AN ASSESSMENT OF DATASETS USED FOR THE EVALUATION OF TECHNIQUES FOR THE DETECTION OF SQL INJECTION VULNERABILITIES

BY

NURUDDEEN TIJJANI WURNO

(SPS/14/MCS/00015)

A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE, THROUGH THE DEPARTMENT OF COMPUTER SCIENCE, FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY, BAYERO UNIVERSITY, KANO, IN PARTIAL FULFILMENT FOR THE AWARD OF MASTER OF SCIENCE DEGREE IN COMPUTER SCIENCE

SUPERVISED BY

KABIRU UMAR (PhD)

JULY, 2019

DECLARATION

I hereby declare that this work isthe product of my research effort undertaken underthe supervision of Dr.Kabir Umar and has not been presented in Bayero University, Kano or elsewhere for the award of Masters in Computer Science. All sources referenced have been duly acknowledged.

Sign: _____ Date: _____

NURUDDEEN T. WURNO

SPS/14/MCS/00015

CERTIFICATION

This is to certify that the research work of this dissertation and its subsequent preparation is undertaken by **NURUDDEEN TIJJANI WURNO(SPS/14/MCS/00015)** was carried out under the Supervision of:

_____ _____

SUPERVISOR Date
Dr. Kabir Umar

_____ _____

H.O.D Date
Dr Yusuf Ibrahim Fagge

APPROVAL

This dissertation titled "**AN ASSESEMENT OF DATASETS USED FOR EVALUATION OF TECHNIQUES FOR DETECTION OF SQL INJECTION VULNERABILITIES"** meets the regulations for the degree ofMaster of Science in Computer Science of Bayero University, Kano and is approved for its contribution to knowledge and literary presentation

Prof. Ahmad Baita Garko	_____	_____
External Examiner	Signature	Date
Dr. Bashir S. Galadanci	_____	_____
Internal Examiner	Signature	Date
Dr. Kabir Umar	_____	_____
Supervisor	Signature	Date
Dr. Yusuf Ibrahim Fagge	_____	_____
Head of Department	Signature	Date
Dr. Ibrahim A. Lawan	_____	_____
SPS Representative	Signature	Date

ACKNOWLEDGMENT

All praise and thanks are due to Allah SWT, The most Gracious, The most Merciful for the gift of life and health throughout the period of this MSc programme.

First and foremost I offer my sincerest gratitude to my Supervisor, Dr. Kabir Umar, who has supported and guided me throughout my dissertation with his patience and expertise while allowing me to work on my own way. One simply couldn't wish for a better friendlier supervisor.

Deepest gratitude to my mentor Prof. Muhammad Yahuza Bello and to all my lecturers in the Computer Science department. Words cannot express my appreciation to my father Malam Tijjani Muhammad Wurno for his advice and concern throughout my MSc programme. Not forgetting my friends and colleagues without your cooperation and kindness, this degree could not have been completed.

Finally, I would like to thank Prof. Sani Lawan Taura for his support, and everyone who was important to the successful realization of this degree, as well as expressing my apology for not mentioning each of you personally.

DEDICATION

This work is dedicated to my lovely kids Abdurrahman Nuruddeen Wurno and Muhammad Nuruddeen Wurno with much affection for lighting up my life.

ABSTRACT

SQL injection vulnerabilities (SQLIVs) have been one of the top software security risks associated with web applications. A lot ofsignificant researchhas been done to study SQL injection attacks and a number of tools/techniques has been proposed by different researchers for the detection of SQL injection vulnerabilities in web applications. Literature investigation reveals that existing tools/techniques for the detection of SQLIVs were evaluated on diverse datasets.However, adoption of the same dataset across studies forms a strong basis for performance comparison across such studies. In this study, an assessment of datasets and performance metricsused for empirical evaluation of techniques for detection of SQLIVs is done by carrying out a comparative analysis of three existing popular open source web vulnerabilities scanners, which includeRe-Inforced PHP Scaanner (RIPS), PHP Application Vulnerabilities Scanner (PAVS) and OWASP Zed Attack Proxy (ZAP), and the tool that performs best is used to evaluate the most frequently useddatasets. RIPS is the tool that outperformed PAVS and ZAP having achieved a recall rate of 87.2%, an accuracy of 82.6% and a precision of 95%. The most frequently used datasets are PHP applications which include; SchoolMate 1.5.4, FaqForge 1.3.2, HotelRS, SugarCRM, Bookstore, Classifieds,and Forum. The most frequently used performance metrics are; Recall, Precision, Accuracy, Efficiency and response time. Hence, a web-based repository wasdeveloped to host these datasets, thereby,forming a set of most frequently used datasets and performance metrics for empirical evaluation of techniques for detection of SQLIVs in web applications.

Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.0 Preamble:

This chapter consists of the introduction of the research work, including a background of the study, the problem statement, aim and objectives, scope of the study, significance of the study, and contributions to knowledge. Finally, the chapter presents how this dissertation is organized.

1.1 Background of the Study

In recent years, the World Wide Web has evolved from a traditional information medium to a vital application bench.It has become the future in today's activitiessuch as; e-commerce,Internet Banking, art galleries, restaurant menus, opening time, retailing, education, societal network, government work etc.The web has become an essential aspect of business(Nagpal, Naresh , & Nanhay , 2017). The Internet is being used almost for every task. Through the web, people started to give their personal information to organizations they subscribed with and this sensitive information could be highly exposed to identitytheft, privacy violation, as well as other cyber threats.Huge volume of sensitive and confidential data is stored in the back-end databases, which makes them lucrative targets for attackers (Kar, Khushboo, Ajit, & Suvani, 2016). Therefore, security of web databases should be a prime concern for web developers.As the information grows on the web, concern for the security of that information grows exponentially(Nagpalet al., 2017).

The recent increase in the growth and use of the internet for a wide range of web-based applications has brought about the increasing popularity of web-based applications(Ogheneovo & Asagba, 2013).Web applications are a major source of information for business process critical for the survival of many organizations. For smooth operations of an organization that utilizes web applications, it is necessary that web applications operate at areasonable level of security(Nadeem et al., 2017). Due to the complexities of web technologies and varieties of risks, it is not an easy task to save the web applications from intruders and threats. Unfortunately, a great number of web developers are unaware of the weaknesses of the security of their web application and this is normal as there are thousands of lines of code which makes it difficult for them to identify the loopholes (AbdulRahman, Alya, Kamarularifin, & Fakariah, 2017).With new technologiesevolving daily, these web applications are facing more and more security problems(Ping, 2017).

With the popularity of web applications, there is also an increase in web application vulnerabilities. Nowadays, attackers use sophisticated tools or Botnets to automatically discover vulnerable web pages from search engines like Google and launch mass injection attacksfrom distributed sources (Kar, Suvasini, & Srikanth, 2015).The amount of known web vulnerabilities is increasing radically. Up to date security reports shows that web-based systems suffer nearly 26 attacks per minute(Huang et al., 2017). Among various security threats against web applications, SQLInjection Attack has been predominantly used for nearly 15 years(Nagpal, Naresh , & Nanhay , 2017).

SQL injection is an attack in which the attacker inserts SQL commands into forms or parameter values (AbdulRahman et al., 2017). SQL injection attack (SQLIA) is a simple and well-understood technique of inserting an SQL query segment usually through GET or POST parameters submitted to a web application. It occurs when untrusted user input data is used to construct dynamic SQL queries without proper validation. By using SQL injection, the attacker could gain unauthorized access to the web application that is linked with an organizations database and would be able to modify, update or steal the critically important information in the database. SQLIA is different from other attack, which does not squander system resource. At the same time, it is a vital imperil to military or banking database servers. The injected code snippet can change the behavior of the application when they show or modify the system information into SQL statements. The SQLIA is top-ranked within threatening web services security risk in 2013 according to data released by the Open Web Application Security Project (OWASP). According to Trust Wave 2012 Global Security report, SQL injection was the number one attack method for four consecutive years (Kar, Suvasini, & Srikanth, 2015). Under such circumstances, a variety of methodshave beenproposed by researchers to detect SQLIA which include; static analysis, dynamic analysis, combined static and dynamic vulnerability analysis, machine learning and taint tracking methods (Wu et al., 2016).

Even though thereare a lot of significant researches that have been conducted to study SQL injection vulnerabilities (SQLIVs), and a number of models/techniques have been proposed to prevent this type of attack. However, not a single model/technique can ensure an adequate level of security to protect the web application (AbdulRahman et al., 2017), which may be because of its diversity and large scope.To understand the evolving risks, there is a need to understand the

modern security risks, understand the limitation of traditional security systems and to develop an intelligentsecurity system.While an SQL injection is easy to prevent with the help of web vulnerability scanners that exist on the market, most of the scanners have the possibility to produce false negative and false positive results. Experiments are carried out with datasets to evaluate the performance of thesetechniques and are subsequently evaluated based on some performance metrics.

An experimental dataset is a collection of web applications, and a set of known attacks (i.e. injected queries) and some legitimate inputs (i.e. genuine queries) for determining the false positive and false negative rates. A false negative is referring to a result which indicates the web application is not vulnerable to the attack when it actually is, whereas the false positive is indicating the web application is vulnerable to the attack while it actually isn't.

This dissertation is aimed at exploring the most commonly used experimental datasets and performance metrics used for evaluation of proposed techniquesfor detection of SQL injection vulnerability. As such,researchers willing to evaluate the performance of their techniques could be properly guided on datasets to use for conducting experiments.

1.2 Problem Statement

For over a decade, SQL injection vulnerabilities (SQLIVs) have been one of the top software security risk associated with web applications (Open Web Application Security Project (OWASP) Top 10 –2004, 2007, 2013, 2011, 2017).

Several significant researcheshave been done to study SQL injection attacks and a number of tools/techniques have been proposed by different researchers for the detection of SQLIVs in web applications.There are existing popular open-source tools such as PAVS, RIPS,and ZAP that are used to perform experiments on the detection of SQLIVs.The performances of these tools/techniques have been evaluated using different datasets.Moreover, adoption of the same dataset across studies forms a strong basis for performance comparison across such studies.

Literature investigation reveals that existing techniques for detection of SQLIVs were evaluated on diverse datasets.There is no commonality of dataset across most studies, and as such, there is no common ground for performance comparison across the studies.

In addition, some experiments were conducted using toy applications as datasets (e.g. Toyrentals, BuyMilkOnline, College portal etc.), whereas some were conducted using highly matured web applications as datasets (e.g. SugarCRM, SchoolMate, Wordpress etc.). Thus, there is no balance across most studies.

These problems emphasize the need for thorough assessment of datasets and performance metrics used for empirical evaluation of techniques for detection of SQLIVs. To the best of our knowledge, there has not been an assessment of these different datasets.

This study aims at carrying out acomparative analysis of threepopular selected open source web vulnerabilities scanners and evaluation of datasets and performance metrics used for empirical evaluation of techniques for detection of SQLIVs in web applications, with the key goal of forming set of most frequently used datasets and performance metrics.

1.3 Aim and Objectives

The general aim of this study is to conduct anassessmentof datasets and performance metricsused forevaluation of techniques for the detection of SQLIVs in web applications.

The objectives of thisstudy include:

i. To perform usage assessment of datasets and performance metrics used for empirical evaluation of techniques for the detection of SQLIVs
ii. To perform a comparative analysis of three existing open source tools for detection of SQLIVs
iii. To perform an evaluation of the most frequently used datasets using the most accurate and precise tool from the result of objective (ii)
iv. Todevelop a web-based repository for the most frequently used datasets for evaluation of techniques for detection of SQLIVs.

1.4Scope of the Study

The scope of this studyis specific to datasets and performance metrics used forempiricalevaluation of tools/techniques for detection of SQL injection vulnerabilities in web application within the last five years (2017 – 2013).

1.5 Significance of the study

The significance of this research work is to form a set of most commonly usedexperimental datasets and performance metrics for the empiricalevaluation of tools/techniques for detection of SQL injection vulnerabilities and make them available online.This willguide researchers on the most appropriate datasets to be used forthe evaluation ofnewly proposedtechniques, and hence, form the basis of comparing performance among thetechniques.

1.6 Contributions to Knowledge

The study was able to make the following contributions

i. The study has conducted a thoroughassessment of different experimental dataset and performance metrics used for empirical evaluation of techniques for detection of SQLIVs, thereby, forming a set of the most appropriate experimental dataset for evaluation of proposedsolution on detection of SQLIV. To the best of our knowledge, there has not beenan assessment of these datasets and performance metrics.
ii. The study also developed an online repository for the most frequently used dataset for empirical evaluation of techniques for detection of SQLIVs in web applications. Thus, researchers could use them for evaluating newly proposed techniques.

1.7 Organization of the Dissertation

The remaining part of thedissertation is organized as follows:Chapter two present Literature Review, Chapter three presents the methodology adopted while conducting the study, Chapter four presents Experimental Results and Analysis, and Chapter fivepresents Summary, Conclusion,andRecommendations**.**

Chapter 2: Literature Review

2.0 Preamble:

This chapter presents a literature review,and in the process, the following issues werediscussed; web application vulnerabilities, SQL injection vulnerabilities, techniques for detection of SQL injection vulnerabilities, datasets and performance metrics for experimental evaluation of techniques for detection of SQL injection vulnerabilities, and lastly presents a summary of the related literature.

2.1 Web Application Vulnerabilities

Web application vulnerabilities have become, in recent years, a major threat to computer system security(Rim, Eric, Mohamed, & Vincent, 2014). Web application vulnerabilityinvolves a system flaw or weakness in web-based applications. They have been around for years, largely due to not validating or sanitizing form inputs, misconfigured web servers, and application design flaws, and they can be exploited to compromise the application's security(Ceccato, Cu , Dennis , & Lionel , 2016). These vulnerabilities are not the same as other common types of vulnerabilities, such as network or asset. They arise because web applications need to interact with multiple users across multiple networks, and that level of accessibility is easily taken advantage of by hackers. Web application vulnerabilities are some of the most common flaws leading to modern data breaches. Once an attacker has found a flaw, or application vulnerability, and determined how to access it, the attacker has the potential to exploit the application vulnerabilities to facilitate a cyber-crime. These crimes target the confidentiality, integrity, or availability (known as the "CIA triad") of resources possessed by an application, its creators, and its users(Nagpal, Naresh , & Nanhay , 2017). Attackers typically rely on specific tools or methods to perform application vulnerabilities discovery and compromise.

While there are many different tools and techniques for exploiting application vulnerabilities, there are handfulvulnerabilities that exist, but we briefly discuss those that are much more common than others as reported by the open web application security project (OWASP) top 10 – 2017 list.

i. **A1:2017-Injection:** Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

ii. **A2:2017-Broken Authentication:** Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.

iii. **A3:2017- Sensitive DataExposure:** Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

iv. **A4:2017-XMLExternalEntities (XXE):** Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

v. **A5:2017-BrokenAccess Control:** Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access to other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

vi. **A6:2017-SecurityMisconfiguration:** Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion.

vii. **A7:2017-Cross-SiteScripting (XSS):** XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or

JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

viii. **A8:2017-InsecureDeserialization:** Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.

ix. **A9:2017-Using Components with Known Vulnerabilities:** Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.

x. **A10:2017-InsufficientLogging &Monitoring:** Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

For this study, we are only considering SQL injection of the Injection flaw as it is rated as number one among the top 10 security risks (OWASP Top 10 Application Security Risk – 2017).

2.2 SQL Injection Vulnerabilities (SQLIVs)

An SQL injection is one of the vulnerabilities of web applications, which makes use of the user input field in a web application to create the SQL statement used to penetrate the back-end database. This attack happens when a web application uses the user's input data without encryption or proper validation in the query command(AbdulRahman et al., 2017).

Structured query language Injection vulnerabilities (SQLIV) is a type of code injection technique that targets databases to steal information from an organization(Sharma, Komal, Chandrakala, & Diksha, 2017). The attacker enters the malicious SQL commands into a SQL statement via the unconstrained user input parameters to manipulate the SQL queries logic. This leads to the threat to all those web applications that access their databases, through SQL commands establish with external input data. Through SQL injection the attacker neglects the authentication phase and

provides confidential information to the attacker. Authorized access to confidential information by a crafted user has unprotected their authority, confidentiality,and integrity. The results could be like the system couldn't deliver proper services to its customers.

SQL injection attacks (SQLIAs) consist of some methods of attacks to maliciously attack database. SQLIA can extract and read confidential or secure data from a database or it can modify any data. The database result, when expected from a malicious database or queries, gives an incorrect or unsolicited result(Agrawal & Upendra, 2017).

It is an attack technique that injects an SQL query in the input of a program, in order to read/write/admin a relational database by affecting the execution of predefined SQL statements. It can be accomplished by placing a meta-character into the input string, which acts as a modifier of the original SQL statement and allows the attacker to alter its behavior(Thomé, Lwin, Domenico, & Lionel, 2018). As an example, consider the code snippet below:

```
$sql="INSERT INTO client_log VALUES'$HTTP_REFERER');";
mysql_query$sql);
```

From the above example,`$HTTP_REFERER`is used to construct an SQL command. The referrer field of an HTTP request is an untrusted value given by the HTTP client; an attacker can set the field to:

```
'); TRUNCATE TABLE client_log
```

This will cause the code in the example to construct the `$sql` variable as:

```
INSERT INTO client_log VALUES(''); TRUNCATE TABLE client_log;
```

Table "`client_log`" will be emptied when this SQL command is executed. This technique, which allows for the arbitrary manipulation of backend database, is responsible for the majority of successful SQL injection attacks(Huang & Lee, 2005).

2.2.1 Types of SQLInjection Attacks (SQLIAs)

There are various methods of attacks present that are used by an attacker to extract data and to make an attack on queries(OWASP, 2012), these methods are classified as follows;

i. **Tautology:** In this technique, malicious contents are added using the conditional statement that always evaluates to true, it works on the injection of tokens to given query statement and it always considered it as true. It is mainlyused by query with 'WHERE' clause. For example, query for login is here

```
"SELECT * FROM employee WHERE login ID = '423'
      and password = 'sss' OR '1' '=' 1'
```

ii. **Union Query Attack:** In this technique, the malicious query is added with the safe query using the UNION keyword, the SQL gives permission to two queries to join and return as one result. Here, an attacker exploits a given variable from a query or it can add some queries by itself to infuse extra information and find out the data from the database. As an example;

Original query: `select acc_no from user where u_id='45'`
Injected Query:`select acc_no from user where u_id='45' UNION select card_no from card_details where u_id='45'`

iii. **Illegal/Logically Incorrect Queries:**in this technique, logically incorrect type of query is performed to have information about some structures of the database. When such a query is given, an error message is displayed from a database including some debugging information. This message displayed is sometimes useful for an attacker to find vulnerabilities present in a database. For example the query below,the data entered gives an irrelevant result which causes an error by which attacker also comes to know about the backend applications which are used.

```
SELECT *FROM users WHERE login='derived' AND password=password (select host from host)
```

iv. **Piggybacked Query:**in this attack technique, the original query is modified by adding some new distinct data in the queryto extract some more information from the database. Attackers use certain delimiters like ";" or"," to join the legitimate query with the illegitimate onewhich used to append an extra query. Example below

```
Select * from users where id="rabnawaz" and pwd=""; Drop table users…"
```

v. **Inference**:in this attack technique, the attacker changes the behavior of database and this is done by two techniques: Blind and timing techniques. In a blind attack, a series of the simple queries are performed to have a guess about the structure of the database. In timingattack, the query processing time is observed to infer some information present in the database.

vi. **Stored Procedures**:In this type of attack, the attacker focuses on procedures that are stored in a database system which is either user-defined or default procedures. A stored procedure is just a code which contains some vulnerabilities such as buffer overflow.

vii. **Alternate Encoding Attacks:**in this type of attack, the attacker changes the pattern of the SQL Injection so that it goes undetected from the common detection and prevention techniques by using hexadecimal, Unicode, octal and ASCII code representation in the SQL Statement. As techniques could not be able to detect the encoded strings and hence, allows these attacks to go undetected.

2.2.2 Consequences of SQL Injection Attacks:

It has been observed that due to access to the database SQLIA has become the dominant web application security risks over the last ten years. Database is very critical for successful operations of any organization. Sensitive information in the database can be used in many ways to serve the attackers purpose (Nadeem et al., 2017).The followings could be the intentions of the attackers to use SQLIA.

i. To gain information about user credentials
ii. To get the database schema
iii. To extract and modify the database
iv. To perform Denial of Services like shutting down the database, dropping tables, etc.
v. Replacements of files with false or tempered information
vi. Execution of remote commands
vii. Shoplifting, account balance change
viii. Interacting with the underlying operating system

Such type of information helps the attacker to proceeds or use more sophisticated attacks.

2.3 Techniques for Detection of SQLIVs

Several techniques have been proposed by different researchers for the detection of SQLIV in web applications. We were able to cast our view on a handful of literature addressing SQLIVs as discussed below.

Amnesia is atechnique that works by combining static and dynamic part for detecting web application vulnerabilities at the runtime(Sharma et al., 2017). The concept behind this solution is that the source code contains enough knowledge to interpret models of the authorized SQL queries generated by the application. They explained that the static part of the technique uses program analysis to automatically build a model of the authorized queries which will be generated by the application. In its dynamic part, it supervises the dynamically generated queries at runtime and checks them for confirmation with the model that was generated at the static part.

JoanAudit(Thomé, Domenico, Lwin, & Lionel, 2017), a static analysis tool for auditing web applications and web services for common injection vulnerabilities namely Cross-site scripting (XSS), SQL injection (SQLi), XML injection (XMLi), XPath injection (XPathi), and LDAP injection (LDAPi)during software development. This tool automatically identifies parts of the program code that is relevant for security and generates an HTML report to guide security auditors audit the source code in a scalable way. It can also automatically fix some cases of vulnerabilities in source code cases where inputs are directly used in sinks without any form of sanitization by using standard sanitization procedures.

Nadeem et al.,(2017), proposed a knowledge base model that is based on dynamic analysis for detection and prevention of SQLIVs. The technique which is composed of Dynamic Analyzer that works as a user would request the page and that request is received and analyzed to check that request is for pages without vulnerabilities and Testerwould generate the possible expected response from the user and user request would be served. On response from the user, the response is compared with the expected result and any discrepancy is observed.

AbdulRahman et al.,(2017), proposed a techniquecalled SDR (Scan-Detect-Report) to scan and detection SQLIV on the web environment based on the Boyer-Moore string matching algorithm. The model is composed of a four-panel reference detection modules made up of Crawler, Parameter testing, exploit and Report. The module will scan for the attribute or criteria of the

web application vulnerabilities and then launch the SQLIA detection Module to perform each of their functions based on the chosen string matching algorithm. The crawler will the website for parameter identification. The captured parameters are then tested to determine whether it is vulnerable to an SQL injection attack, and, if yes, the exploit panel is recommended which will use the same parameters to penetrate the database and will be able to show the names of the table columns and rows affected, and then the report panel will generate a report.

(Appelt, Panichella, & Briand, 2017)proposed a technique that combines machine learning with a Multi-objective genetic algorithm to automatically repair web application firewalls based on successful SQL injection attacks. With a set of legitimate inputs and bypassing SQL injection attacks, and corresponding path conditions extracted from learned classifiers, the technique infers a regular expression that, when added to the WAF's rule set, prevents as many attacks as possible from bypassing, while letting legitimate requests go through.

JOACO tool(Thomé, Lwin, Domenico , & Lionel , 2018), for detectinginjection vulnerabilitiesthrough Security Slicing and Hybrid Constraint Solving. The technique is composed of two steps:Security slicing & Attack conditions generation.It takes as input the byte-code of a Web application written in Java, a catalogue of vulnerabilities, and a list of threat models. The security slicing identifies sinks in the byte-code and for each sink computes the path condition leading to it and the associated context information. The constraint solving takes as input the attack conditions generated in the previous macro-step in the form of a constraint, pre-processes it and then pass it as input to a hybrid constraint solver, which orchestrates a constraint solving procedure for string and integer constraints with their search-based constraint solving procedure.

VuRLE, a tool for automatic detection and repair of vulnerabilities by learning from examples. The tool contains two phases: 1) the learning phase which learns transformative edits and their contexts (i.e., code characterizing edit locations) from examples of vulnerable codes and their corresponding repaired codes, then clusters similar transformative edits and extracts edit patterns and context patterns to create several repair templates for each cluster and context patterns to create severalrepair templates for each cluster, and the repair phase that detects and repairs vulnerabilities by selecting the most appropriate template(Siqi, David , Cong , & Robert , 2017).

VuRLE uses the context patterns to detect vulnerabilities and customizes the corresponding edit patterns to repair them.

Zhu, Guidong, Zhiquan, Boya, & Yongjun, (2017), proposed a technique based on Aho-Corasick (AC) multi-pattern matching algorithm combined with static analysis and dynamic tectonic attack mode to detect and prevent SQL injection attacks effectively. They extended the traditional AC multi-pattern matching algorithm and propose a two-tiered defense of techniques-the first tier is the fine-grained role-based access control (RBAC) modelthat contains roles, activities, granular RBAC SQL operations and data partitioning, and the second tier is an extended AC multi-pattern matching algorithm, which improves the detection efficiency and reduces the SQL statement detection time.

Moh, Santhosh, Sindhusha, & Teng-Sheng, (2016), proposed a technique for detecting SQL injection attacks using a multi-stage log analysis architecture, which combines both pattern matching and supervised machine learning methods. It uses logs generated by the application during attacks to effectively detect attacks and to help in preventing future attacks. The architecture consists of three main parts: (1) Log Generation: this includes a logging system such as Log4j, (2) Data Preprocessing: it handles preprocessing needed for machine learning and pattern matching methods, and (3) Detection Methods: such as Bayes net for machine learning and Kibana for pattern matching.

Kar et al., (2016), proposedSQLiGoTa novel approach to detect SQLIV by modeling SQL queries as a graph of tokens and using the centrality measure of nodes to train a Support Vector Machine (SVM) classifier.SQL queries were normalized into a sequence of tokens preserving the structural composition and captured the interaction between the tokens in form of a graph.The system is designed to work at the database firewall layer and can protect multiple web applications in a shared hosting scenario.

Sunkari (2016), proposed an extremely automated method for detecting and blocking SQL injection attacks in Web applications. The approach is made of Identifying dependable data options and tagging data coming from these options as dependable, Using dynamic tainting he is able to track dependable data at runtime, and Making it possible for only dependable data being SQL keywords and phrases or staff in problem strings.

Mahdi & Mohammad, (2016), proposed a technique to detect SQL injection attacks using hash algorithm. The approach provides clear, simple, actionable guidance for scanning SQL Injection vulnerability in a Web application by design a new black box testing analyzer which based on calculating the hash value for web response.

Umar, Abu Bakar, Hazura, Novia, & Mohd, (2016), proposed a new Pushdown Automaton (PDA) based static analysis technique that uses Context Free Grammar (CFG) analysis for detection of SQLiHs in web applications. The technique provides concrete information that can reliably and confidently guide both human tester/developer and SQLIVs detection tools/techniques as to which part of the source code to concentrate their efforts during detection and fixing of SQL injection flaws in an application. They made use of the computational power of PDA to trace data source of input variables found in dynamic queries to an AEP and consequently identifies all SQLiH in a very effective manner.

Kar, Khushboo, Ajit, &Suvasini,(2016), proposed a techniqueto detect SQL injection attack in real time using improved query normalizationand atwin Hidden Markov Model (HMM) ensemble. The system isintended to work at the database firewall layer, therefore it canprotect multiple web applications hosted on a shared server.Computational overhead was minimized by limiting the investigation to the 'WHERE' clause part of run-time queries. Though, the system has minimal performance impact which is imperceptible over the Internet.

Kar, Suvasini, & Srikanth, (2015), proposed a technique for real-time detection of SQL injection attacks using query transformation and document similarity measure. It adopted the strategy to examine only the 'WHERE' clause part of run-time queries and ignore 'INSERT' queries, which was based on two interesting observations made during the course of research. The technique is able to detect all types of SQL injection attacks by applying document similarity measure on normalized queries, implemented in a tool named SQLiDDS.

Jang & Jin-Young(2014), proposed a technique to detect SQL injection attack using query result size. The technique exhibit a novel scheme that automatically transforms web applications, rendering them safe against SQL injection attacks. The technique was implemented for protecting Java-based web applications, and it dynamically analyzes the developer-intended

query result size for any input and detects attacks by comparing this against the result of the actual query.

Joshi & Geetha(2014), proposed a technique for detecting SQL injection attacks based on machine learning.The technique devised a classifier for detection of SQL Injection attacks by using a combination of Naïve Bayes machine learning algorithm and Role Based Access Control mechanism for detection. The technique detects malicious queries with the help of this classifier.

Abdelhamid, Youcef, & Ahmed(2014), proposed a technique for improving WAFs to detect advanced SQL injection attacks. The technique is a hybrid Injection Prevention System (HIPS) which uses both a machine learning classifier and a pattern matching inspection engine based on reduced sets of security rules, it dissects the HTTP traffic and inspects complex SQL injection attacks in order to cover most evasion techniques and improve security rules management system.

Sheykhkanloo(2014), proposed a technique for the detection of SQL injection attack by employing Neural Networks (NNs). The technique includethree main elements of: a Uniform Resource Locator (URL) generator in order to generate thousands of malicious and benign URLs, a URL classifier the in order to classify the generated URLs to either benign or malicious URLs, and an NN model in order to detect either a given URL is a malicious URL or a benign URL. The URL generator and the URL classifier are employed in order to provide the required malicious and benign URLs for three phases of testing, validating, and theraining of the NN model.

Takeshi (2014), derived the formula to calculatethe parameter of zeta distribution and proposed a technique to detect SQL injection attacks based on the above formula using the approximation function of zeta distribution.

Kumar & Indu(2014), proposed a technique for detection of SQL injection attacks by implementingMessage Authentication Code (MAC) based solution against SQL injection attacks. The technique works both on client and server side. Client-side implements a filter function and server side is based on information theory. MAC of static and dynamic queries is compared to detect SQL injection attack. The technique applies the concept of information

theory for attack detection. Entropy is defined as the information content of a query written by a programmer which should remain intact. When a malicious input alters the static nature of the query, the complexity value changes.

CRAXweb was implemented as an automatic exploit generator for web security issues on real-world web applications, including cross-sitescripting and SQL injection attacks(Huang, Han-Lin, Wai-Meng, & Huan , 2013). The symbolic socket is used as the input for symbolic executions on the web platformsby detecting symbolic queries to SQLservers or symbolic responses to HTTP servers.

ANDROMEDA, an analysis tool that computes data-flow propagations on demand, in an efficient and accurate manner, and additionally features incremental analysis capabilities. The technique uses a security-analysis algorithm featuring local, demand-driven tracking of vulnerable information flowssuchas cross-site scripting (XSS), SQL injection (SQLi) and log forging.(Tripp et al., 2013).ANDROMEDA is currently in use in a commercial product. It supports applications written in Java, .NET,and JavaScript.

DIGLOSSIA(Son, Kathryn , & Vitaly , 2013), a tool that precisely and efficiently detects code injection attacks on server side Web applications generating SQL and NoSQL queries. The key technical innovation is *dual parsing*. To detect injected code in a generated query, DIGLOSSIA parses the query in tandem with its shadow and checks that (1) the two parse trees are syntactically isomorphic, and (2) all code in the shadow query is in shadow characters and, therefore, originated from the application itself, as opposed to user input.

Table 2.1 presents a summary of existing techniques for detection of SQLIVs including the vulnerability analysis approach used in each study.

Table 2.1: Existing Techniques for Detection of SQLIVs

SN	Ref	Title	Vulnerability Analysis Approach	Software Tool	Summary of Technique
1	GS-004	Detection and Prevention of SQL Injection Attacks by Dynamic Analyzer and Testing Model	Dynamic Analysis		The study has proposed a knowledge base model that is based on dynamic analysis which is composed of Dynamic Analyzer and Tester. The model was able to detect and block SQLIA guided by rule set
2	GS-008	SQL Injection Attack Scanner using Boyer-Moore String Matching Algorithm	Dynamic Analysis	SDR(Scan-Detect-Report)	The study proposed a detection model called SDR to scan SQLiV on the web environment based on Boyer-Moore string matching algorithm. The model was able to detect vulnerable applications by scanning for the defined criteria of SQLiA guided by the four-panel reference detection modules made up of Crawler, Parameter testing, exploit and Report.
3	GS-010	Automatically repairing Web Application Firewalls based on successful SQL Injection Attacks	Dynamic Analysis	NSGA-II (A multi-level genetic algorithm)	The study proposed an approach that combines machine learning with a Multi-objective genetic algorithm to automatically repair web application firewalls based on successful SQL injection attacks. The approach was able to prevent many attacks guided by the OWASP core rule set to detect malicious request.

4	GS-011	An Integrated Approach for Effective Injection Vulnerability Analysis of Web Applications through Security Slicing and Hybrid Constraint Solving	Static Analysis	JOACO	The study proposed an integrated approach based on static analysis on a tool JOACO that leverages the synergetic combination of security slicing with hybrid constraint solving. The approach is a general one since it can detect any type of vulnerability whose threat model can be described using regular expression.
5	GS-017	Detecting Security Vulnerabilities in Object-Oriented PHP Programs	Combined Static Analysis with Dynamic Analysis	**OOP**IXY	The study proposed OOPixy, a static analysis tools that detect vulnerabilities in PHP applications. The tool was able to report various types of security vulnerabilities based on inter-procedural data flow analysis to track the values of variables and dynamically analyze data structures.
6	GS-018	A Countermeasure to SQL Injection Attack for Cloud Environment	Dynamic analysis	CCSD (Cloud Computing SQLIA Detection)	The study proposed a mechanism called CCSD to detect SQLIA on applications in a cloud environment. The tool was able to detect SQLIA by comparing the structure of a parse tree generated from user input with the corresponding parse tree stored in the repository.
7	GS-029	A Mutation Approach of Detecting SQL Injection Vulnerabilities	Dynamic analysis	MOSA	The study proposed an approach and implemented a tool MOSA to detect SQL Injection vulnerabilities based on a set of mutation operators. The tool is able to detect vulnerabilities by applying an efficient mutation approach for mining SQL injection vulnerabilities.
8	GS-031	SECSIX: Security Engine for CSRF, SQL Injection,and XSS Attacks	Dynamic Analysis	SECSIX	The study proposed an approach based on dynamic analysis to counter SQLi, XSS,and CSRF. The approach was able to prevent vulnerabilities by scanning all files and returns vulnerable lines in the application code as hotspots.

9	GS-038	A Second-Order SQL Injection Detection Method	Dynamic Analysis		The study proposed a method and implemented a prototype for preventing second-order SQL injection attacks. The method was able to defend vulnerabilities by randomizing the SQL keywords in the trusted constant string which creates new sets of keywords that differ with standard keywords in attack payload that an attacker injects or stores in database or file system.
10	GS-053	Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining	Static Analysis	WAP (Web Application Protection)	The study proposed an approach based on static analysis and implemented a tool for detecting and correcting vulnerabilities in PHP based web applications. The approach was able to detect and correct vulnerabilities using a combination of two techniques; taint analysis to detect vulnerabilities and data mining to discover false positives.
11	GS-057	SQLiGoT: Detecting SQL Injection Attacks using Graph of Token and SVM	Dynamic Analysis	SQLiGoT	The study proposed an approach to detect SQL injection attack and implemented a prototype called SQLiGoT. The approach was able to detect vulnerabilities by modeling SQL queries as graphs of token and using centrality nodes to train an SVM classifier.
12	GS-070	An Automated Security Oracle for Black-Box Testing of SQL Injection Vulnerabilities	Dynamic Analysis	SOFIA	The study proposes the tool SOFIA, a security oracle for SQLi vulnerabilities. The tool learns a safe model characterizing legitimate SQL statements based on information logged during normal system execution, and as such is able to classify new SQL statements by comparing and contrasting them with the safe model.

13	GS-071	Detection of SQL Injection Attacks using Hidden Markov Model	Dynamic Analysis		The study proposed a novel approach to detect SQLIA in real-time using a twin HMM ensemble and improved query normalization. The model was able to detect SQLIV at runtime using the model generated by training with known samples of genuine and injected queries.
14	GS-075	Behind an Application Firewall, Are We Safe from SQL Injection Attacks?	Dynamic Analysis	Xavier	The study proposed a machine-learning driven approach to detect SQLIV in firewalls. The approach was able to detect vulnerabilities by generating SQLIAs bypassing the WAF as a test, then incrementally learning from the test that are blocked or passed by the firewall.
15	GS-077	SQLiDDS: SQL Injection Detection using Query Transformation and Document Similarity	Dynamic Analysis	SQLiDDS	The study proposed a novel approach for real-time detection of SQLi attacks. The approach implemented on a tool SQLiDDS is able to detect all types of SQLIV by applying document similarity measure on normalized queries.
16	GS-079	Security Slicing for Auditing XML, Xpath and SQL Injection Vulnerabilities	Static Analysis	JoanAudit	The study proposed an approach based on state-of-the-art program slicing to assist the security of common injection vulnerabilities. The approach facilitates security auditing for several types of vulnerabilities by first applying static analysis to identify the sinks and sources, then apply specific program slicing techniques to extract minimal and relevant source code that contains statements required for auditing potential vulnerabilities related to the sink.
17	GS-086	Static Detection of Second-Order Vulnerabilities in Web Applications	Static Analysis		The study proposed an approach based on static analysis to detect second-order vulnerabilities and multi-step exploits in web applications. The approach was able to detect vulnerabilities by identifying un-sanitized data flows by connecting input and output points of data in persistent data stores.

18	GS-087	Automated Testing for SQL Injection Vulnerabilities: An Input Mutation Approach	Dynamic Analysis		The study proposed an automated black-box testing approach based on a set of mutation operators that manipulate inputs to create new test input to trigger SQLIAs. The approach is able to detect vulnerability by combining the operators in different ways then applying multiple operators to the same input, thereby generating inputs that contain new attack patterns.
19	GS-089	Automatic Detection and Correction of Web Application Vulnerabilities using Data Mining to Predict False Positives	Combined Static and Dynamic analysis	MAP	The study proposed a hybrid approach for detecting and correcting vulnerabilities in web applications, which is composed of taint analyzer and supervised machine learning. The approach was able to detect vulnerabilities and maintaining low false positives by combining human coded knowledge about the vulnerability versus automatic machine learning for data mining.
20	GS-090	Detection model for SQL Injection Attack: An Approach for Preventing a Web Application from the SQL Injection Attack	Dynamic Analysis		The study proposed a model based on dynamic analysis which is composed of one main detection module consisting of the crawler, parameter testing, exploit and report panels. The model was able to detect and recognize web vulnerabilities by searching for the suspicious and defined criteria of the SQLIA guided by the Boyer Moore string matching algorithm.
21	GS-092	Semantic Security Against Web Application Attacks	Dynamic Analysis		The study proposed an ontology-based technique for detecting and classifying web application attacks. The system is capable of detecting sophisticated attacks effectively and efficiently by analyzing a specific portion of the user request, guided by semantic rules that capture the context of the application, possible attacks,and protocol used.

22	GS-093	Detecting SQL Injection Attacks using Query Result Size	Dynamic Analysis	IDL (Injection Detector Library)	The study proposed a novel technique that modifies applications to guard themselves against SQLIAs. The approach prevents applications from SQLIAs using taint propagation to analyze the code and automatically check the legitimate queries that could be generated by the application, then estimates the size of the generated query result.
23	GS-098	Simulation of Built-in PHP Features for Precise Static Code Analysis	Static Analysis	RIPS	The study proposed a novel approach based on precise static code analysis to detect security vulnerabilities in web applications. The approach was able to accurately and automatically detect taint-style vulnerabilities in PHP applications by performing an intra and inter-procedural data flow analysis to create summaries of data flow within the application and perform context-sensitive string analysis to refine the taint analysis result.
24	GS-099	Search-Based Security Testing of Web Applications	Dynamic Analysis	BIOFUZZ	The study proposed a technique based on a black-box approach for security testing of web applications. The technique was implemented on a tool BIOFUZZ and was able to automatically detect SQLi vulnerabilities through targeted test generation by using searched-based testing to systematically crawls a web application and evolve inputs whose effect on the SQL interaction are assessed at the interface between the web server and database.

25	GS-124	An Automated Black-Box Approach for Web Vulnerability Identification and Attack Scenario Generation.	Dynamic Analysis	Wasapy	The study proposed a methodology based on black-box analysis and implemented a vulnerability scanner (wasapy) that allows automatic identifying residual vulnerabilities of the web application. The approach can automatically identify and exploit vulnerabilities based on web page clustering technique guided by a black box analysis of the targeted application.
26	GS-126	DIGLOSSIA: Detecting Code Injection Attacks with Precision and Efficiency	Dynamic Analysis	DIGLOSSIA	The study proposed the design and implementation of DIGLOSSIA; a tool capable of detecting SQL and NoSQL injection attacks on server-side web applications. The tool was able to detect code injection by following Ray and Ligatti's definition of code and non-code combined with precise character-level taint tracking.
27	GS-129	ANDROMEDA: Accurate and Scalable Security Analysis of Web Applications	Static Analysis	ANDROMEDA	The study proposed a novel approach and implemented ANDROMEDA a sound and highly accurate static security scanner which is composed of demand-driven taint analysis and incremental analysis. The key idea behind the tool is to track vulnerable information flow in a demand-driven manner, without eagerly building any complete representation of the web application.
28	GS-130	Mining SQL injection and Cross Site Scripting Vulnerabilities using Hybrid Program Analysis	Combined Static and Dynamic analysis		The study proposed a prediction model based on hybrid program analysis that is composed of both classification and clustering. The model was able to predict vulnerabilities through the use of a pattern mining approach based on dynamic analysis that classifies input validation and sanitization functions through the systematic execution.

29	GS-136	Predicting SQL Injection and XSS Vulnerabilities through Mining input Sanitization Patterns	Static Analysis	PhpMinerI	The study proposed a model based on static analysis for predicting SQLi and XSS vulnerabilities. The model was implemented on a prototype tool called PhpminerI and was able to detect vulnerabilities through mining input sanitization pattern with a set of static code attribute that characterizes such code pattern, then build prediction models based on the historical information.
30	GS-139	Early Detection of SQL Injection Attack	Dynamic Analysis		The study proposed a light-weight client-side SQLIA detection framework based on conditional entropy metrics. The approach is light-weight and is able to detect SQLIV by extracting query structures from the server side code and convert them to shadow queries, then measure the deviation of information content between the shadow queries and actual input values.
31	GS-142	Detection Method of the Second-order SQL Injection in Web Applications	Combined Static and Dynamic analysis		The study proposed a solution based on combined static and dynamic analysis method to detect second-order SQLi. The solution first analyzes source code to find out the vulnerable data item pair probably containing the second-order SQLi and transforms it into an effective test sequence, after which are combined together with malicious inputs for testing.
32	GS-145	CRAXweb: Automatic Web Application Testing and Attack Generation	Dynamic Analysis	CRAXweb	The study proposed and implemented an automatic exploit generator for web security issues on real-world web applications. The tool exploits vulnerabilities by detecting symbolic queries to SQL server or symbolic responses to HTTP servers.

33	GS-156	A Parse Tree Model for Analyzing and Detecting SQL Injection Vulnerabilities	Combined Static and Dynamic Analysis		The study proposed a model that combines static and dynamic approaches to counter SQLIA. The model is based on the grammatical structure of an SQL statement using a parse tree to test a query by dynamically generating a parse tree and comparing their structures at runtime determining if they match or not.

Table 2.1 presenting asummary of existing techniques for detection of SQLIVs in web applications from studies between 2013-2017, the vulnerability analysis approach used in each technique and the software tool used for detection. Below is a pie-chart showing the distribution of analysis approach across the selected studies.

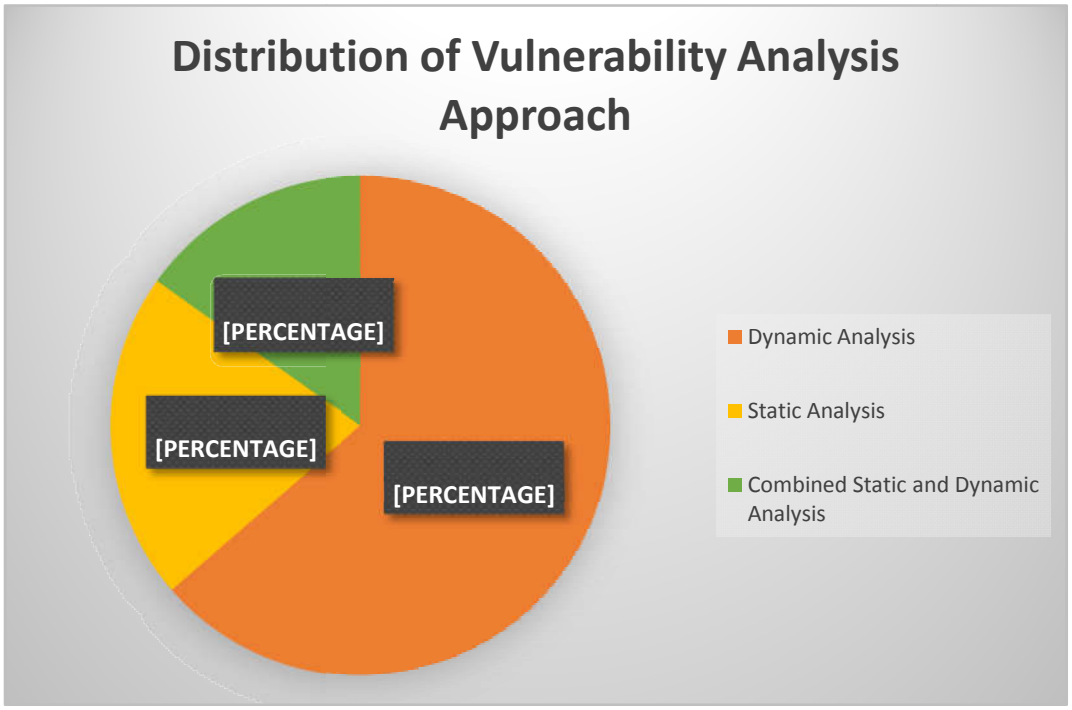

Figure 2.1: Distribution of Vulnerability Analysis Approach used in Studies

As observed in figure 2.1, studies that use dynamic analysis approach in their technique carry 65%, while studies that use static analysis approach in their technique carry 21%, and studies that used combined static and dynamic analysis approach carry 15% of the total selectedstudies. Thus, we can conclude that dynamic analysis is the vulnerability analysis approach that is favored by the research community within this domain.

2.4 Datasets and Performance metrics for Experimental Evaluation of Techniques for Detection of SQLIVs

A dataset is a collection of different values, features, numeric, character or string sequences etc., the dataset may be of any type like attack groups, images, SMS collection, Audio or video, Biometric datasets, etc.Various organizations providea dataset for research purpose based on their types and applications(Shweta, Satyawan , Vishal , & Romil , 2012).

Research comprises different parameters or designing descriptions like type of input and its features, output type and its feature, system at which propose technique is evaluated by different parameters such as accuracy percentage of system,positive behavior like TPR(True Positive Rate) and FPR(False Positive rate), and Negative behavior like TNR(True Negative rate) and FNR(False Negative Rate).Training time defines how much time system takes to learns by taking a decision, Detection time, defines time taken by the system to detect behavior, pattern, fingerprints, signatures etc. Different types of datasets or inputs are required by the system to analyze and provide behavioral study (Shweta et al., 2012).

A performance metric measures techniques behavior, activities,and performance which support a range of stakeholder need for users (Wikipedia). While many metrics are inwardly focused on the performance of a technique, metrics may also focus on the performance against user requirement and value. Performance metrics are used to assess the ability of a technique and consist of measuring criteria such as safety, time, cost, resource utilization, scope etc.

In an attempt to find datasets and performance metrics commonly used for testing web application vulnerabilities, we reviewed the various literature that proposes a solutionon detection and/or prevention of SQL injection vulnerabilities in web applications.

AbdulRahman et al., (2017), proposed an SQLIV detection model called SDR (Scan-Detect-Report) that is based on dynamic analysis and evaluated the model using 10 URLs of PHP web application with respect to efficiency and accuracy.

Nashaat, Karim, & James(2017), proposed a tool to detect vulnerabilities in PHP applications called OOPixy which combines static and dynamic analysis. The tool was evaluated on 5 PHP web applications with respect to precision and recall.

Wu et al.,(2016), proposed a mechanism based on dynamic analysis called CCSD to detect SQLIV on applications in a cloud environment. The technique was evaluated with 5 PHP applications with respect to response time and accuracy.

Nagpal, Naresh, & Nanhay(2017), proposed an approach based on dynamic analysis to counter SQLi, XSS,and CSRF. The approach was evaluated using 3 PHP applications with respect to recall.

Ping(2017), proposeda method and implemented a prototype for preventing second-order SQL injection attacks. The prototype was evaluated using 3 PHP applications with respect to response time.

Medeiros, Nuno,& Miguel (2016), proposedan approach based on static analysis and implemented a tool WAP (Web Application Protection) for detecting and correcting vulnerabilities in PHP based web applications. WAP was evaluated with 10 web applications with respect to accuracy and precision.

Kar, Suvasini, & Srikanth(2016), proposed an approach to detect SQLIV that is based on dynamic analysis and implemented a prototype called SQLiGoT. The approach was evaluated using 5 fully vulnerable web applications with respect to accuracy, recall,and precision.

Ceccato et al.,(2016), implemented the tool SOFIA, a security oracle for SQLIV which is based on dynamic analysis, and was evaluated on 6 PHP and Java applications with respect to accuracy, recall and response time.

Kar, Khushboo, Ajit, & Suvasini(2016), proposed a novel approach based on dynamic analysis to detect SQLIV in real-time. The approach was evaluated using 5 PHP applications with respect to accuracy precision and recall.

Appelt, Cu, & Lionel(2015), proposed a machine-learning driven approach based on dynamic analysis to detect SQLIV in firewalls and evaluated the efficiency of the approach using 3 PHP web applications.

Kar, Suvasini, & Srikanth(2015),proposed a novel approach for real-time detection of SQLIV and was implemented on a tool SQLiDDS. The tool was evaluated using 5 PHP web applications with respect to accuracy, precision,and recall.

Dahse & Thorsten(2014),proposed an approach based on static analysis to detect second-order vulnerabilities and multi-step exploits in web applications. The approach was evaluated using 6 PHP based applications with respect to scan and memory consumption.

Appelt D., Cu, Lionel, & Nadia(2014),proposed an automated black-box testing approach to detect SQLIV. The approach was evaluated based on recall with 2 PHP open source web applications.

Medeiros, Nuno, & Miguel(2014), proposed a hybrid approach that combines static and dynamic analysis for detecting and correcting vulnerabilities in web applications, which is composed of taint analyzer and supervised machine learning. The approach was evaluated using 35 PHP web applications with respect to accuracy and precision.

Buja, Kamarularifin, Fakariah, &Teh(2014), proposed a model based on dynamic analysis for detection of SQLIV and evaluated the model in terms of efficiency and accuracy using 10 URLs of web applications.

Son, Kathryn, & Vitaly(2013), proposed the design and implementation of DIGLOSSIA; a tool capable of detecting SQL and NoSQL injection attacks on server-side web applications. The tool was evaluated using 10 PHP applications with respect to efficiency and accuracy.

Shar, Hee Beng, & Lionel(2013), proposed a prediction model for SQLIV that combines static and dynamic based on hybrid program analysis that is composed of both classification and clustering. The model was evaluated with 6 PHP applications with respect to accuracy, precision,and recall.

Shahriar, Sarah, & Wei-Chuen(2013), proposed a light-weight client-side SQLIV detection framework based on conditional entropy metrics, and was evaluated with 3 PHP web applications.

Yan et al., (2013), proposed a solution based on combined static and dynamic analysis method to detect second-order SQLIV. The efficiency of the technique was evaluated using 4 (2 PHP and 2 Java) web applications.

Appelt, Panichella, & Briand(2017), proposed an approach based on dynamic analysis that combines machine learning with a Multi-objective genetic algorithm to automatically repair web application firewalls based on successful SQL injection attacks. The approach was evaluated using 3 open source and 4 proprietary Java web applications with respect to recall and the hyper-volume metrics.

Thomé, Lwin, Domenico, & Lionel(2018), study implemented an integrated approach based on static analysis on a tool JOACO that leverages the synergetic combination of security slicing with hybrid constraint solving to detect SQLIV. The tool was evaluated using 11 diverse and representative Java web applications with respect to recall and precision.

Thome, Lwin, & Lionel(2015), proposed an approach based on state-of-the-art program slicing to assist the security of common injection vulnerabilities. The approach was evaluated using 5 Java web applications with respect to precision, soundness,and scalability.

AbdulRazzaq et al., (2014), proposed an ontology-based technique based on dynamic analysis for detecting and classifying web application attacks and was evaluated on WebGoat (a Java application) with respect to recall, throughput and response time.

Jang & Jin-Young (2014), proposed a novel technique based on dynamic analysis that modifies applications to guard themselves against SQLIV. The efficiency and accuracy of the technique were evaluated with 5 Java web applications.

Tripp et al.,(2013), proposed a novel approach and implemented ANDROMEDA a sound and highly accurate static security scanner which is composed of demand-driven taint analysis and incremental analysis. The approach was evaluated using 16 Java web applications with respect to accuracy, precision and response time.

Ogheneovo & Asagba(2013), proposed a model that combines static and dynamic approaches to counter SQLIV. The model was evaluated using 6 Java web applications with respect to precision and response time.

Nadeem et al.,(2017),proposed a knowledge base model that is based on dynamic analysis which is composed of Dynamic Analyzer and Tester to detect and prevent SQLIV. The approach was evaluated using 5 ASP.net applications with respect to response time.

Table 2.2 presents a summary of dataset and performance metrics used in existing techniques for detection of SQLIVs from selected studies.

Table 2.2: Experimental Subjects and Performance Metrics used in existing techniques for detection of SQLIV

ID Or Ref	Title of study	Subjects used for experiment	Language of Subjects	Performance metrics used for evaluation
GS-004	Detection and prevention of SQL injection attacks by Dynamic Analyzer and Testing Model	1. portals 2. Classifieds 3. Online shopping 4. University database 5. Financial database	ASP.net	Response time
GS-008	SQL Injection Attack Scanner using Boyer-Moore string matching algorithm	1.www.dracoders.com/games.php 2.senesco.com/newsitem.php 3.www.ath-elite.com.au/trainers.php 4.www.vivactiv.ru/trainings/trainers.php 5.www.pushingpetals.com/buy.php 6.www.suffolkconstruction.com/staffView.php 7.www.votexgym.com/trainer.php 8.www.ureka-sg.com/trainers.php 9.www.centos.org/modules/tinycontent/index.php 10.www.rspba.org/html/newsdetail.php	PHP	Efficiency and accuracy
GS-010	Automatically repairing Web Application Firewalls based on successful SQL Injection Attacks	<u>Open source</u> 1.doPayment 2.expireTicket 3.simulatePayement <u>Proprietary</u> 4.operation 1 5.operation 2 6.operation 3 7.operation 4	Java	Recall Hyper-volume (HV)

GS-011	An Integrated Approach for Effective Injection Vulnerability Analysis of Web Applications through Security Slicing and Hybrid Constraint Solving	1.WebGoat 2.Roller 3.Pebble 4.Regain 5.PSH 6.TPC-APP 7.TPC-C 8.TPC-W 9.RAP 10.Bodgeit 11.OMRS-LUI	Java	Recall Precision
GS-017	Detection Security Vulnerabilities in Object-Oriented PHP Programs	1.Xoops 2.Phpnuke 3.b2evolution 4.Concrete5 5.Magento	PHP	Precision Recall
GS-018	A Countermeasure to SQL Injection Attack for Cloud Environment	1.Employee directory 2.Bookstore 3.Events 4.Classifieds 5.Portals	JSP	Accuracy Response-time
GS-029	A Mutation Approach of Detecting SQL Injection Vulnerabilities	1.HotelRS 2.SugarCRM	PHP	
GS-031	SECSIX: Security engine for CSRF, SQL Injection and XSS attacks	1.College portal 2.Toyrental 3.BuyMilkOnline	PHP	Recall
GS-038	A second-order SQL injection Detection Method	1.webchess 2.sql-labs 3.schoolmate	PHP	Response-time
GS-053	Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining	1.currentcost 2.DVWA 1.0.7 3.emoncms 4.Measureit 1.14 5.Mfm 0.13 6.Multillidae 2.3.5 7. OWASP Vicnum 8.SAMATE 9.Wackopicko 10.ZiPEC 0.32	PHP	Accuracy and Precision
GS-057	SQLiGoT: Detecting SQL Injection Attacks using Graph of Token and SVM	1.Bookstore 2.Forum 3.Classifieds 4.NewsPortal 5.JobPortal	PHP	Accuracy Recall Precision
GS-070	An Automated Security Oracle for Black-Box Testing of SQL Injection Vulnerabilities	1.HotelRS 2.SugarCRM 3.Taskfreak 4.theorganizer 5.Wordpressnewstatpress 6.Wordpresslandingpage	PHP	Accuracy Recall Response-time

GS-071	Detection of SQL Injection Attacks using Hidden Markov Model	1.Bookstore 2.NewsPortal 3.Forum 4.JobPortal 5.Classifieds	PHP	Accuracy Precision Recall
GS-075	Behind an Application Firewall, Are We Safe from SQL Injection Attacks?	1.HotelRS 2.SugarCRM 3.Cyclose	1&2 PHP 3 Java	Efficiency
GS-077	SQLiDDS: SQL Injection Detection using Query Transformation and Document Similarity	1.Bookstore 2.Forum 3.Classifieds 4.NewsPortal 5.JobPortal	PHP	1.Accuracy 2.Precision 3.Recall
GS-079	Security Slicing for Auditing XML, Xpath and SQL Injection Vulnerabilities	1.WebGoat 5.2 2.Roller 5.1.1 3.Pebble 2.6.4 4.Regain 2.1.0 5.PubSub 0.3	Java	Precision Soundness Scalability
GS-086	Static Detection of Second-Order Vulnerabilities in Web Applications	1.OpenConf 5.30 2.HotCRP 2.61 3.osCommerce 2.3.3.4 4.NewsPro 1.1.5 5.MyBloggie 2.1.4 6.Scarf 2007-02-27	PHP	Scan time Memory consumption
GS-087	Automated Testing for SQL Injection Vulnerabilities: An Input Mutation Approach	1.HotelRS 2.SugarCRM	PHP	Recall
GS-090	Detection model for SQL Injection Attack: An Approach for Preventing a Web Application from the SQL Injection Attack	1.http://www.dracoders.com/games.php 2.http://senesco.com/newsitem.php 3.http://www.ath-elite.com.au/trainers.php 4.http://www.vivactiv.ru/trainings/trainers.php 5.http://www.pushingpetals.com/buy.php 6.http://www.suffolkconstruction.com/staffView.php 7.http://www.vortexgym.com/trainers.php 8.http://www.ureka_sg.com/trainers.php 9.http://www.centos.org/modules/tinycontent/index.php 10.http://www.rspba.org/html/newsdetail.php	PHP	Efficiency Accuracy
GS-092	Semantic Security Against Web Application Attacks	WebGoat	Java	Recall Throughput Response-time
GS-093	Detecting SQL Injection Attacks using Query Result Size	1.Bookstore 2.Portal 3.Classifieds 4.Events 5.Empldir	JSP	Efficiency Accuracy

GS-098	Simulation of Built-in PHP Features for Precise Static Code Analysis	1.HotCRP 2.60 2.MyBB 1.6.10 3.osCommerce 2.3.3 4.phpBB2 2.0.23 5.phpBB3 3.0.11	PHP	Precision
GS-099	Search-Based Security Testing of Web Applications	1.WebChess 0.9.0 2.Schoolmate 1.5.4 3.FaqForge 1.3.2 4.geccBBlite 0.2 5.phpMyAddressbook 8.2.5 6.Elemata RC3.0	PHP	Efficiency Accuracy
GS-124	An Automated Black-Box Approach for Web Vulnerability Identification and Attack Scenario Generation.	1.phpBB3 2.SecurePage 3.HardwareStore 4.Insecure 5.DVWA 6.Cyphor 7.Seagull 8.Fttss 9.Riotpix 10.Pligg	PHP	Efficiency
GS-126	DIGLOSSIA: Detecting Code Injection Attacks with Precision and Efficiency	1.MongoPress 2.mongodb-admin 3.mongodb-php-basic 4. rockmongo 5.MongoTinyURL 6.simple-user-auth 7.faqforge 8.schoolmate 9.webchess 10.MyBB with MyYoutube(1.0)	PHP	Precision Efficiency
GS-129	ANDROMEDA: Accurate and Scalable Security Analysis of Web Applications	1.AjaxChat 0.8.3 2.AltoroJ 1.0 3.AppA 4.Blojsom 3.1 5.BlueBlog 1.0 6.Contineo 2.2.3 7.Dlog 3.0-BETA-2 8.Friki 2.1.1-58 9.GestCV 1.0 10.Ginp 1.0 11.JBoard 0.3 12.JPetstore 2.5.6 13.JugJobs 1.0 14.Photov 2.1 15.StrutsArticle 1.1 16.WebGoat 5.1-20080213	Java	Accuracy Response-time Precision
GS-130	Mining SQL injection and Cross Site Scripting Vulnerabilities using Hybrid Program Analysis	1.SchoolMate 1.5.4 2.FaqForge 1.3.2 3.Utopia News Pro 1.1.4 4.Phorum 5.2.18 5.CuteSITE 1.2.3 6.PhpMyAdmin 3.4.4	PHP	Accuracy Recall Precision

GS-136	Predicting SQL injection and XSS Vulnerabilities through Mining input Sanitization Patterns	1.SchoolMate 1.5.4 2.FaqForge 1.3.2 3.WebChess 0.9.0 4.Utopia News Pro 1.1.4 5.Yapig 0.95b 6.PhpMyAdmin 2.6.0-pl2 7.PhpMyAdmin 3.4.4 8.CuteSite 1.2.3	PHP	Recall
GS-139	Early Detection of SQL Injection Attack	1.PHP-Address book 2.Serendipity 3.PHP fusions	PHP	Conditional entropy metrics (H_C, H_V, H_T, H_L)
GS-142	Detection Method of the Second-order SQL Injection in Web Applications	1.SchoolMate 1.5.4 2.WebChess 1.0.0 3.SecondhandBookstore 4.HotelManagementSystem	1&2. PHP 3&4. Java	Efficiency
GS-145	CRAXweb: Automatic Web Application Testing and Attack Generation	1.SchoolMate 1.5.4 2.Webchess 1.0.0rc2 3.Faqforge 1.3.2 4.EVE 5.SimpleGB 1.49.02 6.DedeCms 5.6 7.Django-admin 0.96.1 8.Discuz! 6.0 9.Joomla 1.6	PHP	
GS-156	A Parse Tree Model for Analyzing and Detecting SQL Injection Vulnerabilities	1.Bookstore 2.classifieds 3.Portals 4.Employee Directory 5.Events 6.Checkers 7.Office Talk	JSP	Precision Response-time

FromTable 2.2 presenting the datasets and performance metrics from selected literature, a total of 171 subject applications/ URLs were collected. However, 5 subjects were built using ASP.net, 7 were built using JSP, 120 were built using PHP (out of which 10 are URLs of web applications) and 39 are built using Java. On the other hand,a total of 12 metrics from our selected literaturewere collected.

2.5 Summary

This chapter detailed the background to this work on techniques for detection SQLIVs in web applications. An overview of web application vulnerabilities and SQL injection vulnerabilities are briefly discussed. Techniques for detection and/or prevention of SQLIVs and datasets used in evaluation of the techniques were also reviewed. From the analysis of literature, all these techniques were evaluated with different datasets and performance metrics, and as such there is no common ground for performance comparison across the techniques. Carrying out an assessment of these different datasets and performance metrics and conduct an experiment to evaluate the most frequently used, with the aim of forming set of most appropriate datasets to be used for evaluation of newly proposed techniques for the detection of SQLIVs in web

application are the main contributions presented in this study. Hence a web based repository is developed for these datasets.

Chapter 3: Methodology

3.0 Preamble:

This chapter presents the research methodology adopted in carrying out this research work. In the process, we discuss the phases of the research work which includes; Information synthesis, comparative analysis of tools, evaluation of datasets, and lastly, the design of a dataset repository.

3.1 Overview of Research Methodology

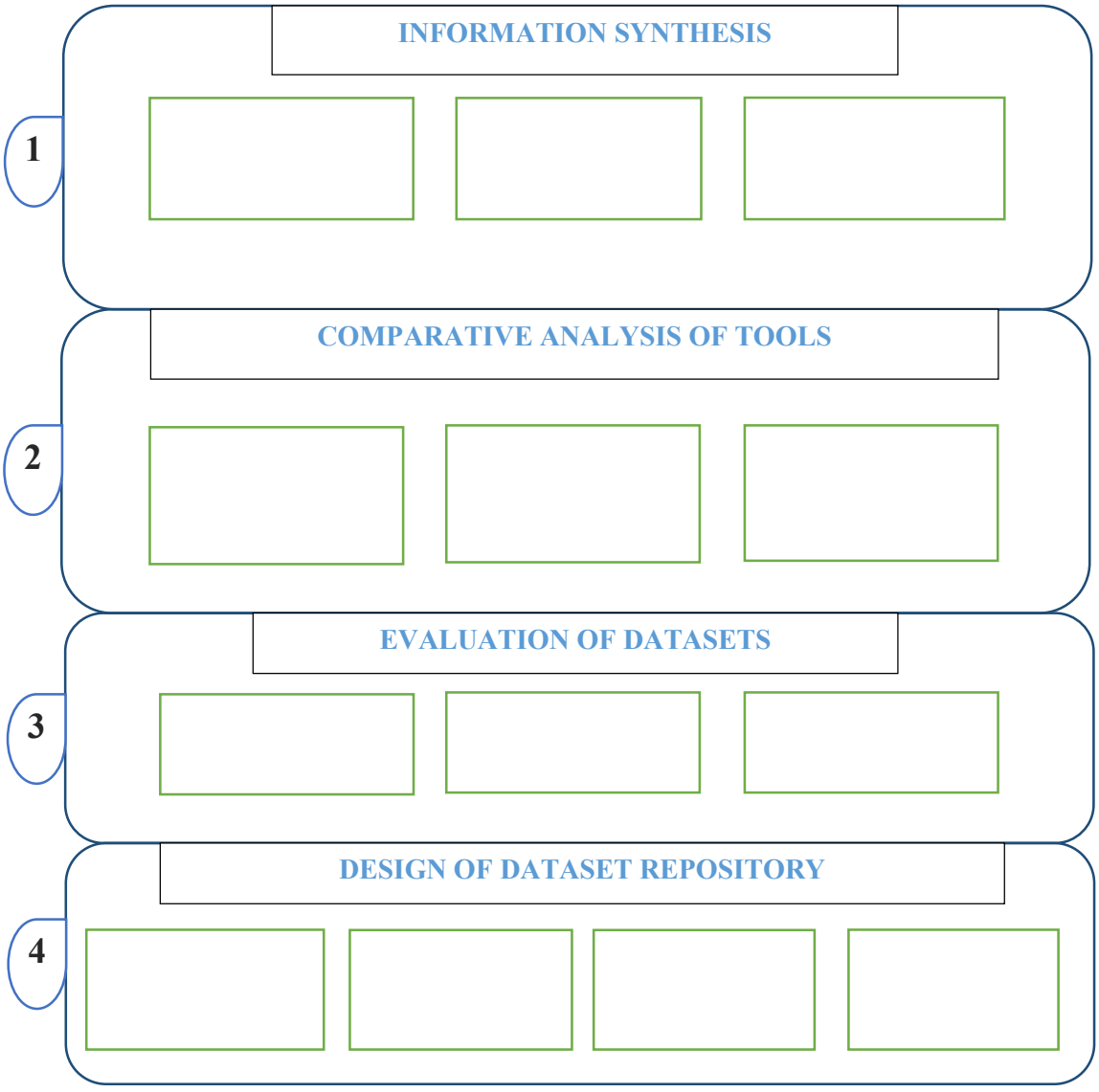

Figure 3.1: An overview of the research methodology

This research work consists of four phases as presented in figure 3.1, and hence, are discussed as follows:

3.2Information Synthesis

This is the first phase of the research work and was based on the analysis of Literatures that present solutions for the detection of SQL injection vulnerabilities from 2013-2017, inclusive.

We collected relevant literature that contains techniques for detection of SQLIVs from reputation journals and conference papers. Each of this literaturewas subjected to a clearly defined inclusion/exclusion criteria based on a specific research question (RQ).

The analysis of literature addressed the specific research question:-

> "What are the most appropriate datasets and performance metrics for experimental evaluation of techniques for detection of SQL injection vulnerabilities in web applications?"

The above question was refined into two key RQs so as to ease the process of selection of primary studies as well as the data collection process.

(RQ1)What are the most appropriate datasets used for evaluation of techniques for detection of SQLIV in web applications from 2013-2017?

(RQ2)What are the most appropriate performance metrics for empirical evaluation of techniques for detection of SQLIVs in web applications from 2013-2017?

In trying to answer the refined research question, we were able to come up with the most frequently used datasets and performance metrics as presented in chapter four.

3.3 Comparative Analysis of Tools

This is the second phase of the research work in which we conducted an experiment in order to perform a comparative analysis of three selected open source web vulnerabilities scannerswith some selected pages from each of the selected datasets (subject applications). The result of the experiment on the comparative analysis reveals the tool with better recall rate, accuracy and precision, and hence we can use this tool to conduct the second experiment on the evaluation of subjects (the third phase of the research work).

The tools used for conducting the experiments are discussed as follows;

PAVS (PHP Application Vulnerability Scanner): PAVS scans the PHP based web application source code and identifies the potential security problems in that application. PAVS also identify the loopholes in PHP configuration file settings. Attacks addressed by PAVS include Cross-site Scripting, SQL Injection, File Manipulation, File Inclusion, Command Execution,and Code Evaluation. Figure 3.2 shows a screen capture of the PAVS tool.

Figure 3.2: PAVS User Interface

RIPS-0.55 (Re-Inforce PHP Security):RIPS is a static code analysis software for the automated detection of security vulnerabilities in PHP applications. It is the most popular static code analysis tool to automatically detect vulnerabilities in PHP applications. By tokenizing and parsing all source code files, RIPS is able to transform PHP source code into a program model and to detect sensitive sinks (potentially vulnerable functions) that can be tainted by user input (influenced by a malicious user) during the program flow. Figure3.3shows a screen capture of the rips-0.55 scanner.

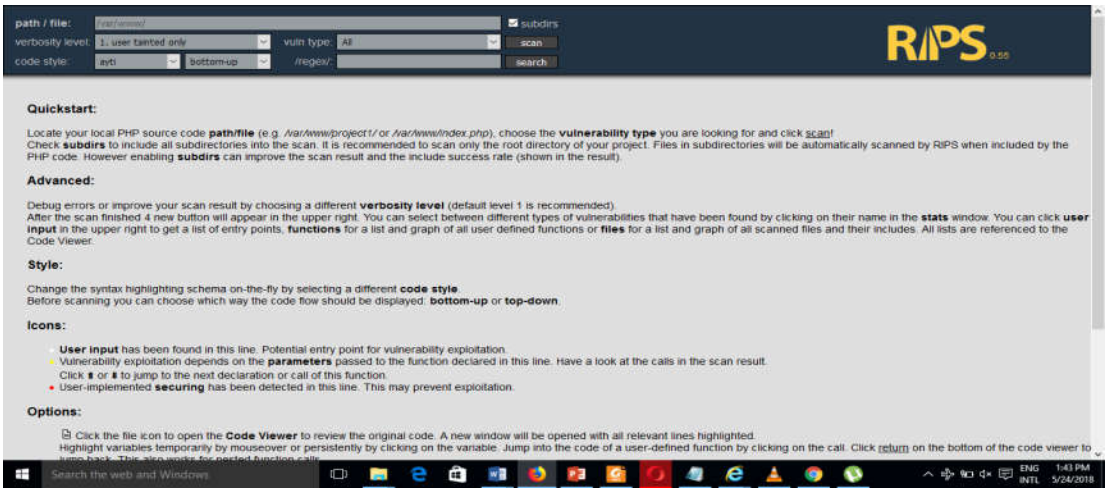

Figure 3.3: RIPS User Interface

OWASP ZAP 2.7.0 (Zed Attack Proxy):ZAP is a free open-source penetration testing tool being maintained under the umbrella of the Open Web Application Security Project (OWASP). ZAP is designed specifically for testing web applications and is both flexible and extensible.At its core, ZAP is what is known as a "man-in-the-middle proxy. It can be used as a stand-alone application, and as a daemon process. Figure 3.4 shows a screen capture of the ZAP user interface.

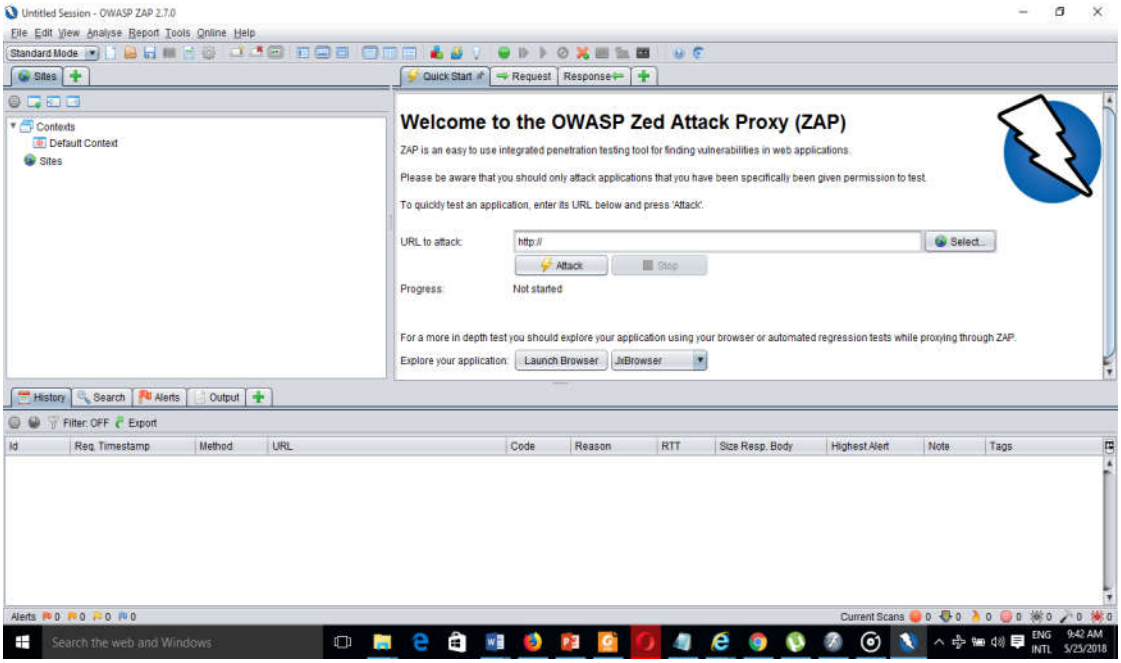

Figure 3.4: OWASP Zed Attack Proxy User Interface

3.3.1 Dataset for Experiment:

The datasets used for the first experiment on comparative analysis of tools are three pages selected at random fromfive of the most frequently used datasets obtained from phase one of this research work, which include SchoolMate 1.5.4, FaqForge 1.3.2,Classifieds 2.0, Bookstore-v02c and Forum_152. Table 3.1, presents a summary of the datasets used for the experiment.

Table 3.1: Datasets used for experiment 1

Subject Applications	**Selected Pages**
SchoolMate_v1.5.4	viewGrades.php visualizeRegistration.php visualizeClasses.php
FaqForge 1.3.2	index.php adminLogin.php adminLogOut.php
Classifieds 2.0	wproducts.php ajax_getPreview.php reviews.php
Bookstore-v02c	bookstore.php bookstore_admin.php bookstore_install.php
Forum_152	index.php admin_viewlog.php lpost.php

3.3.2 Experimental Design:

The experimental setup consisted of WAMPSERVER 2.4 with Apache 2.4.4, PHP 5.4.16 and MySQL 5.6.12, on an Intel Pentium PC. Three pages were selected at random from each subject application and submitted to a group of experts to detect SQLIVs. The result of the work of these experts forms a basis for verifying the accuracy of the tools used for the experiment.

The same pages that were scanned by experts for each subject application (dataset) weresubjected to the selected tools mentioned earlier in order to scan and detect the vulnerabilities in these pages. However, the result presented by each tool will be compared to thatofexperts so as to verify false positives and false negatives.

3.3.3 Performance Metrics for Comparing Tools:

Based on the result of the experiment carried out on the selected dataset, we choose recall, accuracy,and precision for performance comparison of the selected tools as they are the most commonly used performance metrics from analysis of literature. The tool which achieved the highest recall, accuracy,and precision will be chosen for conductingthe second experiment on the full subject applications in order to evaluate them for vulnerabilities. Hence, in the process, we discuss these performance metrics and present the formulae for computing them. We made use of the following abbreviations;

i. True Positives (TP): The number of actual attacks that are correctly classified as attacks;
ii. False Positives (FP): The number of legitimate statements that are incorrectly classified as attacks;
iii. True Negatives (TN): The number of legitimate statements that are correctly classified as safe;
iv. False Negatives (FN): The number of actual attacks that are incorrectly classified as safe;

Recall (R): is defined as the ratio of the correctly detected attacks (*TP*) over the total actual attacks (which is the sum of the reported (*TP*), and the *FN* that were not detected). It measures how good a technique is in finding actual vulnerabilities. Mathematically,

$$R = {}^{\mathrm{TP}}/_{(TP + FN)} \qquad (3.1)$$

Precision (P): is defined as the ratio of the correctly detected attacks (*TP*), over the number of reported errors (which includes the reported *TP* and *FP*). It measures the actual vulnerabilities that are correctly predicted. Mathematically,

$$P = {}^{TP}/_{(TP + FP)} \qquad (3.2)$$

Accuracy (A): is defined as the correctness of a technique to detect vulnerabilities successfully. It measures the total number of instances well classified. Mathematically,

$$A = {(TP + TN)}/{(TP + TN + FP + FN)} \quad (3.3)$$

The results obtained from the experiment based on the dataset and tools used for the experiment are presented in chapter four and we discuss the result based on these evaluation metrics.

3.4 Evaluation of Datasets

This is the third phase of the research work, in which the second experiment was conducted on the evaluation of datasets using the tool that happens to be the most accurate and precise from the result of the first experiment conducted in the second phase of this research work.

3.4.1 Subjects Chosen:

The dataset used for the experiment were extracted from table 2.2 of chapter two. We choose open source PHP/MySQL applications, as they are the most commonly used in most of our literature. The datasets used for our experiments are;

i. **SchoolMate 1.5.4:**a PHP/MySQL solution for elementary, middle and high schools. It is a free software application meant for course management. The application runs on all major operating systems.

ii. **FaqForge 1.3.2:** is a web-based document creation & management tool. It is a simple application to manageFAQs and can be used to create manuals, guides, HOWTOs, & other web-based documents requiring a hierarchical structure. Can be used on any OS running a webserver capable of running PHP/MySQL.

iii. **Classifieds:** is a web application for publishing different types of classified advertisements on a website, possibly multilingual. It is built on top of a PHP MVC Framework and uses a MySQL database.

iv. **Bookstore:** is a web application for the management of book sales for an onlinebookstore, where the user will be able to see and to make the purchase of the book.

v. **Forum:** is a PHP/MySQL online discussion Forum with a frames-based layout, Reply-to-user, e-mail notification, HTML-posting, signatures, friends & kill files, user promotion/demotion,and full admin tools.

Table 3.2 presents a summary of the chosen datasets with their size in terms of lines of code and the number of files each subject application contains.

Table 3.2: Datasets used for the Experiment 2

Subject Applications	LOC	Number of Pages
SchoolMate_v1.5.4	8,145	63
FaqForge 1.3.2	1,710	17
Classifieds 2.0	10,949	280
Bookstore-v02c	16,957	7
Forum_152	12,324	589

3.4.2 Experimental Design:

The experimental setup here also consisted of WAMPSERVER 2.4 with Apache 2.4.4, PHP 5.4.16 and MySQL 5.6.12, on an Intel Pentium PC. In this case, the complete set of the dataset was used for the experiment without removing any page, so as to verify their vulnerabilities.

3.5 Design ofDataset Repository

This is the fourth phase of the research work, which involves designingweb repository for the most frequently used datasetspresentedintable 3.2 of theprevious phase of this research work.We start by designing an architecture for the web repository, then draw a class and activity diagrams for the system.

3.5.1 Architecture of Dataset Repository:

Figure 3.5, presents the architectural design of the dataset repository which consists of two users: User and Admin. Let's look at it from a data flow perspective;

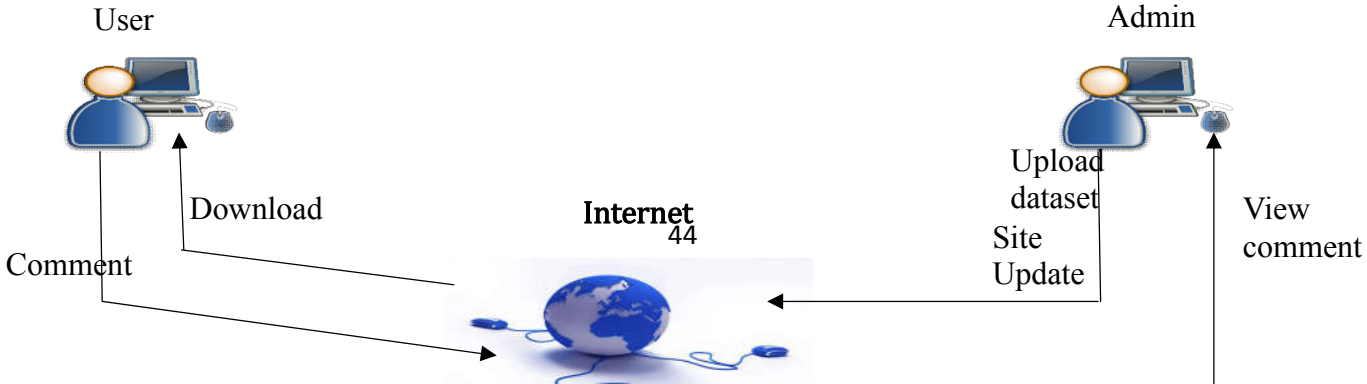

Figure 3.5: Architectural Design of Dataset Repository

The client user can download a dataset which flows upwards from the database repository to the server through the internet to the client's browser.The client can also write a comment base on his/her experience with the services benefited from which flows downward from the client's browser through the internet to the server and then to the database.On the other hand, the admin user can read the clients comment which flows upwards from the repository to the server through the internet to the admin's browser. The admin user can upload a dataset to the repository which flows downward from the admin's browser through the internet to the server to the repository. The admin user can also do site update.

3.5.2 Class Diagram of Dataset Repository:

Figure 3.6, presents a class diagram for the dataset repository, and hence, shows the attribute of each user and the type of function they perform.

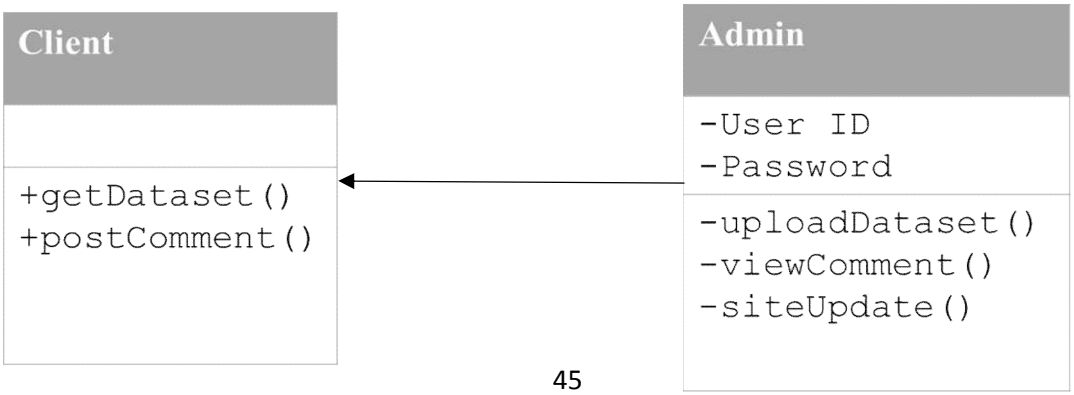

Figure 3.6: Class diagram of dataset repository

3.5.3 Activity Diagram of Dataset Repository:

Figure 3.7, presents an activity diagram for the dataset repository, thus showing the flow of activity that occurs within the components of the repository.

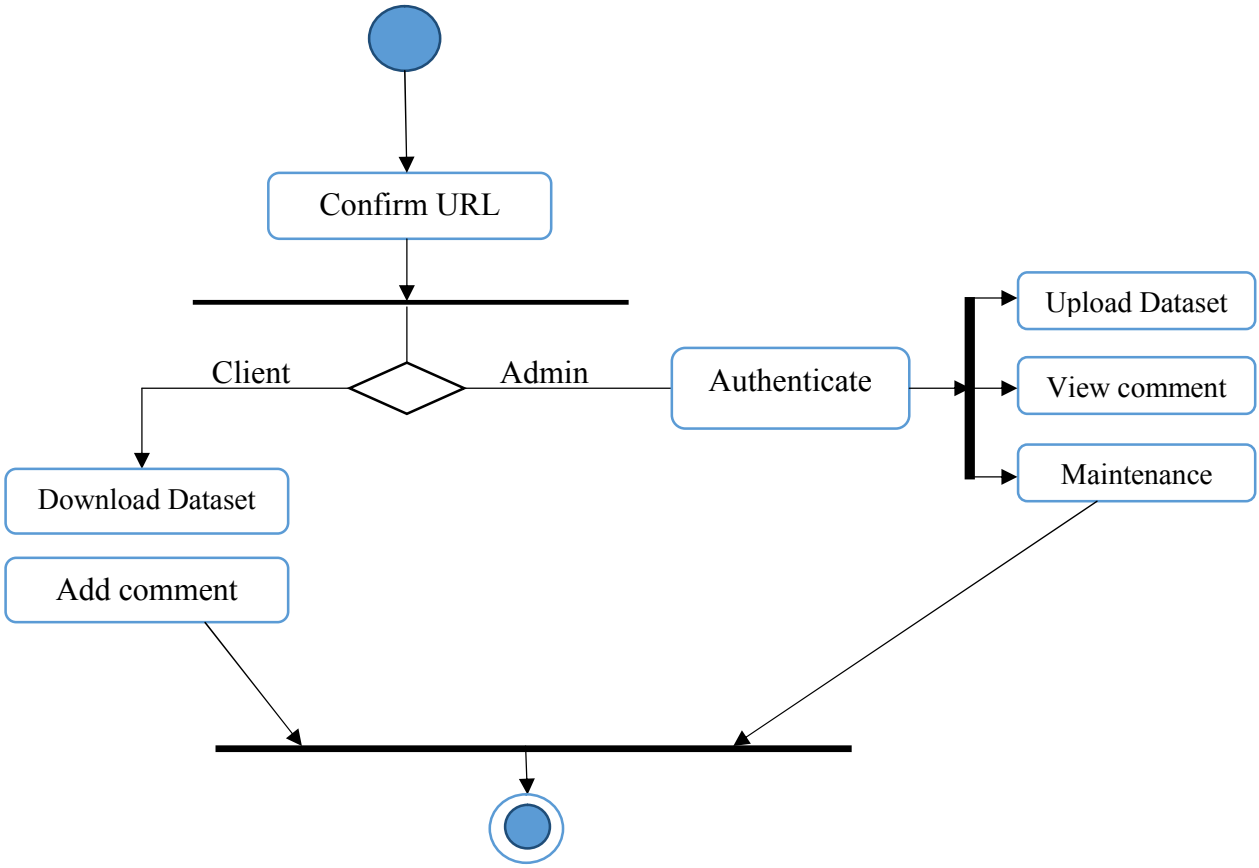

Figure 3.7: Activity Diagram of Dataset Repository

3.5.4 Operation of Dataset Repository:

A web-based repository was developed to host the most frequently used dataset for empirical evaluations of solutions for the detection of SQLIVs. Clients can easily download the subjects (experimental datasets) available in the repository and also leave a comment. There is a separate path for the admin to perform functions such as uploading the dataset, carrying out site update and reading client's comments. In order to achieve the implementation of the dataset repository described by the architecture in figure 3.2, we use Micromedia Dreamweaver 8 (a PHP 5 IDE) to build the front-end and MySQL database was used to build the back-end of the dataset repository. We chose PHP because, it is the most popular server-side programming language of all recognized websites with a share of 81.4%(W3Techs, 2013). The implementation of the repository is presented in chapter four.

Chapter 4: Experimental Results and Analysis

4.0 Preamble:

This chapter presents the results of the analysis of literature as well as that of the experiments conducted on comparative analysis of tools and evaluation of datasets.

4.1 Result of Literature Analysis

The result of our literature analysis is presented in this section, starting from the search process to data analysis.

4.1.1 Search Process:

The search was conducted in March 2018, using Google Scholar as the source for the search, and no restriction was made on any particular conference or journal so as to avoid the possibility of missing any relevant primary study.

The search process resulted in a total of 840 articles.After a careful review of the title, abstract and keywords of the articles, a total number of 152 articles were found to be relevant to our main research question. Finally, we subjected these relevant articles to inclusion and exclusion criteria guided by our quality assessment questions, and 33 articles were selected to be included in this study. Table 4.1, presents a summary of the resultofthe search process.

Table 4.1: Results of Article Search Process

Article's year of Publication	Total No. of Retrieved Articles	Total No. of Relevant Articles	Total No. of Selected Articles
2017	180	49	9
2016	180	19	4
2015	120	13	3
2014	180	39	9
2013	180	33	8
Total	840	152	33

Data collection was conducted on the selected articles, however, the data collected from selected literature was tabulated in different forms so as to ease the process of data analysis.

4.1.2 Datasets (Subject Applications):

This section presents the analysis of datasets used for the empirical evaluation of solutions in our selected literature (Addressing RQ1).

After careful analysis of selected articles, it was observed that a total of 171 subject applications/ URLs were collected. However, 5 subjects were built using ASP.net, 7 were built using JSP, 120 were built using PHP (out of which 10 are URLs of web applications) and 39 are built using Java. Table 4.2, presents the most frequently used among these subject applications based on language.

Table 4.2: Most Frequently used Datasets from Selected Literatures

S/N	Language of subject Application	Name of subject applications	Number of LOC	Frequency
1	JSP	Employee directory	5,658	3
2		Bookstore	16,957	3
3		Events	7,242	3
4		Classifieds	10,949	3
5		Portals	16,453	3
6	PHP	HotelRS	1,556	4
7		SugarCRM	352,026	4
8		Classifieds	10,949	3

9		Bookstore	16,959	3
10		Forum	12,324	3
11		NewsPortal	N/A	3
12		JobPortal	N/A	3
13		SchoolMate 1.5.4	8,145	5
14		FaqForge 1.3.2	90,148	4
15	Java	WebGoat	24,608	2

From table 4.2, a total number of 15 subject applications that are the most frequently used in our selected literatures(achieving objective (i), [i.e., perform an assessment of dataset used for evaluation of techniques for detection of SQLIVs]) were collected (out of which 9 are PHP applications, 5 are JSP applications and 1 a Java application). It is observed that the subject applications with higher frequency are built using PHP with Schoolmate 1.5.4 having the highest frequency of 5 followed by FaqForge, HotelRS,and SugarCRM with a frequency of 4 each, while the subject application with the least frequency is WebGoat a Java application having a frequency of 2 only.

4.1.3 Performance Metrics:

This section presents an analysisof performance metrics used for the empirical evaluation of selected articles (Addressing RQ2). Table 4.3, presents the performance metrics collected from analysis of data collection with their frequency showing the most frequently used among them.

Table 4.3: Performance Metrics used in Selected Literatures.

SN	Metric	Description	Frequency
1	Response time	The length of time taken for a system to react to a given input, request or service (It is the sum of the service time and wait time).	7
2	Efficiency	A measurable concept, quantitatively determined by the ratio of useful output to total input (effectiveness of a technique).	8
3	Accuracy	The closeness to the true value seen as a degree of agreement of results of one same conceived entity, measured by different methods.	12
4	Recall	The fraction of relevant instances that have been identified over the total amount of relevant instances (sensitivity).	13
5	Hypervolume (HV)	Takes a value within the interval [0,1] and it measures the volume in the objective space that is dominated by a set of non-dominated solution	1
6	Precision	a measure of the details in which quantity is expressed	13
7	Soundness	A property that provides the initial reason for counting a system (technique) as desirable.	1
8	Scalability	The capability of a system (process) to handle a growing amount of work, or its potential to accommodate that growth.	1
9	Scan time	The amount of time a system (technique) takes to finish processing a request.	1
10	Memory consumption	The amount utilized memory allocated to a given process to complete execution.	1
11	Throughput	The maximum amount of processed data in a given amount of time.	1
12	Conditional entropy metrics (H_C, H_V, H_T, H_L)	Is a measurement technique that measuresthe information content of SQL queries as part of determining thedeviation between the shadow query and an original query with malicious input.	1

As presented in table 4.3, a total of 12 metrics from our selected literaturewere collected. And it is observed that Recall and Precision have the highest frequency of 13, followed by Accuracy with a frequency of 12, then Efficiency and Response-time that have a frequency of 8 and 7 respectively. Figure 4.1, shows a bar chart presenting the distribution of the performance metrics in table 4.3

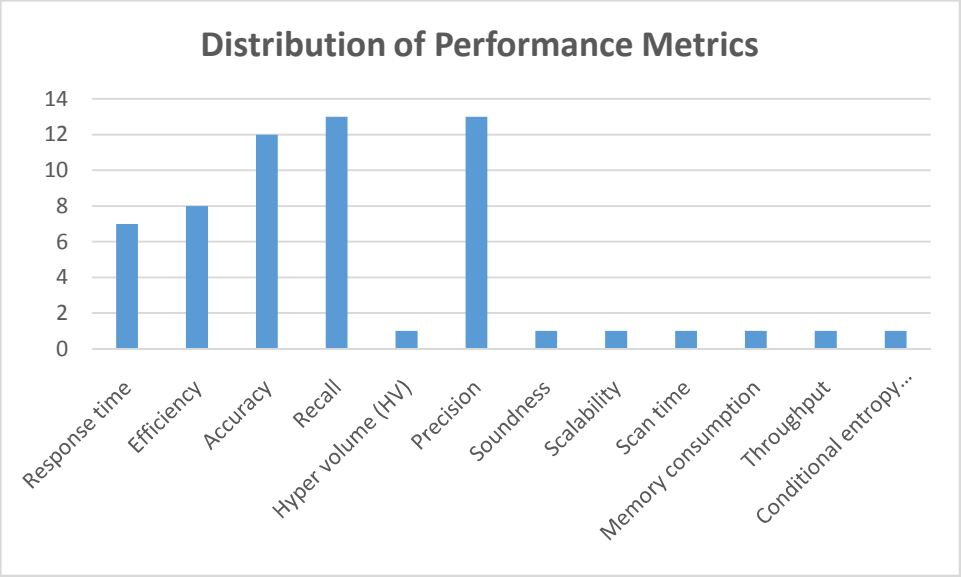

Figure 4.1: Distribution of Performance Metrics used in Selected Literature

As shown in figure 4.1, there are five metrics that are most frequently used for evaluating the performance of techniques for detection of SQLIVs in the selected literature.

4.2Comparative Analysis of Tools

This section presents the result of experiment 1, described in section 3.3.2 of chapter three.Three pages selected at random from each subject application and their source code was properly checked by a group of experts to find the SQLIVs for every selected page in each dataset. Then selected tools were used to scan the same datasets. The following figures present the tools in operation and the result obtained are presented in table 4.4.

Figure 4.2: Running Scan with PAVS tool

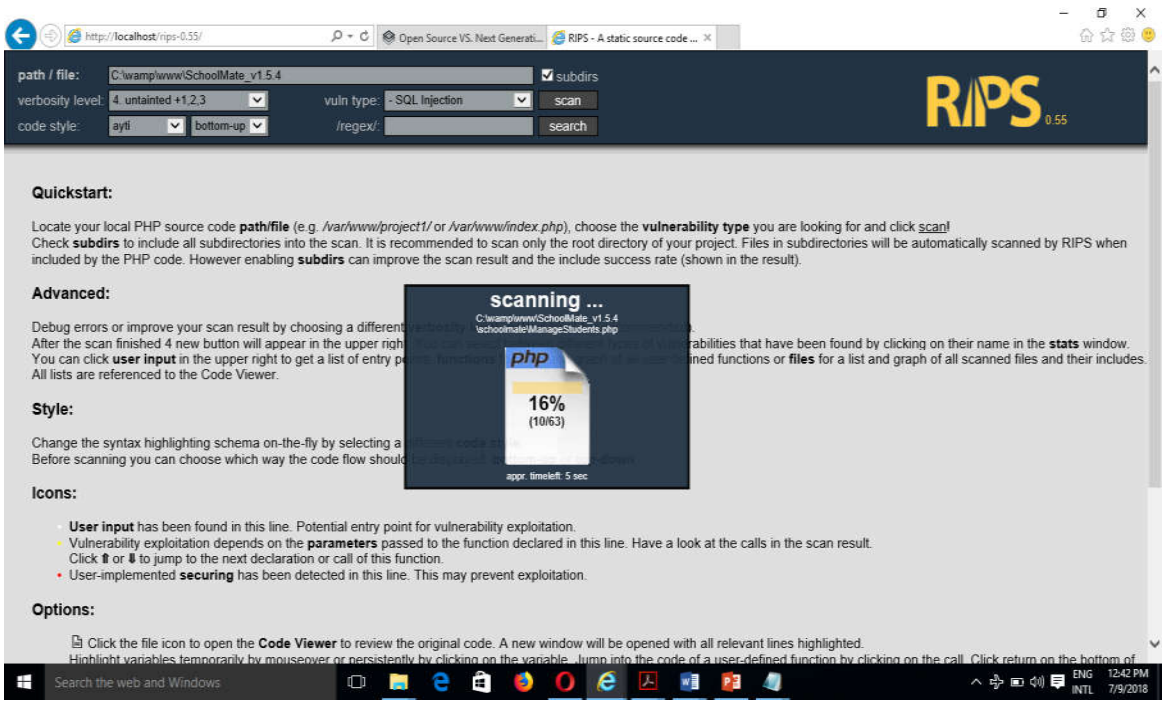

Figure 4.3: Running Scan with RIPS tool

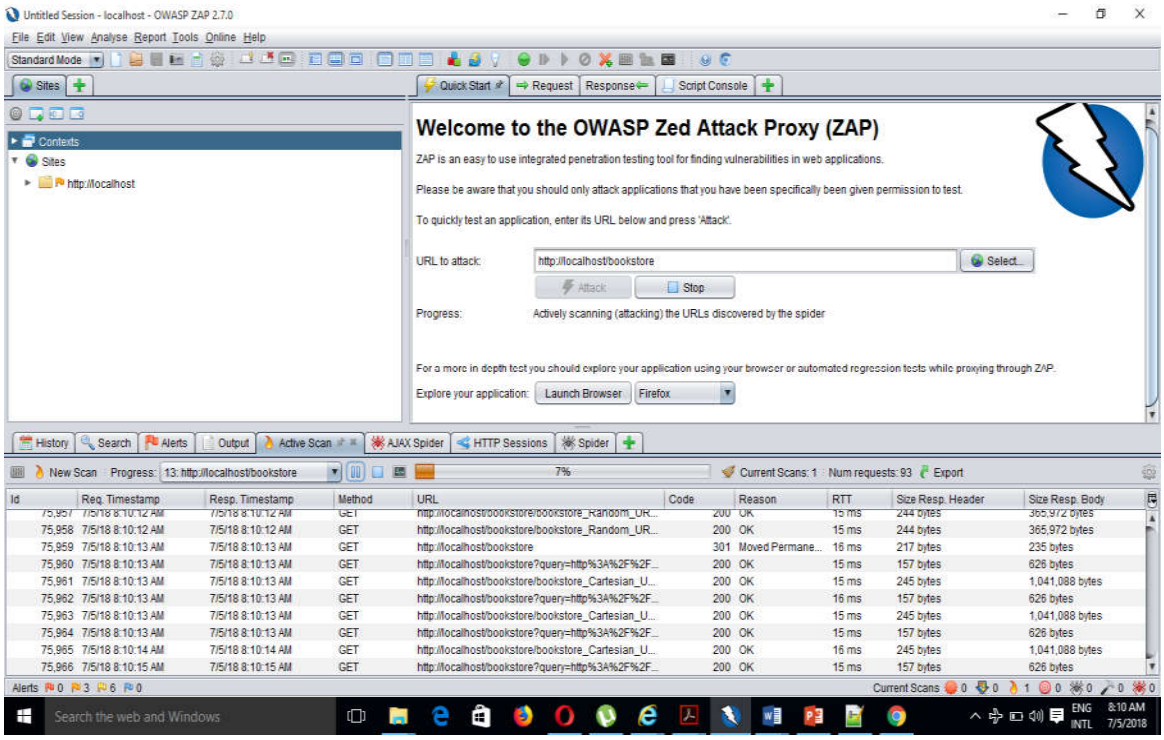

Figure 4.4: RunningScan with OWASP ZAP

4.2.1 Comparative Analysis Result:

The result of theexperimenton comparative analysis of tools is presented in table 4.4, including the result of experts and that of the tools. The first column contains the subject applications and the second column is the selected pages used for the experiment.

Table 4.4: Results Obtained from Experiment 1

Subject Applications	Pages	Expert Detection (TP)	**PAVS**			**RIPS**			**ZAP**		
			TP	FP	FN	TP	FP	FN	TP	FP	FN
SchoolMate 1.5.4	viewGrades.php	4	2	1	3	3	0	1	1	0	3
	visualizeRegistration.php	2	1	0	1	1	0	1	0	0	2
	visualizeClasses.php	1	1	0	0	1	0	0	0	0	1
FaqForge 1.3.2	index.php	0	0	0	0	0	0	0	0	0	0
	adminLogin.php	0	0	0	0	0	0	0	0	0	0
	adminLogOut.php	0	0	0	0	0	0	0	0	0	0
Bookstore-v02c	bookstore.php	2	1	0	1	1	0	1	0	0	2
	bookstore_admin.php	9	8	1	2	9	0	0	5	0	4
	bookstore_install.php	1	2	1	0	1	0	0	0	0	1
Forum_152	index.php	2	2	0	0	2	0	0	0	0	2
	admin_viewlog.php	3	3	0	0	3	0	0	2	1	2
	lpost.php	2	1	0	1	2	0	0	1	0	1
Classifieds 2.0	wproduct.php	6	6	1	1	7	2	1	2	0	4
	ajax_getPreview.php	3	2	0	1	2	0	1	1	0	2
	reviews.php	2	2	0	0	2	0	0	2	1	1
TOTAL		**37**	**31**	**4**	**10**	**34**	**2**	**5**	**14**	**2**	**25**

Discussion: From the results presented in table 4.4, experts were able to detect a total of 37 vulnerabilities from the sample datasets, and they are the true positive (TP) results. PAVS was able to detect 31 vulnerabilities with 4 false positives and 10 false negatives; while RIPS was able to detect 34 vulnerabilities with 2 false positives and 5 false negatives; and ZAP detected 14 vulnerabilities with 2 false positives and 25 false negatives. Figure 4.5 present a chart showing the result of table 4.4

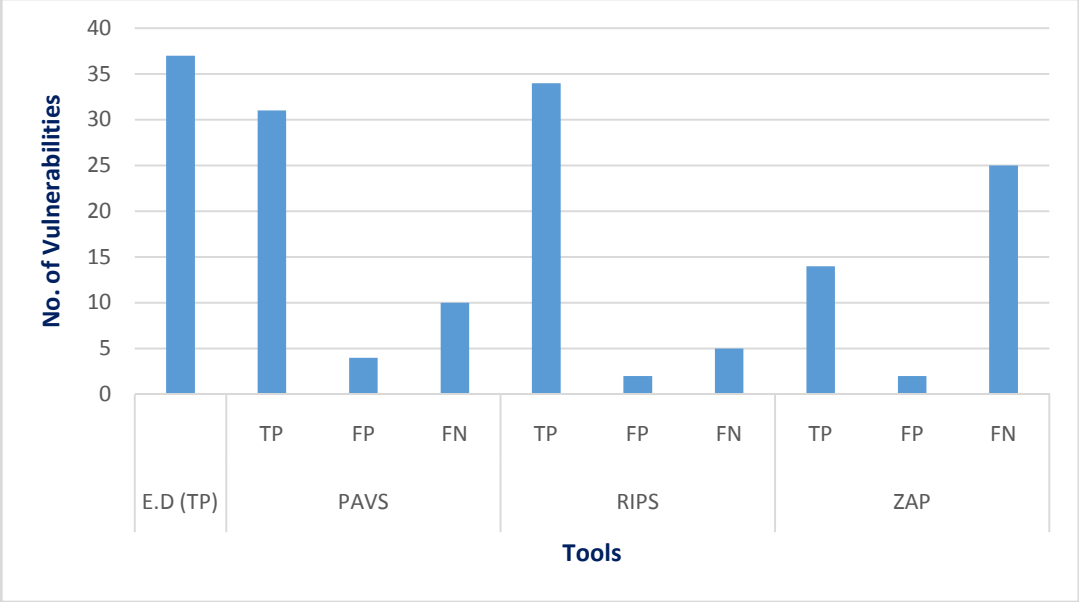

Figure 4.5: Chart showing result of comparative analysis

4.2.2 Performance Metrics Result:

To get the recall, precision and accuracy of each tool, we use the formula for calculating them presented in section 3.4.1.3

- For PAVS (TP = 34, TN = 0, FP = 5, FN = 13), therefore,

  R = 75.6 %

  P = 87.1%

  A = 65.4%

- For RIPS (TP = 38, TN = 0, FP = 2, FN = 6), therefore,

  R = 87.2 %

  P = 95.0%

  A = 82.6%

- For ZAP, (TP = 14, TN = 0, FP = 2, FN = 30), therefore,

R = 35.9 %

P = 87.5%

A = 30.4%

Table 4.5 summarizes the percentage recall, accuracy and precision of the selected tools.

Table 4.5: Performance Metrics Result

Performance Metric	Expert Detection	PAVS	RIPS	ZAP
Recall (%)	100	75.6	87.2	35.9
Accuracy (%)	100	65.4	82.6	30.4
Precision (%)	100	87.1	95.0	87.5

From table 4.5presentsa summary of the recall, accuracy,and precision of selected tools.We draw charts showing the distribution of recall in figure 4.6, precision in figure 4.7andthat of accuracy in figure 4.8

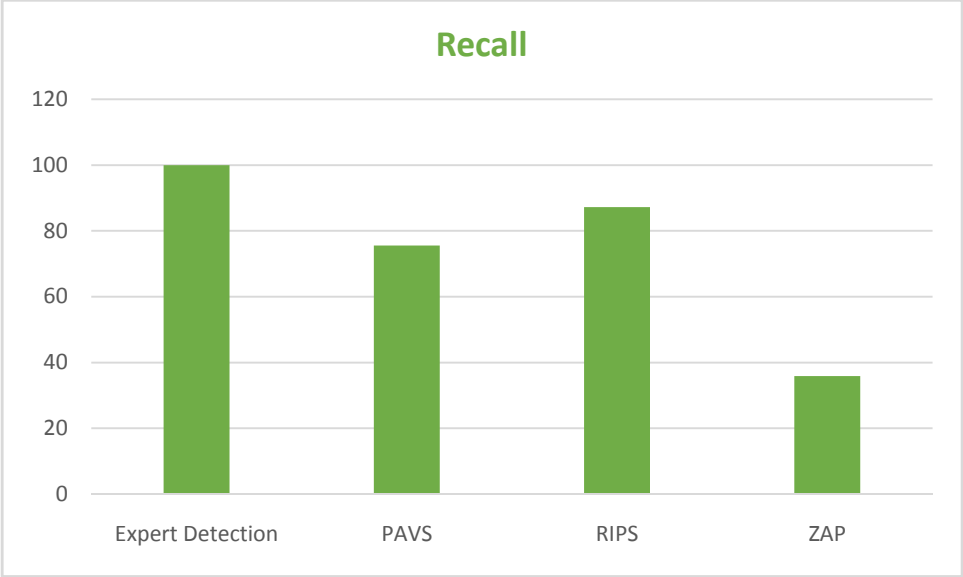

Figure 4.6: Distribution of Recall across Selected Tools

From figure 4.6, RIPS is the tool with the highest percentage of recall, which was due to the high true positive result and lowerfalse negatives. PAVS achieved a recall of 75.6 %, which was due

to a more false negative result than that of RIPS, while ZAP was poor in terms of recall which was due to very low true positive and very high false negatives.

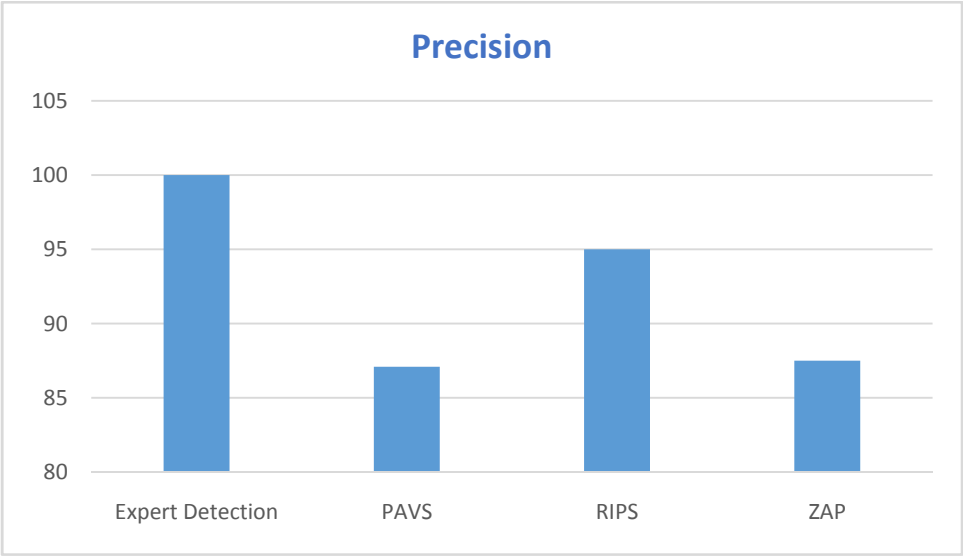

Figure 4.7: Distribution of Precision across Selected Tools

From figure 4.7, RIPS has the highest precision following expert detection which is as a result of the high number of true positives and less false positive that was only detected on the wproduct.php page of the classified dataset; while ZAP and PAVS has almost the same precision with a little difference of 0.4 %, andthis was due to lower true positive results recorded by the tools in almost all the datasets.

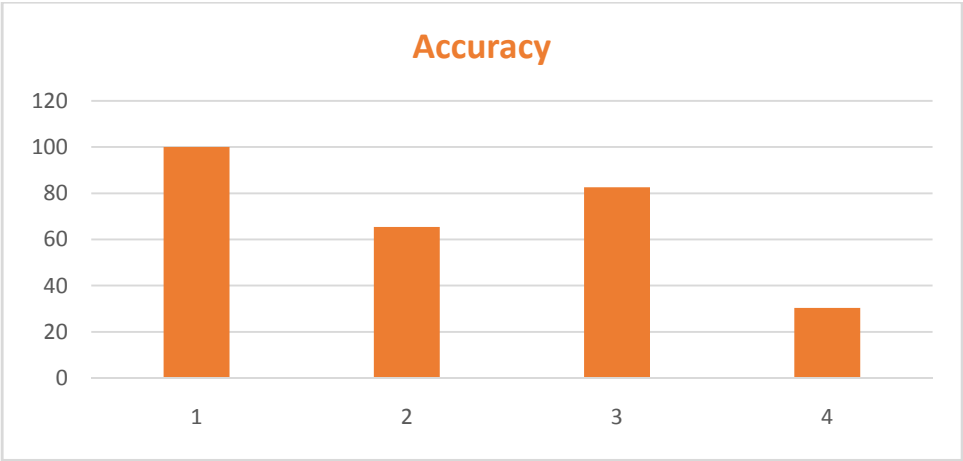

Figure 4.8: Distribution of Accuracy across Selected Tools

From figure 4.8, RIPS is the more accurate tool which was due to the high true positive result and lower false positives and false negatives. PAVS is the tool following RIPS which was due to a high number of false negative, while ZAP was the least accurate which was due to very low true positiveand high false negatives.

Hence, from the result of performance metrics, RIPS have a higher recall rate and it is more accurate and precise compared to PAVS and ZAP tools achieving objective (ii), (i.e. to perform a comparative analysis of three existing open source web vulnerabilities scanner), as such it was used toconduct the second experiment.

4.3 Evaluation of Datasets

The second experiment was conducted as described in section 3.4.2 of chapter three,in order to evaluate the most frequently used experimental dataset for empirical evaluation of solutions for SQLIV in studies from selected literatures using RIPS (achieving objective (iii) [i.e.To perform evaluation of most frequently used datasets using the most accurate and precise tool from result of objective]). We used the complete applications for this experiment. Table 4.6 presents the results of the second experiment, including the size of each subject in terms of lines of codes and the number of pages contained.

Table 4.6: Dataset Evaluation Result

Subject Applications	LOC	Number of Pages	**SQLIVs (Detected by RIPS)**
SchoolMate_v1.5.4	8,145	63	262
FaqForge 1.3.2	1,710	17	0
Classifieds 2.0	10,949	280	180
Bookstore-v02c	16,957	7	11
Forum_152	12,324	589	15
Total	**49,995**	**948**	**468**

Discussion: From the result of the experiment in Table 4.6 presenting the result of the second experiment, the first column contains the subject applications used for the experiment, the second column indicates the number of lines of code for the subject applications, the third column contains the number of files scanned for vulnerabilities, and the last columns present the SQLIVs found for each subject applications. Thus, it is observed that a total number of 948 files containing 49,995 lines of code was scanned and 468 SQLIVs were found using RIPS.

Figure 4.9 shows a distribution of vulnerabilities found across the subjects, while figure 4.10 is a line graph showing the vulnerabilities found per lines of code.

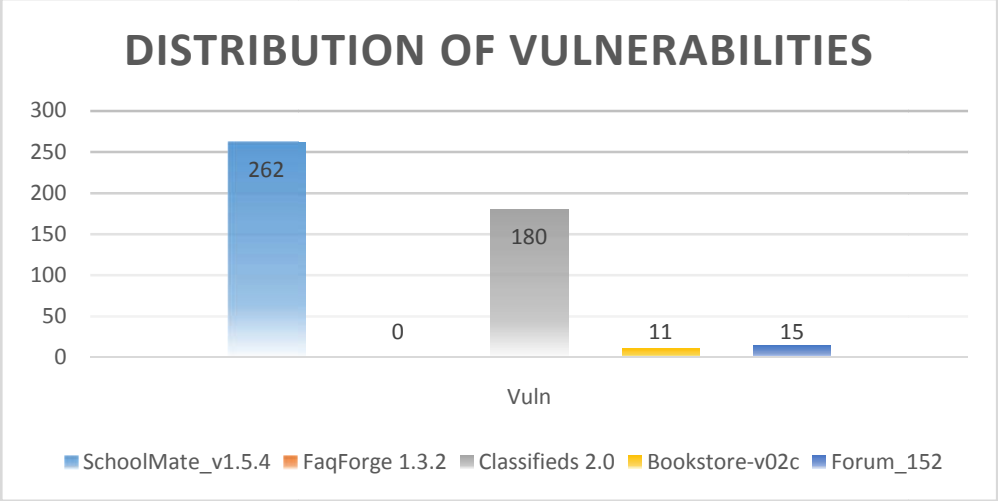

Figure 4.9: Distribution of SQILVs across Subjects

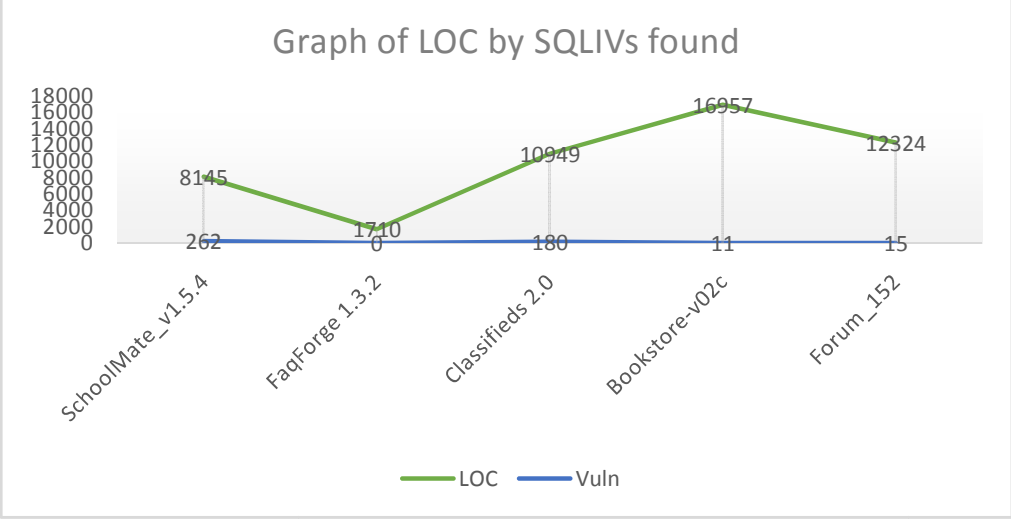

Figure 4.10: Graph of LOC by SQLIVs for each Subject

From figure 4.10, it is observed that schoolmate with 8,145 lines of code contains about 262 vulnerabilities, while forum with 12,324 LOC contains only 15 vulnerabilities and classified that is close to it in size has 180 vulnerabilities. So also bookstore that is having the highest number of LOC 16,957 has 11 vulnerabilities and faqforge with the least number of LOC 1,710 has no vulnerabilities.

Thus, we can conclude that the vulnerabilities found in a subject have no direct effect with respect to the size of that subject.

4.3 Operation of Dataset Repository

This section presents the result of the implementation of the dataset repository (achieving objective (iv) [i.e. to develop a web-based repository of the most frequently used datasets]).

4.3.1 Technology used:

In order to achieve the implementation of the dataset repository described by the architecture in figure 3.2, we use Micromedia Dreamweaver 8 (a PHP 5 IDE) to build the front-end and MySQL database was used to build the back-end of the dataset repository.

4.3.2 Test Running:

The dataset repository was tested on a localhost WAMPSERVER 2.4 with Apache 2.4.4, PHP 5.4.16 and MySQL 5.6.12. Figure 4.11 presents a subject being downloaded from the implemented dataset repository works.

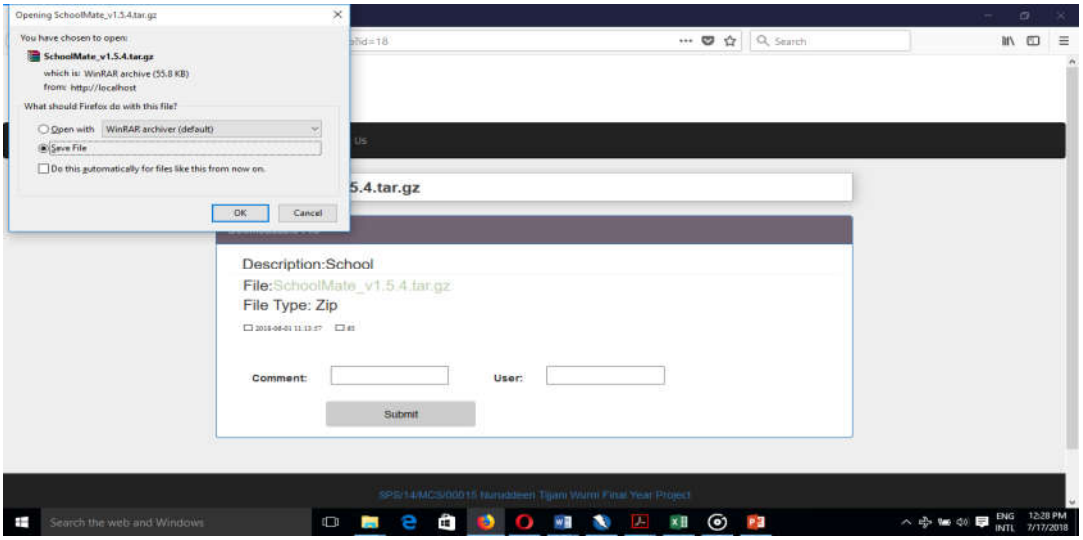

Figure 4.11: Downloadable File from Dataset Repository

Chapter 5: Summary, Conclusion,andRecommendations

5.0 Preamble:

This chapter presents a summary of this research work,a conclusionand suggests recommendations.

5.1 Summary

SQL injection vulnerabilities are among the most common threats to web applications and services. However, several significant researcheshave been done to study SQLIVs and a number of techniques for the detection of SQLIV have been proposed by different researchers.Literature investigation has shown that not a single datasets have been used to evaluate all these techniques. In this study,we conducted a thorough assessment of existing experimental dataset from selected literature with the goal of forming a set of most commonly used among them.This study was carried out in three phases as follows; In the first phase we carried out a search and selection process of literature containing experimental datasets used for empirical evaluation of solutions for SQLIV in web applications, theninformation analysis on the selected literature was carried out in order to get the most frequently used experimental datasets and performance metrics.In the second phase we conducted two experiments on five of the most frequently used datasets; the first experiment is on comparative analysis of tools, in which three open source tools (PAVS, RIPS,and ZAP) were assessed based on recall, accuracy,and precision, consequently,the tool that achieved the best recall rate, precision,and accuracy among them will be used to conduct the second experiment. The second experiment is on the evaluation of datasets using RIPS in order to report and document their vulnerabilities. And lastly, in the third phase, a web-based repository was developed to host the most frequently used dataset obtained from the previous phase.

5.2 Conclusion

Having a reliable, publicly available experimental dataset for evaluation of proposed solutions for the detection of SQLIVs is one of the fundamental concern of researchers in this domain.This research work has carried out a comparative analysis of three existing popular open source web vulnerabilities scanners, and perform an assessment experimental datasets and performance metrics used for empirical evaluation of techniques for detection of SQLIVs in web applications,

thereby forming a set of most frequently used experimental datasets. Hence, a web-based repository was developed for the most frequently used experimental datasets.

The study was able to make the following findings:

i. Among the three tools compared, RIPS outperformed PAVS and OWASP ZAP in terms of recall, accuracy,and precision, having achieved a recall rate of 87.2%, an accuracy of 82.6% and a precision of 95%
ii. The study found that the most frequently used datasets for empirical evaluation of solutions for SQLIVs include; SchoolMate_1.5.4 and FaqForge_1.3.2 which have both been used in five studies, followed by HotelRS and SugarCRM which have both been used in four studies, while Bookstore, Forum,and Classifieds have all been used in three studies
iii. The performance metrics that are most commonly used in evaluation of techniques for detection of SQLIVs include;recall and precision which have been used in thirteen studies, followed by accuracy which has been used in twelve studies, then efficiency which has been used in eight studies, and lastly response time which have been used in seven studies

5.3 Recommendations

The set of datasets and performance metrics presented in this research work is recommended to be used by researchers to evaluate their newly proposed techniques for detection of SQLIVs as this will provide ground for performance comparison across most techniques.

Infuture,the web-based repository should be enhanced to contain more experimental datasets that are more appropriate for conducting an empirical evaluation of techniques for the detection of SQLIVs.In particular,a vulnerability verification task based on the report of the CVE reference of vulnerabilities found in each of the datasetsshould be included if it exists.

References

Abdelhamid, M., Youcef , B., & Ahmed , S. (2014). Improving Web Application Firewalls to detect advanced SQL injection attacks. *10th International Conference on Information Assurance and Security* (pp. 35 - 40). OKINAWA, Japan: IEEE.

AbdulRahman, T. F., Alya , B. G., Kamarularifin , A., & Fakariah, A. M. (2017). SQL Injection Attack Scanner Using Boyer-Moore String Matching Algorithm. *Journal of Computers, 12*(2), 183-189.

AbdulRazzaq, Khalid , L., H. , A. F., Ali , H., Zahid , A., & Peter , B. C. (2014). Semantic security against web application attacks. *Journal of Information Sciences, 254*, 19-38.

Appelt, D., Cu , N. D., Lionel , B. C., & Nadia , A. (2014). Automated Testing for SQL Injection Vulnerabilities: An Input Mutation Approach. *Proceedings of the 2014 International Symposium on Software Testing and Analysis* (pp. 259 - 269). San Jose, CA, USA: ACM.

Appelt, D., Panichella, A., & Briand, L. (2017). Automatically Repairing Web Application Firewalls Based on Successful SQL Injection Attacks. *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)* (pp. 339-350). Luxembourg: IEEE.

Bandhakavi, S., Bisht, P., Madhusudan, P., & Venkatakrishnan, V. N. (2007). CANDID: Preventing SQL Injection Attacks using Dynamic Candidate Evaluations. *14th ACM Conference on Computer and Communications Security(CCS)* (pp. 12-23). New York, USA: ACM.

Bisht, P., Madhusudan, P., & Venkatakrishnan, N. V. (2010). CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. *ACM*.

Buja, G., Kamarularifin, B. A., Fakariah, M. A., & Teh, A. F. (2014). Detection Model for SQL Injection Attack: An Approach for Preventing a Web Application from the SQL Injection Attack. *Symposium on Computer Applications & Industrial Electronics* (pp. 60-64). Penang, Malaysia: IEEE.

Ceccato, M., Cu , N. D., Dennis , A., & Lionel , B. C. (2016). SOFIA: An Automated Security Oracle for Black-Box Testing of SQL-Injection Vulnerabilities. *31st IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 167-178). Singapore, Singapore: IEEE.

Coffey, J., White,, L., Wilde, N., & Simmons, S. (2010). Locating Software Features in a SOA Composite Application. *2010 IEEE 8th European Conference In Web Services (ECOWS)* (pp. 99–106). IEEE.

Dahse, J., & Thorsten , H. (2014A). Static Detection of Second-Order Vulnerabilities in Web Applications. *Proceedings of the 23rd USENIX Security Symposium.191*, pp. 989-1003. San Diego, CA, USA: USENIX Association.

Dahse, J., & Thorsten , H. (2014B). Simulation of Built-in PHP Features for Precise Static Code Analysis. *NDDS Internet Society*, 119 - 127.

Dharam, R., & Sajjan, S. G. (2013). Runtime Monitors to Detect and Prevent Union Query based SQL InjectionAttacks.

Halfond, W. G., & Orso, A. (2006). AMNESIA: Analysis and Monitoring for NEutralizing SQLInjection. *IEEE and ACM International Conference on Automated Software Engineering(ASE)* (pp. 174-183). ACM.

Halfond, W. G., Orso, A., & Manolios, P. (2007). Using Positive Tainting and Syntax-Aware Evaluation to Counter SQL Injection Attacks. *14th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)* (pp. 175-185). ACM.

Hidhaya, S. F., & Geetha, A. (2012). Deployment of Reverse Proxy for the Mitigation of SQL Injection Attacks using Input-Data Cleansing Algorithm. *International Journal on Cryptography and Information Security (IJCIS), 2*(4), 67-81. doi:10.5121/ijcis.2012.2408

Huang, S.-K., Han-Lin, L., Wai-Meng, L., & Huan , L. (2013). CRAXweb: Automatic Web Application Testing and Attack Generation. *2013 7th International Conference on Software Security and Reliability.* Taiwan.

Joshi, A., & Geetha , V. (2014). SQL Injection Detection using Machine Learning. *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)* (pp. 1111 - 1115). Surathkal: IEEE.

Kar, D., Khushboo , A., Ajit , S. K., & Suvasini , P. (2016). Detection of SQL Injection Attacks using Hidden Markov Model. *2nd IEEE International Conference on Engineering and Technology (ICETECH), 17th & 18th March 2016, Coimbatore, TN.* India.

Kar, D., Suvasini , P., & Srikanth , S. (2015). SQLiDDS SQL Injection Detection Using Query Transformation and Document Similarity. *Springer International Publishing Switzerland*, 377-390.

Kar, D., Suvasini , P., & Srikanth , S. (2016). SQLiGoT: Detecting SQL Injection Attacks using Graph of Tokens and SVM. *Computers & Security*, 4(9), (pp 254-260). Elsevier.

Kumar, D. G., & Madhumita , C. (2014). Detection Block Model for SQL Injection Attacks. *International Journal of Computer Network and Information Security*, 56-63. Springer.

Kumar, M., & Indu, L. (2014). Detection and Prevention of SQL Injection attack. *International Journal of Computer Science and Information Technologies, 5*(1), 374-377.

Lashkaripour, Z., & GhaemiBafghi, A. G. (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.

Liu, A., Yuan, Y., Wijesekera, D., & Stavrou, A. (2009). SQLProb: A Proxy-based Architecture towards Preventing SQL Injection Attacks. *In Proceedings of the 2009 ACM Symposium on Applied Computing* (pp. 2054-2061). Honolulu, Hawaii, USA: ACM.

Mahdi, M., & Mohammad, A. H. (2016). Using Hash Algorithm To Detect Sql Injection Vulnerability. *International Journal Of Research In Computer Applications And Robotics, 4*(1), 26-32.

Manmadhan, S., & T, M. (2012, November). A METHOD OF DETECTING SQL INJECTION ATTACK TO SECURE WEB APPLICATIONS. *International Journal of Distributed and Parallel Systems (IJDPS), 3*(6).

Medeiros, I., Nuno , N. F., & Miguel , C. (2014). Automatic Detection and Correction of Web Application Vulnerabilities using Data Mining to Predict False Positives. *International World Wide Web Conference Committee (IW3C2)* (pp. 1-11). Seoul, Korea.: ACM.

Medeiros, I., Nuno , N., & Miguel , C. (2016). Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining. *IEEE*.

MITRE. (2017, February). *MITRE CVE.* Retrieved March 29, 2018, from Common Vulnerabilities Exposure (CVE): htttp://cve.mitre.org/

Moh, M., Santhosh , P., Sindhusha , D., & Teng-Sheng , M. (2016). Detecting Web Attacks Using Multi-Stage Log Analysis. *2016 6th International Advanced Computing Conference.* USA.

Mui, R., & Frankl, P. (2010). Preventing SQL Injection through Automatic Query Sanitization with ASSIST. *Fourth International Workshop on Testing, Analysis and Verification of Web Software* (pp. 27-38). New York, USA: Springer.

Mui, R., & Frankl, P. (2011). Preventing Web Application Injections with Complementary Character Coding. *ESORICS 2011* (pp. 80-99). Verlag Berlin Heidelberg: Springer.

Nadeem, R. M., Saleem, M. R., Bashir, R., & Habib, S. (2017). Detection and Prevention of SQL Injection Attack by Dynamic Analyzer and Testing Model. *International Journal of Advanced Computer Science and Applications (IJACSA), 88*, 209-214.

Nagpal, B., Naresh , C., & Nanhay , S. (2017). SECSIX: security engine for CSRF, SQL injection and XSS Attacks. *Springer*.

Nashaat, M., Karim , A., & James , M. (2017). Detecting Security Vulnerabilities in Object-Oriented PHP Programs. *Journal of computer security, 10*, 442 - 451.

Ogheneovo, E. E., & Asagba, P. (2013). A Parse Tree Model for Analyzing And Detecting SQL Injection Vulnerabilities. *West African Journal of Industrial & Academic Research*, 33-49.

OWASP TOP 10 Security Threats. (2017, November 20). Retrieved March 29, 2018, from OWASP: https://www.owasp.org/index.php/category

Ping, C. (2017). A second-order SQL injection detection method. *Computer & Security, 20*, 592 - 597.

Rim, A., Eric, A., Mohamed, K., & Vincent, N. (2014). An automated black box approach for web vulnerability identification and attack scenario generation. *Journal of the Brazilian Computer Society, Springer Verlag (Germany)*, 1-16.

Sekar, R. (2009). An Efficient Black-box Technique for DefeatingWeb Application Attacks. *Proceedings of the 16th annual network and distributed system security symposium (NDSS).* San Diego, CA, USA.

Shahriar, H., Sarah , N., & Wei-Chuen , C. (2013). EARLY DETECTION OF SQL INJECTION ATTACKS. *International Journal of Network Security & Its Applications (IJNSA), Vol.5, No.4*.

Shar, L. K., & Hee Beng , T. K. (2013). Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns. *Elsvier, 55*, 1767-1780.

Shar, L. K., Hee Beng , K., & Lionel , B. C. (2013). Mining SQL Injection and Cross Site Scripting Vulnerabilities using Hybrid Program Analysis.

Sharma, D., Komal, K., Chandrakala, D., & Diksha, B. (2017). Using AMNESIA to secure web applications and database against SQL injection attacks. *International Research Journal of Engineering and Technology (IRJET), 04*(04), 908-910.

Sheykhkanloo, N. M. (2014). Employing Neural Networks for the Detection of SQL Injection Attack. *ACM*.

Shweta, k., Satyawan , M., Vishal , T., & Romil , R. (2012, December 2). Research dataset Evaluation for Technologies. *International Journal Of Computer Architecture And Mobility, 1*(2).

Siqi, M., David , L., Cong , S., & Robert , D. H. (2017). VuRLE: Automatic Vulnerability Detection and Repair by Learning from Examples. Xidian, China.

Son, S., Kathryn , M. S., & Vitaly , S. (2013). Diglossia: Detecting Code Injection Attacks with Precision and Efficiency. *ACM*.

Stivalet, B. C., & Aurelien, D. (2015, 10 28). *PHP Vulnerability Test Suite.* Retrieved from Software Assurance Reference Dataset (SARD): https://SARD/View_all_test_cases(PHP Vulnerability Test Suite)

Su, Z., & Wassermann, G. (2006). The Essence of Command Injection Attacks inWeb Applications. *Proceedings of the Symposium on Principles of Programming Languages(POPL)* (pp. 372-382). New York, USA: ACM.

Sun, S.-T., & Beznosov, K. (2010). Retrofitting Existing Web Applications with Effective Dynamic Protection Against SQL Injection Attacks. *ACM*.

Sunkari, V. (2016). Protect Web Applications against SQL Injection Attacks Using Binary Evaluation Approach. *International Journal of Innovations in Engineering and Technology, 6*(4), 484-490.

Sunkari, V., & Dr.C.V., G. R. (2014). Preventing Input Type Validation Vulnerabilities Using Network Based Intrusion Detection Systems. *IEEE*, 702-706.

Takeshi, M. (2014). SQL Injection Attack Detection Method using the Approximation Function of Zeta Distribution. *IEEE International conference on Systems, Man and Cybernetics.* San Diego,CA, USA.

The MITRE Corporation. (2012). Retrieved from CVE-Common Vulnerabilities and Exposures: http://www.cve.mitre.org/

Thomé, J., Alessandra , G., & Andreas , Z. (2014). Search-Based Security Testing of Web Applications. *SBST.* Hyderabad, India.

Thomé, J., Domenico, B., Lwin, S. K., & Lionel, B. C. (2017). JoanAudit: A Tool for Auditing Common Injection vulnerabilities. *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering.* Paderborn, Germany.

Thome, J., Lwin , S. K., & Lionel , B. (2015). Security Slicing for Auditing XML, XPath, and SQL Injection Vulnerabilities. *IEEE*, 553-564.

Thomé, J., Lwin, K. S., Domenico , B., & Lionel , B. (2018). An Integrated Approach for Effective Injection Vulnerability Analysis of Web Applications through Security Slicing and Hybrid Constraint Solving. *security and trust*.

Tripp, O., Marco , P., Patrick , C., Radhia , C., & Salvatore , G. (2013). ANDROMEDA: Accurate and Scalable Security Analysis of Web Applications.

Umar, K., Abu Bakar, S. M., Hazura , Z., Novia , A., & Mohd , A. T. (2016). Enhanced Pushdown Automaton based Static Analysis for Detection of SQL Injection Hotspots in Web Application. *Indian Journal of Science and Technology, 9*(28).

W3Techs. (2013, December). *W3Techs.* Retrieved May 24, 2018, from Usage of Server-side Programming Languages for Websites: http://w3techs.com/technologies/overview/programming_language/all

Wu, T.-Y., Chien-Ming, C., Xiuyang, S., Shuai, L., & Jerry, L. C.-W. (2016). A Countermeasure to SQL Injection Attack for Cloud Environment. *Springer Science+Business Media*.

Yan, L., Xiaohong, L., Ruitao , F., Zhiyong , F., & Jing , H. (2013). Detection Method of the Second-Order SQL Injection in Web Applications. *Springer International Publishing Switzerland*, 154–165.

Zhu, Y., Guidong, Z., Zhiquan , L., Boya , N., & Yongjun , S. (2017). A Two-Tiered Defence of Techniques to Prevent SQL Injection Attacks. *Innovative Mobile and Internet Services, Springer International Publishing*.