UNIVERSITY OF NAIROBI

COLLEGE OF BIOLOGICAL & PHYSICAL SCIENCES

SCHOOL OF COMPUTING & INFORMATICS

HYBRID MULTI-AGENTS SYSTEM VULNERABILITY SCANNER FOR DETECTING SQL INJECTION ATTACKS IN WEB APPLICATIONS

BY:

MUTAI HILLARY

(P53/80165/2015)

SUPERVISOR:

DR. ROBERT OBOKO

A Research Project Submitted in Partial Fulfillment of the Master of Science Degree in Distributed Computing Technology in the University of Nairobi School of Computing and Informatics.

October 2016

DECLARATION

This project is my original work and, to the best of my knowledge, this research work has not been submitted for any other award in any University.

Sign: ___________________________

Date: ___________________________

Mutai Hillary

(P53/80165/2015)

This project has been submitted in partial fulfillment of the requirements for the Degree of Masters of Science in Distributed Computing Technology at the University of Nairobi with my approval as the University supervisor.

Sign: ________________________ Date: ______________________

Dr. Robert Oboko.

School of Computing and Informatics

University of Nairobi

DEDICATION

This Thesis is dedicated to my parents, my wife Edith Lagat, my children Ron Lagat and Jayden lagat and everyone that has supported me in development of my thesis.

ACKNOWLEDGEMENT

I take this opportunity to thank Almighty God for this far he has helped me. Secondly, I would like to give gratitude to my lectures and fellow students for the support they accorded me. Moreover, my appreciation and gratitude goes to my supervisor, Dr. Robert Oboko for his relentless guidance and support during the period I was undertaking this research project. My sincere gratitude also goes to the entire panel team Prof Elisha Opiyo, Dr. Robert Oboko, Dr. Agnes Wausi, for their comments, advices and support that helped in improving my project.
I thank my colleages at work for the support you gave, Last but not least, much appreciation goes to my family for their unyielding support and encouragement during this project.

ABSTRACT

Web applications form part of our daily life due to their appropriateness, flexibility, availability, usability and interoperability. This has allowed most organizations map their businesses globally and facilitate information exchange. They embrace a multi-tier architectural design where the third tier is a database and the core component within an organization. The issue of concern in web applications is dealing with security. There has been a dramatic increase in web application vulnerabilities being reported as attackers improve their skill and competencies to defeat the existing techniques. The main objective of this study was to investigate agent-based vulnerability scanners systems for detecting SQL injection attacks in web applications and formulate system requirements. To achieve this objective, a desktop review was used to test the time taken to scan, the accuracy and number of vulnerabilities detected by three existing systems i.e. Vega, Wapiti and Zap. The test was performed across three web application i.e webgoat, vicnum and genhoud. Vega – Performed better in detecting SQL injections but the scanning time was high, it also showed a better representation of vulnerabilities detected because it categorized the vulnerabilities as either high, medium or low. Wapiti – was above average, it was able to take average time in scanning web applications, however, it could not discover all SQL vulnerabilities. Zap- did not perform well in the time taken to scan web vulnerability and its discovery. The gaps in the existing systems under study led to the development of a hybrid multi-agents system Ron Scanner to address the limitation of the existing systems. Ron Scanner – Performed better than all the others tools tested. It recorded a mean scan time of 16.5 % which is the lowest as compared to other vulnerabilities. The results demonstrate that the proposed hybrid multi-agent system is able to perform a scan on a web application faster than Vega, Wapiti, and Zap scanners. The mean scan time is 2.2 sec lower and the mean vulnerabilities detected is 0.4sec higher in our proposed hybrid multi-agent system. Additionally, the system is more accurate in detecting SQL vulnerabilities. From the findings, the author recommends the use of a hybrid multi-agents system to detect SQL web applications vulnerabilities, as it provides a better coverage with no false positive and false negative limited time to scan, improved detection trend and accuracy as compared with already existing vulnerabilities scanners.

Keywords: Web vulnerability scanners, Multi-agents, SQL injection attacks, and web-based applications.

TABLE OF CONTENTS

DEFINITION OF TERMS AND ABBREVIATION

API	Application program interface
ATM	Automated Teller Machine
CSS	Cross-Site Scripting
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transport **Protocol** Secure
MAS	Multi Agent Systems
MaSE	Multi-agent System Engineering
OWASP	Open Web Application Security Project
PHP	PHP Hypertext Preprocessor
SDN	Software-Defined Networking
SQL	Structured Query Language
URL	Uniform Resource Locator
VB	Visual Basic
W3AF	Web Application audit and attack framework
WAVSEP	Web Application Vulnerability Scanner Evaluation Project
ZAP	Zap Attack Proxy

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

1.0 INTRODUCTION

1.1 Background Information

Web applications are vital in our daily life. They offer appropriateness, flexibility, availability, usability and interoperability. Organizations and individuals are mapping their businesses into the world through web applications. Web applications allow more information exchange including financial operations, payments of daily bills, reaching out to clients among other activities. Most web applications, and the underlying databases, often contain confidential or sensitive information, including customer details, financial records, credit cards details and user credentials. This information is highly valuable to an organization; hence, this makes web application an ideal target for attacks. Web applications have become a desirable target for cyber–criminals. SQL injection attacks on web applications have experienced a significant rise in recent years. Owasp (2013) top ten vulnerability list ranked SQL injection as the most vulnerable attack.

Muchai, Kimani, Kigen, Mwangi, & Shiyayo (2015) found out that in the recent times, there has been a rise in the number of criminal activities on the web applications that target website information. Cyber hackers have advanced to a degree where it is almost impossible to detect intrusion unless a person uses advanced, continuous monitoring and detection methods. In 2012, Muchai et al. (2015) studied top 3 methods used by cyber criminals. The author mentioned key loggers, stealing of password and Automatic Teller Machine (ATM) skimming was widespread. Compared to 2012, the top 3 vulnerabilities in 2015 were ransomware, database transaction manipulation, and social engineering.

In most cases, web applications continue being a desirable target for web vulnerabilities as hackers continue to improve their ways. Hackers are becoming more sophisticated as a result of new tools to perform penetration tasks. It is important for those in charge of database security to design new techniques and programs. These programs should be customized to guarantee information system security. Kala (2014) studied safety and integrity of web applications contents. In his study, he concluded that numerous attacks on the web application servers exploit weaknesses of a system. Further, he pointed out that compromised web contents by intruders and attackers is an area of concern, not only for the government institutions but to all organizations, web application owners as well as individuals who access web applications.

Attackers have advanced their techniques. They capture system information security vulnerabilities to get valuable information from a database. They achieve this aim by generating a query and applying it to the desired database to access sensitive information. The process used by these hackers is referred to as SQL injection. Having easy access to the internet needs a better understanding of communications taking place between a user and web applications as demonstrated in the figure below.

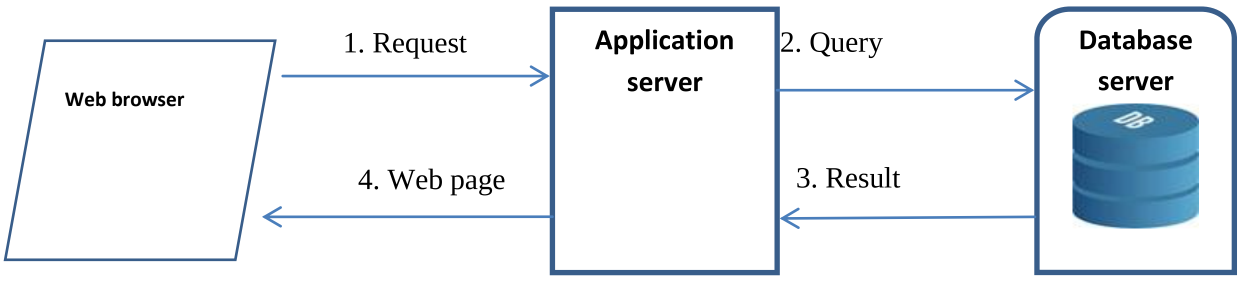


Figure 1: Web Application Architecture

Typically, a web application receives input from a user to retrieve information. It rely on the validation put in place. Validation is required to enable web application check on the validity of input from users, to build a query for accessing a database. According to Rawat et al. (2011), if data is not validated it will be susceptible to SQL injection.

One of the approaches used in SQL injection is the use of multiple agents systems. Multiple agents systems solve problems for individual agents or unified system. Moreover, a hybrid multi-agent system is used to detect SQL injection vulnerabilities; it has best detection trend and accuracy.

1.2 Problem Statement

Most web applications, and the underlying databases, often contain confidential or sensitive information, including customer details, financial records, credit cards details and user credentials. This information is highly valuable to an organization; hence, this makes web application an ideal target for attacks. This applications have therefore become a desirable target for cyber–criminals. In addition, SQL injection attacks on web applications have experienced a significant rise in recent years with SQL injection ranking as the most vulnerable attack (Owasp 2013).

The literature provides notable highlights on the security of web application regarding SQL injection vulnerabilities. Most studies have confirmed the shortcomings of SQL web scanners according to Phalguna Rao et al. (2013). However, some IT practitioners and researchers have proposed different methods as a solution to SQL injection problem. Most studies have shown that existing techniques have not been 100% accurate, they suffer from various weaknesses. Kumar Singh & Roy (2012) confirmed that these shortcomings include incomplete implementations, multiple frameworks, a longer span of time taken to scan and False positive and false negative. Therefore, A hybrid Multi-Agents system need to be developed to address the shortcomings of the existing systems.

1.3 Main Objectives

The main objective of this study is to investigate agent-based vulnerability scanners systems for detecting SQL injection attacks in web applications and formulate system requirements.

1.3.1 Specific Objectives

The specific objectives are

a) Develop hybrid multi-agent prototype system using an appropriate technology, which addresses the problem of SQL injection attacks and dynamically tests for the effectiveness of web application vulnerabilities in both development and production environments.

b) Analyse the developed system agaist set metrics.
c) To test and validate the effectiveness of the system on selected web applications.
d) To identify various open source vulnerabilitis scanning tools for web applications.

1.4 Justification of the Study

Users of web application incur losses when they are frauded by hackers. They spend a lot of resources in recovering damages in the process. At the same time, web application users are not in a position to proactively detect vulnerabilities created by the hackers. A hybrid Multi-Agents system is thus necessary to detect and report any vulnerabilities in web application in addition to safeguarding sensitive information and minimizing loses resulting from web application attacks.

1.5 Scope

The study was limited to the use of multi-agents hybrid approach towards solving SQL injection attacks in development and production web applications. It involves the development of a multi-agent hybrid approach system based that implements the detection of SQL injection attacks.

CHAPTER TWO

2.0 Literature Review

2.1 Introduction

The use of computers, tablets, and smartphones have significantly increased over the past few years (Stuttard & Pinto, 2011). Conversely, web applications have been developed to perform practically every useful function online. These include but not limited to; shopping online, interactive web pages, web search, auctions, banking, gambling and social networking. Web applications are however faced with some weaknesses which vary regarding complexity, detection, and recovery. For example in a report published by (Owasp 2013) , 86% of all websites tested
by whitehat, Sentinel had at least one significant vulnerability. The most severe security risks organization face presently is linked to open web applications. These vulnerabilities are weak authentication, SQL injection, Cross-Site Scripting (CSS), session management, sensitive data disclosure, security misconfiguration, and cross-site request forgery among other vulnerabilities (Figure 2). Besides, making use of components which have known vulnerabilities, as well as unvalidated forwards and redirects, constitute the top most web

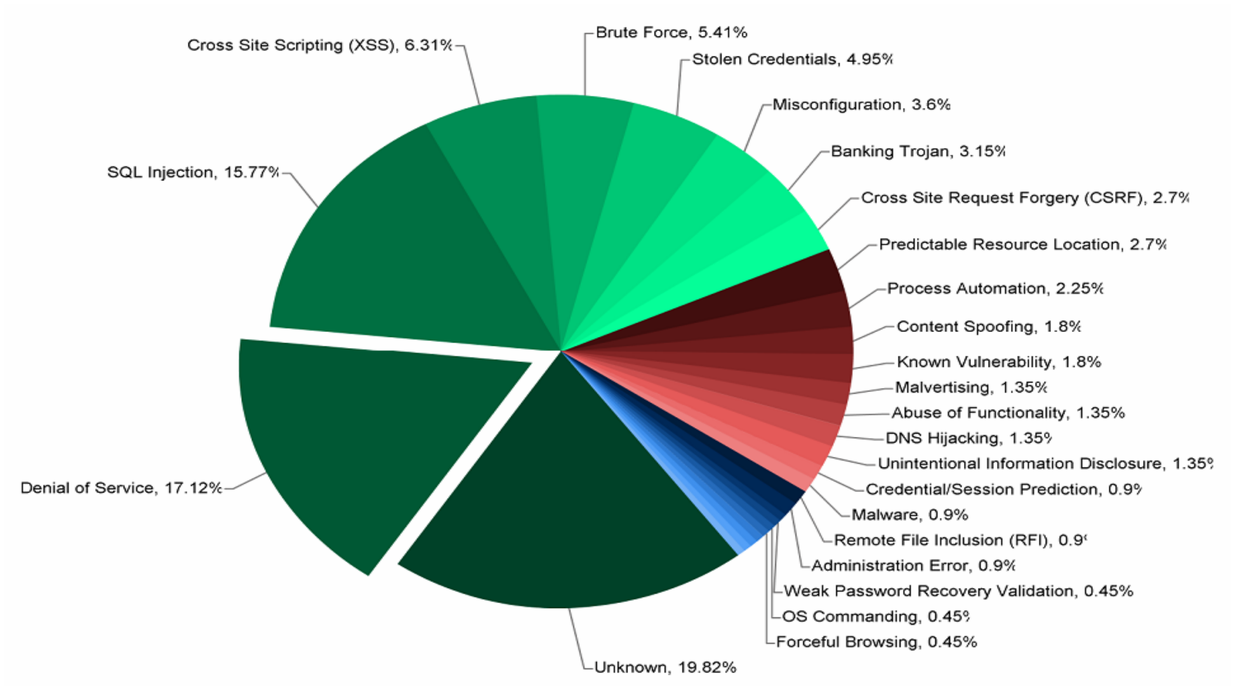


Figure 2: Web Application Vulnerabities (Owasp 2013)

According to (Shema n.d.), most organizations rely on open web application to implement or reengineer business processes. Such application include web application with dynamic content. Although web applications are developed with security concept in mind, many developers are less experienced regarding security. Hence, these open web applications are vulnerable to network vulnerabilities. Manually analyzing all the applications for loopholes and prioritizing their importance for remediation can be a daunting task without organized efforts and using automated tools to improve accuracy and efficiency.

Many organizations have mapped their business from the traditional in-house system into the world. These allow people to continuously respond to a request from both inside and outside the organizations' corporate network with the help of gadgets such as laptops, tablets, or smartphones.With an increasing use of these devices, hackers may take advantage of connectivity to gain unauthorized access to vital company information. For this reason, it is imperative for any organization to ensure that they protect their web application and reduce the possibility of a security breach to their electronic system. Subjecting a web application to automated penetration during the testing process elicit comparatively quick results. Currently, many tools for testing vulnerabilities exist and they are either open source or commercial.

Vulnerabilities linked to web application are steadily increasing. Yu et al., (2011) shows that the weaknesses found in web applications enables attackers gain entry to unauthorized systems, accounts and obtain confidential and sensitive data. Access to such information has the potential of eroding trust among the concerned parties.

According to (Petukhov & Kozlov n.d.), the most effective way to detect vulnerabilities in web applications is by manually reviewing the code. This procedure suffered various challenges which include; takes much time to consume, expert skills is required, and prone to errors that are overlooked. This lead to various security experts actively developing automated approaches to detect various web security vulnerabilities. The approaches are categorized into three broad testing categories. They are black box, white box and grey box.

The black-box approach was designed to analyse user generated actions on web applications. In this analysis, the process assumes there is no source code for the web application. The technique behind this method is that a user submit many but various patterns of SQL in a web application forms or pages. Analysis is then done to assess the results. When an application shows errors, it is assumed that the application shows some traces of vulnerability. However, the black box technique does not gurantee completeness and accuracy based on the result it captures.

On the other hand, the white-box technique anchors its analysis on the server side when assessing web based applications. When using this method, the application's source code is assumed to be available. Analysis techniques such as using static or dynamic procedures can be invoked. In his paper, Kumar (2015) made a complete survey of these techniques and made several statements, however this approach suffers from susceptiblity to false positives and false negative which is necessitated by imprecisions in analysis. Moreover, this challenge is further compounded by the dynamic and amplified use of scripting languages. While static analysis methods often execute analysis with precision, it's emphasis is anchored on the control path.

The white box approach also suffers from dynamic analysis which is performed on paths that are already executed. This poses a challenge regarding the covered paths during an execution (Mirjalili et al. 2014). While this is a major weakness of this approach, internal access to web application through dynamic analysis make the technique precise.

The grey-box approach combines both the black box techniques and white-box techniques. The primary objective of this approach is to generate all the vulnerabilities that can be detected by white box method and test them using black-box approach. However, the approach inherits the weaknesses of black box approach.

The focus of this research is detection and prevention of SQL injection web vulnerabilities using Hybrid multi-agent approach in open web applications. Literature survey began by searching professional journals in leading e-libraries, searching Google Scholar to get the latest articles and web application security companies such as Owasp (2013). There was an overwhelming number of papers written on SQL Injection. The review has demonstrated that there are sizeable approaches available which have widely been used in detecting and preventing SQL Injections. These techniques range from traditional approaches to current techniques such as Hybrid approach. In this study, the review concentrated on various approaches available for improving detection and prevention of SQL Injections, investigation of existing approaches of SQL Injections in web applications. In this paper, the author point out where existing works depart from the point of focus in this study, revealing the gaps in literature that this study aims to address.

2.2 SQL Injection

SQL injection refers to a database interpreted language. It form a third layer of any web application. In this layer, the SQL constructs statements are incorporated using data supplied by text or users.Web application usage by various organization and individuals in the last ten years has attracted extensive discussion in practice and research regarding SQLinjection vulnerabilities. This is because SQL web vulnerabilities have contributed to a significant web application insecurity issues. Johari & Sharma (2012) point out that if SQL statements are constructed in an unsecure way, the application developed is likely to be vulnerable to attacks linked to SQL Injection. This is to say that,   if the data supplied by the users is not validated correctly, the user may alter or trigger malicious SQL statements. These malicious statements can arbitrarily amend the database contents or make the target machine to execute a malicious code.

2.3 Challenges Involved in Prevention Implementation

During development, web application developers face various difficulties in their process to totally secure web applications. To mitigate these problems, web developers have to assess the application state during the time of development. Thus, developers should embrace aspects such as developer priorities and technological approach as Etienne Janot, Pavol Zavarsky (2008) explains.

Researchers implementing SQL injections attack solutions have faced various challenges. Etienne Janot, Pavol Zavarsky (2008) cite various challenges including identification of entry points as one of the task encountered when implementing a protection solution. Identifying entry gates becomes a problem with multiple data input channel entry points such as Cookies, GET, POST and User-agent headers. This is because entry points have to be known for the effectiveness of protection schemes. However, it becomes more difficult with big web applications and complex architects.

Another challenge is, queries which are segmented and implemented across various multiple modules makes it harder to trace and sanitize entry points.

Evasion techniques is another challenge, attacks continuously evolves which overcome the commonly used approaches like black list mechanism since whitelisting is normally cumbersome to use. However http attacks need to be normalized before detection are applied hence blacklisting is limitted and cannot offer full protection.

Keeping upto date validation rules with evolving database is another challenge. However this could be resolved by bringing database structure closer to the application.

The finding from these studies partially informs the present study. In this study, the author was interested in how users requestcan be redirected efficiently to reduce response time when users are accessing Content Delivery Network (CDN) on a replica server. The need to optimally redirect user request contributes to less network resource wastage while enhancing user website experience. Even though there are glaring similarities, two principal points exist of deviation of this body of knowledge from this study. First, this study focuses on a comparison of extant request directions and their ability to enhance user response time while Chen et al. (2016) literature concentrates on an improved load balancer compared to the traditional load balancers. The second point of deviation lies in the dependent variable. This study focuses on understanding how Software-Defined Networking (SDN) technology can be used in a CDN network to improve response time and guarantee optimum performance. Part of these processes involves evaluating load balancers with other techniques in the discipline.

2.4 Target of attacker

According to Phalguna Rao et.al.,(2014), the attacker searches fields for user input and the parameters linked to it that are exposed to SQL injections in a web application. Different types of databases responds uniquely to attacks and queries directed to them. The attacker "Fingerprint" these information to the database thus able to know the database’s type and version housing the web application. Armed with these information, an attacker can perform targeted attack on the database.

2.5 Web application vulnerability testing tools

Stuttard & Pinto (n.d.) stated that web application vulnerability scanners are programmed softwares that examine web applications to determine security vulnerabilities. These scanners are able to identify security breaches such as command execution, server configuration errors, SQL injection, and directory traversal among other breaches. Testing tools can be acquired commercially. However, there are many other tools available as open source. Whether it is a commercial or an open source tool, each of these tools have their strengths and weaknesses. Majority of these tools crawl a web application and identify application layer vulnerabilities either by inspecting them for suspicious attributes or manipulating Hypertext Transfer Protocol (Http) messages. For very complex cases these tools emulate attacks originating from peripheral hackers. They also provide advance techniques for depicting different forms of vulnerabilities. Majority of testing tools which work by penetration use fuzz testing method. Fuzzing is the most advanced testing appraoch that covers wide range of cases. The technique allows the application to accept invalid data as an input. This process limit the chances of vulnerabilities.

Jaiswal 2014, explained that the penetration testing tools are efficient and fast. They are able to detect security breaches quickly. Moreover, unlike conventional black box testing, any person with little technical know-how on security can use penetration.

However, despite being effecient, penetration testing tools have disadvantages. This tools cannot find all vulnerabilities. Penetration testing tools are poor at finding vulnerabilities like access control flaws, identification of hard coded backdoors , multi-vector attack , information disclosure and encryption flaws. Further, the use of random data also fails to uncover vulnerabilities unless the fuzzy process is repeated several times. Penetration testing tools do not have any specific goal to work toward, and, therefore, try to attack any possible risk. (Mirjalili et al., 2014).

2.5.1 Web Vulnerability scanning tools

The following open source web scanners were used by the researcher in this study, Vega, Zap and Wapiti.

Vega is an open source scanner. It is also used as a testing platform. Vega is a powerful program with capability of finding and validating SQL injections as well as ensuring safety of sensitive information when scanning for vulnerabilities. Developers designed Vega with an automatic scanner suited for accelerated tests.

Vega is designed with an intercepted proxy for tactical inspection. The Vega scanner spots SQL injection and other security flaws in a database. Similarly, the program have an Application Program Interface (API) which makes it easier to extend based on the language being used by the web application on the internet. Vega classifies the scan alert summary into four categories namely high, medium, low or Info. It provides a report with each of the categories as mentioned in the groups above. The report consists of all vulnerabilities found, and the quantity..

Wapiti is an open source web scanning tool that performs a security audit of a web application. It employs a black box approach i.e. it does not study the code. Instead, it checks all the web pages and identifies forms found on a web page where it can insert or reject data. Based on the information derived from its website, it is clear that Wapiti can detect SQL injection among others web vulnerabilities.

Wapiti has an architectural which support POST, Http and GET techniques of web application attack. Its characterized by its ability to provide comprehensive reports after completion of a scan, authentication using various methods such as NTLM, Kerberos or Basic, ability to define or limit the scope of the scan to a folder, web page or a domain, update to understand recently release web development technologies such as HTML5.

Zap Attack Proxy (ZAP) is an open source web scanning tools that use a Graphical User Interface (GUI) interface. The application suits both new developers and experienced programmers. To utilize the application, simply input the URL of the application you would like to scan and wait for the scan to be completed then review the generated report.

2.6 Multi-Agent system

2.6.1 An agent

An agent according to Weiss (1999), is a software or a robot with computational capability (Figure 3). The software perceives and act independently based on its environment. To execute these process, the program depends on its “experience”, hence, it is regarded as an "intelligent entity". It performs its actions in a flexible and rational manner in different environmental circumstances given it is a perceptual and effectual equipment.

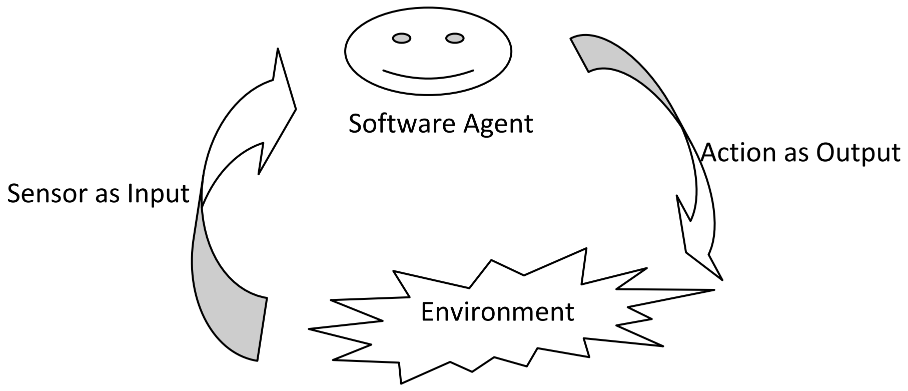


Figure 3: Agent in its environment

2.6.2 Multi-Agent System (MAS)

MAS designate multiple entities (agents) in a distinctive environment. The system is composed of automous collection of agents capable of defining their objectives and actions. Further, they interact and collaborate with one another by passing messages (communication). In a multi-agent system environment, the agents collectively work to address a particular problem in context. The system provide a perfect platform for negotiation, competition, coordination and cooperation among diverse functional units. Communication is made possible by using an interaction protocol to accomplish their roles.

2.6.3 Role of Agents in Web Application Security

Multi-agent systems has positively impacted web applications. Being a distributed systems, it has a number of advantages. Multiple agents operates in parallel, hence this results in an increased overall speed. Agents operate in an asynchronouse which increases its efficiency. when a single or several agents fail it does not intefer with the whole system. This is because other agents in the system undertake the role, thus increasing agent robustness and reliability. Agent system has the capability of being scaled. This can be done based on the magnitude of the problem to be solved. Scaling can be achieved by adding new agents; adding new agents does not hinder the operations of other agents in the system. Scaling increases agents flexibility. Compared to centralized systems, agent system are far much more cost-effective since agent system is consists of simple subsystems with low unit cost. Agents are designed autonomously. Individual agents are disntinctively designed and developed by developers.

2.7 Agent Development Methodology

2.7.1 Multi-agent System Engineering Methodology (MaSE)

The MaSE methodology is used in the distributed agent paradigm. The methodology has analysis and design phases. The analysis phases comprises of detailed stages that include; capturing system goals, use cases, refining roles of agents among others. In design phase, four phases are applied. These phases include builting classes for agents, creating conversations, assembling classes for agents and designing the system.

This study used MaSE methodology for the design and analysis of the system.

2.8 Related Work

2.8.1 Manual approaches

In this section various techniques about the manual approaches including defensive programming used to detect and prevent input manipulation of web application vulnerabilities in particular SQL injection (Kumar Singh & Roy, 2012). Kumar Singh & Roy, (2012) stated that the input text or data manipulations could be avoided by designing the system to prevent user input from comprising malicious characters or keywords. Moreover, they stated that input filters could be achived using white or black list approach by programmers. However, many developers of web applications pay little attention on risk linked to SQL injections on applications that incorporate back-end databases which gives the attackers an opportunity to perform their attacks. Code review approach detects bug at low cost. However, it consumes a lot of time in comparison with automated static analysis. There are high chances that it may be implemented by developes because of tight timelines and race to ship an application according to Bhat et al. (2013).

In this research, a Hybrid multi-agent technique and approach was used to help advance validation of user's input by delivering data on a system.

2.8.2 Static and Dynamic Code Analysis Approach

The rise of web application vulnerabilities has attracted extensive discussion in practice and research. This is because SQL injection has contributed to a significant security threat in web application databases (OWASP (2013) vulnerability list report). Traditionally, developers use data validation by checking through the system entry points. They use commands such as GET and POST (Rawat et al., (2011). However, a major rise and demand for more secure web applications and significant complex applications create a challenge on the security of web applications. Rawat et al., (2011);Adam Kie‘zun et. al., (2010), in their paper, presented an improved approach for detecting vulnerabilities such as SQL injection attacks in web application, this technique is implemented using Ardilla. Ardilla automatically creates model input that; expose SQL Injection, symbolically traces taints through implementation including database access and transforms the inputs to generate concrete results. However, this approach has various limitations: the approach is based on the source code of the web application, its design is limited to specific application like PHP applications, therefore, this approach requires a developer availablity. Also, it requires learning skills because a developer will need to adjust the source code. This approach faces a major challenge in a case the pioneer developer abandons the project, it would be cumbersome patching the vulnerabilities.

In an attempt to come up with a more efficient approach, (Jeom-Goo Kim 2011) presented a technique that uses combination of dynamic and static assessment to inhibit dissemination of SQL query by the user in SQL inquiry attributes standards. The process is achieved by using a utility that has the capability of detecting SQL static query element in web driven application.This is achieved at run time by detecting SQL queries. This approach compares SQL derived from normal users with SQL query produced dynamically by a potential attacker. Moreover, this approach has challenges in regard to learning and adjusting the source code.

These pieces of studies inform this study by providing an understanding of various components to be incorporated in the current approach to enhance efficiency in SQL injection detection, reliability , lower rate of false positive and false negative and reduction in the span of time taken to scan web applications for SQL injections. Furthermore, lack of cross platform application has also been noted extensively in literature enabling clarity in defining the multi platform of interest in this study.

AMNESIA, according to William et. Al. (2012) is an approach that amalgamate runtime monitoring and static in a web application in detecting and preventing SQL injection attacks. The static part is used to put together a legal query using program analysis and the other dynamically generates queries automatically using monitoring during runtime.
A database server detects structural query injected by a form support approach. If queries resist the approach, then the technique prevents operation of the queries on the database server using these steps; identifying the hotspot, building SQL query models, instrumenting application and runtime monitoring. The major shortcoming of this approach is it is only implemented on Java supported application.

2.8.3 Use of API Approach

McClure, R., McClure, R., Krüger (2013) found the concept behind SQL DOM which is simpler than others depending on developers ability to perform the complex defensive coding techniques in building dynamic SQL queries using strings.

However, an API was used to enhance the security. The Sqldomgen is used to perform analysis at compile time on the database schema, hence constructing various set of SQL query classes integrated with IDE which developers calls directly to construct SQL queries.
This approach maps various variations of SQL queries according to tables and columns.
Moreover, McClure, R., Krüger, 2013, stated that this approach was categorized into various classes with strong-typed methods including SQL statements, table columns and where conditions. validation of data types is automatically achieved by mapping the data types in the database.
In this approach, development of column classes strings replaces quote with doublequote in strings at runtime to sanitize them. However this approach suffers from various weaknesses and limitations which include writing new query ,rewriting query generated codes, overheads for developers training and code rewriting, its full-object policy comes at a cost and stored procedures remain unprotected.

2.8.4 Hybrid Approach

Pankaj Sharma, Rahul Johari, S.S. Sarma 2012, proposed an approach which is an extension of replica based hybrid technique to curb SQL Injection attack (MHAPSIA) works in production web environment which provides protection from SQL injection. This approach is categorized into a production environmental mode and safe mode. In safe mode, a secure query model for SQL is created while in the production environmental, dynamic queries are validated against a secure query model in safe mode, and approves typical input request against sanitizer mode.

Other studies such as Al-Amro and El-Qawasmeh (2012) have investigated SQL Injection with an intention of improving the security of web applications using an algorithm.These studies have broadly examined algorithm and describe the leaks depends on analyzing the web application source code. the study gave 12 steps given each performs specific kind of leaks where SQL injection vulnerabilities are discussed.
Al-Amro and El-Qawasmeh (2012) contributed significantly to the Hybrid Approach by developing a program that inspects various forms of writing; in code, no standard, and the availability of some alternatives commands. The proposed algorithm combines only two compiled languages for use, i.e., VB and C#. the study is limited in that other languages like Java are left out.

Bhat et al. (2013) presented a hybrid approach which combines an audit and signature based method. In the signature based technique, the authors used a token wise string to present a detection mode for SQL injection. They used a technique known as the Hirschberg algorithm. The algorithm was based on the principle of divide and conquer . The technique was adopted to minimize the complexity of time and space. In audit technique, the authors analyzed transactions to determine presence of malicious access. However, this technique harbored several challenges such as ineffienct detection of attacks. It further generated false positives and false negatives.

A significant gap in this studyis focus on one single approach to SQL injection detection rather than taking a combination of various methods. Besides, the literature points out that most existing web application SQL injection scanners have not been 100% in detecting web vulnerabilities. Therefore, this study seeks to contribute to these bodies of knowledge in SQL injection by investigating existing approaches and proposing a better approach to efficiency detection of SQL injection with a minimal rate of false positive and false negative.

2.9 The Gap

The literature provides notable highlights on the security of web application regarding SQL injection detection rate. Studies such as Johari & Sharma (2012) have confirmed the shortcomings of SQL web scanners. Other studies have shown an existing opportunity for combining various techniques to detect SQL injection. The current techniques suffer from weakness which include; unfinished implementations, frameworks that are complex, runtime overheads and it is manual. Further, the technique has higher prevalences for false positives and negatives. The gap evidenced in the literature necessitated the undertaking of this study. The study seek to fill the gap by designing and implementing hybrid multi-agents system approach, to detect web applications vulnerabilities, provide a better coverage with no false positive and false negative, based on short span of time taken to scan and increased detection trend and accuracy.

2.10 Proposed architecture

The study proposed an architecture in which situated agents in various modules continuously detects SQL injection web vulnerabilities (Figure 4). The agents used a statement from the web application to detect SQL injections. On receiving the statement from web application, the agent analyzes the statement for any suspicious activity and either make a decision for the statement to be executedif no suspicious activity is found, then the statement is sent to database for execution. Upon existence of a suspicious activity, the statement is compared against the existing specification and it is send to the audit agent module. If the analysis and the auditing module processes have been fulfilled, it will provide a complete transaction. Hence, and SQL injection alert is generated.

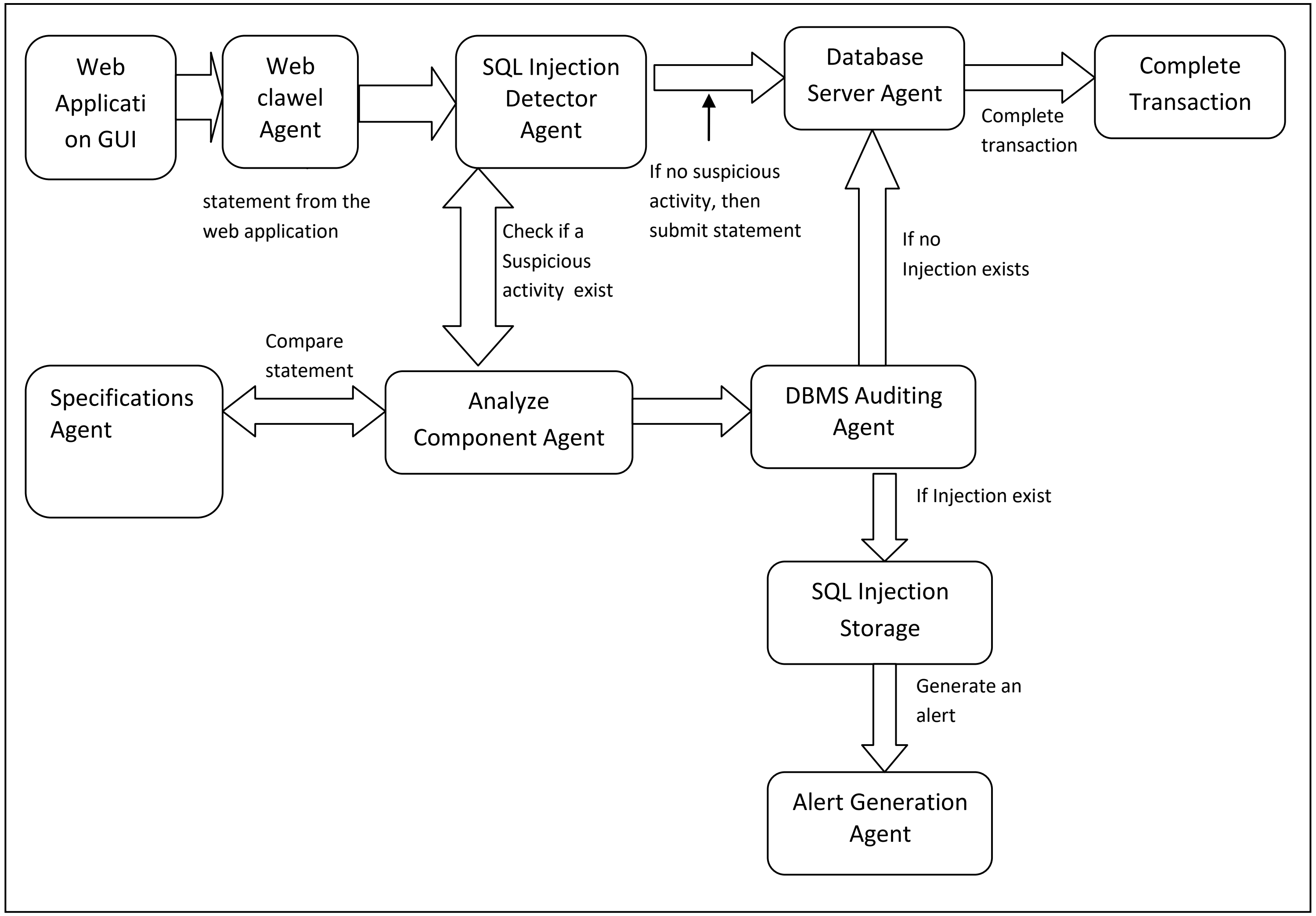


Figure 4: Representation of the proposed architecture

CHAPTER THREE

3.0 METHODOLOGY

3.1 System Design

Multiagent Systems Engineering (MaSE) was used as a methodology for system design in this study. MaSE provided a guide in analysis and design. Two steps were involved, analysis and design.

3.1.1 Analysis Phase

In the analysis phase, the first step entailed capturing the system goalswhich involved initial specification of the system and transormation to a set of system goals. These goals were analyzed and mapped to a Hierarchical diagram. The second step used cases to built a set of Sequence. This would assist in identifying the initial roles together with message routes by the system analyst. The third step is refining roles; here constructed goals are transformed with Sequence diagrams into roles. When roles have been created, each role is associated with tasks that describe the behavior exhibited to achieve its goals successfully.

3.1.2 Design Phase

The second step was design phase. The design phase involved builting agent classes using defined roles in the analysis phase. Design stage provide a class diagram for the agent at the end.This step illustrated the general system organization that involved classes of agent and the conversations. The second step in the design phase, was construction of conversations. The conversation is constructed from messages and agents states for individual path communicating using a Con-current Task Method, increasing messages and agent states for increased performance. Assembling agents step, creates the parts of the agent classes. This process was achieved in two phases. The phases specified the architecture of the agent and defined the features and components that constitute the architecture.

The outcome of the analysis and design phase is an Agent Architecture diagram. At design step, the classes defining agents are instantiated to real agents. The numbers, location and types of agents are shown in a deployment diagram (Figure 11).

3.1.3 Limitations of Mase Methodology

MaSE methodology has the following weakness according to Dam & Winikoff n.d.)

1) Gap analysis and design phases: There exist a gap hence one should analyze the role requirements to get important information for suitable architectural structure of an agent

3.1.4 Justification of Mase Methodology

MaSE methodology major strength is tracking changes in the development process. Everything that is created in analysis and design phases can be traced from the beginning and from the end in various steps. An example is derived goal from Capturing phase which can be tracked into a task, role, and agent class. Also MaSE supports requirement development, analysis and implementation.

3.2 Hybrid multi-agent System

This sytem was designed with an objective to reduce scanning time in web application and achieve increased detection accuracy. In this study, accuracy of the Hybrid multi-agent system was fully optimized. Additionally, the fuzzing and crawling components were engineered to work efficiently and deliver acceptable results.
Accuracy was given a lower priority while scanning time was awarded a priority. In a real world scenario , it would not be practical for users to wait for time consuming web scanners. As a matter of fact an application that takes long to scan would be ignored by users.

3.3 Simulation Design

This program aimed at testing and validating the hybrid muilt agent system. User's types the URL and click the start.The scanning involves crawling, parsing and discovery of vulnerabilities, this process goes through each web application link while scanning for vulnerabilities, and the analysis is done to display report of discovered vulnerabilities and their location (Figure 5).

The scan of web application involved crawling and parsing. Scanning process leadto discoverey of the vulnerabilities, which was achieved by reviewing each web application link while scanning for the web vulnerabilities until the end of the process. After the completion of the process, the results were analysed to display the discovered vulnerabilities and their location. The results were analysed further to generate a report.

The process steps involved the following steps:

Input – URL of the web application is entered before the start of the scan

Processing - Processing entails crawling entire web pages, fuzzing and identifying weaknesses and exuding inputs of web applications to confirm SQL flaws.

Output - Completion of a process make the program to display the results.

3.3.1 Contents of the Scanning Report

The reports includes features including number of vulnerabilities discovered,type of the vulnerable discovered and the location or webpage where the vulnerability has been detected.

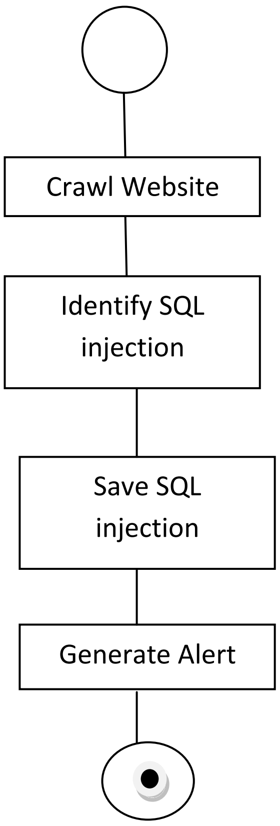


Figure 5: hybrid muilt agent simulation diagram

3.4 Target Population

The study targeted opens source web vulnerability testing tools . Any open source web application vulnerability scanning tool was eligible to be used in the study . further the study targeted web application which have know SQL vulnerabilities

3.5 Sampling Procedure

Purposive sampling based on various categories or classifications was used to select the tools used in the study. The tools selected from lists available on various online portals that classify open source web scanning tools in reference to various factors such as their capability and detecting vulnerabilities.

The web site had all well know vulnerabilities in advance which the tools was benchmarked with. Web applications from WAVSEP by chen (2014) were developed to analyse and detect the degree of accuracy of scanners being used. The goal was to increase broad understanding of detection barriers, and determine how each scanning tool could navigate diverse vulnerabilities discovered. The metrics used in this research included , detection accuracy, number of vulnerabilities detected, time taken to scan web application, consistency and stability. These metrics are similar to those used by McQuade (2014)

3.5.1 Sampling Size

Commons (2012) suggested that to design the sample, factors such as sampling frame, parameters, target population, suitable sampling technique and sample size of the sample should be considered. Purposive sampling was used to select web application with known vulnerabilities as well as the tools for scanning the chosen applications.This was because purposive sampling accords the researcher the leeway to target cases that had the required information. However all the selected tools were benchmarked with OWASP top ten list of vulnerabilities . Analysis was performed against the set metrics to chosen tools which was a representative of the sample. The algorithms of these tools were analysed and used to inform on the required improvement.

3.5.2 Tools that were not selected for this research

1. Commercial tools were not considered in this study since the source code is not available for scrutiny . In addition , these tools are pretty expensive and out of reach for some peole.
2. Tools that are no longer available for download.
3. Tools that after installation could not work well for one reason or the other.
4. Tools that have not been updated for a while since accurancy is not guarantted.

3.6 Data Collection

This study relied primarily on quantitative data. Data was collected on the detection accuracy, the number of vulnerabilities detected, realibility, consistency and stability. Data was collected through observation and examination of the reports displayed at the end of the scanning process. These metrics have been used in a study that was conducted by (Mcquade 2014). Each web scanning tool was tested against the web application with all the relevant settings configured.

Metrics.

i. Detection accuracy – vulnerabilities detected by the web vulnerability tools . This is expressed in terms of percentage.

ii. Time – the time taken by the any of the tools under study to scan a given web application

iii. Consistency and realiablity – this was arrived at after running the same tool several times against the same web application under the same conditions and configurations and comparing the results.

3.7 Tools used in the experiment

The open source vulnerable tools selected were Wapiti, Zap and Vega, while Ron Scanner was developed by the researcher to test and validate the hybrid multi-agent system.

CHAPTER FOUR

4.0 SYSTEM ANALYSIS AND DESIGN

In this study, the system analysis and design was guided by the MaSE methodology.

4.1 System specification

4.1.1 Overview

Currently, SQL injection is ranked as the topmost web application vulnerability. The proposed system detected SQL injection and reported the injection. The SQL detection involved the checking of database entry points and string text. A normal SQL injection usually detect against the already identified SQL injection stored in a database. A blind SQL injection might not exist in the already identified SQL injection database , but the system will detect and provide the event logs.

The process of SQL injection detection entailed detection of SQL injection attacks, based on the scaning of the specified website URL.The process of SQL injection identification was done by agents, forming a multi-agent system.

4.1.2 Inputs and outputs

4.1.2.1 Inputs

Ron scanner system is scanning an existing web application both in production and development environment; therefore the URL of that web application is the input. Since the system was detecting SQL injections, it would be necessary to define the SQL injections. A list of past SQL injections that are in the current Injections database tracking system provided additional input to the system.

4.2.2.2 Outputs

The system outputed a list of identified SQL injections and their status. The status was either “Vulnerable”, “Not Vulnerable”.

4.1.3 Data management

The data used by the system was stored in a database. The system saved all the identified SQL Injections in database. Likewise, the status of each SQL injection attack was saved in the database.

The list of past errors described in section 4.1.2.1 above were saved in database.

4.1.4 System failure

On the event that this prototype system failed, a progress tracker agent indicated the point of failure. The output showed the progress of the SQL injection identification, and if the progresses of the errors stagnate at a particular agent, then it would be an indication of system failure.

4.2 System analysis

System analysis phase produced a series of roles that described the function of the system and what the system required to achieve the overall system requirements. An entity that was performed within a system was reffered to as a role. MaSE has each role assigned to accomplish a certain responsibility. Each role work in unison to enable the system achieve sub-goals or overall system goals. Assigning roles involved a series of steps;

1. Goal Identification- A goal was identified from user requirements and structured into an Hierarchical Diagram.
2. Identifying use cases - The use cases provided sequence diagrams that were used to identify initial roles and communication routes.
3. Thesystem goals were then translated into a set of roles

4.2.1 Identifying goals

Goal identification was the initial step in the analysis phase, At this step initial system specification are morphed into a planned set of system goals.

a). Capturing goals

Capturing goals was initiated by the process of extracting scenarios during specification. They described the scenario's goals. The following were scenarios in the initial specification:

1. The system was responsible for detecting,and reporting SQL injections
2. SQL injections identification involved scanning of the target website, and detecting SQL injections.

3. A knowledge base of web vulnerabilities was used to inform the system on the type of vulnerabilities to report.
4. An SQL injections reporting component generated an Alert.
5. If a new SQL injection attack was available, the attack was saved in the database

Goals were then derived from the scenarios. The following were the derived goals:

1. Scan a website
2. Identify New SQL injections
3. Identify SQL injections
4. Generate an Alert
5. Reduce scan time of a website
6. Low false positive
7. Low false negative
8. Save new SQL injection

b) Structure the goals

The goals were put in hierarchies depending on the importance, level or detail. This resulted in a goal hierarchy diagram (Figure 6)

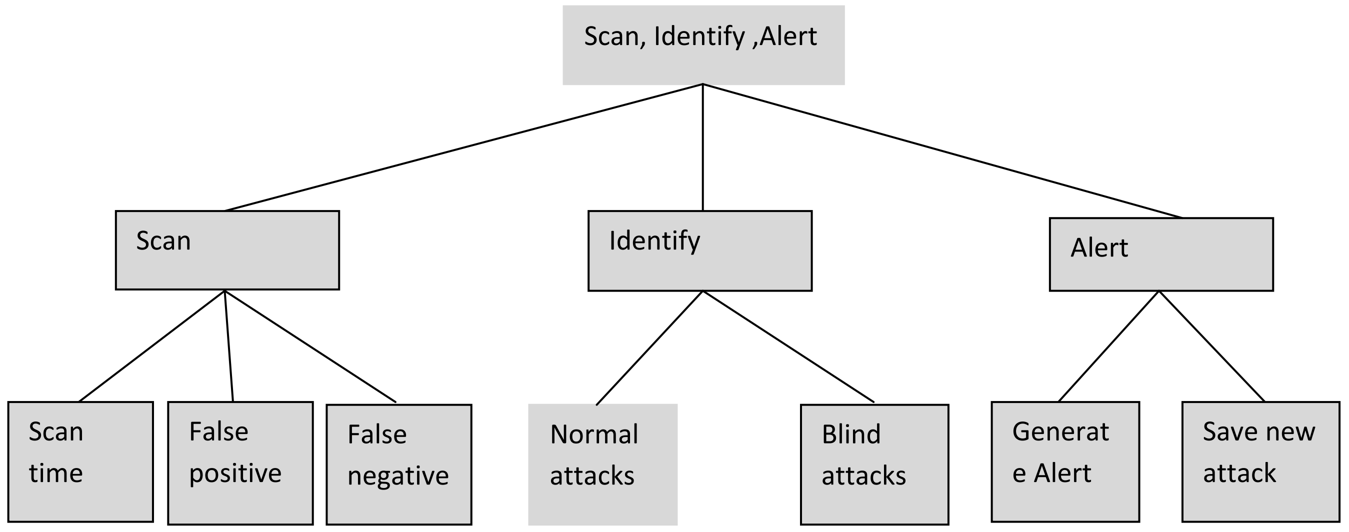


Figure 6: Goal hierarchy diagram

4.2.2 Applying Use Cases

a) Creating use cases

In web application system, there were events that occured and this events were defined by Use cases. Use cases illustrated exactly what the user think the system should accomplish (Figure 7). Use cases prompt users for more details or might need clarification regarding existing information about the goals of the system. The Use cases were created to achieve identification of paths of communication.

b) Actors

The main actors were the agents that are responsible for the scanning,identification, reporting SQL injection attacks (Figure 7).

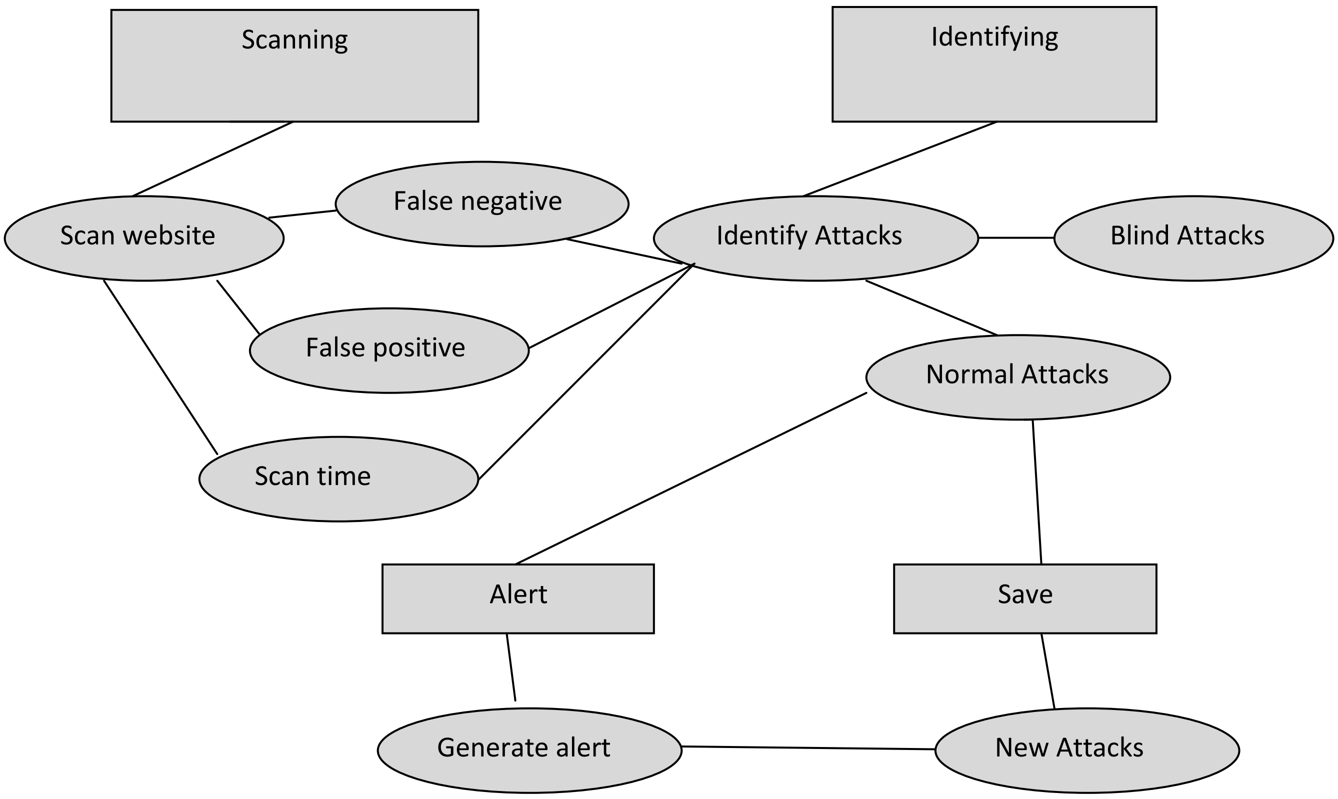


Figure 7: Use cases diagram

4.2.3 Refine roles

This step was important. Refining roles helped map sequence diagrams into structured goals. It enabled morphs sequence diagrams into roles according to their tasks. The goals helped in assigning tasks to be executed. The illustration below captures a role model (Figure 8).

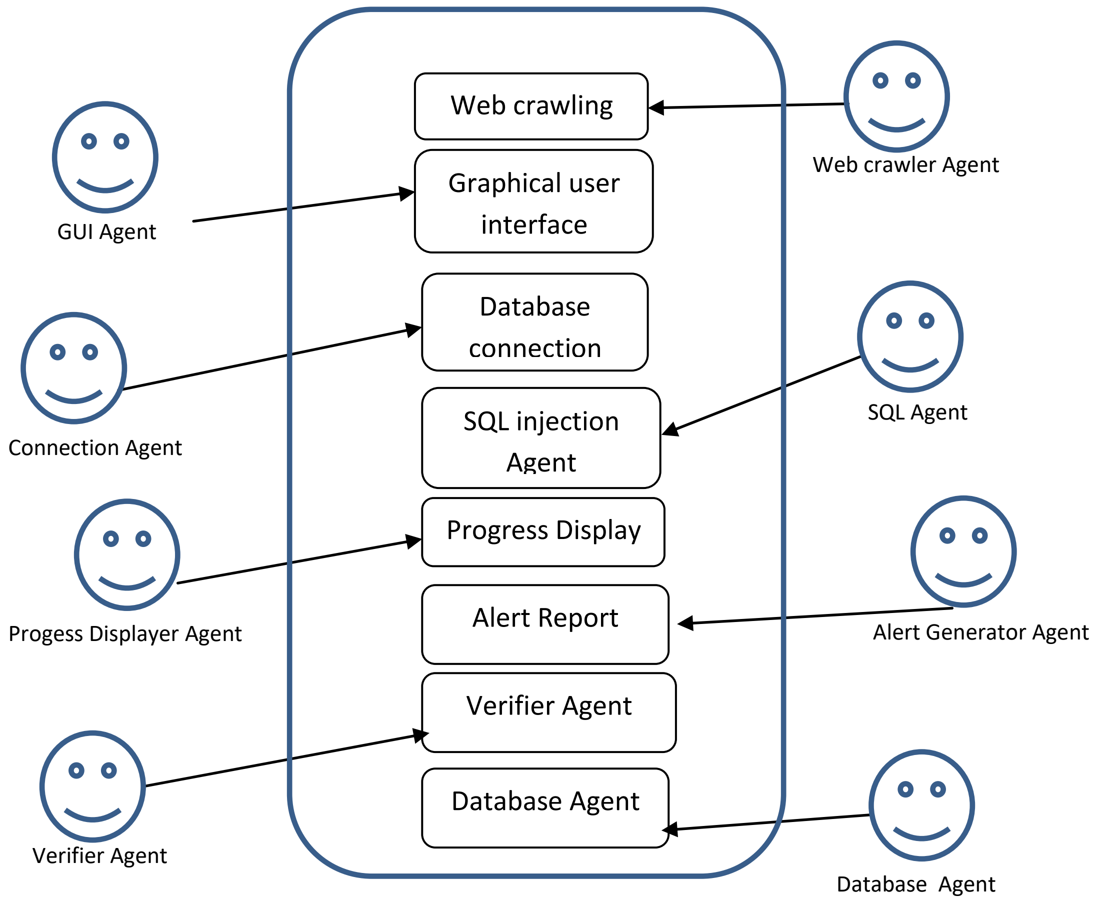


Figure 8: Role model Diagram

4.3 System Design

4.3.1 Overall architecture

Situated agents in the system detected SQl injections web vulnerabilities. On identifying an injection attack, an agent created and generated an alert, and saved it in the database. The system made scan web application both in development and production environment.

4.3.2 Crawling

Crawling flow chart is illustrated in figure 9 below. Crawling involves the following steps;

- Identify the root of the website (the home URL)
- Mark the pages as visited and push it into a queue
- Tranverse down to identify the immediate sub folders/ sub urls
- For each url in the url queue
    - Tranverse down to indentify sub urls
    - Mark them as visited and push them into queue
    - Repeate step 5 untill a dead end is reached
    - Once dead end is reached remove the url in the immediate top level from the queue

      Urls in the visted urls array / list it the complete set of urls for the web application

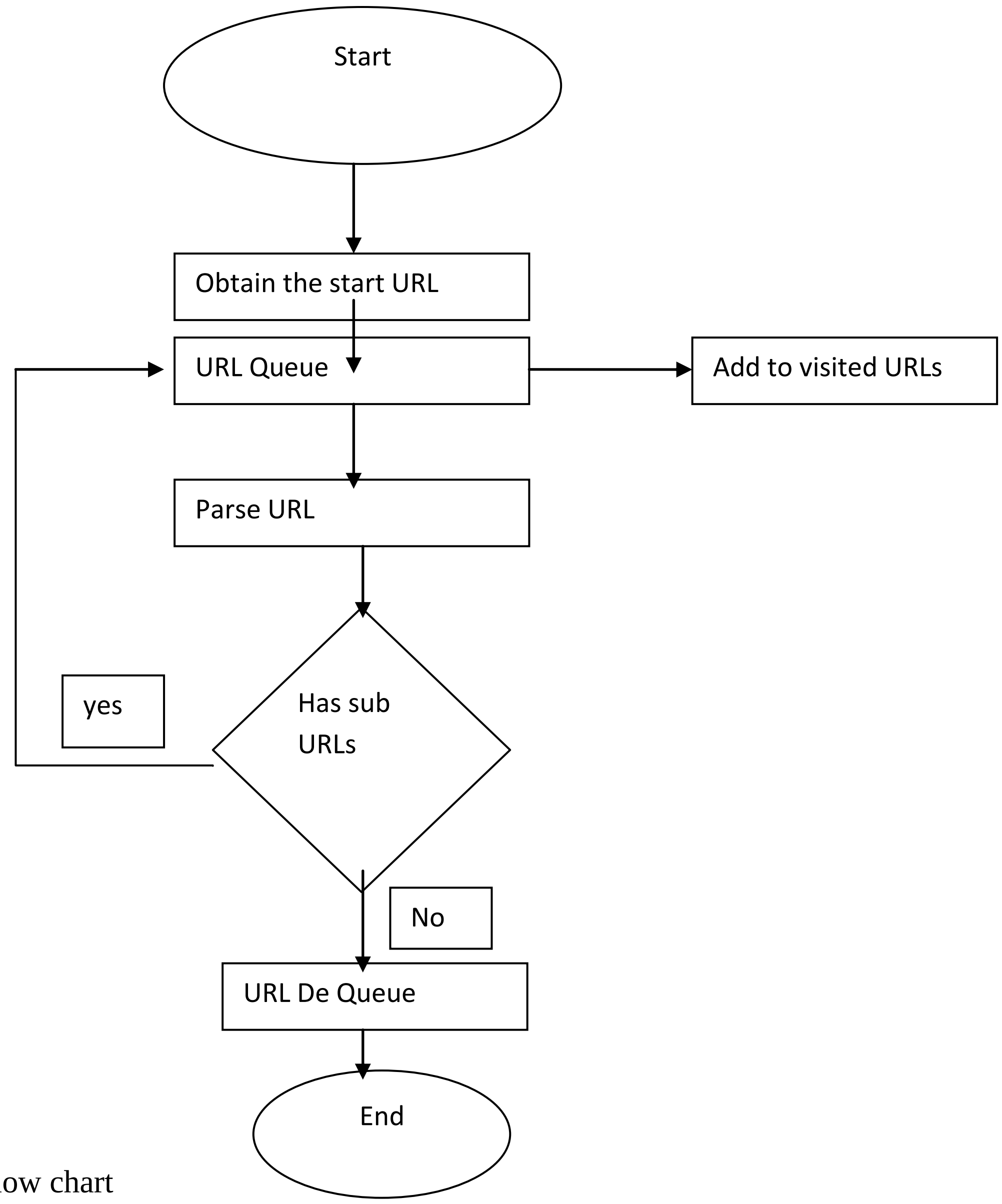


Figure 9: Crawling flow chart

4.3.3 Scanner Agent

A scanner agent accomplish scanning tasks through a series of steps (Figure 10). For a particular URL in a series of URL visited;

a. Parameters acts as identifiers

b. The parameters were added into a list of parameter

c. Execute scripts / test cases under for (Sql Injection)

d. Verify the response to identify malicious character set

e. Remove parameter from parameter queue

f. Report vulnerabilities

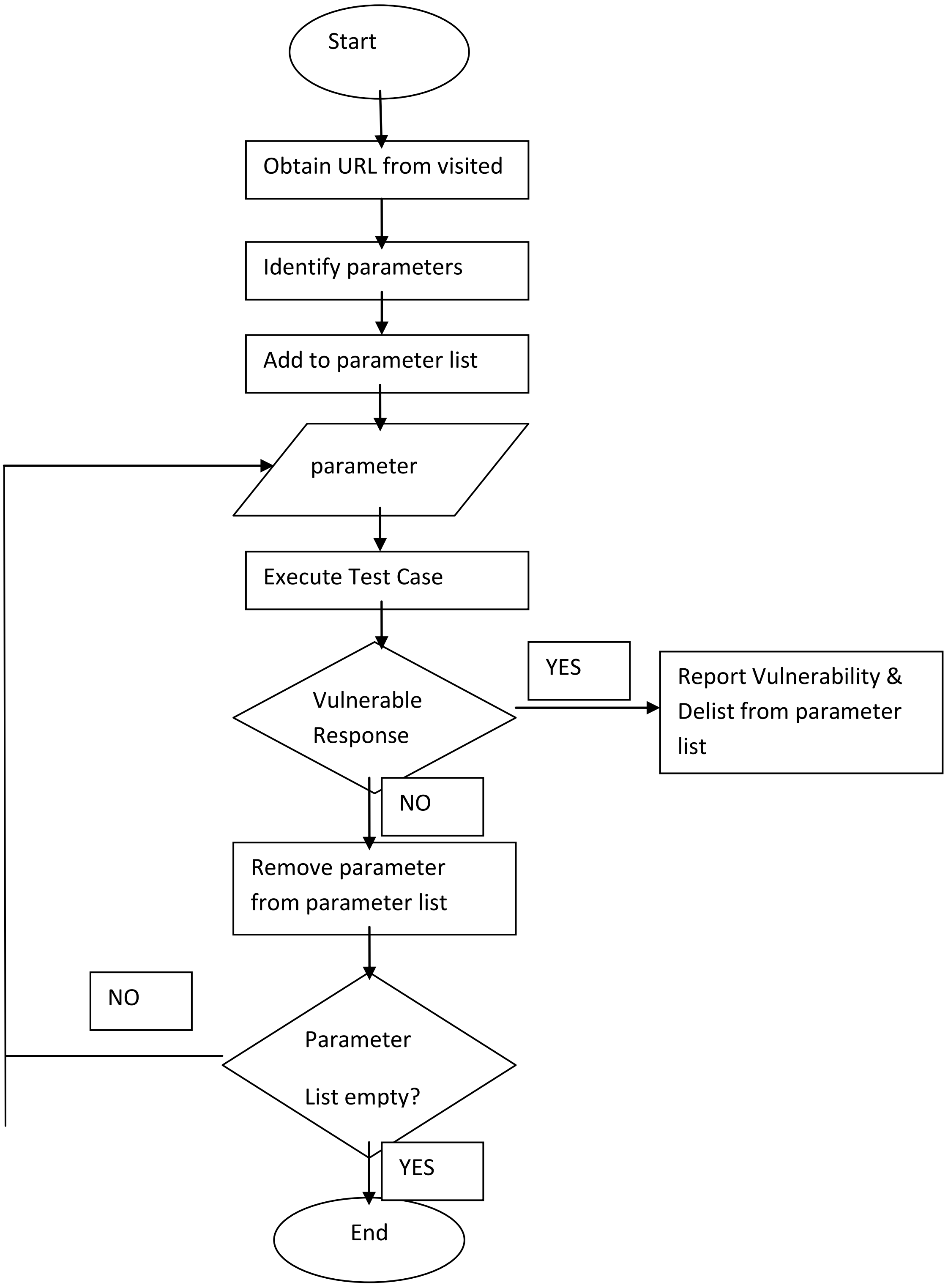


Figure 10: Scanner Diagram

4.4 Pseudocode

Start the application , enter the main URL and type details that web app should use to perform crawl. Add the main URL and sub urls into the list of visited URLs. queue URL to perform a search .

```
{
While the queue is not empty
IF the URL protocol is not HTTP or HTTPS then
        Break;
Go back to while
Mark this URL as ready  searched URL
If there exist a on the site then
If file includes . Disallow. Statement then
Break;
Go back to while
Open the URL
If the open URL is not HTML file then
Break;
Go back to while
Iterate the HTML file
While the html text contains another link
{
If robots.txt file exist on URL/site then
If file includes . Disallow .statement then
Break;
Go back to while
If the open URL is HTML file then
If the URL isn’t marked as searched then
Mark this URL as already searched URL
Else if type of file is user requested
Add to list of files found
}
}
```

4.5 Database design

The database stores the details of the identified error, the status of whether it is reported or not and the status of whether it is fixed or not. Moreover, it store the details of the fix applied on the target software.

A reported error might have one or more fixes, and therefore the relationship between the identified error and the fix will be one-to-many relationship. Main tables was created, for storing dected injections (Table 1).

After determining the data to be stored, and applying the normalization rules, the author came up with the main table described below:

Table 1: Main Database table

Name	**Data type**	**Length**	**Description**
VMA_ID	Int	4	Primary key
VMA_Main_URL	Text	100	Name of the main website
VMA_Sub_URL	Text	20	Name of the child main website link
Injection Status	Text	Max	Check whether the link is vulnerable or not
Scan Status	Text	300	A web link of uploaded screen shots
Alert	Text	20	Name of the computer where the vulnerability occurred
Injection type	Text	20	A status to indicate attack type
Date	DateTime	8	date and time when the error occurred

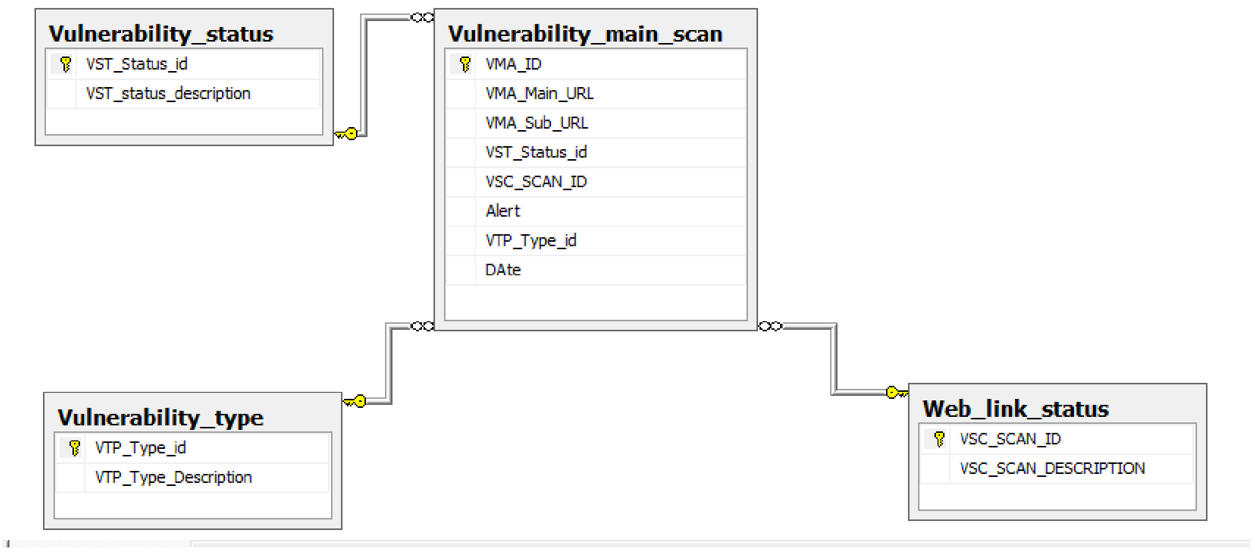


Figure 11: Database diagram

CHAPTER FIVE: SYSTEM IMPLEMENTATION AND TESTING

5.0 Tools required

To accomplish the goal of this study, the following tools were used:

1. Java Development Kit 8.1
2. Net Beans IDE 8.1
3. Jade 4.4.0
4. MySQL

5.1 System development

The system was developed using Java. The agents were run in JADE platform. The development was broken down into modules which are described below;

5.1.1 Website scanning module

Website crawler checks the validity of the website and crawl over the inner website links.This allowed complete scanning of the website by going through link by link.Link identification was done by reading crawler agent

5.1.2 SQL Injection Identification

The SQL injections were identified by use of SQL agent module system. The database agent created and checked if the attacks were new and saved it in the database.

5.1.3 Progress Displaying

This track the process of the system and guide the client of the processes taking place in the system.

5.1.4 vulnerability reporting

This handles reporting and display of vulnerabilities identified during the scanning process.

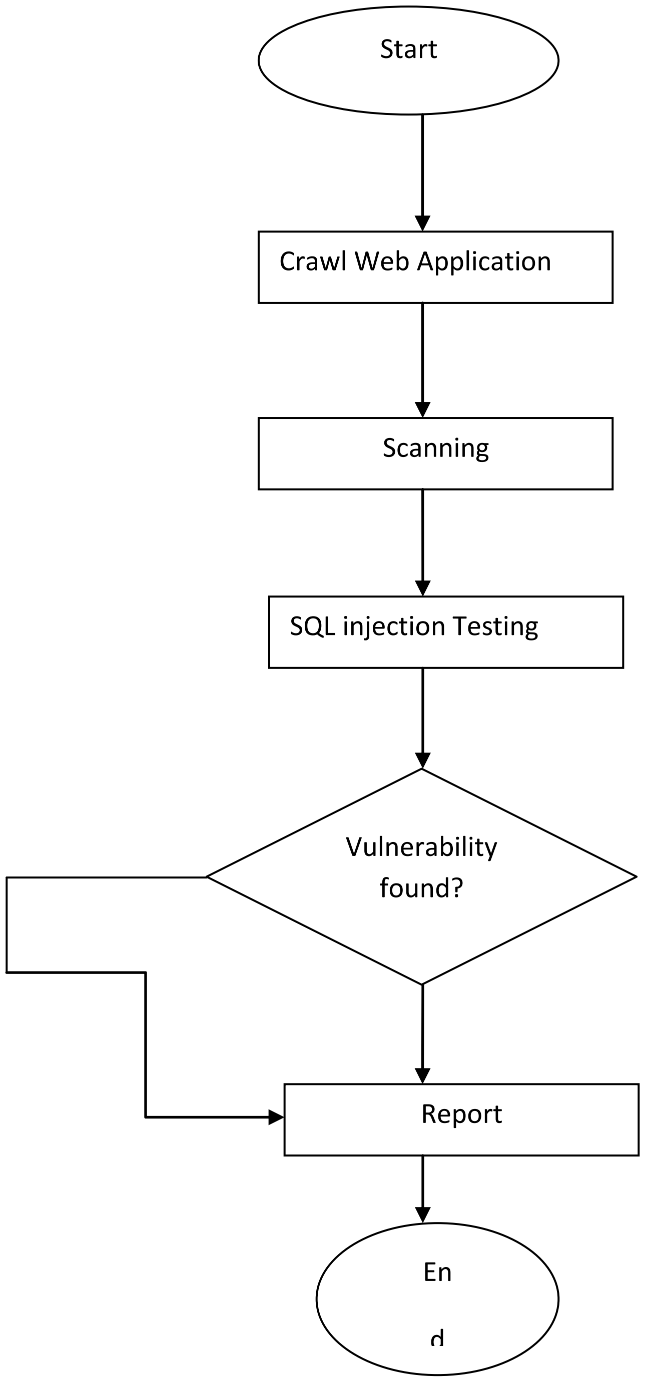


Figure 12: SQL indentification diagram

5.2 Configuration

Once the system was developed, the following configurations needed to be done before running the system:

1. Putting all the known SQL injection in database. These are the SQL injections that had occurred in the past.
2. Setting/updating the name of the initiating agent .

5.3 Testing and Experimentation

The system was set up in a testing environment, which was composed of websites in both production and development.

5.3.1 Testing procedure for SQL injections attacks

In this study, the selected web application vulnerability tools executed on aforementioned web application. There afterwards, the results were recorded. The test procedure is described below;

a. Start the web scanning tool

b. Enter the web application URL to be tested

c. Initiate scanning process.

d. Give the process some seconds to complete. If scanning is successful, a report is produced and displayed accompanied with the results. Web application vulnerabilities discussed part 2.3 uses this principle for scanning.

e. Repeat this process for all the tools.

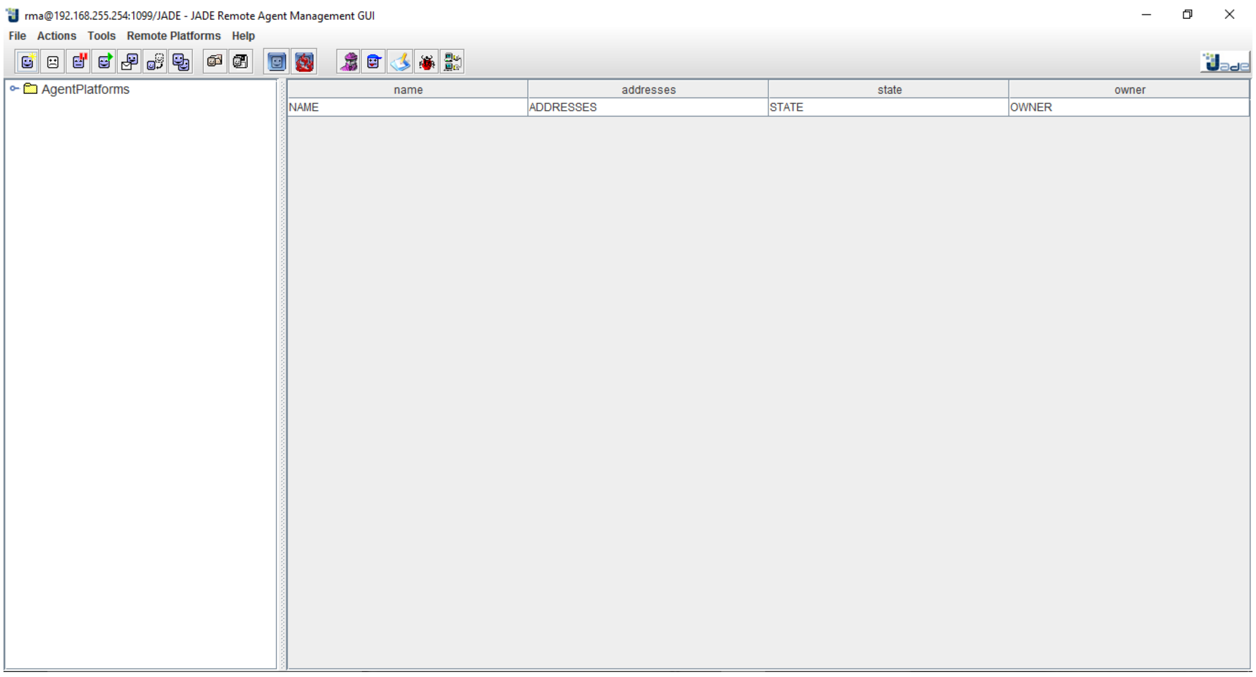
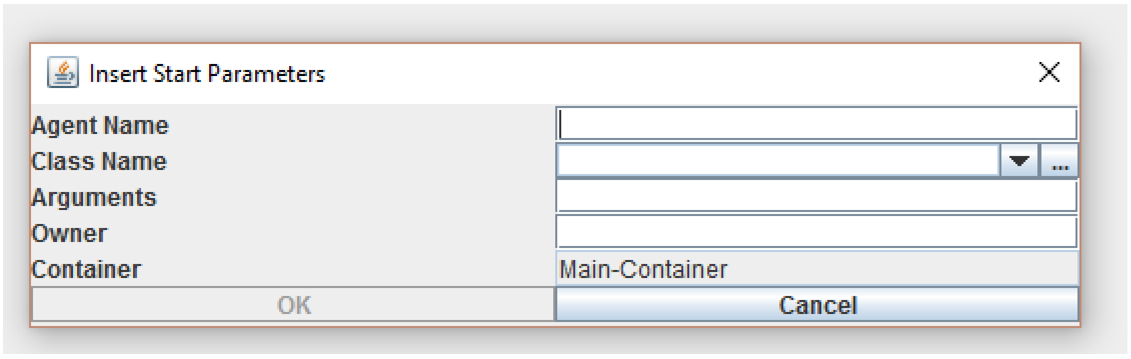

rma@192.168.255.254:1099/JADE - JADE Remote Agent Management GUI
File Actions Tools Remote Platforms Help
AgentPlatforms
name
addresses
state
owner
NAME
ADDRESSES
STATE
OWNER

Figure 13: Jade agents main window

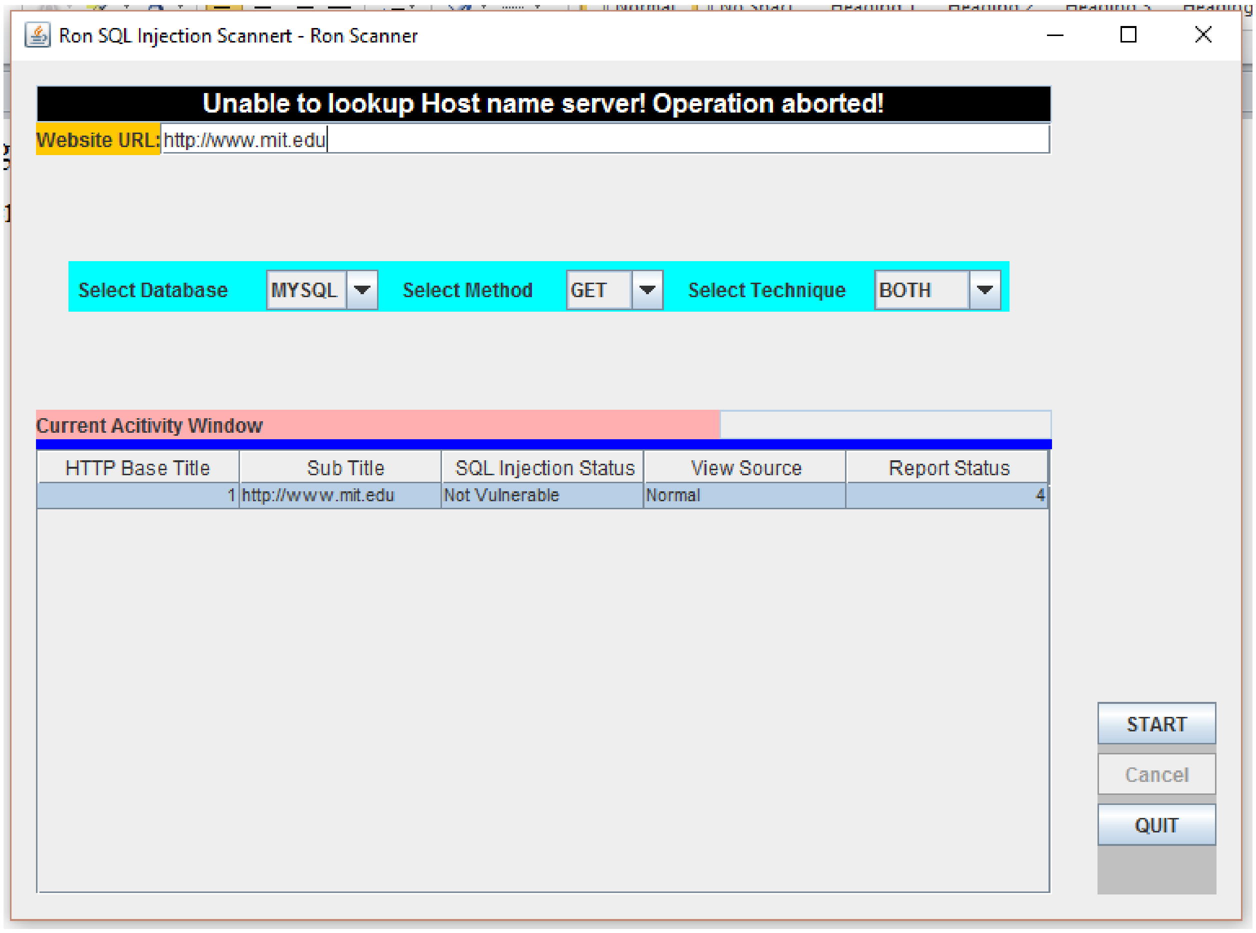
5.3.2 Steps for launching Ron Scanner

The agents are started as shown in the screenshot below. Enter the name of the starting and its class. An agent host environment needs to be installed on the computer to run the system. This is the container where the agents resided.


Insert Start Parameters

Agent Name	
Class Name	
Arguments	
Owner	
Container	Main-Container

Figure 14: Main screen for starting mobile agents


Ron SQL Injection Scannert - Ron Scanner

Unable to lookup Host name server! Operation aborted!

Website URL: http://www.mit.edu

Select Database: MYSQL · Select Method: GET · Select Technique: BOTH

Current Acitivity Window

HTTP Base Title	Sub Title	SQL Injection Status	View Source	Report Status
1	http://www.mit.edu	Not Vulnerable	Normal	4

Figure 15: Main screen for Ron Scanner

5.3.3 Data Analysis

This section comprises of data analysis as stipulated in the research methodology, the presentation of findings in tables as well as summary and interpretation on findings with regard to the vulnerabilities that exist in various web applications.

5.3.4 Data presentation

Data colleted was analysed using descriptive statistical software packages. Descriptive statistics such as frequencies, percentage and mean were used (Cohen et al. n.d.). The research results are presented in a form of bars graphs, pie charts and tables for ease of interpretation . The tools were ranked based to the metrics set.

5.3.5 Limitation and Assumptions

This study was conducted based on the assumption that different web vulnerability detection tools have different capabilities. The study was limited by the choice of the tools to use in detection of SQL injections. Different tools are built with different vulnerabilities in mind and can be used on different platforms. This means that there is a possibility of choosing "tool A" to perform a test, which "tool A " may not well suited to discover.

5.3.6 Testing for Efficiency and Scan time

This is a test to ensure that the website is scanned in lesser time and the system identified SQL injections both blind and normal injections.

5.3.7 System testing for false positive and false negative

This is a test to ensure that the website is scanned and identified SQL injections both blind and normal injections should have low false negative and positive. The results of the research are described in the chapter 6.

CHAPTER SIX

6.0 Evaluation and Results

The multi-agent system was designed with an aim of improving weaknesses that were found with existing web vulnerabilities . A black box approach was adopted with an aim of improving application scanners . The tool used to test and validate the proposed hybrid multi-agent system demonstrated the improved capability of the scanner.

6.1 Simulation implementation

In this section , the simulation implementation is discussed . All the technologies used are listed below. The following items are discussed.

I. Coding – explanation of the source codes used is done and sample of the code is attached as part of the appendix

II. Testing – a series of test were conducted to test and validate hybrid multi- agent system

III. Installation – installation instructions are attached as part of the appendix

IV. Documentation – user manaual is provided as part of the appendix

Implementation tools

V. The following tools were used during the development of the simulation to test the hybrid multi- agent system.

   a. Windows operationg system

   b. Approach – object oriented

   c. Programming language : java

6.2 Choice of the programming language

The platform chosen for the development of the program was java . This choice was arrived at since the researcher is well versed with the language and has a wealth of experience in developing applications using java.

6.3 Development of the simulation.

The simulation was divided into various Agents. Each agent deals with the a certain process of a discovery of SQL web vulnerability. The source code samples are provided as part of the appendix.

6.3.1 SQL injection Discovery

The scanning method described in the hybrid multi-agent system used SQL agent to checks for SQL injection by looking for existance of Boolean, keywords and special characters in the text fields of a web application. This comprised of all the special charaters including (<[&]=‘,+>) it also comprised the Boolean charaters such as , ‘AND’ ‘or ‘’or’ and other keywords, for example, Delete, Truncate and Update among others. SQL injections occurs due to invalidated user input. For instance , when a user logs in using a username and password , ‘’SELECT * from systemusers Where username=’v_username’ and password=’v_password’. SQL injections testing tries this “Select * from users where username =x or 1=1” since one is always equal to 1 this query is true for all the records in the database . If real inputs where a user access a web browser or application , these values will be analyzed against the database, else if a disparity is identified, the results is transmitted to an evaluator analyzing vulnerability and resetting the http occurs.

6.4 Web application scanning Results

The simulation results were evaluated by comparing the performance of the open source scanner and the Ron scanner under the set metrics.

6.4.1 Time taken to scan scan various applications

To maintain realiability of the study the reasearcher administered 15 test on the web vulnerability scanners, the assumption was that all the web vulnerabilities were at similar conditions during the test. The results are as shown in the following tables and figures

Table 2: Vega test results across three web application under the set metrics i.e. scanning time and number of vulnerabilities discovered.

Web Applications	Mean (Seconds)	Vulnerabilities discovered
webgoat	94	3
Vicnum	59	2
genhound	49	2
mean	67.3	2.3
Standard Deviation	23.6	0.58

Table 3: Wapiti test results across three web application under the set metrics i.e. scanning time and number of vulnerabilities discovered.

Web Applications	Mean (Seconds)	Vulnerabilities discovered
webgoat	114	2
Vicnum	71	2
genhound	74	1
mean	86.3	1.7
Standard Deviation	24	0.58

Table 4: ZAP test results across three web application under the set metrics i.e. scanning time and number of vulnerabilities discovered.

Web Applications	Mean (Seconds)	Vulnerabilities discovered
webgoat	124	1
Vicnum	69	1
genhound	91	0
mean	94.7	0.7
Standard Deviation	27.9	0.6

Table 5: Ron scanner test results across three web application under the set metrics i.e. scanning time and number of vulnerabilities discovered.

Web Applications	Mean (Seconds)	Vulnerabilities discovered
webgoat	73	3
Vicnum	42	3
genhound	32	2
mean	49	2.7
Standard Deviation	21.4	0.58

Table 6: Comparing the scanning time taken by Vega, Wapiti, ZAP and Ron Scanner to scan various application.

Scanners	Vega	Wapiti	ZAP	Ron Scanner
Duration in seconds	67.3	86.3	94.7	49
Standard deviation	23.6	24	27.9	21.4
Percentage(%)	22.6	29.0	31.9	16.5

The shorter the duration the more efficient the application

Table 7: Comparing the number of vulnerabilities detected by Vega, Wapiti, ZAP and Ron Scanner to scan various application.

Scanners	Vega	Wapiti	ZAP	Ron Scanner
Mean (Number of vulnerabilities detected)	2.3	1.7	0.7	2.7
Standard deviation	0.58	0.58	0.6	0.58
Percentage(%)	31.1	23	9.4	36.5

The higher the mean the more the numbers of vulnerabilities detected

6.5 Data representation

A visual representation of the tools accurancy

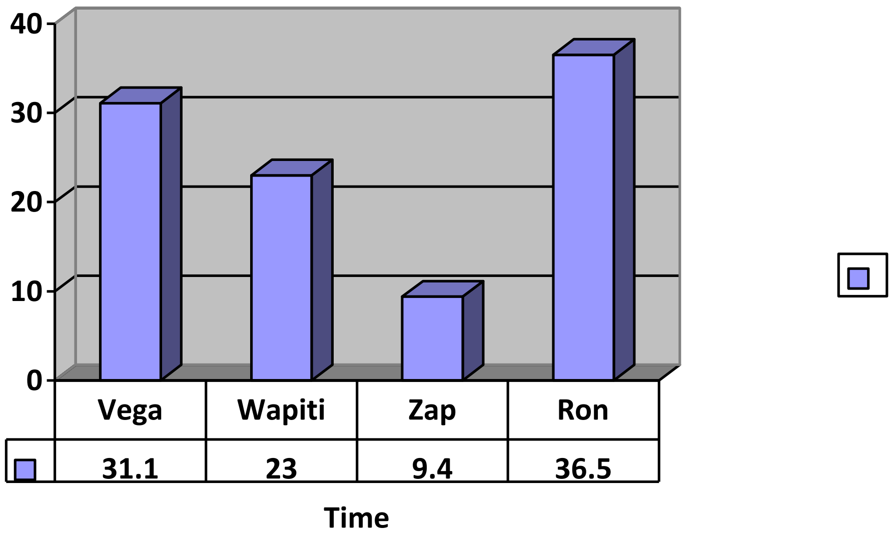


Figure 16: Scanning Tools Accurancy

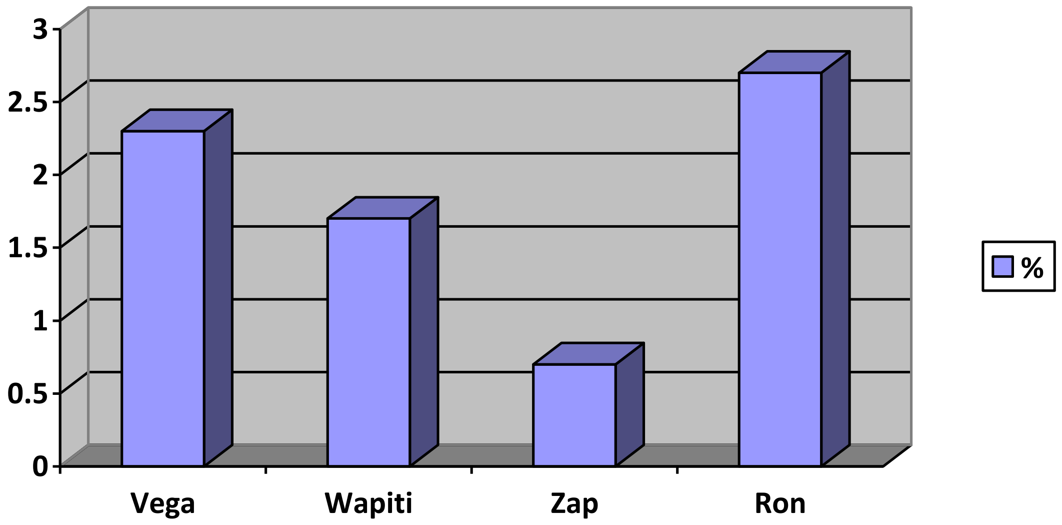
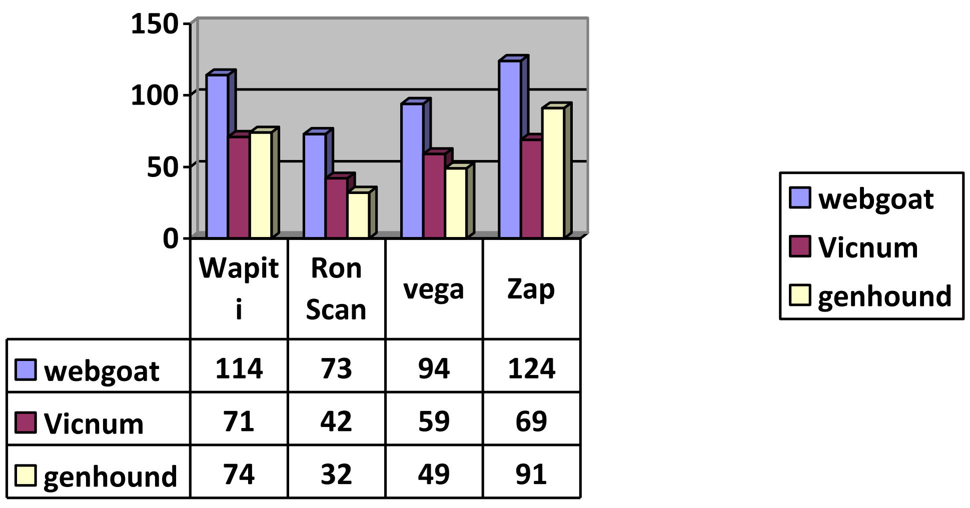


Figure 17: Scanning Tools Consitency



	Wapit i	Ron Scan	vega	Zap
webgoat	114	73	94	124
Vicnum	71	42	59	69
genhound	74	32	49	91

Figure 18: Web Tools Vs Scan Time

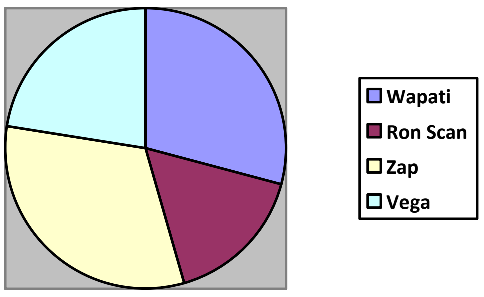


Figure 19: **Time take to scan for web vulnerabilities**

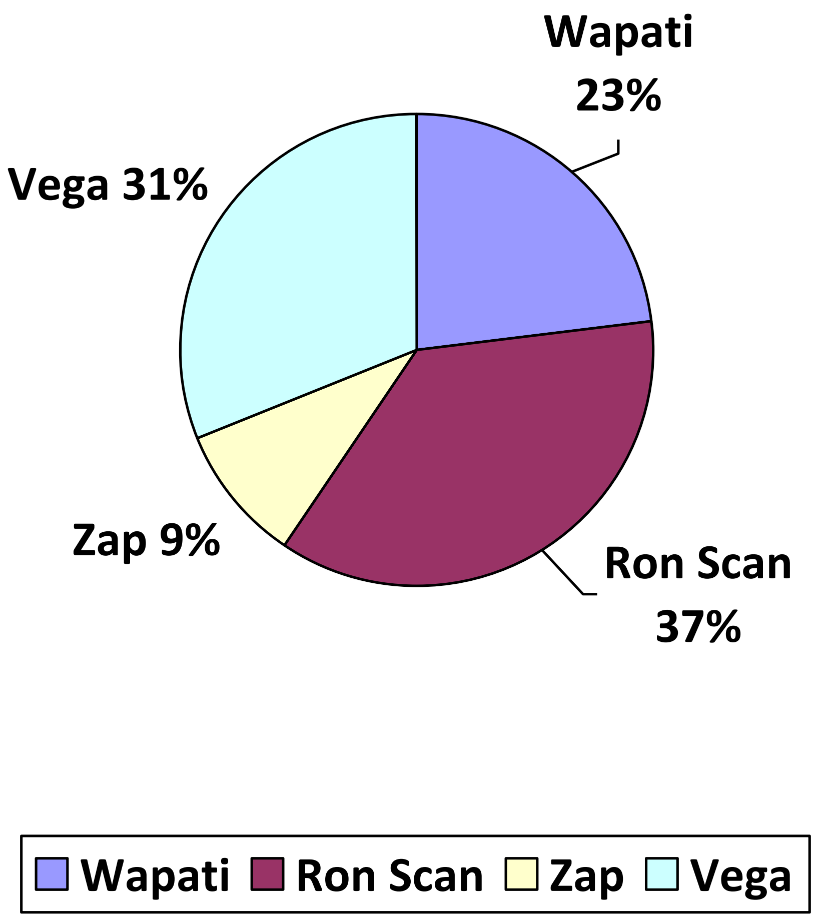


Figure 20: number of vulnerabilities discovered during scan of web application

6.6. Summary results for the tools used

Vega – Performed better in detecting SQL injections but the scanning time was higher compared to Ron scan,it shows better representation of vulnerabilities detected because it categorizes the vulnerabilities as either high , medium or low.

Ron Scanner – Perform better than all the others tools tested. It was able to take the least of time scanning web application and it was also more consistent in its performance results.

Wapiti - This tool can be rated as above average, it was able to take average time in scanning web applications also, it was average in consistent performance results, however it could not discover all SQL vulnerabilities and runs smoothly with minimal errors.

Zap- Performance can be classified as poor . it did not perform well in time taken to scan web vulnerability and its discovery. As indicated by van der loo (2011), the tools fails to excel in detection of web vulnerabilities and it also took the longest scan time.s

6.7. Discussion

web vulnerability scanners comparative study has been done by various reseachers worldwide. Various studies have noted that while various web application tools exists, they differ in operations and vulnerability. However, what remain is that the vulnerabilities. Despite being similar in working principles, tools and web applications, vulnerabities never changes.

From the results, different patterns of behavior were observed in scan time taken and the number of vulnerabilities detected.Ron scan recorded a mean scan time of

16.5 % which is shown to be the lowest as compared to other vulnerabilities.

The results demonstrate that our proposed hybrid multi-agent system is able to perform a scan on a web application faster than other selected vulnerability tools and more accurate in detecting SQL vulnerabilities.The mean scan time is 2.2 sec lower and the mean vulnerabilities detected is 0.4sec higher in our proposed hybrid multi-agent system.

In a study conducted by Doup et al. n.d.(2011) to test vulnerability, using tools such burp scanner and the IBM’s rational app scan noted crawling contemporary applications is a major problem for many WVS. The finding highlighted by Doup et al. n.d.(2011) reflects this research. Thus, a hybrid system should be considered to increase performance for scanning vulnerabilities.

Patole & Kothimbire (2014) showed that open source WVS are weak in identifying vulnerabilities. They also take longer to scan for vulnerabilities. A study by Patole & Kothimbire (2014) is in tandem with this study. This study designed a sophisticated algorithm to mitigate this concern and increase vulnerability identification and detection.

(Zlatkovski & Mileva 2013) analysed various web vulnerabilities scanners and their results showed that time take to take to perform scan varies with different tools and many shown to take longer time, the reaseacher was able to perform SQL injections in less time by fuzzing web applications using hybrid multi-agent system.

Shelly (2011) carried out identical study by employing incursion tools to evaluate effectiveness of available WVS. Both open source and commercial tools were used in the study. These tools included W3AF, Wapiti, N-stalker and W3AF. The tools were subjected to a customized edition of BuggyBank web applications. The apparatus employed were tested for XSS, SQL and other vulnerabilities. Researchers found out that testing WVS in a secure and non-secure applications is a favorable technique for discovering web vulnerabilities . Additionally, the study observed that in discovering non-traditional instances of SQLI , further studies need to be undertaken to increase detection techniques used by these tools. In this study, the researcher used permutation and heuristic in the detection process.

The hybrid multi-agent system is capable of mitigating concerns highlighted by other previous studies. This is achieved through using multiple multi agents during the process of vulnerability and subsequent improvement of the existing vulnerability detection techniques. For example, this study observed that WVS use GET and POST strategies to detect weaknesses in an application. These two methods require sufficient time to scan. Despite the time factor, they provide accurate results.

6.8 Attack Analysis Proficiencies

By analyzing how each of the web scanning tools discovered vulnerabilities , this information provided the reaseacher with an insight on how the tools sampled works and shed more light on the areas which can be considered for future research and enhancements.

In a nutshell most of the tools would do the crawling process using the POST and GET parameters. Once the inputs on the web application have been detected , the scanning tool would attempt some values in the application and analyze the response. Since these tools have been developed using different methods , they use different approaches in their detection mechanism. For instance some of the tools would use numerical values such as 1,2,3,4 while other tools would use letters of alphabet or even leave the field blank . then option used by the tools had an impact on the results produced.

The number of web pages detected by the various tools was not the same. This is simply because the WVS use different crawliong methods . some of the tools used the POST method while others used the GET method.

CHAPTER SEVEN

7.0 CONCLUSION AND RECOMMENDATIONS

From analysis of collected data, results and discussion; the following was concluded and recommended based on the objectives of the study.

a). Develop hybrid multi-agent prototype system using an appropriate technology, which addresses the problem of SQL injection attacks and dynamically tests for the effectiveness of web application vulnerabilities in the development and production environments.

This study developed a hybrid multi-agent based prototype hybrid multiagent system Ron Scanner. Hence, this study has met this objective successfully.

b). Analyse the developed system against set metrics

The metrics used by this study to evaluate our prototype were; time taken to scan web applications , detections accuracy, consistency and realiability.Thus, this study has successfully achieved this objective.

c). To test and validate the effectiveness of the system on selected web applications

This study developed a program to simulate the functionality of the multi-agent system . This program was subjected to the same test and compared its performance with the selected open web scanning tools.

d). To identify various open source vulnerabilitis scanning tools for web applications

In this study, Literature review was done on the existing web vulnerability tools, and three tools were chosen depending on various factors

7.1 Conclusion

The open source tools have the capacity to detect vulnerabilities in the test cases performed. However, none of the tools have the capacity to detect all vulnerabilities. the same conclusion was arrived at by (Mcquade 2014). The research concluded that there is no easier way or any black box vulnerabilities as identified for comparison purposes by WAVSEP.

7.2 Conclusion on specific tools

Wapiti – Produced impressive results , with a fairly easy to interprete the report. As a matter of fact , it reported the highest numbers of SQL injections in the webGoat applications

Vega – provides one of the best reports when comnpared to all other tools used in this study the vulnerabilities detected are classifiesd into four categories namely , high, medium, low and info. See appendix for a sample of vega report. This is categorization is very useful and provide a guide to the user on the vulnerabilities that should be given priority when sealing the weakness. The tool is easy to use and provides a user friendly graphical user interface.

ZED attach proxy – populary known as Zap took long time to scan the applications. However its able to discover some vulnerabilities.

Ron Scan produced good results when compared with other tools which were being used. However, Ron Scan has a higher degree of accuracy in detection compared to others. The performance of this tool is not 100% perfect, however, the tool significantly detect several weaknesses unlike the others.

7.3 Conclusion about hybrid multi-agent system Ron Scanner

The hybrid system presented in this study is extensive. This is in regard to execution and detection method against vulnerabilities found in web application. the hybrid multi agent system executes faster and scan web application for vulnerabilities. It produces a report for an evaluator to identify the vulnerabilities that have been discovered.

However, since the hybrid multi-agents designed in this study did achieve 100% when scanning for existing vulnerabilities, a robust crawing algorithm component should be increased. This will enable “deep” crawing to identify vulnerabilities. Additionally, the result shows that the hybrid system designed should be optimized to shorten scanning time. Studies aimed at creating and implementing a sophisticated multi agents should be pursued to increase capacity of detecting more vulnerabilities.

7.4 Suggestion for further research

It has been proved that agents can be used to do the work for us by specifying to them the terms of reference, otherwise known as ontologies . It is recommeed that in future more vulnerabilities will be solved by multi- agents and therefore further research on the extending and refining the use of agents should be pursued to verify their usefulness since the project has concentrated on the two most serious vulnerabilities according to OWASP top 10 , 2013.

REFERENCES

Bhat, M.N., Veeranjaneyulu, N. & Raghunath, A., 2013. International Journal of Advanced Research in Computer Science and Software Engineering A Hybrid Approach for handling SQLI Vulnerabilities in Web Applications. , 3(12), pp.604–609.

Cohen, L. et al., *Research Methods in Education*,

Commons, S., 2012. *Social Science Research: Principles, Methods, and Practices*,

Dam, K.H. & Winikoff, M., Comparing Agent-Oriented Methodologies.

Doup, A., Cova, M. & Vigna, G., Why Johnny Can ' t Pentest : An Analysis of Black-box Web Vulnerability Scanners.

H, K.C. & Kala, V., 2014. Report on Vulnerabilities in Web Applications. , 4(9).

Jaiswal, A., 2014. Security Testing of Web Applications : Issues and Challenges. , 88(3), pp.26–32.

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

Kumar Singh, A. & Roy, S., 2012. A network based vulnerability scanner for detecting SQLI attacks in web applications. *2012 1st International Conference on Recent Advances in Information Technology, RAIT-2012*, pp.585–590.

Kumar, R., 2015. A Comparative Study and Analysis of Web Service Testing Tools. , 4(1), pp.433–442.

Mcquade, K., 2014. Open Source Web Vulnerability Scanners : The Cost Effective Choice ? , pp.1–13.

Mirjalili, M., Nowroozi, A. & Alidoosti, M., 2014. A survey on web penetration test. , (November).

Muchai, C. et al., 2015. Achieving Enterprise Cyber Resilience Through Situational Analysis. *Kenya Cyber Security Report 2015*. Available at: http://serianu.com/downloads/KenyaCyberSecurityReport2015.pdf.

Owasp, 2013. OWASP Top 10 - 2013. *OWASP Top 10*, p.22. Available at: http://owasptop10.googlecode.com/files/OWASP Top 10 - 2013.pdf.

Patole, M.S. & Kothimbire, S.D., 2014. A Review on Web Security Mechanism Performance Evaluation Using Vulnerability and Attack Injection. , 3(11), pp.2585–2588.

Petukhov, A. & Kozlov, D., Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing.

Phalguna Rao, K., BSasankar, A. & Chavan, V., 2013. Analysis of Detection and Prevention Techniques Against SQL Injection Vulnerabilities. , 4, pp.50–55.

Rawat, R., Singh Dangi, C. & Patil, J., 2011. Safe Guard Anomalies against SQL Injection Attacks. *International Journal of Computer Applications*, 22(2), pp.11–14. Available at: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=058F712989513E110B872D4117C676F7?doi=10.1.1.206.2986&rep=rep1&type=pdf\nhttp://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.206.2986\nhttp://www.ijcaonline.org/archives/volume22/number2/2558-351.

Shema, M., *Compliments of,*

Stuttard, D. & Pinto, M., *No Title,*

Thiyagarajan, A. et al., 2015. Methods for Detection and Prevention of Sql Attacks in Analysis of Web Field Data. , 4(4), pp.657–662.

Yu, Y. et al., 2011. Analysis and Suggestions for the Security of Web Applications. , pp.236–240.

Zlatkovski, D. & Mileva, A., 2013. EVALUATION AND TESTING OF SEVERAL FREE / OPEN SOURCE WEB. , (Ciit), pp.221–224.

LIST OF APPENDIX

Appendix 3: Sample code

```
public static void processPage(String URL) throws SQLException, IOException{
		//check if the given URL is already in database
		String sql = "select * from Record where URL = '"+URL+"'";
		ResultSet rs = db.runSql(sql);
		if(rs.next()){


		}else{
			//store the URL to database to avoid parsing again
			sql = "INSERT INTO  `Crawler`.`Record` " + "(`URL`) VALUES " +
"(?);";
			PreparedStatement stmt = db.conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS);
			stmt.setString(1, URL);
			stmt.execute();


			//get useful information
			Document doc = Jsoup.connect("http://www.mit.edu/").get();
			if(doc.text().contains("research")){
        System.out.println(URL);

    }

    //get all links and recursively call the processPage method

    Elements questions = doc.select("a[href]");

    for(Element link: questions){

        if(link.attr("href").contains("mit.edu"))

            processPage(link.attr("abs:href"));
```

Appendix 6: How to run the system

1. Install Wampserver 3.0

2. Install Java Runtime Environment

3. Install the Agent Host application in every computer in the network

4. Install Agent Client application in the computer where the agents will be launched

5. Start the agent Host application in all the computers in the network

6. Start the agent Client application and use it to send agents to the computers on the network.