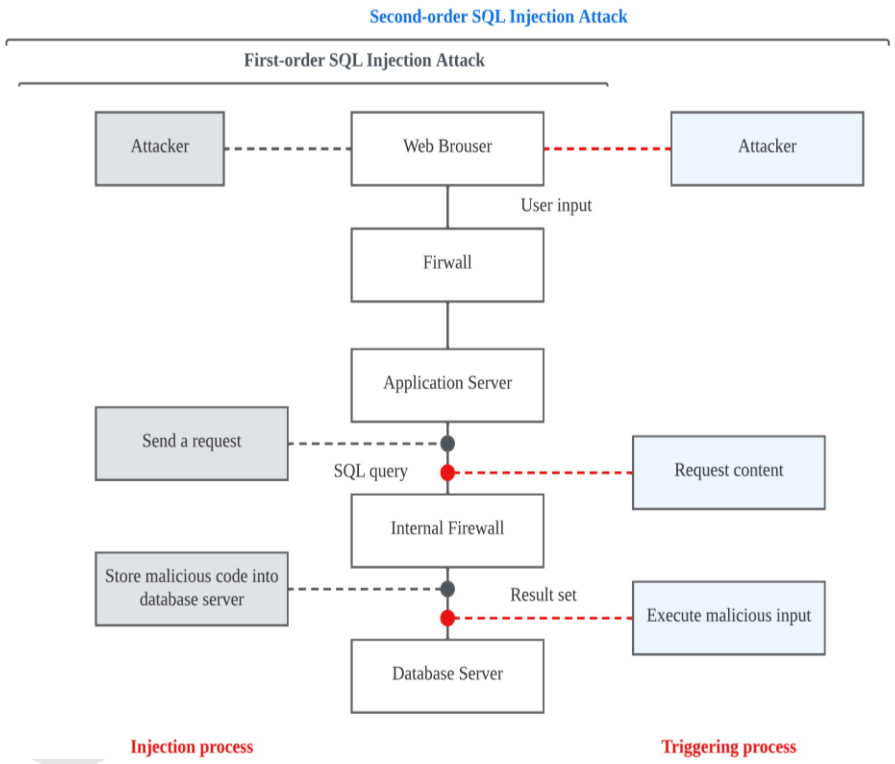

Figure 1.1 : Second-order SQLI mechanism in web application

A technique for detecting SQLIA at the application level cannot defend against the second-order SQLIA attack because the malicious input supplied by the attacker is concatenated with the SQL statement at the database level (Dahse & Holz, 2014). Indeed, second-order vulnerability does not appear when the user submits regular content but requires unique SQL injection attack strings to trigger it. Therefore, ordinary tools can hardly detect second-order vulnerabilities (Liu & Wang, 2018). Furthermore, preventing first-order SQLIVs using available techniques such as prepared statements and escape techniques is not sufficient to prevent second-order SQLIVs. The developer might successfully escape user input and deem it safe. However, later when the data is reused to create different queries, the previously sanitised input may result in a second-order SQL injection attack.

Although different web programming languages provide different data validation mechanisms for protection against SQLIVs, however, they do not guarantee secure web applications. Inexperienced developers and those rushing to get a product to market may not employ language-provided mechanisms properly. Moreover, experienced programmers often create applications with software errors and vulnerabilities.

Therefore, testing online applications for SQLIV detection and removal before deployment is indispensable to safeguarding them from exploitation. As in conventional software applications, software applications' testing always makes it easier to detect and fix errors.

Due to the importance of producing secure web applications, the research community has investigated the area of automated detection and removal of SQLIVs over the years and proposed many approaches to the problem.

Static analysis techniques are among the most widely used approaches for the detection and removal of SQLIVs during the test phase of web application development. Although there has been a considerable number of static analysis techniques for SQLIVs detection, the area of second-order SQLIVs detection has not been adequately explored (Cao et al., 2018; Kronjee et al., 2018; Liu & Wang, 2018; Medeiros et al., 2016; Trinh et al., 2014; Yan et al., 2014a; Yan et al., 2018). In fact, very few static analysis techniques addressed the detection of second-order SQLIVs.

Nevertheless, a thorough investigation of the literature reveals that most of the testing and analysing techniques proposed to automate the SQLIVs assessment were incapable of handling the second-order SQLIVs. They only focused on the detection of first-order SQLIVs since they only analysed SQL queries generated at the application level, but they ignored those generated at the database level. This may be due to two common explanations: First, when the first-order vulnerability is detected and prevented, the second-order vulnerability is not exploitable anymore. Second, when successfully escaped, malicious input is deemed safe. However, the downside of these propositions is that the attack can be launched later in different times and contexts by exploiting the second-order vulnerabilities that make use of that data to create different SQL queries. Indeed, the mechanism to detect SQLIV, which may lead to SQLIA on the application level, may not afford to detect second-order SQLIV as the malicious inputs supplied by the attacker are concatenated with the SQL statement at the database level, not the application level.

In addition, most existing SQLIV removal techniques are predominantly manual Steiner et al. (2017) and Umar et al. (2014a). They only automate fix generation and leave the actual source code modification for applying the auto-generated fix in the hands of the developer, despite the fact that manual bug fixing is prone to errors and human limitations.

Obviously, it would be highly desirable to have a technique that can analyse the source code of vulnerable web applications for identifying the vulnerable paths to second-order SQLIAs and produce a reliable and secure version ready for deployment in to live environment.

This type of technique would reduce the human efforts and expenses associated with the testing phase of web application development, resulting in higher quality software.

The aforementioned weaknesses of existing techniques motivate further research in the area, with the objective of defining an accurate and precise method of achieving automated detection and removal of second-order SQLIVs for web applications.

Consequently,this study introduces a new static analysis technique for the automated detection of second-order SQLIVs. In addition, the technique inserts fixes to remove the detected vulnerabilities automatically.

Static analysis is devoted to identifying possible or candidate pairs of vulnerable paths (target paths) to second-order SQLI. It statically detects when data comes from tainted sources and is stored in the back-end database for the purpose of using and retrieving them again without proper sanitisation. Then, the technique applies escaping techniques to the detected vulnerabilities to remove them.

1.2 Problem Statement

The process of automated web application testing for SQLIVs detection and remediation is particularly delicate, challenging, and costly due to the complex infrastructure of web applications and the extreme heterogeneity of SQL injection attack vectors (Akrout et al., 2014; Di Lucca & Fasolino, 2006; Doğan et al., 2014; Li et al., 2014). Detection and prevention of second-order injections can be particularly difficult because the injection point is located separately from where the attack occurs.

Several black-box vulnerability scanning techniques have been developed to support web application testing for SQLIV detection because they are easy to use, automated, and independent of the underlying web application technology (Akrout et al., 2014; Aliero et al., 2019; Chen & Wu, 2010; Djuric, 2013; Huang et al., 2003; Kals et al., 2006; Patil et al., 2016; Thomé et al., 2014). However, black-box testing techniques cannot guarantee precision and completeness as they do not explore all possible program paths of applications. Moreover, existing black-box vulnerability scanners are limited to detecting first-order SQLIVs, and they are not capable of detecting second-order SQLIVs. This tendency is due to two main reasons: First, black box techniques cannot confirm whether the injected code is already in storage or not. Second, they may have trouble linking the initial injection event with triggering the stored injected code. Furthermore, black-box scanners are based on the idea of knowing little about the internal workings of the application. In the case of a first-order injection, this is less relevant because the scanner can directly verify that the attack worked. However, with a second-order SQL injection, the scanner must not only implement the attack but also has to find a way to force the application to trigger the attack without knowing the application's source code, i.e., it must select the right attack vectors that are able to detect and exploit the second-order SQLIVs.

Empirical evidence has shown that existing black box scanners have difficulty confirming that the attack code has been successfully injected into the database and maintaining the state of the database, which is critical to perform a second pass and search for new pages that would execute the injected attack code and launch the second-order SQLIA (Anagandula & Zavarsky, 2020; Bau et al., 2012; Deepa & Thilagam, 2016; Doupé et al., 2010; Hofman & Ibrahimi, 2022; Khoury et al., 2011; Parvez et al., 2016; Stanford et al., 2010).

White-box testing approach, specifically static source code analysis, is found very attractive in addressing the aforementioned weaknesses of black-box vulnerability scanners and their inability to support code modification for automated vulnerability removal. However, the research on second-order SQL injection technology and the detection accuracy of the existing static analysis techniques for second-order vulnerabilities is either unsatisfactory or such vulnerabilities are completely overlooked (Dolatnezhad & Amini, 2019; Fernando & Abawajy, 2013; Ping, 2017; Saidu Aliero et al., 2015; Xiao et al., 2017; Yan et al., 2014b).

Existing static analysis approaches utilise taint analysis and similar code analysis techniques to detect SQLIVs by tracking the flow of intruders or tainted input values throughout the application itself (Backes et al., 2017; Jovanovic et al., 2006; Medeiros et al., 2016; Su & Wassermann, 2006; Xie & Aiken, 2006; Yan et al., 2018). However, these techniques are not able to track the flow of input values across databases until the final query, which makes it difficult to detect second-order SQLIVs. The attacker can store malicious code in the database and trigger its execution at a later time by exploiting improper sanitisation of the data retrieved from the database, resulting in a second-order SQL injection attack. In the context of this thesis, improper sanitisation refers to the failure to adequately filter, validate, or otherwise handle user-supplied input before it is stored in the database or after it is retrieved from the database in a manner that ensures that the data can be safely used to construct an SQL command.

Despite several existing studies on the applicability of taint analysis techniques, to the best of our knowledge, very few works, such as Dahse & Holz (2014) and Yan et al. (2014a), have addressed the automated detection of second-order SQL injection vulnerabilities, while none have targeted the automated removal of second-order SQL injection vulnerabilities.

Unfortunately, existing approaches to remediating SQLIVs can be divided into two extremes: On the one hand, some approaches only identify the vulnerability and then implement or generate a fix that can address the vulnerability without modifying the underlying code, leaving its remediation to the programmer (Abadi et al., 2011; Dysart & Sherriff, 2008; Mui & Frankl, 2010; Panda, 2017; Rafnsson et al., 2020; Scholte et al., 2012; Siddiq et al., 2021; Tasevski & Jakimoski, 2020; Thomas & Williams, 2007; Umar et al., 2014b). On the other hand, there are techniques that automatically remove vulnerabilities by modifying the source code. These techniques identify the root cause of the vulnerability and then modify the source code to eliminate it. This is accomplished by either applying scaping methods to the user input (Medeiros et al., 2016), inserting parameterised queries (Rafnsson et al., 2020), or validating user input (Tommy et al., 2017). However, these techniques are limited to addressing the first-order vulnerabilities by handling the user input securely and preventing an attack from being launched at the injection point since they only remove the vulnerability at the injection point and not at the triggering point, which is not sufficient to protect the application from second-order attacks, since the attack can be launched later by exploiting the second-order vulnerabilities that use the malicious input stored in the database. Moreover, manual removal of such vulnerabilities is tedious, error-prone and costly.

Obviously, vulnerability detection alone does not make web applications secure. Actual remediation of detected vulnerabilities is required to secure the web application. Therefore, an approach to automate the detection and removal of second-order SQL injection vulnerabilities is highly desirable, even though this is still an open research area in the current literature on web application vulnerabilities.

The aforementioned weaknesses and gaps clearly reveal the shortcomings and inadequacies of existing techniques at achieving automated detection and removal of second-order SQLIVs, thus, signifying the utmost importance of further research in this area.

In order to address the issues raised above, this thesis suggests using a static analysis technique to improve the automated detection and removal of second-order SQLIVs in web applications' source code. Static analysis was chosen as the method for the proposed solution because: (i) static analysis can be used to identify vulnerabilities early in the development process before the code is deployed. This can help prevent vulnerabilities from being introduced into production systems and reduce the risk of exploitation, (ii) static analysis tools can be automated to efficiently and consistently analyse large code bases, which can be particularly useful for identifying second-order vulnerabilities that are more difficult to identify manually, (iii) static analysis can provide detailed information about the source of the vulnerability, (iii) static analysis can provide detailed information about the source of the vulnerability, the exact location of the injection and triggering points, and the nature of the vulnerability, which improves developer awareness of the risks associated with second-order vulnerabilities, (iv) static analysis has the potential to explore all possible execution paths, which means a greater chance of finding such vulnerabilities, and (v) static analysis methods design many types of rules to detect vulnerabilities and therefore have the potential to identify second-order vulnerabilities.

1.3 Objectives of The Research

The main objective of this research work is to propose a new static analysing technique for detecting second-order SQL injection vulnerabilities in web applications' source code and automatically removing them. In order to achieve the main objective, the following are the specific objectives of this thesis:

i. To propose a technique to detect and remove second-order SQLIV of a web application by analysing its source codes

ii. To implement the proposed technique that enables automatic detection and removal of second-order SQLIV in a web application.

iii. To evaluate the effectiveness of the proposed technique.

1.4 Scope of the Study

Software security testing is the process of identifying whether the security features of software implementation are consistent with the design. Software security testing can be divided into security functional testing and security vulnerability testing. The software development process involves several development phases, including requirements, design, coding, testing, and deployment. It is essential to take care of the security aspects of the web application at each stage (Deepa & Thilagam, 2016). As stated in the literature, several testing techniques specifically target each development phase mentioned above (Jovanovic, 2009; Luo, 2001). The approach designed in this study specifically focuses on software security testing for the detection and removal of second-order SQLIVs during the testing phase of web application development.

The scope of software vulnerabilities is very broad, diverse, and complex. However, previous reports on software security consider injection vulnerabilities the most severe and prevalent vulnerabilities among other web application vulnerabilities (Acunetix, 2020; OWASP, 2021; SANS/CWE, 2019; TRUSTWAVE, 2018). SQLIVs are top-ranked as the most severe and common injection vulnerabilities with hazardous consequences. The lack of effective mechanisms for addressing the detection of second-order SQLIVs which are associated with an increasing trend of reprocessing submitted data and optimising its use increases the risks of an attack. Therefore, the focus of this study is to address the problem of detecting and removing second-order SQLIVs in web applications. Figure 1.2 illustrates the research direction. The bold lines that are connected to the green boxes present the direction focused on this study, while the dashed lines represent other paths that are not considered in this study.

Several techniques for removing SQL injection vulnerabilities include input validation, parameterized queries, and sanitization. This research uses the sanitisation technique to remove second-order SQL injection vulnerabilities in the source code. Sanitization is a technique that removes potentially dangerous characters or metadata from user input to prevent SQL injection attacks. This technique requires fewer changes to the code than the other techniques, making it an effective solution for preventing SQL injection attacks, especially when applied automatically. In addition, sanitization can be faster and more lightweight compared to parameterized queries and input validation. This can be useful for applications that require high performance or processing large volumes of data.

Several programming languages exist for developing web applications, such as JSP, Python, PHP, and so forth. This research is concerned with web applications developed by PHP as a subject of security testing. PHP is the most common programming language used on the server (Hauzar & Kofroň, 2012; Positive Technologies, 2014; W3techs, 2021). Moreover, PHP is particularly prone to programming mistakes that may lead to web application vulnerabilities, such as SQL injections (Backes et al., 2017).

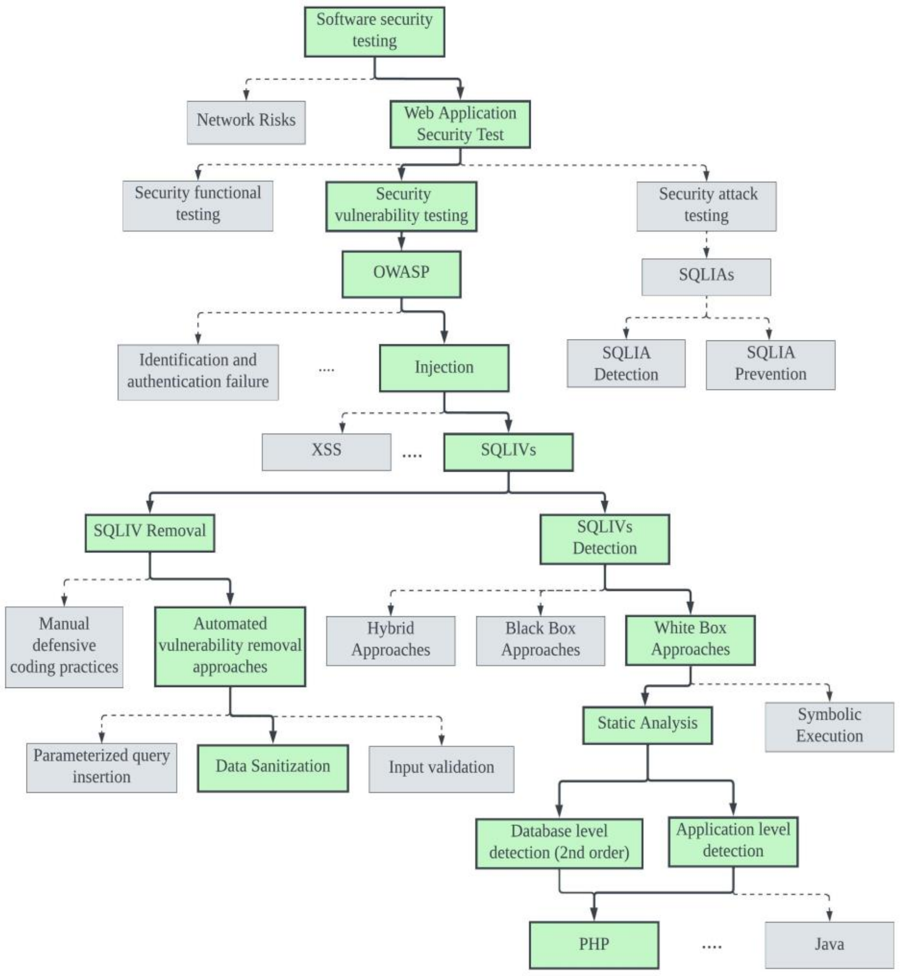
Software security testing
Network Risks
Web Application Security Test
Security functional testing
Security vulnerability testing
Security attack testing
OWASP
SQLIAs
Identification and authentication failure
....
Injection
SQLIA Detection
SQLIA Prevention
XSS
....
SQLIVs
SQLIV Removal
SQLIVs Detection
Manual defensive coding practices
Automated vulnerability removal approaches
Hybrid Approaches
Black Box Approaches
White Box Approaches
Static Analysis
Symbolic Execution
Parameterized query insertion
Data Sanitization
Input validation
Database level detection (2nd order)
Application level detection
PHP
....
Java

Figure 1.2 : Scope of the research

1.5 Contributions of the Study

This study made several contributions to the body of knowledge that include but are not limited to the following:

a) It provides an automated technique based on static program analysis for an effective analysis of web application source code to detect and remove second-order SQLIVs.

b) It provides a support tool named SoSQLiP to automate the process of detecting and removing second-order SQLIVs proposed by our technique.

c) It provides empirical evidence that the proposed technique is effective in testing web applications compared to the existing technique.

Furthermore, this new technique will benefit developers of web applications by enabling them to test their source codes and get rid of second-order SQLIVs before deploying their applications.

1.6 Thesis Organization

The thesis comprises six chapters. A brief description of each chapter is given below.

This chapter provides an overview of the research area. It pinpoints the research problem, objectives, scope of the study, main contributions, and the structure with which the chapters are organised. The remaining chapters are organised as follows.

Chapter 2 provides a thorough review of key areas that serve as the foundation for this study. This chapter discusses the existing approaches and techniques used to detect and remove the SQLIVs of web applications and highlights the limitations, gaps, and issues of existing SQLVs detection and removal approaches and techniques. The reviewed literature provides a base for the technique proposed in this study.

Chapter 3 presents the methodology of the study. It shows the materials and methods used for achieving the objectives of the study, namely, to propose a static analysis technique to detect and remove the vulnerable points to second-order SQL injection attacks, to implement a prototype software tool, and to discuss a test strategy to evaluate the performance of the proposed technique.

Chapter 4 presents the newly proposed technique to detect and remove second-order SQL injection vulnerabilities in the web application source file. The chapter discusses the performance of the proposed technique and its architecture. It also explains the development and implementation of SoSQLiP software prototype.

Chapter 5 presents a comprehensive set of experiments that were carried out to empirically evaluate the new technique, SoSQLiP, in terms of its ability to detect second-order SQLIVs, second-order SQLIVs detection precision, second-order SQLIVs detection recall, second-order SQLIVs detection F-Measure, and percentage of the SQLIVs removed. In addition, the chapter contains the experimental results, analysis, and discussion.

Chapter 6 provides a summary and highlights the contributions and limitations of this study. It also gives directions for future research.

REFERENCES

Abadi, A., Feldman, Y. a, & Shomrat, M. (2011). Code-motion for API migration: Fixing SQL injection vulnerabilities in Java. *WRT 2011 - Proceedings of the 4th Workshop on Refactoring Tools, Co-Located with ICSE 2011*, 1–7. https://doi.org/10.1145/1984732.1984734

Abdulridha Hussain, M., Alaa Hussien, Z., Ameen Abduljabbar, Z., Ma, J., Al Sibahee, M. A., Abdulridha Hussain, S., Omollo Nyangaresi, V., & Jiao, X. (2022). Provably throttling SQLI using an enciphering query and secure matching. *Egyptian Informatics Journal*. https://doi.org/10.1016/J.EIJ.2022.10.001

Acunetix. (2020). Web Application Vulnerability. *Acunetix*, *October*, 26. https://w3techs.com/technologies/history_overview/programming_language/ms/y

Akrout, R., Alata, E., Kaaniche, M., & Nicomette, V. (2014). An automated black box approach for web vulnerability identification and attack scenario generation. *Journal of the Brazilian Computer Society*, *20*(1), 1–16. https://doi.org/10.1186/1678-4804-20-4

Alazab, A., Alazab, M., Abawajy, J., & Hobbs, M. (2011). *Web Application Protection against SQL Injection Attack*. *Icita*, 1–7.

Alhuzali, A., Eshete, B., Gjomemo, R., & Venkatakrishnan, V. N. (2016). Chainsaw: Chained automated workflow-based exploit generation. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 641–652. https://doi.org/10.1145/2976749.2978380

Alhuzali, A., Gjomemo, R., Eshete, B., & Venkatakrishnan, V. N. (2018). NAVEX: Precise and scalable exploit generation for dynamic web applications. *Proceedings of the 27th USENIX Security Symposium*, 377–392.

Aliero, M. S., Ghani, I., Qureshi, K. N., & Rohani, M. F. (2019). An algorithm for detecting SQL injection vulnerability using black-box testing. *Journal of Ambient Intelligence and Humanized Computing*, *11*(1), 249–2660. https://doi.org/10.1007/s12652-019-01235-z

Aliero, M. S., Ghani, M., & Khan, M. M. (2015). Review on sql injection protection methods and tools. *Jurnal Teknologi*, *77*(13), 49–66. https://doi.org/10.11113/jt.v77.6359

Almadhy, W., Alruwaili, A., & Hendaoui, S. (2022). *Using SQLMAP to Detect SQLI Vulnerabilities*. *22*(1), 234–240.

Aloraini, B., Nagappan, M., German, D. M., Hayashi, S., & Higo, Y. (2019). An empirical study of security warnings from static application security testing tools. *Journal of Systems and Software*, *158*. https://doi.org/10.1016/j.jss.2019.110427

Alotaibi, M., Al-hendi, D., Alroithy, B., AlGhamdi, M., & Gutub, A. (2019). Secure Mobile Computing Authentication Utilizing Hash, Cryptography and Steganography Combination. *Journal of Information Security and Cybercrimes Research*, *2*(1), 73–82. https://doi.org/10.26735/16587790.2019.001

Alwan, Z. S., & Younis, M. F. (2017). Detection and Prevention of SQL Injection Attack: A Survey. *International Journal of Computer Science and Mobile Computing*, *6*(8), 5–17. www.ijcsmc.com

Amankwah, R., Kwaku, P., & Yeboah, S. (2017). Evaluation of Software Vulnerability Detection Methods and Tools: A Review. *International Journal of Computer Applications*, *169*(8), 22–27. https://doi.org/10.5120/ijca2017914750

Anagandula, K., & Zavarsky, P. (2020). An Analysis of Effectiveness of Black-Box Web Application Scanners in Detection of Stored SQL Injection and Stored XSS Vulnerabilities. *Proceedings - 2020 3rd International Conference on Data Intelligence and Security, ICDIS 2020*, 40–48. https://doi.org/10.1109/ICDIS50059.2020.00012

Anis, A., Zulkernine, M., Iqbal, S., Liem, C., & Chambers, C. (2018). Securing web applications with secure coding practices and integrity verification. *Proceedings - IEEE 16th International Conference on Dependable, Autonomic and Secure Computing, IEEE 16th International Conference on Pervasive Intelligence and Computing, IEEE 4th International Conference on Big Data Intelligence and Computing and IEEE 3*, 618–625. https://doi.org/10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00112

Anley, C. (2002). Advanced SQL injection in SQL server applications. *White Paper, Next Generation Security Software*. http://www.ngssoftware.com

Austin, A., & Williams, L. (2011). One Technique is Not Enough: A Comparison of Vulnerability Discovery Techniques. *2011 International Symposium on Empirical Software Engineering and Measurement*, 97–106. https://doi.org/10.1109/ESEM.2011.18

Avancini, A., & Ceccato, M. (2011). Security testing of web applications: A search-based approach for cross-site scripting vulnerabilities. *Proceedings - 11th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2011*, 85–94. https://doi.org/10.1109/SCAM.2011.7

Awang, N. F., & Manaf, A. A. (2015). Automated security testing framework for detecting SQL injection vulnerability in web application. *Communications in Computer and Information Science*, *534*, 160–171. https://doi.org/10.1007/978-3-319-23276-8_14

Backes, M., Rieck, K., Skoruppa, M., Stock, B., & Yamaguchi, F. (2017). Efficient and Flexible Discovery of PHP Application Vulnerabilities. *Proceedings - 2nd IEEE European Symposium on Security and Privacy, EuroS and P 2017*, 334–349. https://doi.org/10.1109/EuroSP.2017.14

Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C., & Vigna, G. (2008). Saner: Composing static and dynamic analysis to validate sanitization in web applications. *Proceedings - IEEE Symposium on Security and Privacy*, 387–401. https://doi.org/10.1109/SP.2008.22

Bateman, H. V. (1998). Software vulnerability analysis. *Purdue University, De-Partment of Computer Sciences*, 274.

Bau, J., Wang, F., Bursztein, E., Mutchler, P., & Mitchell, J. (2012). Vulnerability Factors in New Web Applications: Audit Tools, Developer Selection & Languages. *Seclab.Stanford.Edu*. http://seclab.stanford.edu/websec/scannerPaper.pdf

Belyaev, M. V, Shimchik, N. V, Ignatyev, V. N., & Belevantsev, A. A. (2018). Comparative Analysis of Two Approaches to Static Taint Analysis. *Programming and Computer Software*, *44*(6), 459–466. https://doi.org/10.1134/S036176881806004X

Bhanderi, A., Rawal, N., & Student, P. G. (2007). A Review on Detection Mechanisms for SQL Injection Attacks. *International Journal of Innovative Research in Science, Engineering and Technology (An ISO*, *3297*, 12446–12452. https://doi.org/10.15680/IJIRSET.2015.0412145

Bisht, P., Sistla, A. P., & Venkatakrishnan, V. N. (2010). TAPS: Automatically preparing safe SQL queries. *Proceedings of the ACM Conference on Computer and Communications Security*, 645–647. https://doi.org/10.1145/1866307.1866384

Braz, L., Fregnan, E., Calikli, G., & Bacchelli, A. (2021). Why don’t developers detect improper input validation? *Proceedings - International Conference on Software Engineering*, 499–511. https://doi.org/10.1109/ICSE43902.2021.00054

Cao, K., He, J., Fan, W., Huang, W., Chen, L., & Pan, Y. (2018). PHP vulnerability detection based on taint analysis. *2017 6th International Conference on Reliability, Infocom Technologies and Optimization: Trends and Future Directions, ICRITO 2017*, *2018-Janua*, 436–439. https://doi.org/10.1109/ICRITO.2017.8342466

Chaki, S. M. H., & Mat Din, M. (2019). A Survey on SQL Injection Prevention Methods. *International Journal of Innovative Computing*, *9*(1), 47–54. https://doi.org/10.11113/ijic.v9n1.224

Chen, J. M., & Wu, C. L. (2010). An automated vulnerability scanner for injection attack based on injection point. *ICS 2010 - International Computer Symposium*, 113–118. https://doi.org/10.1109/COMPSYM.2010.5685537

CHEN, Y., PAN, Z., CHEN, Y., & LI, Y. (2022). DISOV: Discovering Second-Order Vulnerabilities Based on Web Application Property Graph. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*. https://doi.org/10.1587/transfun.2022eap1045

Chess, B., & Mcgraw, G. (2004). Static analysis for security. *IEEE Security and Privacy*, *2*(6), 76–79. https://doi.org/10.1109/MSP.2004.111

Chess, B., & West, J. (2007). Secure Programming with static Analysis. In *Addison-Wesley Professional*. https://doi.org/10.1017/CBO9781107415324.004

Choudhury, H., Roychoudhury, B., & Saikia, D. K. (2016). Efficient Detection of Multi-step Cross-Site Scripting Vulnerabilities. *International Journal of Network Security*, *18*(6), 1041–1053. https://doi.org/10.1007/978-3-319-13841-1

Chowdhury, I., & Zulkernine, M. (2011). Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture*, *57*(3), 294–313. https://doi.org/10.1016/j.sysarc.2010.06.003

Dahse, J., & Holz, T. (2014). Static Detection of Second-Order Vulnerabilities in Web Applications. *23rd USENIX Security Symposium (USENIX Security 14)*, 989–1003. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/dahse

Damele, B., & Stampar, M. (2016). *sqlmap: automatic SQL injection and database takeover tool*. SQLMap. http://sqlmap.org/

Deepa, G., & Thilagam, P. S. (2016). Securing web applications from injection and logic vulnerabilities: Approaches and challenges. *Information and Software Technology*, *74*, 160–180. https://doi.org/10.1016/j.infsof.2016.02.005

Di Lucca, G. A., & Fasolino, A. R. (2006). Testing Web-based applications: The state of the art and future trends. *Information and Software Technology*, *48*, 1172–1186. https://doi.org/10.1016/j.infsof.2006.06.006

Díaz, G., & Ramón Bermejo, J. (2013). Static analysis of source code security: Assessment of tools against SAMATE tests. *Information and Software Technology*, *55*, 1462–1476. https://doi.org/10.1016/j.infsof.2013.02.005

Ding, S., Tan, H. B. K., Shar, L. K., & Padmanabhuni, B. M. (2013). Towards a hybrid framework for detecting input manipulation vulnerabilities. *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, *1*, 363–370. https://doi.org/10.1109/APSEC.2013.56

Djuric, Z. (2013). A black-box testing tool for detecting SQL injection vulnerabilities. *2013 Second International Conference on Informatics & Applications (ICIA)*, 216–221. https://doi.org/10.1109/ICoIA.2013.6650259

Doğan, S., Betin-Can, A., & Garousi, V. (2014). Web application testing: A systematic literature review. *Journal of Systems and Software*, *91*(1), 174–201. https://doi.org/10.1016/j.jss.2014.01.010

Dolatnezhad, S., & Amini, M. (2019). Preventing SQL Injection Attacks by Automatic Parameterizing Raw Queries Using Lexical and Semantic Analysis Methods. *Scientia Iranica*, *6*, 0–0. https://doi.org/10.24200/sci.2019.21229

Doupé, A., Cova, M., & Vigna, G. (2010). Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6201 LNCS*, 111–131. https://doi.org/10.1007/978-3-642-14215-4_7

Draib, N., Sultan, A. B. M., Ghani, A. A. B. A., & Zulzalil, H. (2019). Evaluation of SQL injection vulnerability detection tools. *International Journal of Engineering and Advanced Technology*, *9*(1), 1747–1751. https://doi.org/10.35940/ijeat.A2648.109119

Dysart, F., & Sherriff, M. (2008). Automated fix generator for SQL injection attacks. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, *May*, 311–312. https://doi.org/10.1109/ISSRE.2008.44

Evans, D., & Larochelle, D. (2002). Improving security using extensible lightweight static analysis. *IEEE Software*, *19*(1), 42–51. https://doi.org/10.1109/52.976940

Faker, S. A., Muslim, M. A., & Dachlan, H. S. (2017). A Systematic Literature Review on SQL Injection Attacks Techniques and Common Exploited Vulnerabilities. *International Journal of Computer Engineering and Information Technology*, *9*(12), 284–291.

Fangqi, S., Liang, X., & Su, Z. (2011). Static detection of access control vulnerabilities in Vue applications. *20th USENIX Conference on Security*, 11–21. https://doi.org/10.1088/1742-6596/1646/1/012021

Felderer, M., Büchler, M., Johns, M., Brucker, A. D., Breu, R., & Pretschner, A. (2016). Security Testing: A Survey. In *Advances in Computers* (Vol. 101, pp. 1–51). https://doi.org/10.1016/bs.adcom.2015.11.003

Fernando, H., & Abawajy, J. (2013). Malware detection and prevention in RFID systems. *Studies in Computational Intelligence*, *460*, 143–166. https://doi.org/10.1007/978-3-642-34952-2_6

Fu, X., & Qian, K. (2008). SAFELI - SQL injection scanner using symbolic execution. *TAV-WEB 2008 - Proceedings of the Workshop on Testing, Analysis and Verification of Web Software*, 34–39. https://doi.org/10.1145/1390832.1390838

Gautam, B., Tripathi, J., Singh, S., & Student, M. T. (2018). A Secure Coding Approach For Prevention of SQL Injection Attacks. *International Journal of Applied Engineering Research*, *13*(11), 9874–9880. http://www.ripublication.com

Ghafarian, A. (2018). A hybrid method for detection and prevention of SQL injection attacks. *Proceedings of Computing Conference 2017*, *2018-Janua*(July), 833–838. https://doi.org/10.1109/SAI.2017.8252192

Goseva-Popstojanova, K., & Perhinschi, A. (2015). On the capability of static code analysis to detect security vulnerabilities. *Information and Software Technology*, *68*, 18–33. https://doi.org/10.1016/j.infsof.2015.08.002

Halfond, W. G. J., Viegas, J., & Orso, A. (2008). A Classification of SQL Injection Attacks and Countermeasures. In *Preventing Sql Code Injection By Combining Static and Runtime Analysis*. https://doi.org/doi=10.1.1.95.2968

Hanif, H., Md Nasir, M. H. N., Ab Razak, M. F., Firdaus, A., & Anuar, N. B. (2021). The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches. *Journal of Network and Computer Applications*, *179*(August 2020), 103009. https://doi.org/10.1016/j.jnca.2021.103009

Harer, J. A., Kim, L. Y., Russell, R. L., Ozdemir, O., Kosta, L. R., Rangamani, A., Hamilton, L. H., Centeno, G. I., Key, J. R., Ellingwood, P. M., Antelman, E., Mackay, A., McConley, M. W., Opper, J. M., Chin, P., & Lazovich, T. (2018). Automated software vulnerability detection with machine learning. *ArXiv:1803.04497*. http://arxiv.org/abs/1803.04497

Hauzar, D., & Kofroň, J. (2012). On security analysis of PHP web applications. *Proceedings - International Computer Software and Applications Conference*, 577–582. https://doi.org/10.1109/COMPSACW.2012.106

Hofer, T. (2010). Evaluating Static Source Code Analysis Tools. *Master's Thesis, School of Com- Puter and Communications Science, Ecole Polytechnique Federale de Lausanne*.

Hofman, S. J., & Ibrahimi, E. (2022). State of the Art: Performance Overview of Black-Box Web Application Scanners. *19th SC@RUG 2022 Proceedings 2021-2022*, 9–14.

Hu, J., Zhao, W., & Cui, Y. (2020). A Survey on SQL Injection Attacks, Detection and Prevention. *ACM International Conference Proceeding Series*, 483–488. https://doi.org/10.1145/3383972.3384028

Huang, Y., Huang, S.-K., Lin, T.-P., & Tsai, C.-H. (2003). Web application security assessment by fault injection and behavior monitoring. *Proceedings of the Twelfth International Conference on World Wide Web - WWW '03*, 148. https://doi.org/10.1145/775152.775174

Johari, R., & Sharma, P. (2012). A survey on web application vulnerabilities (SQLIA, XSS) exploitation and security engine for SQL injection. *Proceedings - International Conference on Communication Systems and Network Technologies, CSNT 2012, May 2012*, 453–458. https://doi.org/10.1109/CSNT.2012.104

Johnson, J. (2022). *Internet and social media users in the world 2022*. Statista. https://www.statista.com/statistics/617136/digital-population-worldwide/

Joseph, S., & K.P, J. (2016). *Evaluating the Effectiveness of Conventional Fixes for SQL Injection Vulnerability Secure Software Development View project Mobile Application Security View project Swathy Joseph Evaluating The Effectiveness Of Conventional Fixes For SQL Injection Vulnera*. https://doi.org/10.1007/978-81-322-2529-4_44

Joshi, C., & Singh, K. (2016). Performance Evaluation of Web Application Security Scanners for More Effective Defense. *International Journal of Scientific and Research Publications*, *6*(6), 660. www.ijsrp.org

Jovanovic, I. (2009). Software Testing Methods and Techniques. *The IPSI BgD Transactions on Internet Research*, *5*(1), 30–41. internetjournals.net

Jovanovic, N., Kruegel, C., & Kirda, E. (2006). Pixy: A static analysis tool for detecting Web application vulnerabilities. *2006 IEEE Symposium on Security and Privacy (S&P'06)*, 260–263. https://doi.org/10.1109/SP.2006.29

Jovanovic, N., Kruegel, C., & Kirda, E. (2010). Static analysis for detecting taint-style vulnerabilities in web applications. *Journal of Computer Security*, *18*(5), 861–907. https://doi.org/10.3233/JCS-2009-0385

Kagorora, F., Li, J., Hanyurwimfura, D., Camara, L., & Student, P. G. (2017). Effectiveness of Web Application Security Scanners at Detecting Vulnerabilities behind AJAX/JSON. *International Journal of Innovative Research in Science, Engineering and Technology (An ISO*, *4*(6), 11068–11076. https://doi.org/10.15680/IJIRSET.2015.0406079

Kals, S., Kirda, E., Kruegel, C., & Jovanovic, N. (2006). SecuBat: A Web Vulnerability Scanner. *Proceedings of the 15th International Conference on World Wide Web - WWW '06*, 247. https://doi.org/10.1145/1135777.1135817

Kaur, D., & Kaur, P. (2016). Empirical Analysis of Web Attacks. *Physics Procedia*, *78*(December 2015), 298–306. https://doi.org/10.1016/j.procs.2016.02.057

Kaur, P., & Kour, K. P. (2016). SQL injection: Study and augmentation. *Proceedings of 2015 International Conference on Signal Processing, Computing and Control, ISPCC 2015*, 102–107. https://doi.org/10.1109/ISPCC.2015.7375006

Khoury, N., Zavarsky, P., Lindskog, D., & Ruhl, R. (2011). An analysis of black-box web application security scanners against stored SQL injection. *Proceedings - 2011 IEEE International Conference on Privacy, Security, Risk and Trust and IEEE International Conference on Social Computing, PASSAT/SocialCom 2011*, 1095–1101. https://doi.org/10.1109/PASSAT/SocialCom.2011.199

Kieyzun, A., Guo, P. J., Jayaraman, K., & Ernst, M. D. (2009). Automatic creation of SQL Injection and cross-site scripting attacks. *2009 IEEE 31st International Conference on Software Engineering*, 199–209. https://doi.org/10.1109/ICSE.2009.5070521

Kim, M., & Lee, D. H. (2014). Data-mining based SQL injection attack detection using internal query trees. *Expert Systems with Applications*, *41*(11), 5416–5430. https://doi.org/10.1016/j.eswa.2014.02.041

Kim, S., Kim, R. Y. C., & Park, Y. B. (2016). Software Vulnerability Detection Methodology Combined with Static and Dynamic Analysis. *Wireless Personal Communications*, *89*(3), 777–793. https://doi.org/10.1007/s11277-015-3152-1

Kindy, D. A., & Pathan, A.-S. K. (2013). A Detailed Survey on various aspects of SQL Injection in Web Applications: Vulnerabilities, Innovative Attacks and Remedies. In *International Journal of Communication Networks and Information Security (IJCNIS)* (Vol. 5, Issue 2). http://irep.iium.edu.my/31262/1/364-882-1-PB.pdf

Kronjee, J., Hommersom, A., & Vranken, H. (2018). Discovering software vulnerabilities using data-flow analysis and machine learning. *ACM International Conference Proceeding Series*, 1–10. https://doi.org/10.1145/3230833.3230856

Kumar, P., & Pateriya, R. K. (2012). A survey on SQL injection attacks, detection and prevention techniques. *2012 3rd International Conference on Computing, Communication and Networking Technologies, ICCCNT 2012*, 1–5. https://doi.org/10.1109/ICCCNT.2012.6396096

Lala, S. K., Kumar, A., & Subbulakshmi, T. (2021). Secure web development using OWASP guidelines. *Proceedings - 5th International Conference on Intelligent Computing and Control Systems, ICICCS 2021*, *Iciccs*, 323–332. https://doi.org/10.1109/ICICCS51141.2021.9432179

Lam, M. S., Martin, M., Livshits, B., & Whaley, J. (2008). Securing web applications with static and dynamic information flow tracking. *Proceedings of the 2008 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation - PEPM '08*, 3–12. https://doi.org/10.1145/1328408.1328410

Landi, W. (1992). Undecidability of Static Analysis. In *From acm Letters on Programming Languages and Systems* (Vol. 1, Issue 4). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.9722&rep=rep1&type=pdf

Lashkaripour, Z., & Ghaemi Bafghi, A. (2013). A Simple and Fast Technique for Detection and Prevention of SQL Injection Attacks (SQLIAs). *International Journal of Security and Its Applications*, *7*(5), 53–66. https://doi.org/10.14257/ijsia.2013.7.5.05

Li, Y. F., Das, P. K., & Dowe, D. L. (2014). Two decades of Web application testing - A survey of recent advances. In *Information Systems* (Vol. 43, pp. 20–54). https://doi.org/10.1016/j.is.2014.02.001

Liu, M., & Wang, B. (2018). A Web Second-Order Vulnerabilities Detection Method. *IEEE Access*, *6*, 70983–70988. https://doi.org/10.1109/ACCESS.2018.2881070

Luo, L. (2001). Software testing techniques. *Institute for Software Research International Carnegie Mellon University Pittsburgh, PA*, *15232*(1–19), 19. http://www.cs.cmu.edu/~luluo/Courses/17939Report.pdf

Makiou, A., Begriche, Y., & Serhrouchni, A. (2014). Improving Web Application Firewalls to detect advanced SQL injection attacks. *2014 10th International Conference on Information Assurance and Security, IAS 2014*, 35–40. https://doi.org/10.1109/ISIAS.2014.7064617

Marashdih, A. W., Zaaba, Z. F., & Suwais, K. (2022). Predicting input validation vulnerabilities based on minimal SSA features and machine learning. *Journal of King Saud University - Computer and Information Sciences*, *34*(10), 9311–9331. https://doi.org/10.1016/j.jksuci.2022.09.010

McClure, R. A., & Krüger, I. H. (2005). SQL DOM: Compile time checking of dynamic SQL statements. *Proceedings - 27th International Conference on Software Engineering, ICSE05*, 88–96. https://doi.org/10.1109/icse.2005.1553551

Medeiros, I., Neves, N., & Correia, M. (2013). *WAP : Automatic Detection and Correction of Web Application Vulnerabilities*. 1–13.

Medeiros, I., Neves, N., & Correia, M. (2016). Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining. *IEEE Transactions on Reliability*, *65*(1), 54–69. https://doi.org/10.1109/TR.2015.2457411

Medeiros, N., Ivaki, N., Costa, P., & Vieira, M. (2020). Vulnerable Code Detection Using Software Metrics and Machine Learning. *IEEE Access*, *8*(2020), 219174–219198. https://doi.org/10.1109/ACCESS.2020.3041181

Mishra, N., Chaturvedi, S., Sharma, A. K., & Choudhary, S. (2014). XML-based authentication to handle SQL injection. *Advances in Intelligent Systems and Computing*, *236*, 739–749. https://doi.org/10.1007/978-81-322-1602-5_79

Mitropoulos, D., Louridas, P., Polychronakis, M., & Keromytis, A. D. (2017). Defending Against Web Application Attacks: Approaches, Challenges and Implications. *IEEE Transactions on Dependable and Secure Computing*, *5971*(c), 1–1. https://doi.org/10.1109/TDSC.2017.2665620

Mohosinaand, A., & Zulkernine, M. (2012). DESERVE: A framework for DEtecting program SEcuRity Vulnerability Exploitations. *Proceedings of the 2012 IEEE 6th International Conference on Software Security and Reliability, SERE 2012*, 98–107. https://doi.org/10.1109/SERE.2012.22

Mui, R., & Frankl, P. (2010). Preventing SQL Injection through Automatic Query Sanitization with ASSIST. *Electronic Proceedings in Theoretical Computer Science*, *35*, 27–38. https://doi.org/10.4204/eptcs.35.3

Muraleedharan, A. (2015). A Robust Method for Prevention of Second Order and Stored Procedure based SQL Injections. *International Journal of Computer Applications*, 20–23.

Musciano, C., & Kennedy, B. (2000). *HTML & XHTML : The Definitive Guide 4th edition*. http://ommolketab.ir/aaf-lib/9j7j12gg2cfghkxh1rdsr6j6md9vop.pdf

O'Leary, R. (2017). APPLICATION SECURITY STATISTICS REPORT. *WhiteHat Security*, *36*(8), 1725–1744. https://doi.org/10.1016/S0045-6535(97)10063-7

Olivo, O., Dillig, I., & Lin, C. (2015). Detecting and exploiting Second Order denial-of-service vulnerabilities in web applications. *Proceedings of the ACM Conference on Computer and Communications Security*, *2015-Octob*, 616–628. https://doi.org/10.1145/2810103.2813680

OWASP. (2016). *SQL Injection Prevention Cheat Sheet*. https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

OWASP. (2017). *Top 10-2017 Top 10 - OWASP*. https://www.owasp.org/index.php/Top_10-2017_Top_10

OWASP. (2021). *OWASP Top 10:2021*. OWASP.Org. https://www.owasptopten.org/the-release-of-the-owasp-top-10-2021

Pan, Z., Chen, Y., Chen, Y., Shen, Y., & Li, Y. (2022). LogInjector: Detecting Web Application Log Injection Vulnerabilities. *Applied Sciences (Switzerland)*, *12*(15). https://doi.org/10.3390/app12157681

Panda, M. (2017). Performance analysis of encryption algorithms for security. *International Conference on Signal Processing, Communication, Power and Embedded System, SCOPES 2016 - Proceedings*, 278–284. https://doi.org/10.1109/SCOPES.2016.7955835

Parrend, P., Navarro, J., Guigou, F., Deruyver, A., & Collet, P. (2018). Foundations and applications of artificial Intelligence for zero-day and multi-step attack detection. *EURASIP Journal on Information Security*, *2018*(1), 1–21. https://doi.org/10.1186/s13635-018-0074-y

Parvez, M., Zavarsky, P., & Khoury, N. (2016). Analysis of effectiveness of black-box web application scanners in detection of stored SQL injection and stored XSS vulnerabilities. *2015 10th International Conference for Internet Technology and Secured Transactions, ICITST 2015*, 186–191. https://doi.org/10.1109/ICITST.2015.7412085

Patil, S., Marathe, N., & Padiya, P. (2016). Design of efficient web vulnerability scanner. *2016 International Conference on Inventive Computation Technologies (ICICT)*, 1–6. https://doi.org/10.1109/INVENTIVE.2016.7824873

Pereira, J. D. A., Campos, J. R., & Vieira, M. (2019). An exploratory study on machine learning to combine security vulnerability alerts from static analysis tools. *2019 9th Latin-American Symposium on Dependable Computing, LADC 2019 - Proceedings*, *Ml*. https://doi.org/10.1109/LADC48089.2019.8995685

Ping, C. (2017). A second-order SQL injection detection method. *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 1792–1796. https://doi.org/10.1109/ITNEC.2017.8285104

Ping, C. (2018). x-A second-order SQL injection detection method. *Proceedings of the 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2017*, *2018-Janua*, 1792–1796. https://doi.org/10.1109/ITNEC.2017.8285104

Positive Technologies. (2014). *Web Application Vulnerability Statistics*. http://fortune.com/global500/

Qasaimeh, M., Khairallah, T. Z., Shamlawi, A., & Khairallah, T. (2018). BLACK BOX EVALUATION OF WEB APPLICATION SCANNERS: STANDARDS MAPPING APPROACH. *Article in Journal of Theoretical and Applied Information Technology*, *31*(14). https://www.researchgate.net/publication/329990368

Rafnsson, W., Giustolisi, R., Kragerup, M., & Høyrup, M. (2020). Fixing Vulnerabilities Automatically with Linters. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *12570 LNCS*, 224–244. https://doi.org/10.1007/978-3-030-65745-1_13

Sadeghian, A., Zamani, M., & Abdullah, S. M. (2013). A Taxonomy of SQL Injection Attacks. *2013 International Conference on Informatics and Creative Multimedia*, 269–273. https://doi.org/10.1109/ICICM.2013.53

Saidu Aliero, M., Ghani, I., Zainudden, S., Murad Khan, M., & Bello, M. (2015). Review on sql injection protection methods and tools. In *Jurnal Teknologi* (Vol. 77, Issue 13, pp. 49–66). https://doi.org/10.11113/jt.v77.6359

Salih Ali, N. (2018). Investigation framework of web applications vulnerabilities, attacks and protection techniques in structured query language injection attacks. *Int. J. Wireless and Mobile Computing, 14*(2), 103–122. https://doi.org/10.1504/IJWMC.2018.091137

Sampaio, L., & Garcia, A. (2015). Exploring Context-Sensitive Data Flow Analysis for Early Vulnerability Detection. *Journal of Systems and Software*, *113*, 337–361. https://doi.org/10.1016/j.jss.2015.12.021

SANS/CWE. (2019). CWE - 2019 CWE Top 25 Most Dangerous Software Errors. In *SANS Institute*. https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html

Satyanarayana, L. V., & Sekhar, M. V. B. C. (2011). Static Analysis Tool for Detecting Web Application Vulnerabilities. *International Journal of Modern Engineering Research (IJMER), 1*(1), 127–133.

Scholte, T., Robertson, W., Balzarotti, D., & Kirda, E. (2012). Preventing input validation vulnerabilities inweb applications through automated type analysis. *Proceedings - International Computer Software and Applications Conference*, 233–243. https://doi.org/10.1109/COMPSAC.2012.34

Scott, D., & Sharp, R. (2002). Abstracting application-level web security. *Proceedings of the 11th International Conference on World Wide Web, WWW '02*, 396–407. https://doi.org/10.1145/511446.511498

Selecte, R., Khalid, M. N., Farooq, H., & Iqbal, M. (2020). *Predicting Web Vulnerabilities in Web Applications Based on Machine Learning* (Vol. 910, Issue April). Springer Singapore. https://doi.org/10.1007/978-981-13-6095-4

Shar, L. K., Briand, L. C., Tan, H. K., & Member, S. (2015). Web Application Vulnerability Prediction Using Hybrid Program Analysis and Machine Learning. *IEEE Transactions on Dependable and Secure Computing*, *12*(6), 688–707. https://doi.org/10.1109/TDSC.2014.2373377

Shar, L. K., & Tan, H. B. K. (2013). Defeating SQL injection. *Computer*, *46*(3), 69–77. https://doi.org/10.1109/MC.2012.283

Sharma, C., & Jain, S. C. (2014). Analysis and classification of SQL injection vulnerabilities and attacks on web applications. *2014 International Conference on Advances in Engineering and Technology Research, ICAETR 2014*. https://doi.org/10.1109/ICAETR.2014.7012815

Siddiq, M. L., Jahin, M. R. R., Ul Islam, M. R., Shahriyar, R., & Iqbal, A. (2021). SQLIFIX: Learning Based Approach to Fix SQL Injection Vulnerabilities in Source Code. *Proceedings - 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2021*, 354–364. https://doi.org/10.1109/SANER50967.2021.00040

Singh, J. P. (2017). Analysis of SQL Injection Detection Techniques. *Theoretical and Applied Informatics*, *28*(1&2), 37–55. https://doi.org/10.20904/281-2037

Singh, N., Dayal, M., Raw, R. S., & Kumar, S. (2016). SQL Injection: Types, Methodology, Attack Queries and Prevention. *2016 International Conference on Computing for Sustainable Global Development (INDIACom)*, 2872–2876.

Som, S., Sinha, S., & Kataria, R. (2016). Study on SQL Injection Attacks: Mode, Detection and Prevention. *International Journal of Engineering Applied Sciences and Technology*, *1*(8), 23–29. http://www.ijeast.com

Stanford, C., Bau, J., Bursztein, E., Gupta, D., & Mitchell, J. (2010). State of The Art: Automated Black Box Web Application Vulnerability Testing. *Security and Privacy (SP), 2010 IEEE Symposium On*, 332–345. http://www.owasp.org/images/2/28/Black_Box_Scanner_Presentation.pdf

Steiner, S., de Leon, D. C., & Alves-Foss, J. (2017). A structured analysis of SQL injection runtime mitigation techniques. *Proceedings of the Annual Hawaii International Conference on System Sciences, 2017-Janua*, 2887–2895. https://doi.org/10.24251/hicss.2017.349

Su, Z., & Wassermann, G. (2006). The Essence of Command Injection Attacks in Web Applications. *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, *41*(1), 372–382. https://doi.org/10.1145/1111320.1111070

Tasevski, I., & Jakimoski, K. (2020). Overview of SQL injection defense mechanisms. *2020 28th Telecommunications Forum, TELFOR 2020 - Proceedings*, 3–6. https://doi.org/10.1109/TELFOR51502.2020.9306676

Thomas, S., & Williams, L. (2007). Using automated fix generation to secure SQL statements. *Proceedings - ICSE 2007 Workshops: Third International Workshop on Software Engineering for Secure Systems, SESS'07*, 9. https://doi.org/10.1109/SESS.2007.12

Thomé, J., Gorla, A., & Zeller, A. (2014). Search-based security testing of web applications. *7th International Workshop on Search-Based Software Testing, SBST 2014 - Proceedings*, 5–14. https://doi.org/10.1145/2593833.2593835

Thomé, J., Shar, L. K., & Briand, L. (2016). Security slicing for auditing XML, XPath, and SQL injection vulnerabilities. *2015 IEEE 26th International Symposium on Software Reliability Engineering, ISSRE 2015*, 553–564. https://doi.org/10.1109/ISSRE.2015.7381847

Tomasdottir, K. F., Aniche, M., & Van Deursen, A. (2020). The Adoption of JavaScript Linters in Practice: A Case Study on ESLint. *IEEE Transactions on Software Engineering*, *46*(8), 863–891. https://doi.org/10.1109/TSE.2018.2871058

Tommy, R., Sundeep, G., & Jose, H. (2017). Automatic Detection and Correction of Vulnerabilities using Machine Learning. *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*, 1062–1065. https://doi.org/10.1109/CTCEEC.2017.8454995

Traphagen, K., & Traill, S. (2014). How Cross-Sector Collaborations are Advancing STEM Learning. *Toxicological Sciences : An Official Journal of the Society of Toxicology*, *137*(April), 41 p. https://doi.org/10.1093/toxsci/kft286

Trinh, M.-T., Chu, D.-H., & Jaffar, J. (2014). S3: A Symbolic String Solver for Vulnerability Detection in Web Applications. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14*, 1232–1243. https://doi.org/10.1145/2660267.2660372

TRUSTWAVE. (2018). *Trustwave Global Security Report Introduction the State of Security*. https://www2.trustwave.com/rs/815-RFM-693/images/Trustwave_2018-GSR_20180329_Interactive.pdf

Umar, K., Sultan, A. B., Zulzalil, H., Admodisastro, N., & Abdullah, M. T. (2014a). *On the Automation of Vulnerabilities Fixing for Web Application*. *c*, 221–226.

Umar, K., Sultan, A. B., Zulzalil, H., Admodisastro, N., & Abdullah, M. T. (2014b). Prevention of attack on Islamic websites by fixing SQL injection vulnerabilities using co-evolutionary search approach. *2014 the 5th International Conference on Information and Communication Technology for the Muslim World, ICT4M 2014*, 1–6. https://doi.org/10.1109/ICT4M.2014.7020604

Vaseghipanah, M., Bayat, N. K., Asami, A., & Shahmirzadi D, M. A. (2016). SQL Injection Attacks: A Systematic Review. *International Journal of Computer Science and Information Security (IJCSIS), 14*(12), 678–696. https://s3.amazonaws.com/academia.edu.documents/51650658/77_Paper_301116170_IJCSIS_Camera_Ready_678-696.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1520163083&Signature=l5TbtUtYOowCTeWZAQjeSZhUEV8%3D&response-content-disposition=inline%3B filename%3DS

Verma, N., & Kaur, A. (2015). A Detailed Study on Prevention of SQLI attacks for Web Security. *International Journal of Computer Applications Technology and Research, 4*(4), 308–311. https://doi.org/10.7753/ijcatr0404.1018

W3techs. (2021). *Usage Statistics and Market Share of PHP for Websites, August 2021*. https://w3techs.com/technologies/details/pl-php

W3Techs. (2023). *Usage Statistics and Market Share of PHP for Websites, January 2023*. W3Techs. https://w3techs.com/technologies/details/pl-php

Wang, Y., Wang, D., Zhao, W., & Liu, Y. (2015). Detecting SQL vulnerability attack based on the dynamic and static analysis technology. *Proceedings - International Computer Software and Applications Conference, 3*, 604–607. https://doi.org/10.1109/COMPSAC.2015.277

Wichman, L. (2010). *Mass SQL Injection for Malware Distribution Mass SQL Injection for Malware Distribution GIAC (GWAPT) Gold Certification*. https://www.sans.org/reading-room/whitepapers/malicious/mass-sql-injection-malware-distribution-33654

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering. In *Experimentation in Software Engineering* (Vol. 9783642290). https://doi.org/10.1007/978-3-642-29044-2

WVS, A. (2022). *Acunetix WVS*. https://www.acunetix.com/

Xiao, L., Matsumoto, S., Ishikawa, T., & Sakurai, K. (2017). SQL injection attack detection method using expectation criterion. *Proceedings - 2016 4th International Symposium on Computing and Networking, (CANDAR)*, 649–654. https://doi.org/10.1109/CANDAR.2016.74

Xie, Y., & Aiken, A. (2006). Static detection of security vulnerabilities in scripting languages. *15th USENIX Security Symposium, 15*, 179–192. http://www.usenix.org/event/sec06/tech/full_papers/xie/xie_html/

Yan, L., Li, X., Feng, R., Feng, Z., & Hu, J. (2014a). Detection method of the second-order sql injection in web applications. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8332 LNCS*, 154–165. https://doi.org/10.1007/978-3-319-04915-1_11

Yan, L., Li, X., Feng, R., Feng, Z., & Hu, J. (2014b). *Structured Object-Oriented Formal Language and Method*. *7787*, 154–165. https://doi.org/10.1007/978-3-642-39277-1

Yan, X. X., Wang, Q. X., & Ma, H. T. (2018). Path sensitive static analysis of taint-style vulnerabilities in PHP code. *International Conference on Communication Technology Proceedings, ICCT*, *2017-Octob*, 1382–1386. https://doi.org/10.1109/ICCT.2017.8359859

Zhang, K. (2019). A machine learning based approach to identify SQL injection vulnerabilities. *Proceedings - 2019 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019*, 1286–1288. https://doi.org/10.1109/ASE.2019.00164