The Islamic University of Gaza
Deanery of Graduate Studies
Faculty of Engineering
Computer Engineering Department

Modern Approach for WEB Applications Vulnerability Analysis

Rami M. F. Jnena

120090823

Supervisor

Prof. Mohammad A. Mikki

A thesis proposal to partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering

(2013) 1434H

ACKNOWLEDGMENT

It is a pleasure to thank the many people who made this thesis possible.

This work would not have been possible without the support from my supervisor Prof. Dr. Mohammad Mikki under whose guidance, I chose this topic. And I would like to gratefully acknowledge his supervision that has been abundantly helpful and has assisted me in numerous ways. I specially thank him for his infinite patience. The discussions I had with him were invaluable.

My thankfulness also goes to my mother and my father.

I would also like to thank my wife who encouraged me strongly to complete this thesis and study in spite of the great burdens that I went through.

I want to thank my brothers for their support and patience during my studies and thesis.

I also want to thank my sisters for their support during my studies and thesis. Their appreciation towards the education has pushed me forward in my studies.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The numbers of security vulnerabilities that are being found today are much higher in applications than in operating systems. This means that the attacks aimed at web applications are exploiting vulnerabilities at the application level and not at the transport or network level like common attacks from the past. At the same time, quantity and impact of security vulnerabilities in such applications has grown as well. Many transactions are performed online with various kinds of web applications. Almost in all of them user is authenticated before providing access to backend database for storing all the information. A well-designed injection can provide access to malicious or unauthorized users and mostly achieved through SQL injection and Cross-site scripting (XSS).

In this thesis we are providing a vulnerability scanning and analyzing tool of various kinds of SQL injection and Cross Site Scripting (XSS) attacks. Our approach can be used with any web application not only the known ones. As well as it supports the most famous Database management servers, namely MS SQL Server, Oracle, and MySQL.

We validate the proposed vulnerability scanner by developing experiments to measure its performance. We used some performance metrics to measure the performance of the scanner which include accuracy, false positive rate, and false negative rate. We also compare the performance results of it with performance of similar tools in the literature.

ABSTRACT ARABIC

عدد الثغرات الأمنية التي يتم العثور عليها اليوم هي أعلى بكثير مما كانت عليه في تطبيقات نظم التشغيل. هذا يعني أن الهجمات التي تستهدف تطبيقات الويب واستغلال نقاط الضعف على مستوى التطبيق وليس على مستوى النقل أو شبكات الحاسوب مثل الهجمات المشتركة أصبحت من الماضي. وفي الوقت نفسه، نمت كمية وتأثير الثغرات الأمنية في هذه التطبيقات بشكل كبير. وحيث انه يتم تنفيذ العديد من المعاملات عبر الإنترنت مع أنواع مختلفة من تطبيقات الويب. علما بأنه يتم مصادقة المستخدم قبل توفير إمكانية الوصول إلى قاعدة البيانات فانه من خلال نظام حقن ثغرات مصمم تصميما جيدا سيوفر الولوج لهذه الانظمة الحصول على معلومات مهمة، ان عملية الحقن الاساسية تتم من خلال تقنيتي SQL Injection وCross Site Scripting .

في هذا البحث، قمنا بتصميم وتطوير نظام مسح وكشف للثغرات الامنية الخاصة بانظمة الويب، يستطيع النظام اكتشاف الثغرات من نوعي SQL Injection وCross Site Scripting . كما وانه غير مقيد بنوع تطبيق الويب وانما يمكن استخدامه لكافة انظمة الويب. مع الدعم الكامل لاشهر انظمة قواعد البيانات MS SQL Server و Oracle و MySQL .

تم عمل فحص وتقييم للفكرة المطروحة من خلال تجارب شاملة لقياس الاداء. تم استخدام مقاييس الاداء الاساسية لهذا النوع من الانظمة وهي، مقياس الدقة، مقياس معدل الايجابية الكاذبة، و مقياس معدل السلبية الكاذبة. كما وتم مقارنة نتائج الاداء مع ادوات وانظمة مماثلة.

Chapter1
INTRODUCTION

1.1.Motivation

A "computer system" is more than hardware and software; it includes the policies, procedures, and organization under which that hardware and software is used. Security holes can arise from many areas or combination of these them. This leads no sense to restrict the study of vulnerabilities to hardware and software problems [67].

When attacker breaks into a computing system, he takes advantage of lapses in procedures, technology, or management (or some combination of those factors), permitting unauthorized access or actions. The precise failure of the controls is termed a vulnerability or security flaw; mistreatment that failure to violate the security policy is termed exploiting the vulnerability. One who attempts to exploit the vulnerability is called an attacker [67].

Another more general definition from [68] defines Vulnerability analysis as “the act of determining which security holes and vulnerabilities may be applicable to the target network”.

Vulnerability analysis, also known as vulnerability assessment [69], “is a process that defines, identifies, and classifies the security holes (vulnerabilities) in a computer, network, or communications infrastructure”. In addition, vulnerability analysis can forecast the effectiveness of proposed countermeasures and evaluate their actual effectiveness after they are put into use.

Vulnerability analysis consists of several steps:

- Define and classify target system.
- Assign relative levels of importance to the target system resources.
- Identify potential threats to each resource.
- Develop or setup a method to deal with the most serious potential problems.
- Define and implement procedures to minimize the consequences if the attack for the target system resource.

In order to develop reliable and robust web applications, we have to use vulnerability metrics that let us monitor, analyze, and quantify application behavior under a range of faults and attacks. In this research we will present a scanning tool for analyzing web applications vulnerability in real time. This scanner lets us quantify how attacks and faults impact network performance and services, discover attack points, and examine how critical the web application components behave during an attack or system fault.

1.2. Attack analysis

Most network attackers overcome the target system with a brute forced traffic to consume all system resources (such as CPU cycles, memory, network bandwidth, and packet buffers). These attacks degrade service and can eventually lead to a complete shutdown.

There are two common types of attacks [72]*:*

- *Server attacks:* these attacks include TCP SYN, Smurf IP, ICMP flood, and Ping of Death attacks. For example, the attacker may make brute force requests to a victim server with spoofed source IP addresses. Due to TCP/IP protocol stack vulnerabilities, the victim server cannot complete the connection requests and wastes all of its system resources. This will result denial of service on the target attacked system.
- *Routing attacks*: the main strategy in routing attacks is distributed denial-of-service (DDoS) attacks which focuses on routers devices. When a router is compromised, it will forward traffic according to the attackers' intent. Similar to server attacks, the attackers aim to consume all router resources, forcing the router to drop all incoming packets, thus negatively affecting network performance and behavior.

Vulnerability Analysis researches in networks and internet still in its beginning; this gives researchers much room for improvements. Several tools, which are based on modeling network specifications, fault trees, graph models, and performance models, works on vulnerability analysis by checking logs of systems and monitor performance metrics [70, and 71].

There are three common types of vulnerability analysis techniques [72]:

- Network specifications survivability analysis. This approach injects fault and intrusion events into a given network specification, and then visualizes the effects in scenario graphs. This is done by using model checking, Bayesian analysis, and probabilistic analysis, which provides a multifaceted network view of a desired service.
- Attack trees. This approach determines which attacks are most feasible and therefore most likely in a given environment, and quantifies vulnerability by mapping known attack scenarios into trees. Attack trees assume that all vulnerability paths are known and can be defined as possible or impossible. This can change as new attacks are discovered, however, to sudden render a previously impossible node possible.
- Graph-based network-vulnerability analysis. This approach analyzes risks to specific network assets and examines the possible consequences of a successful attack. As input, the analysis system requires a database of common attacks (broken into atomic steps), specific network configuration and topology information, and an attacker profile. Nodes identify an attack stage, such as the machine class the attacker has accessed and the user privilege level he or she was compromised. Using graph methods lets you identify the attack paths with the highest probability of success.

1.3. Penetration Testing

A penetration test is “an authorized attempt to violate specific constraints stated in the form of a security or integrity policy”. This method implies a metric for determining whether the study has succeeded. In addition, it provides a framework in which to examine those aspects of procedural, operational, and technological security mechanisms relevant to protecting the particular aspect of system security in question.

Another study does not have a specific target; instead, the goal is to find some number of vulnerabilities or to find vulnerabilities within a set period of time. The strength of such a test depends on the proper interpretation of results. Briefly, if the vulnerabilities are categorized and studied, and if conclusions are drawn as to the nature of the flaws, then the analysts can draw conclusions about the care taken in the design and implementation. But a simple list of vulnerabilities, although helpful in closing those specific holes, contributes far less to the security of a system.

In practice, penetration testing study is affected by many constraints; resources and time are the most constraints that affect it. If these constraints arise as aspects of policy, they improve the test because they make it more realistic.

1.4. Layers of Penetration Testing

Penetration testing is designed to characterize the effectiveness of security mechanisms and controls to attackers. Attacker's point of view conducts penetration test studies, and the environment in which the tests are conducted is that in which a putative attacker would function. Different attackers, however, have different environments; for example, insiders have access to the system, whereas outsiders need to acquire that access. There are two layers for a penetration testing study, external attacker with access to the system and internal attacker with access to the system.

1. ***External attacker with access to the system:*** in this layer, in order to launch the attack the testers/attacker have access to the system and can proceed to log in or to invoke network services available to all hosts on the network. This layer requires an access account from which the testers can achieve their goal or using a network service that can give them access to the system. Common forms of attack at this stage are guessing passwords, looking for unprotected accounts, and attacking network servers. To provide the desired access an implementation of flaws in servers are required.
2. ***Internal attacker with access to the system:*** in this layer, the testers have an account on the system and can act as authorized users of the system. The test typically involves gaining unauthorized privileges or information and, from that, reaching the goal. At this stage, the testers acquire (or have) a good knowledge of

the target system, its design, and its operation. Attacks are developed on the basis of this knowledge and access.

In some cases, information about specific layers is irrelevant and that layer can be skipped. For example, penetration tests during design and development skip layer 1 because that layer analyzes site security. A penetration test of a system with a guest account will usually skip layer 2 because users already have access to the system. Ultimately, the testers must decide which layers are appropriate.

1.5. Methodology at Each Layer in Penetration Testing Methodology

The penetration testing methodology springs from the Flaw Hypothesis Methodology. The usefulness of a penetration study comes from the documentation and conclusions drawn from the study and not from the success or failure of the attempted penetration. Such a conclusion can only be drawn once the study is complete and when the study shows poor design, poor implementation, or poor procedural and management controls. Also important is the degree of penetration. If an attack obtains information about one user's data, it may be deemed less successful than one that obtains system privileges because the latter attack can compromise many user accounts and damage the integrity of the system.

1.6.Flaw Hypothesis Methodology

The Flaw Hypothesis Methodology was developed at System Development Corporation and provides a framework for penetration studies [67]. It consists of five steps, information gathering, flaw hypothesis, flaw testing, flaw generalization, and flaw elimination.

1. ***Information gathering.*** In this step, the testers become familiar with the system's functioning. They examine the system's design, its implementation, its operating procedures, and its use. The testers become as familiar with the system as possible.

2. ***Flaw hypothesis.*** Drawing on the knowledge gained in the first step and on knowledge of vulnerabilities in other systems, the testers hypothesize flaws of the system under study.
3. ***Flaw testing.*** The testers test their hypothesized flaws. If a flaw does not exist (or cannot be exploited), the testers go back to step 2. If the flaw is exploited, they proceed to the next step.
4. ***Flaw generalization.*** Once a flaw has been successfully exploited, the testers attempt to generalize the vulnerability and find others similar to it. They feed their new understanding (or new hypothesis) back into step 2 and iterate until the test is concluded.
5. ***Flaw elimination.*** The testers suggest ways to eliminate the flaw or to use procedural controls to ameliorate it.

1.7. Vulnerability Classification

Security flaws from various perspectives are described by vulnerability classification frameworks. Some frameworks describe vulnerabilities by classifying the techniques used to exploit them. Other frameworks characterize vulnerabilities in terms of the software and hardware components and interfaces that make up the vulnerability. And others classify vulnerabilities by their nature; this is done by discovering techniques for finding previously unknown vulnerabilities.

Vulnerability analysis goal is to develop methodologies that provide the following abilities:

1. Specify, Design, and implement a computer system without vulnerabilities.
2. Analyze a computer system to detect vulnerabilities.
3. Address any vulnerability introduced during the operation of the computer system.
4. Detect attempted exploitations of vulnerabilities.

1.8. Aim of this Thesis

The main goal of this master thesis is to present a new analyzing tool for main two web applications vulnerabilities, which are mainly SQL Injection and Cross Site Scripting (XSS). To achieve this goal, a dynamically generate test requests that are applied specifically to a given web application will be applied by the analysis tool. By doing this analysis, our scanning will be apple to detect vulnerabilities of any web application regardless if it's for known web application or custom web application. The analysis tool will conduct two tests; these t101ests will identify the common web applications vulnerabilities that are SQL Injections and Cross Site Scripting (XSS). These tests will be applied on web applications input parameters so the tests will be parameter-based tests.

1.9.Methods Used

To accomplish the proposed solution, the following methods have been used in sequence:

- Study the basic principles of web applications vulnerability analysis.
- Study and learn the main scripting languages used for implementation of code.
- Review the existing techniques of vulnerability analysis.
- Identify the major and common vulnerabilities on web applications and study their mechanisms.
- Design a set of related analysis mechanisms and algorithms.
- Demonstrate the validity of the proposed solution to detect vulnerabilities on different types of web applications.

1.10. Our approach

The new scanning tool has been implemented in Perl scripting language under Linux environment. The evaluation method used is an automatic exploiting for the detected vulnerabilities which will verify the existence of vulnerability and minimize the false positives that may exist by the scanning tool.

1.11. Organization of this Thesis

The rest of the thesis is organized as follows; Chapter 2 presents a complete review of vulnerability analysis with statistics about known vulnerabilities of web applications and their impact in web developments. A brief review of well-known scanning and analyzing tools of vulnerabilities with detailed description of most modern tools, as well as, categories of lately proposed solutions are discussed in chapter 3. Chapter 4 presents a formal description of our design of analysis tool with required parameters, assumptions, and all prerequisite mechanisms needed to make this comprehensive work. Validation and evaluation results are provided in chapter 5. The report ends with conclusion in chapter 6 which summarizes this thesis and gives some hints for future work on this research subject.

Chapter 2
VULNERABILITY ANALYSIS

2.1 Introduction

In computer security, vulnerability is a weakness which allows an attacker to reduce a system's information assurance.

Vulnerability is the intersection of three elements: a system susceptibility or flaw, attacker access to the flaw, and attacker capability to exploit the flaw [46]. In order to exploit vulnerabilities, the attacker must have at least one applicable tool or technique that can connect to a system weakness security hole.

According to NIST SP 800-37, “vulnerability analysis and assessment is an important element of each required activity in the NIST Risk Management Framework (RMF)”. This RMF comprises six steps, into each of which vulnerability analysis and assessment is to be integrated [45]:

- Information System Categorization.
- Security Controls Selection.
- Security Controls Implementation.
- Security Controls Assessments.
- Information Systems Authorization.
- Security Controls Monitoring.

Integration is done by the vulnerability assessment tools, by automating the detection, identification, measurement, and understanding of vulnerabilities found in ICT components at various levels of a target ICT system or infrastructure. Vulnerability is an attribute or characteristic of a component that can be exploited by either an external or internal agent (hacker or malicious insider) to violate a security policy of (narrow definition) or cause a deleterious result in (broad definition) either the component itself, and/or the system or infrastructure of which it is apart. Such “deleterious results” include unauthorized privilege escalations or data/resource accesses, sensitive data disclosures or privacy violations, malicious code insertions, denials of service, etc [45].

Such tools are often referred to as vulnerability scanners, because their means of vulnerability detection is to scan targets (usually network services and nodes, and the operating systems, databases, and/or Web applications residing on those nodes) in an attempt to detect known, and in some cases also unknown, vulnerabilities [45].

Improving the scanning techniques of Web Application scanners will allow them to achieve better performance and, therefore, increase their credibility. However, in order to understand and improve web application scanners, the common vulnerabilities that they aim to detect must be understood first. This chapter is organized as follows: how are vulnerability assessments tools work will be discussed in Section 2.1, some of the specific vulnerabilities that web application scanners attempt to probe for will be discussed in Section 2.2, several of the most popular and researched web application scanners will be discussed in Section 2.3.

2.2 How Vulnerability Assessment Tools Work

Vulnerability assessment tools generally work by attempting to automate the steps often employed to exploit vulnerabilities: they begin by performing a "footprint" analysis to determine what network services and/or software programs (including versions and patch levels) run on the target. The tools then attempt to find indicators (patterns, attributes) of, or to exploit vulnerabilities known to exist, in the detected services / software versions, and to report the findings that result. Caution must be taken when running exploit code against "live" (operational) targets, because damaging results may occur. For example, targeting a live Web application with a "drop tables" Standard Query Language (SQL) injection probe could result in actual data loss. For this reason, some vulnerability assessment tools are (or are claimed to be) entirely passive. Passive scans, in which no data is injected by the tool into the target, do nothing but read and collect data. In some cases, such tools use vulnerability signatures, i.e., patterns or attributes associated with the likely presence of a known vulnerability, such as lack of a certain patch for mitigating that vulnerability in a given target. Wholly passive tools are limited in usefulness (compared with tools that are not wholly passive) because they can only surmise the presence of vulnerabilities based on circumstantial evidence, rather than testing directly for those vulnerabilities.

Most vulnerability assessment tools implement at least some intrusive “scanning” techniques that involve locating a likely vulnerability (often through passive scanning), then injecting either random data or simulated attack data into the “interface” created or exposed by that vulnerability, as described above, then observing what results. Active scanning is a technique traditionally associated with penetration testing, and like passive scanning, is of limited utility when performed on its own, as all the injected exploits would be “blind”, i.e., they would be launched at the target without knowing its specific details or susceptibility to the exploits. For this reason, the majority of vulnerability assessment tools combine both passive and active scanning; the passive scanning is used to discover the vulnerabilities that the target is most likely to contain, and the active scanning is used to verify that those vulnerabilities are, in fact, both present and exposed as well as exploitable. Determining that vulnerabilities are exploitable increases the accuracy of the assessment tool by eliminating the false positives, i.e., the instances in which the scanner detects a pattern or attribute indicative of a likely vulnerability that which, upon analysis, proves to be either (1) not present, (2) not exposed, or (3) not exploitable. It is the combination of passive and active scanning, together with increased automation, which has rendered automated penetration testing suites more widely useful in vulnerability assessment.

Most vulnerability assessment tools are capable of scanning a number of network nodes, including networking and networked devices (switches, routers, firewalls, printers, etc.), as well as server, desktop, and portable computers. The vulnerabilities that are identified by these tools may be the result of programming flaws (e.g., vulnerabilities to buffer overflows, SQL injections, cross site scripting [XSS], etc.), or implementation flaws and misconfigurations. A smaller subset of tools also provides enough information to enable the user to discover design and even architecture flaws.

The reason for “specialization” of vulnerability assessment tools, e.g., network scanners, host scanners, database scanners, Web application scanners, is that to be effective, the tool needs to have a detailed knowledge of the targets it will scan. Web application and database vulnerability scanners look for vulnerabilities that are traditionally ignored by network- or host-level vulnerability scanners [45].

Even custom-developed Web application and/or database application often use common middleware (e.g., a specific supplier’s Web server, such as Microsoft® Internet Information Server [IIS] or Apache®), backends (e.g., Oracle® or PostgreSQL), and technologies (e.g., JavaScript®, SQL) that are known or considered likely to harbor certain types of vulnerabilities that cannot be identified via signature based methods used by network- and host-based vulnerability analysis tools. Instead, Web Application scanners and database scanners directly analyze the target Web application or database, and attempt to perform common attacks against it, such as SQL injections, XSS, least privilege violations, etc [45].

2.3 Web Vulnerability attack Threats

The numbers of security vulnerabilities that are being found today are much higher in applications than in operating systems. This means that the attacks aimed at web applications are exploiting vulnerabilities at the application level and not at the transport or network level like common attacks from the past[48].

The Open Web Application Security Project (OWASP) has put together what is considered the definitive standard list of top threats to Web applications. It is called “The OWASP Top 10 Project” and it represents a general consensus on the major areas of threat by category. The Top 10 threats [Figure 2.1] as they exist currently are as follows:

1. Injection:

Injection flaws, such as SQL, 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 unauthorized data.

2. Cross Site Scripting (XSS):

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. 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.

3. Broken Authentication and Session Management:

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities.

4. Insecure Direct Object References:

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

5. Cross-Site Request Forgery (CSRF):

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

6. Security Misconfiguration:

Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults. This includes keeping all software up to date, including all code libraries used by the application.

7. Insecure Cryptographic Storage:

Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes.

8. Failure to Restrict URL Access:

Many web applications check URL access rights before rendering protected links and buttons. However, applications need to perform similar access control checks each time these pages are accessed, or attackers will be able to forge URLs to access these hidden pages anyway.

9. Insufficient Transport Layer Protection:

Applications frequently fail to authenticate, encrypt, and protect the confidentiality and integrity of sensitive network traffic. When they do, they sometimes support weak algorithms, use expired or invalid certificates, or do not use them correctly.

10. Invalidated Redirects and Forwards:

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

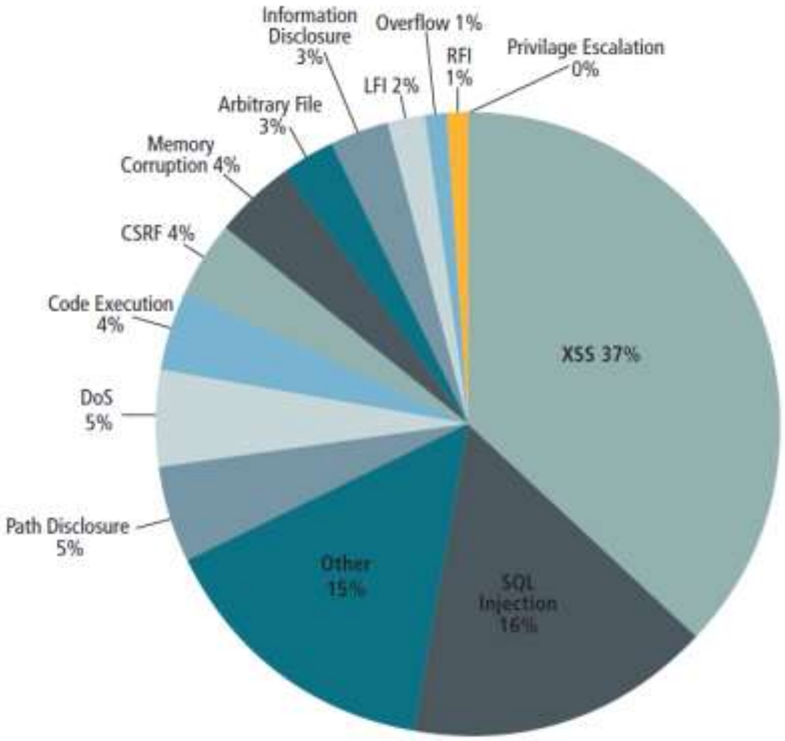


Figure 2.1: Top 10 Threats

This list only includes the ten most critical web application risks that exist today, but other important vulnerabilities exist, including buffer overflow exploits and malicious file execution (included on the previous version of the OWASP Top 10 for 2007 [47]). Because the total number of web application vulnerabilities that exist is extremely large, only the most relevant vulnerabilities are implemented and analyzed in this research. The web application vulnerabilities that are most relevant to this research include SQL injection, and cross-site scripting (XSS). Information about these vulnerabilities has been taken from [48].

2.3.1 SQL Injection

This section will discuss a brief overview of different kinds of SQL injection attacks and their defenses. [48] Describes the basic definition and fundamental information regarding SQL injection techniques.

Attacks

SQL injection occurs when malicious input is passed into a database interpreter without being properly validated or encoded. In this type of attack the client is attacking the web server database. The input that the attacker passes into the interpreter is crafted to be a legitimate SQL statement, but instead of returning the data that the application’s developer intended, the interpreter now returns the data requested by the attacker. This type of attack is severe because not only can it expose all sensitive user and business related data, but it could even go as far as executing operating system commands or giving an attacker complete control of a web application. An example of a valid SQL query which displays information for the user “Rami” is:

SELECT info FROM users WHERE username = 'Rami';

An attacker could use the malicious user name “' OR 1=1 –” to cause the interpreter to display all of the user information data in the database. The corresponding SQL query would be:

SELECT info FROM users WHERE username = '' OR 1=1 –

This is one of the simplest types of SQL injection, but works because the leading single quote causes the query to break out of the single quote delimited data. Therefore the always true “OR 1=1” is appended to the query, and thus displays all of the user information data in the database. The double dashes “–” at the end of the query cause all of the text that would follow it to be commented out, because “–” is the comment symbol in this SQL language.

Adding the comment symbol is necessary in this attack because it nullifies the rest of the syntax that the web application would normally append to the end of database query to complete the operation. Therefore, the only query that is being executed is the attacker’s injected sequence, and not the web applications expected query.

Even more dangerous attacks are possible against certain SQL versions and databases as well. An example of this would be if an attacker took advantage of a web application that implements both regular and administrator users, and therefore normally logs in users with default roles, but could also log in a user with administrator roles. If the administrator user has advanced functionality and has the ability to access all of the web application’s data, then the web application can be completely compromised if an attacker takes control of the administrator account. An SQL injection attack could accomplish this if a web application uses email addresses as user names and associates each user name in the database as either a regular or an administrator user. In this example the attacker will exploit a generic “Change Mailing Address” field on a web page and associate an email address of their choosing to the administrator account. The attacker would enter the following in the “Change Mailing Address” field on the web page:

```
'; UPDATE users SET username = 'attacker@email.com' WHERE

username LIKE '%admin%'; –
```

The semi-colon “;” will end the first query and allow for the attacker’s query to be executed. This query will cause the email address that the attacker entered to replace the email address that matches the pattern most like “admin”. All that is necessary to perform this attack is for the attacker to “guess and check” until they know that the table holding the accounts is in fact “users”, and that the field holding the user names is in fact “username”. After this, the user name most closely matching “admin” will be replaced with the attacker’s email address, but will continue to have administrative capabilities. Now that the attacker has replaced the administrator’s email address with his own, he can click the “Forgot

Password” button that most user-based web applications provide, and have the administrator’s password sent to him in the convenient “Password Reminder” email.

The previously mentioned attack is not always easy to execute because it is not trivial to find out the name of the table and column being used in the SQL database. This challenge is overcome by using two other types of SQL injection: blind SQL injection and error-based SQL injection. Blind SQL injection uses a series of true and false questions to take advantage of the predictability of the WHERE condition in SQL, since 1=1 will always return true. Therefore, if a record is returned when using blind SQL injection, the attacker’s injected condition must have been true. Error-based SQL injection is a specific type of SQL injection that uses SQL error messages to determine the structure of the database. SQL injection statements are crafted by the attacker in a way such that the attacker can use the error responses to systematically unveil table names, column names, column data types, and even specific data entries [48].

Defenses

The best way to prevent SQL injection is to have all interpreters separate untrusted data from database queries and to only accept expected input [41, 50]. In order to achieve this, all data originating from a client should have special characters escaped or sanitized into a valid format, should use an API which avoids the use of an interpreter entirely, or should use prepared statements and parameterized interfaces. The “mysqli->prepare” mechanism will create prepared statements for MySQL in PHP, and the following code sequence will escape all special characters:

```
if ( !get_magic_quotes_gpc() ) {

$safe_string = mysql_real_escape_string( $original_string );

}
```

Also, to protect against blind SQL injection and error message based SQL injection, SQL error reporting should be disabled in conjunction with the other safety measures mentioned above. In order to disable error reporting for MySQL the "@" character should precede commands to suppress on-screen error reporting. These defensive measures should be implemented to mitigate SQL injection attacks.

2.3.2 Cross Site Scripting

Three of the main types of cross-site scripting (XSS) attacks, as well as some defensive techniques to protect against them, will be reviewed in this section.

Attacks

XSS occurs when a web application includes malicious code in a web page that is sent to a client's browser without proper content validation. In this type of attack the web page server is attacking the client machine. When the web page is viewed by the client it will execute the malicious script that the attacker embedded into the web page. XSS is the most prevalent web application security flaw [47] due in a large part to its simplicity and resulting severity. Some of the attacks that this type of vulnerability can result in are the hijacking of a user's session, the defacement of websites, the insertion of hostile content, and the redirection of users' requests.

Reflective XSS is a type of XSS attack that can occur when a victim follows a URL which contains malicious scripting that is executed when the web page is rendered. This is commonly done by sending victims legitimate looking e-mail messages that contain malicious script in the message's URL. Once the HTTP request from the URL is processed, the HTML content is received and displayed in the victim's browser, thus executing the malicious script. An example of a URL containing a reflective XSS attack that would execute some type of malicious script included the function "malicious()" would be:

http://www.targetsite.com/display.php?user=<script>malicious()</script>

Stored XSS is another type of XSS attack. This type of attack occurs when the malicious script is uploaded into the database back end of a server without input validation, and is later retrieved by the web application to be embedded into a web page. This causes every user who visits the infected web page to execute the malicious script in his or her browser. An example of where this vulnerability can be found is a web application that uses a comment section to allow users to view and leave feedback about a product. If an attacker were to leave a comment which included malicious script, the script would be stored as a comment for that product in the database, and then executed every time a user clicks to view the page holding the comments for that product. An example of this type of attack is a script crafted to steal a user's cookie and save it in a remote site for exploitation at a later time so that they can perform actions as if they were the victim (such as bank transactions, e-mail correspondence, etc...). The following script would execute such an attack if stored in a web application's database and then executed in a client's web browser:

```
<script>document.write('<img src="“http://www.attackersite.com'
+document.cookie+'"“) </script>
```

A third type of XSS attack is Document Object Model, or DOM-based, XSS. This is a different kind of XSS attack because it occurs on the client side when the user is processing the content, instead of on the server side when the web application is retrieving information to put in a web page. The Document Object Model is the standard model that represents HTML and XML content of a web page. The DOM can be modified in this type of attack to execute a malicious script in the victim's browser. An example of this type of attack would be to exploit a web page that uses some embedded JavaScript in to set the default language for the client using a variable in the URL. An example of this would be:

http://www.mysite.com/index.html#default=English

The malicious script that would exploit this would simply need to replace the variable “English” in the URL. A URL that shows this type of DOM-based XSS attack would be:

http://www.mysite.com/index.html#default=<script>malicious()</script>

Because everything after the # in a URL is not sent to the server by the browser, the malicious script would not be detected by the server even if the server was performing input validation. Therefore the script would be echoed into the page (DOM) when the browser renders it, and would result in the attacker’s script being executed [49].

Defenses

The same rules described above that apply to protecting against SQL injection, apply to protecting against XSS as well. All user supplied input should be validated and properly escaped before being included in the output web page [50]. This requires the same escaping technique as before, which in PHP is:

```
if ( !get_magic_quotes_gpc() ) {

$safe_string = mysql_real_escape_string( $original_string );

}
```

Also, proper output encoding will ensure that the browser treats the possibly dangerous content as text, and not as active content that could be executed. The “htmlentities()” and “htmlspecialchars()” functions in PHP will check output to make sure that it is HTML encoded.

The best way to avoid DOM-based XSS vulnerabilities is to have all client-side input passed to the server for proper validation. However, if using variables in the DOM cannot be avoided, then input validation should occur in the script itself. A check to confirm that the string being written to the HTML page consists of only alphanumeric characters should be completed so that no scripting characters are allowed. A downside of this defensive technique is that the security check is viewable in the HTML code to the attackers, and therefore is easily understandable and attackable [49]. An example of a script which checks that only alphanumeric characters exist in a string is:

```
if (original_string.match(/^[a-zA-Z0-9]+$/)) {

document.write(original_string);

}
```

These defensive techniques, although not infallible due to the mentioned limitations, will add a level of protection against attacks that aim to exploit XSS vulnerabilities.

2.4 Web Application Vulnerability Scanners

There are several web application vulnerability scanners that test for popular vulnerabilities in web servers and web applications. These tools can either be academic research projects, free/open-source applications, or commercial software products. Tools are developed in academia by members of universities who are interested in improving and studying web application vulnerability scanners, but are generally not available for purchase or commercial use. The open-source/free tools are available to the public, but are generally not as up-to date and accurate as the commercial tools. These tools do however, give users the ability to customize their tool and gain a greater understanding of the security of their web applications. Commercial tools usually give more comprehensive results than open-source/free tools, but can cost anywhere from just under $100.00 to over $6000.00 [8, 50]. Specific web vulnerability scanners from these three categories that automatically scan for and detect the most common web application vulnerabilities will be reviewed in this section.

2.4.1 Web Application Scanners in Academia

One of the categories of web application vulnerability scanners includes those that are developed in academia. These scanners are different from free/open-source and commercial scanners because the researchers who work on them are continuously evaluating them and also discuss not only where their design succeeds, but where their design is limited and requires future work. These scanners are not available for public use, so they cannot be used in this analysis of web vulnerability scanner limitations, but reviewing the techniques and methods used by these scanners will help in understanding how other web application scanners work [48].

Huang et al. developed a web application scanner called WAVES that attempts to reduce the number of potential side effects of black-box testing [51, 52]. The auditing process of web application scanners can cause permanent modifications, or even damage, to the state of the application it is targeting. This is a drawback that both commercial and open-source/free web application scanners share, and is why the authors introduced a testing methodology that would allow for harmless auditing. Their experimental results found that WAVES was unable to detect any new vulnerability that were not already detected by a static source code analyzer they had developed. Also, WAVES was unable to discover all of the vulnerabilities that the static source code analyzer had found (detected only 80% of the vulnerabilities found by the static analyzer). The authors believe their tool failed in part because it did not have complex procedures able to detect all data entry points, and because it was unable to observe HTML output.

Another academic black-box approach was developed by Antunes and Viera as described in [53]. Their web vulnerability scanner was used to identify SQL injection vulnerabilities in 262 publicly available web services. The first step in their approach was to prepare for the tests by obtaining information regarding the web service in order to generate the workload (valid web service calls). The second step was to execute the tests. This was accomplished by using a workload emulator that acted as a web service consumer, and by using an attack load generator that automatically generated attacks by injecting them into the workload test calls. The final step in their approach was to analyze the responses by using a set of well-defined rules which would identify vulnerabilities and exclude potential false-positives. Their results showed that they achieved a detection coverage rate of 81% in the scenario where they had access to the known number of vulnerabilities, and maintained a false-positive rate of 18% in their optimistic interpretation. These results are better than those of the commercial tools that the authors analyzed, and suggest that it is possible to improve the effectiveness of vulnerability scanners [48].

2.4.2 Free/Open-Source Web Application Scanners

Many open-source and free web application scanners are available for black-box testing and analysis. Some of these applications provide extensive functionality with the ability to be customized and expanded to meet the needs of users. Others however do not provide a great deal of usability and have a limited amount of functionality, and therefore can only test for a few web application vulnerabilities. Three of the more thorough and robust free/open-source scanners, Grendel-Scan [54], Wapiti [55], and W3AF [56], will be reviewed.

Grendel-Scan [54] is an open-source web application security testing tool which has an automated testing module for detecting common web application vulnerabilities. It has the ability to find simple web application vulnerabilities, but its designers state that no automated tool can identify complicated vulnerabilities, such as logic and design flaws. Grendel-Scan tests for SQL injection, XSS attacks, and session management vulnerabilities, as well as other vulnerabilities.

Wapiti [55] is a free web application vulnerability scanner and security auditor. It performs black-box analysis by scanning the web pages of a web application in search of scripts and forms where data can be injected. After the list of scripts and forms is gathered, Wapiti injects payloads to test if the scripts are vulnerable. Wapiti scans for remote file inclusion errors, SQL and database injections, XSS injections, and other vulnerabilities.

W3AF [56] is exactly what it stands for, a Web Application Attack and Audit Framework. The goal of the project is to create a framework which can find and exploit web application vulnerabilities easily. The project's long term objectives are for it to become the best open source web application scanner, and the best open source web application exploitation framework. Also, the designers want the project to create the biggest community of web application hackers, combine static code analysis and black box testing into one framework, and become the NMAP [57] of the web. W3AF incorporates a great deal of plug-ins into its framework, and is capable of testing for SQL injection, XSS attacks, buffer overflow, malicious file execution, and session management vulnerabilities. Table 2.1 provides a comparison of the vulnerabilities that the free and open source web application scanners search for.

Vulnerability Type	Grendel-Scan	Wapiti	W3AF
SQL Injection	X	X	X
Cross Site Scripting	X	X	X
Session Management	X	-	-
Malicious File Execution	-	X	X
Buffer Overflow	-	-	X

Table 2.1: A comparison of the relevant vulnerabilities detected by free/open-source web application scanners.

2.4.3 Commercial Web Application Scanners

Commercial web application scanners are generally licensed to companies or organizations that wish to test their web applications for vulnerabilities so that they can fix security holes before they are maliciously exploited. Since a data breach can result in the loss of personal information of thousands of customers, and the loss of millions of dollars, companies are willing to pay large sums of money for these applications. These commercial applications compete against each other for market share, and therefore do not want to disclose their scanner's limitations or restrictions. However, an approach to analyze these limitations and restrictions is proposed in this thesis. Some of the features of popular commercial web application scanners will be discussed below [48].

Cenzic [58] sells a web application scanner tool called Hailstorm which utilizes stateful testing. Stateful testing tools are designed to behave like human testers by taking what seem to be an application's insignificant or disparate weaknesses, and combining them together into serious exploits. The key benefits that Hailstorm claims are the ability to identify major security flaws in target applications, to help with internal compliance policies, to avoid vulnerabilities that lead to downtime, and to assess applications for commonly known vulnerabilities. Cenzic provides a 7-day free trial of Hailstorm Core which can detect vulnerabilities including SQL injection, XSS , and session management.

Acunetix Web Vulnerability Scanner [8] is another black-box tool which claims in-depth checking for SQL injection, XSS, and other vulnerabilities with its innovative AcuSensor Technology. This technology is supposed to quickly find vulnerabilities with a low number of false-positives, pinpoint where each vulnerability exists in the code, and report the debug information as well. Acunetix also includes advanced tools to allow penetration testers to fine tune web application security tests, and has many more features to scan websites with different scan options and identities. The only vulnerability that the free edition of the software detects is XSS, but a 30-day trial version of the product is available that also can detect SQL injection, file execution, session management, and manual buffer overflow attacks.

N-Stalker [59] provides a suite of web security assessment checks to enhance the overall security of web applications. It is founded on the technology of Component-oriented Web Application Security Scanning, and allows users to create their own assessment policies and requirements, enabling them to check for more than 39,000 signatures and infrastructure security checks. Vulnerabilities checked for include SQL injection, XSS attacks, buffer overflows, and session management attacks, but the evaluation edition only lasts for a 7-day period.

Netsparker [60] is a web application vulnerability scanner developed by Mavituna Security Ltd. Netsparker is focused on eliminating false-positives, and uses confirmation and exploitation engines to ensure that false-positives are not reported. The engines also allow the users to see the actual impact of the attacks instead of text explanations of what the attack could do. Because of the techniques Netsparker uses, Mavituna Security claims that it developed the first false-positive free web application scanner. Netsparker scans for all types of XSS injection, SQL injection, malicious file execution, and session management vulnerabilities.

Burp Scanner [61] is a web application vulnerability scanner that is part of Burp Suite Professional. Burp Suite Professional is the commercial version of Burp Suite, which is an integrated platform for attacking and testing web applications. Burp Suite provides a number of tools, including an interception web proxy, web spider, application intruder, session key analyzer, and data comparer. The professional version includes Burp Scanner which can operate in either passive or active mode, or either manual scan or live scan mode. The vulnerabilities it searches for include SQL injection, XSS injection, and session management vulnerabilities.

Rational AppScan [62] is licensed by IBM for advanced web application security scanning. The AppScan tool automates vulnerability assessments and tests for SQL injection, XSS attacks, buffer overflows, and other common web application vulnerabilities. AppScan can generate advanced remediation capabilities in order to ease vulnerability remediation, simplify results with the Results Expert wizard, and test for emerging web technologies. Rational AppScan provides an unlimited evaluation period for its standard edition; however, with the evaluation license the software is only capable of testing a test web site provided by AppScan.

BuyServers Ltd. [63] sells a web vulnerability scanner called Falcove which is a 2-in-1 scanning and penetration tool, meaning that it not only tries to detect vulnerabilities, but is capable of exploiting them as well. Falcove utilizes a crawler feature that checks for web vulnerabilities, audits dynamic content (password fields, shopping carts), and generates penetration reports that explain the security level of the tested web site. However, BuyServers Ltd. no longer supports the trial version of the product that detects SQL injection, XSS, and file execution attacks.

HP's WebInspect [64] software provides web application security testing and assessment for complex web applications. WebInspect claims fast scanning capabilities, broad security assessment coverage, and accurate web application security scanning results. HP also believes WebInspect identifies security vulnerabilities that are undetectable by traditional scanners by using innovative assessment technologies such as simultaneous crawl and audit, and on current application scanning. HP WebInspect scans for data detection and manipulation attacks, session and authentication vulnerabilities, and server and general HTTP vulnerabilities, but does not currently provide a working evaluation version of the product.

NT OBJECTives' NTOSpider [65] is a web application security scanner that claims to provide automated vulnerability assessment with unprecedented accuracy and comprehensiveness. NTOSpider identifies application vulnerabilities and ranks threat priorities, as well as produces graphical HTML reports. NT OBJECTives' proprietary S3 Methodology and Data

Sleuth intelligence engine are employed for automation and accuracy, and checks vulnerabilities on a case-by-case basis, which provides context-sensitive vulnerability checking. NTOSpider checks for SQL injection, XSS attacks, and session management vulnerabilities, but does not provide a trial version for evaluation.

Table 2.2 provides a summary of the relevant vulnerabilities that each of the evaluation versions of the commercial web application scanners detect.

Vulnerability Type	Hailstorm	N-Stalker	Netsparker	Acunetix	Burp Scanner
SQL Injection	X	X	X	X	X
Cross Site Scripting	X	X	X	X	X
Session Management	X	X	X	X	X
Malicious File Execution	-	-	X	X	-
Buffer Overflow	-	X	-	X	-

Table 2.2: A comparison of the relevant vulnerabilities detected by evaluation versions of commercial web application scanners.

Chapter 3
RELATED WORK

According to Curphey and Araujo [1], there are eight categories of web application security Assessment tools: source code analyzers, web application (black box) scanners, database scanners, binary analysis tools, runtime analysis tools, configuration management tools, HTTP proxies, and miscellaneous tools. The most common of these web application assessment tools are source code analyzers and web application scanners. Source code analyzers generally achieve good vulnerability detection rates, but are only useful if the web application's source code is available. On the other hand, web application vulnerability scanners are the tools which most closely mimic web application attacks, but have been known to perform rather poorly [3, 17, 44, 52].

There are two main approaches to test web applications for vulnerabilities [5]:

> **White box testing:** consists of the analysis of the source code of the web application. This can be done manually or by using code analysis tools like Ounce [6] or Pixy [7]. The problem is that exhaustive source code analysis may be difficult and cannot find all security flaws because of the complexity of the code.
>
> **Black box testing:** consists in the analyses of the execution of the application in search for vulnerabilities. In this approach, also known as penetration testing, the scanner does not know the internals of the web application and it uses fuzzing techniques over the web HTTP requests.

In practice, black-box vulnerability scanners are used to discover security problems in web applications. These tools operate by launching attacks against an application and observing its response to these attacks. To this end, web server vulnerability scanners such as Nikto [10] or Nessus [11] dispose of large repositories of known software flaws. While these tools are valuable components when auditing the security of a web site, they largely lack the ability to identify a priori unknown instances of vulnerabilities. As a consequence, there is the need for a scanner that covers a broad range of general classes of vulnerabilities, without specific knowledge of bugs in particular versions of web applications [9].

Security testing a Web application or Web site requires careful thought and planning due to both tool and industry immaturity. Finding the right tools involves several steps, including analyzing the development environment and process, business needs, and the Web application's complexity. M. Curphey and R. Arawo [1] Describes the different technology types for analyzing Web applications and Web services for security vulnerabilities, along with each type's advantages and disadvantages. Their analysis is based on collective experiences and the lessons we've learned along the way.

Fonseca, J. CISUC proposed a method to evaluate and benchmark automatic web vulnerability scanners using software fault injection techniques. The most common types of software faults are injected in the web application code which is then checked by the scanners. The results are compared by analyzing coverage of vulnerability detection and false positives. Three leading commercial scanning tools are evaluated and the results show that in general the coverage is low and the percentage of false positives is very high [42].

Fonseca, J. CISUC [4] proposed a methodology to inject realistic attacks in Web applications. The methodology is based on the idea that by injecting realistic vulnerabilities in a Web application and attacking them automatically we can assess existing security mechanisms. To provide true to life results, this methodology relies on field studies of a large number of vulnerabilities in Web applications. The paper also describes a set of tools implementing the proposed methodology. They allow the automation of the entire process, including gathering results and analysis. We used these tools to conduct a set of experiments to demonstrate the feasibility and effectiveness of the proposed methodology. The experiments include the evaluation of coverage and false positives of an intrusion detection system for SQL injection and the assessment of the effectiveness of two Web application vulnerability scanners. Results show that the injection of vulnerabilities and attacks is an effective way to evaluate security mechanisms and tools.

Stefan Kals, Engin Kirda, Christopher Kruegel, and Nenad Jovanovic [9] developed SecuBat, a generic and modular web vulnerability scanner that, similar to a port scanner, automatically analyzes web sites with the aim of finding exploitable SQL injection and XSS vulnerabilities. Using SecuBat, they were able to find many potentially vulnerable web sites. To verify the accuracy of SecuBat, they picked one hundred interesting web sites from the potential victim list for further analysis and confirmed exploitable flaws in the identified web pages. Among thier victims were well-known global companies and a finance ministry. More than fifty responded to request additional information or to report that the security hole was closed.

SecuBat vulnerability scanner [9] consists of three main components: First, the crawling component gathers a set of target web sites. Then, the attack component launches the configured attacks against these targets. Finally, the analysis component examines the results returned by the web applications to determine whether an attack was successful.

Scott and Sharp [41] discuss web vulnerabilities such as XSS. They propose to deploy application-level firewalls that use manual policies to secure web applications. Their approach would certainly protect applications against a vulnerability scanner such as SecuBat. However, the problem of their approach is that it is a tedious and error-prone task to create suitable policies.

Huang et al. [13] present a vulnerability detection tool that automatically executes SQL injection attacks. As far as SQL injection is concerned, our work is similar to theirs. However, their scanner is not as comprehensive as our tool because it lacks any detection mechanisms for XSS vulnerabilities where script code is injected into applications. The focus of their work, rather, is the detection of application level vulnerabilities that may allow the attacker to invoke operating-level system calls (e.g., such as opening a file) for malicious purposes.

There are many commercial web application vulnerability scanners available on the market that claim to provide functionality similar to our scanner (e.g., Acunetix Web Vulnerability Scanner [8]). Unfortunately, due to the closed-source nature of these systems, many of the claims cannot be verified, and an in-depth comparison with our scanner is difficult. For example, it appears that the cross-site scripting analysis performed by Acunetix is much simpler than the complete attack scenario presented in our approach. Also, no working proof-of-concept exploits are generated.

AMNESIA is a model-based technique that combines static analysis and runtime monitoring [15, 14]. In its static phase, AMNESIA uses static analysis to build models of the different types of queries an application can legally generate at each point of access to the database. In its dynamic phase, AMNESIA intercepts all queries before they are sent to the database and checks each query against the statically built models. Queries that violate the model are identified as SQLIAs and prevented from executing on the database. In their evaluation, the authors have shown that this technique performs well against SQLIAs. The primary limitation of this technique is that its success is dependent on the accuracy of its static analysis for building query models. Certain types of code obfuscation or query development techniques could make this step less precise and result in both false positives and false negatives.

SQLGuard [17] and SQLCheck [16] also check queries at runtime to see if they conform to a model of expected queries. In these approaches, the model is expressed as a grammar that only accepts legal queries. In SQLGuard, the model is deduced at runtime by examining the structure of the query before and after the addition of user-input. In SQLCheck, the model is specified independently by the developer. Both approaches use a secret key to delimit user input during parsing by the runtime checker, so security of the approach is dependent on attackers not being able to discover the key. Additionally, the use of these two approaches requires the developer to either rewrite code to use a special intermediate library or manually insert special markers into the code where user input is added to a dynamically generated query.

WebSSARI detects input-validation related errors using information flow analysis [18]. In this approach, static analysis is used to check taint flows against preconditions for sensitive functions. The analysis detects the points in which preconditions have not been met and can suggest filters and sanitization functions that can be automatically added to the application to satisfy these preconditions. The WebSSARI system works by considering as sanitized input that has passed through a predefined set of filters. In their evaluation, the authors were able to detect security vulnerabilities in a range of existing applications. The primary drawbacks of this technique are that it assumes that adequate preconditions for sensitive functions can be accurately expressed using their typing system and that having input passing through certain types of filters is sufficient to consider it not tainted. For many types of functions and applications, this assumption is too strong.

Livshits and Lam [19] use static analysis techniques to detect vulnerabilities in software. The basic approach is to use information flow techniques to detect when tainted input has been used to construct an SQL query. These queries are then flagged as SQLIA vulnerabilities. The authors demonstrate the viability of their technique by using this approach to find security vulnerabilities in a benchmark suite. The primary limitation of this approach is that it can detect only known patterns of SQLIAs and, because it uses a conservative analysis and has limited support for untinting operations, can generate a relatively high amount of false positives.

Huang and colleagues [19] propose WAVES, a black-box technique for testing Web applications for SQL injection vulnerabilities. The technique uses a Web crawler to identify all points in a Web application that can be used to inject SQLIAs. It then builds attacks that target such points based on a specified list of patterns and attack techniques. WAVES then monitors the application's response to the attacks and uses machine learning techniques to improve its attack methodology. This technique improves over most penetration-testing techniques by using machine learning approaches to guide its testing. However, like all black-box and penetration testing techniques, it cannot provide guarantees of completeness.

Fu et al. suggested a Static Analysis approach to detect SQL Injection Vulnerabilities. The main aim of SAFELI approach is to identify the SQL Injection attacks during compile-time. It has a couple of advantages. First, it performs a White-box Static Analysis and second, it uses a Hybrid-Constraint Solver. On one hand where the given approach considers the byte-code and deals mainly with strings in case of White-box Static Analysis, on the other through Hybrid-Constraint Solver, the method implements an efficient string analysis tool which is able to deal with Boolean, integer and string variables. Its implementation was done on ASP.NET Web applications and it was able to detect vulnerabilities that were ignored by the black-box vulnerability scanners. This approach is an efficient approximation mechanism to deal with string constraints. However, the approach is only dedicated to ASP.NET vulnerabilities [25].

A secured database testing approach has been suggested for web applications by Haixia and Zhihong. This approach suggested the following methodology:-

I. Detection of potential input points of SQL injection.

II. Automatic generation of test cases.

III. Running the test cases to make an attack on the application to find the database vulnerability.

The mechanism suggested here is shown to be efficient as it was able to detect the input points of SQL Injection exactly and on time as per expectation. An analysis on this technique makes it clear that the approach needs improvement in the development of attack rule library and detection capability [26].

Ruse et al.'s Approach suggested using automatic test case generation for detection of SQL injection vulnerabilities. The idea is to create a specific model that deals with SQL queries automatically. Furthermore this technique also identifies the relation between sub-queries. This technique is shown to identify the casual set and obtain 85% and 69% reduction respectively while experimenting on few sample examples. Moreover, no false positive or false negative were produced and it has been able to detect the root cause of the injection. Although this approach claimed an apparent efficiency, it has a huge drawback that this approach has not been tested on real life existing database with real queries [27].

Thomas et al.'s approach suggested an automated prepared statement generation algorithm to eliminate vulnerabilities related to SQL Injection. Their research work used four open source projects namely: (i) Net-trust, (ii) ITrust, (iii) WebGoat, and (iv) Roller. The experimental results show that, their prepared statement code was able to successfully replace 94% of the SQL injection vulnerabilities in four open source projects. The only limitation observed was that the experiment was conducted using only

Java with a limited number of projects. Hence, the wide application of the same approach and tool for different settings still remains an open research issue to investigate [28].

SQL-IDS approach has been suggested by Kemalis and Tzouramanis in [29] and it uses a novel specification-based methodology for detecting exploitations of SQL injection vulnerabilities. The method proposed here does query-specific detection which allows the system to perform concentrated analysis at almost no computational overhead. It also does not produce any false positives or false negatives. This is a very new approach and in practice it's very efficient; however, it is required to conduct more experiments and do comparison with other detection methods under a flexible and shared environment.

McClure and Krüger suggested a framework SQL DOM (strongly-typed set of classes with database schema). The existing flaws have been considered closely during access of relational databases from Object-Oriented Programming Language's point of view. The focus lies mainly in identifying hurdles in interacting with databases through Call Level Interfaces. The solution proposed here is based on SQL DOM object model to handle this kind of issues by creating a secure surrounding i.e., creation of SQL statement through object manipulation for communication. When this technique was evaluated qualitatively it showed many advantages for: testability, readability, maintainability and error detection at compile. Although this proposal is efficient, there still exists scope of further improvements with latest and more advanced tools such as CodeSmith [30].

Ali et al.'s approach has been adopted by Ali et al. in which a hash value technique has been followed to improve user authentication mechanism. Hash values for user name and password has been used. For testing this kind of framework SQLIPA (SQL Injection Protector for Authentication) was developed. Hash values for user name and password is created for the first time user account is created and they stored in the user account table in a database. The framework requires further improvement in order to minimize the overhead time which was 1.3 ms even though tested on few sample data. Hence simply minimizing the overhead time is not sufficient but also to test this framework with large amount of data is required [31].

Su and Wassermann implemented their algorithm with SQLCHECK on a real time environment. It checks whether the input queries conform to the expected ones defined by the programmer. A secret key is applied for the user input delimitation. The analysis of SQLCHECK shows no false positives or false negatives. Also, the overhead runtime rate is very low and can be implemented directly in many other Web applications using different languages. Table 3 shows the number of attacks attempted as well as prevented [32]. It also shows the number of valid uses attempted and allowed, and the mean and standard deviation of times across all runs of SQLCHECK for the application under check. It is a very efficient approach; however, once an attacker discovers the key, it becomes vulnerable. Furthermore, it also needs to be tested with online Web applications [32, 33].

Dynamic Candidate Evaluations approach has been proposed by Bisht et al. called CANDID (Candidate evaluation for Discovering Intent Dynamically). It is a Dynamic Candidate Evaluations technique in which SQL injection is not only detected but also prevented automatically. Mechanism behind this method is that it dynamically extracts the query structures from every SQL query location which is intended by the developer. So, basically it resolves the problem of manually changing the application to produce the prepared statements. Although tool using this mechanism has been shown to be efficient in some cases, it failed for many other cases. An example for its failure is when applied at a wrong level or when an external function is dealt with. Furthermore it also fails in many cases due to limited capability of this technique [34].

Cross Site Scripting (XSS) vulnerability analysis has many approaches [20], Interpreter-based Approaches and Syntactical Structure Analysis. Pietraszek, and Berghe use Interpreter based approach of instrumenting interpreter to track untrusted data at the character level and to identify vulnerabilities they use context-sensitive string evaluation at each susceptible sink [21]. This approach is sound and can detect vulnerabilities as they add security assurance by modifying the interpreter. But approach of modifying interpreter is not easily applicable to some other web programming languages, such as Java (i.e., JSP and servlets) [22].

On the other hand, A successful inject attack changes the syntactical structure of the exploited entity, stated by Su, and Wassermann in [22] and they present an approach to check the syntactic structure of output string to detect malicious payload. Augment the user input with metadata to track this sub-string from source to sinks. This metadata help the modified parser to check the syntactical structure of the dynamically generated string by indicating end and start position of the user given data. If there is any abnormality then it blocks further process. These processes are quite success while it detect any injection vulnerabilities other than XSS. Only checking the syntactic structure is not sufficient to prevent this sort of workflow vulnerabilities that are caused by the interaction of multiple modules [23].

Gary and Zhendong [24] presented a static analysis for finding XSS vulnerabilities that directly addresses weak or absent input validation. thier approach combines work on tainted information flow with string analysis. Proper input validation is difficult largely because of the many ways to invoke the JavaScript interpreter; they faced the same obstacle checking for vulnerabilities statically, and they address it by formalizing a policy based on the W3C recommendation, the Firefox source code, and online tutorials about closed-source browsers. They provide effective checking algorithms based on thier policy. And they implemented their approach and provided an extensive evaluation that finds both known and unknown vulnerabilities in real-world web applications.

A number of black-box testing, fault injection and behavior monitoring to web applications approaches has been used by Y. Huang, S. Huang, Lin, and Tsai in order to predict the presence of vulnerabilities [35]. This approach combines user experience modeling as black-box testing and user-behavior simulation [36]. There are many other projects where a similar kind of approach has been followed like APPScan [37], WebInspect[38], and ScanDo [39]. As all these approaches were used to detect errors in the development life cycle, they might not be able to provide instant web application protection [40] and they cannot guarantee the detection of all flaws as well [41].

Su and Wassermann in [32] suggested an approach which states that when there is a successful injection attack there is a change in the syntactical structure of the exploited entity. So, they have presented an approach where syntactic structure of output string is checked to detect malicious payload. For tracking sub-string from source to sinks they increased the user input with metadata. The modified parser was helped by this metadata to check the syntactical structure of the dynamically generated string by indicating start and end point of the user given input. Moreover the process was blocked if there was a sign of any abnormality. This approach was found to be quite successful while it detects any injection vulnerabilities other than XSS. Hence, it is not sufficient to avoid this sort of workflow vulnerabilities which are result of interaction between multiple modules [43].

Interpreter-based approach has been suggested by Pietraszek, and Berghe in which there is use of instrumenting interpreter to track untrusted data at the character level and for identifying vulnerabilities that use context-sensitive string evaluation at each susceptible sink [44]. This technique is good and also able to detect vulnerabilities as security assurance is added by modifying the interpreter, however this approach of modifying interpreter is not easily feasible to some other famous and widely used web programming languages, such as Java, Jsp, Servlets [32].

Chapter 4

THE PROPOSED APPROACH METHODOLOGY AND DESIGN

4.1 Introduction

Many scanners which are considered as web application scanners identify vulnerabilities that are known in specific pages and applications. For example, the Content Management Systems (CMS), such as Joomla and wordpress, have many vulnerabilities for with web scanners contains a signature for it, but these scanners don’t have signatures for vulnerabilities that may present in a custom web application built for specific client. In order to detect and examine these custom vulnerabilities, the web scanner have to dynamically generate test requests that are applied to the given web application.

In order to start analyzing vulnerabilities of the web application, the scanner need to get data from the web application, these data will be the GET and POST requests with parameters. Getting data from web application is called crawling; this is very effective technique that can be used to record web application pages parameters and requests.

Our approach performs tests for two major web applications vulnerabilities, which are, SQL Injection and Cross Site Scripting (XSS). The main reason for choosing these vulnerabilities is their criticality and importance, as they were reported by OWASP as the most vulnerability of web applications [47].

4.2 Tools used

To implement the vulnerability analysis scanner many tools were used. This includes, Perl Programming Language, Linux OS: BackTrack 5, and virtual machine software (VM Ware).

4.2.2 Perl Programming Language

Perl Programming language was originally developed by Larry Wall in 1987 as a general-purpose Unix scripting language to make report processing easier. Since then, it has undergone many changes and revisions. The latest major stable revision is 5.16, released in May 2012. Perl borrows features from other programming languages including C, shell scripting (sh), AWK, and sed. The language provides powerful text processing facilities without the arbitrary data length limits of many contemporary UNIX tools, facilitating easy manipulation of text files. Perl gained widespread popularity in the late 1990s as a CGI scripting language, in part due to its parsing abilities. In addition to CGI, Perl is used for graphics programming, system administration, network programming, finance, bioinformatics, and other applications [66].

Perl has many features since The overall structure of Perl derives broadly from C. Perl is procedural in nature, with variables, expressions, assignment statements, brace-delimited blocks, control structures, and subroutines. Perl also takes features from shell programming. All variables are marked with leading sigils, which unambiguously identify the data type (for example, scalar, array, hash) of the variable in context. Importantly, sigils allow variables to be interpolated directly into strings. Perl has many built-in functions that provide tools often used in shell programming (although many of these tools are implemented by programs external to the shell) such as sorting, and calling on system facilities [66].

Perl is often used as a glue language, tying together systems and interfaces that were not specifically designed to interoperate, and for "data munging",[66] that is, converting or processing large amounts of data for tasks such as creating reports. In fact, these strengths are intimately linked. The combination makes Perl a popular all-purpose language for system administrators, particularly because short programs can be entered and run on a single command line.

Perl code can be made portable across Windows and Unix; such code is often used by suppliers of software (both COTS and bespoke) to simplify packaging and maintenance of software build- and deployment-scripts.

4.2.3 Linux Operating System (Backtrack Distribution)

BackTrack Distribution is a penetration testing and security auditing platform with advanced tools to identify, detect, and exploit any vulnerabilities uncovered in the target network environment. Applying appropriate testing methodology with defined business objectives and a scheduled test plan will result in robust penetration testing of your network.

BackTrack is a merger between three different live Linux penetration testing distributions—IWHAX, WHOPPIX, and Auditor. In its current version, BackTrack is based on Ubuntu Linux distribution version.

BackTrack contains a number of tools that can be used during your penetration testing process. The penetration testing tools included in Backtrack can be categorized into the following:

- **Information gathering:** This category contains several tools that can be used to get information regarding a target DNS, routing, e-mail address, websites, mail server, and so on. This information is gathered from the available information on the Internet, without touching the target environment.
- **Network mapping:** This category contains tools that can be used to check the live host, fingerprint operating system, application used by the target, and also do port scanning.
- **Vulnerability identification:** In this category we can find tools to scan vulnerabilities (general) and in Cisco devices. It also contains tools to

carry out fuzzing and analyze Server Message Block (SMB) and Simple Network Management Protocol (SNMP).

- **Web application analysis:** This category contains tools that can be used in auditing web application.
- **Radio network analysis:** To audit wireless networks, Bluetooth and Radio Frequency Identifier (RFID), you can use the tools in this category.
- **Penetration:** This category contains tools that can be used to exploit the vulnerabilities found in the target machine.
- **Privilege escalation:** After exploiting the vulnerabilities and gaining access to the target machine, we can use tools in this category to escalate privilege to the highest privilege.
- **Maintaining access:** Tools in this category will be able to help us in maintaining access to the target machine.
- **Voice over IP (VOIP):** To analyze VOIP we can utilize the tools in this category.
- **Digital forensics:** In this category we can find several tools that can be used to do digital forensics such as acquiring hard disk image, carving files, and analyzing hard disk image.
- **Reverse engineering:** This category contains tools that can be used to debug a program or disassemble an executable file.

4.3 Development Description

In this section, our approach details will be described. The main steps of our approach are described in figure 4.1. First, the scanner will do manual crawling for the web application pages. Second, it will loop through all log requests that were recorded by the crawler and filter these requests as they are GET or POST requests, in this steps the scanner will check for requests parameters as well. Third, for each parameter of each request the scanner will perform our main tests which are SQL INJECTION and XSS tests. Fourth, the scanner will check for web directory listing vulnerability test, which if it's found the scanner will check for HTTP PUT test. The scanner reports each transaction of the done tests so it will generate a full report at the end of work

The approach steps can be listed as followed order:

1. Web application crawling.
2. Web pages requests with parameters filtering.
3. SQL Injection test.
4. Cross Site Scripting (XSS) test.
5. Directory listing test.
6. Report Generation.

4.3.1 Web application crawling

The first step that the scanner will do is obtaining data about the target web application. This process will be called Application crawling. This is very effective technique that can be used to record web application pages parameters and requests.

Web Scanner can use web crawling in two ways, automatic web crawling and manual web crawling. Firstly, Automatic crawling software is a type of bot; In general, it starts with a list of URLs to visit, called the seeds. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl frontier. URLs from the frontier are recursively visited according to a set of policies. Examples of these crawlers are: GNU Wget utility, Aspseek, GRUB. Secondly, Manual crawling which can be done using a local web proxy with manually accepting and recording web requests. The main benefit of both techniques is to build a list of web requests for all web application pages and links so they can be passed to the vulnerability scanner later.

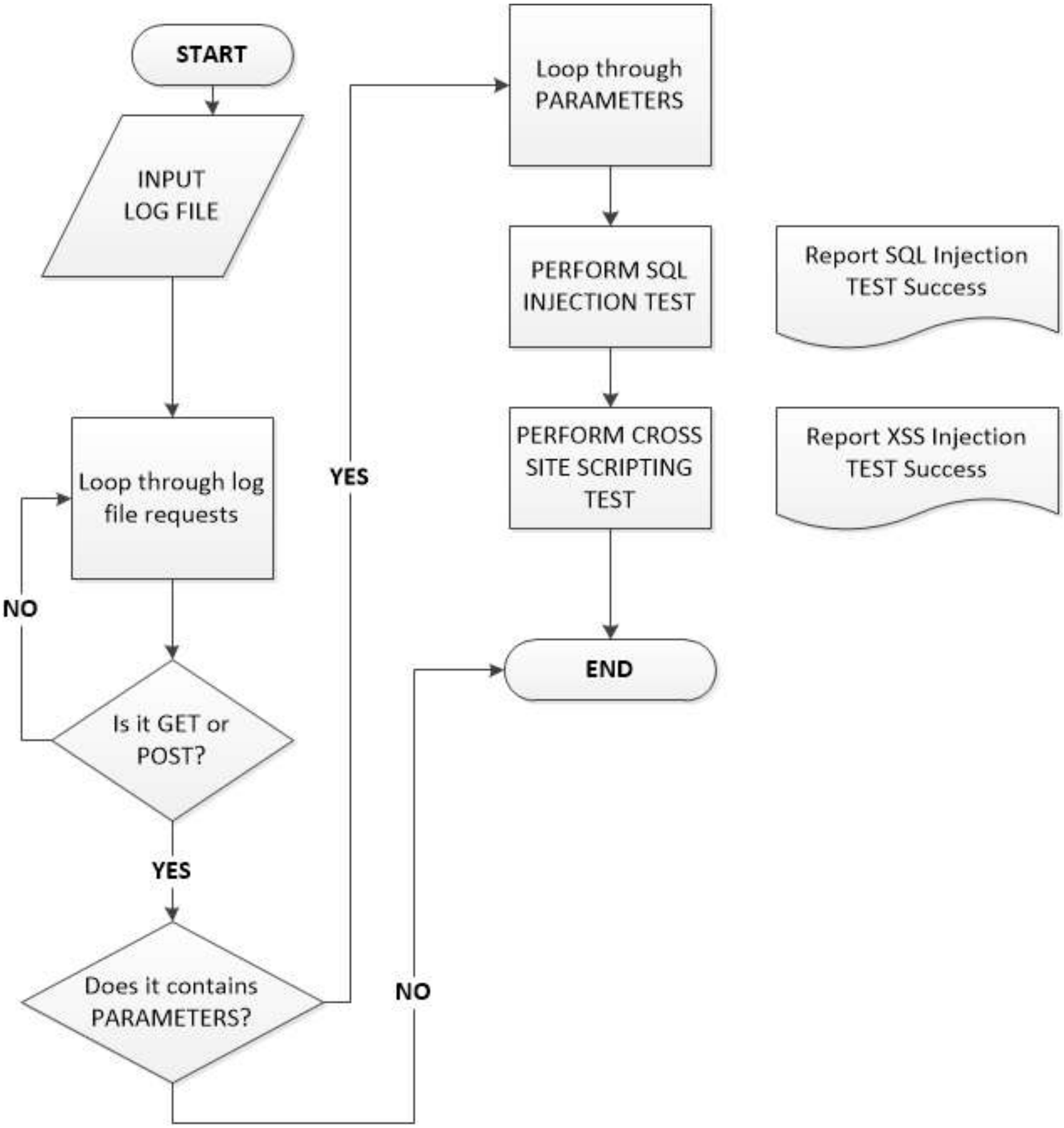


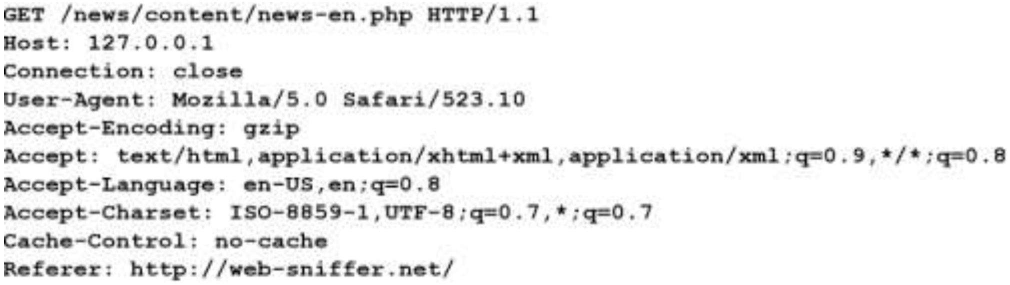
Figure 4.1: Our approach flowchart

Even the automatic web crawling is easier and faster, it can’t discover all application pages for many reasons. Firstly, the crawling software must be able to parse Web forms and generate logical form submissions to the application (i.e. web crawler couldn’t submit form with reCAPTCHA field). Otherwise, the application's business logic prevents the user from reaching subsequent pages or areas of the application. Secondly, automated web crawling software follow every link and/or form a given web application presents, they might cause unanticipated events to occur. For example, if a hyperlink presented to the user allows certain transactions to be processed or initiated, the agent might inadvertently delete or modify application data or initiate unanticipated transactions on behalf of a user. For these reasons, most experienced testers normally prefer the manual crawling technique because it allows them to achieve a thorough crawl of the application while maintaining control over the data and pages that are requested during the crawl and ultimately are used to generate test requests [9].

Our approach will rely on a manual application crawl to discover all testable pages and requests. In order to complete this task, we will use local proxy server utility to record all application requests in a log file as manually to crawl the application.

4.3.2 Web pages requests with parameters filtering

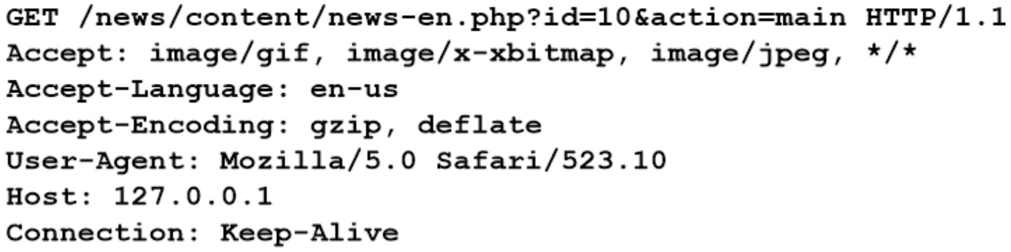
Our scanner works on HTTP (GET, POST) requests and response parameters. HTTP works through a series of requests from the client and associated server responses back to the client. Each request is independent and results in a server response. A typical raw HTTP request is shown in figure 4.2.



```
GET /news/content/news-en.php HTTP/1.1
Host: 127.0.0.1
Connection: close
User-Agent: Mozilla/5.0 Safari/523.10
Accept-Encoding: gzip
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7
Cache-Control: no-cache
Referer: http://web-sniffer.net/
```

Figure 4.2: RAW HTTP request

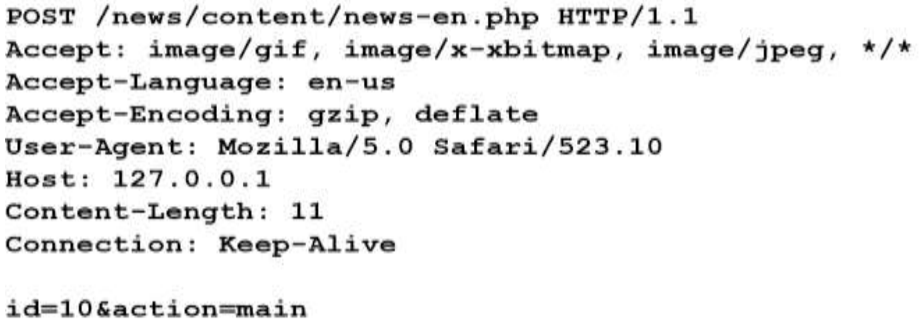
Our scanner depends on GET and POST methods, in GET method the request can contain parameters that will be send with the HTTP request. A typical GET request with parameters in shown in figure 4.3. The request can contain multiple parameter data separated by & mark. Our scanner will parse and use each one of the parameters to check its vulnerability tests.



```
GET /news/content/news-en.php?id=10&action=main HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 Safari/523.10
Host: 127.0.0.1
Connection: Keep-Alive
```

Figure 4.3: GET method

The POST method passes the data parameters of the request after all request headers, as show in figure 4.4. as well as, our scanner will parse these parameters in order to use it in the vulnerability test.

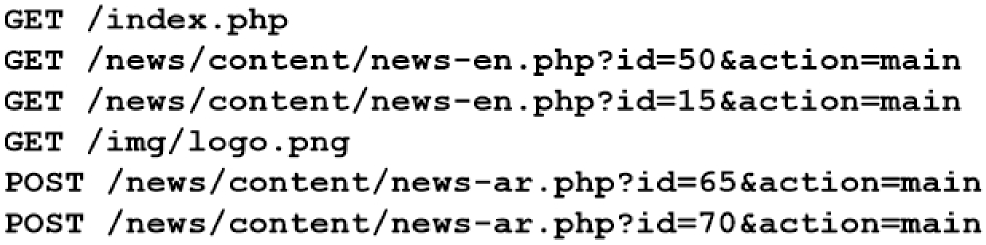


```
POST /news/content/news-en.php HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 Safari/523.10
Host: 127.0.0.1
Content-Length: 11
Connection: Keep-Alive

id=10&action=main
```

Figure 4.4: POST method

The scanner will parse all HTTP requests and generate a new list containing only the type of request (GET, POST), the path of the web page requested, and the passed parameters of the request. Figure 4.5 shows a sample list generated by the scanner.



```
GET /index.php
GET /news/content/news-en.php?id=50&action=main
GET /news/content/news-en.php?id=15&action=main
GET /img/logo.png
POST /news/content/news-ar.php?id=65&action=main
POST /news/content/news-ar.php?id=70&action=main
```

Figure 4.5: HTTP requests parse result list

4.3.3 SQL Injection

SQL injection can be present in any front-end application accepting data entry from a system or user, which is then used to access a database server. In this section, we will focus on our techniques in identifying and testing of SQL injection vulnerabilities.

In a Web environment, the Web browser is a client acting as a front end requesting data from the user and sending it to the remote server which will create SQL queries using the submitted data. Our main goal at this stage is to identify anomalies in the server response and determine whether they are generated by SQL injection vulnerability.

Our two main techniques for testing SQL Injection vulnerabilities are testing by inference (parameter based) and UNION-Blind injection, figure 4.6 represents the block diagram of SQL Injection proposed methodology.

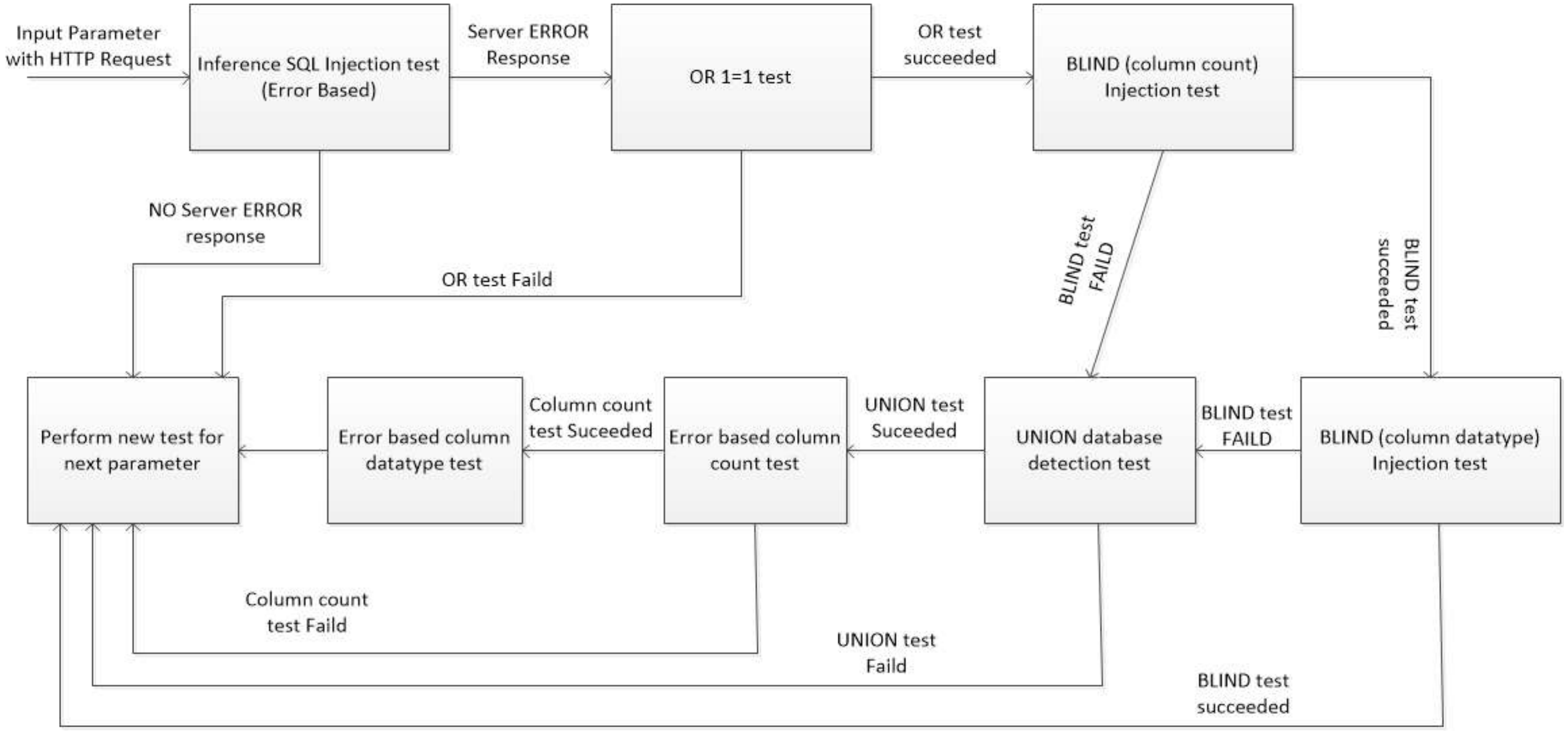

Input Parameter with HTTP Request
Inference SQL Injection test (Error Based)
Server ERROR Response
OR 1=1 test
OR test succeeded
BLIND (column count) Injection test
NO Server ERROR response
OR test Faild
BLIND test FAILD
BLIND test succeeded
Perform new test for next parameter
Error based column datatype test
Column count test Suceeded
Error based column count test
UNION test Suceeded
UNION database detection test
BLIND test FAILD
BLIND (column datatype) Injection test
Column count test Faild
UNION test Faild
BLIND test succeeded

Figure 4.6: Block Diagram

1- SQL Injection by inference:

Testing for SQL Injection by inference requires three steps:

a. Identify all the data entry on the Web application.
b. Specify the kind of request that might trigger anomalies.
c. Detect anomalies in the response from the server.

Once our scanner identified all the data accepted by the application, it will modify it and analyze the response from the server. If the web application is vulnerable its response will include an SQL error that reported directly from the database server.

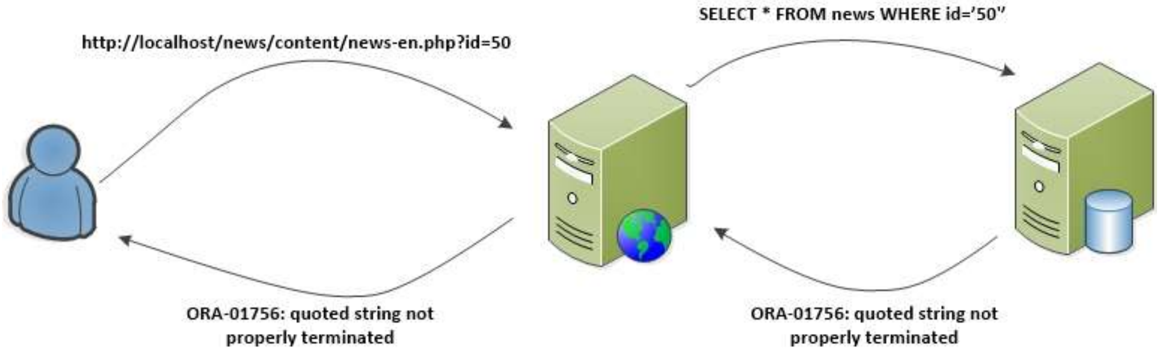

http://localhost/news/content/news-en.php?id=50
SELECT * FROM news WHERE id='50''
ORA-01756: quoted string not properly terminated
ORA-01756: quoted string not properly terminated

Figure 4.7: Database server error response

Figure 4.7 illustrates the scenario of a request from the scanner which triggers an error on the database. Depending on how the application is coded, the file returned in step 4 will be constructed and handled as a result of one of the following:

a. The SQL error is displayed on the page and is visible to the user from the Web browser.
b. The SQL error is hidden in the source of the Web page for debugging purposes.
c. Redirection to another page is used when an error is detected.
d. An HTTP error code 500 (Internal Server Error) or HTTP redirection code 302 is returned.

e. The application handles the error properly and simply shows no results, perhaps displaying a generic error page.

In order to identify the SQL injection vulnerability the scanner needs to determine the type of response the application is returning. These responses are categorized into two types, Common database server errors and Generic error.

Common database servers errors are known as full SQL error and even they are different from database server to another it's the easiest way to determine if the parameters are vulnerability hole to the web application, Table 4.1 represent different error response messages from database servers. The scanner will check for error messages returned and search for common errors in the return error message.

No.	Database Server	Error Message
1.	SQL Server	Server Error in '/' Application. Syntax error converting the nvarchar value 'Microsoft SQL Server 2000 – 8.00.760 (Intel X86) Dec 17 2002 14:22:05 Copyright (c) 1988-2003 Microsoft Corporation Enterprise Edition on Windows NT 5.2 (Build 3790:) ' to a column of data type int. Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.
2.	MySQL Server	Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1
3.	Oracle Server	Exception Details: System.Data.OleDb.

		OleDbException: One or more errors occurred during processing of command. ORA-00933: SQL command not properly ended

Table 4.1: Common database servers' errors.

Generic errors returns from web application server are second kind of errors returned from database server that will be tested by our scanner; these generic errors are returned instead of known error messages described in table 4.2 and they may indicate a potential SQL injection vulnerability. In order to test for generic errors the scanner will inject SQL code into parameter and analyses server response which will be in many various forms. These forms categorized into two types, HTTP Code Error and Different Response size error as in Table 4.2.

Error Type	Error Criteria
500 server error	500 status codes returned in the test response, but not in the original page response.
Generic error message	Generic error message (string including unable to, error, or cannot) returned in the test response, but not in the original page response.
Small (length) response	Test response was 100 characters or less in length, and the original page response was greater than 100 characters in length.
Detailed database error	Database error message detected in the test response, but not in the original page response.
No error	None of the error classification criteria were met.

Table 4.2: Generic errors criteria

2- OR Injection test:

OR 1=1 Injection test is used to determine whether there are additional injection testing can be performed using the current parameter and its associated request. In this stage we exploit the SQL code and check if it can be exploited in order to complete our tests.

Since each query has different coding criteria we have to automate this exploit, this function inserts several different OR 1=1 test strings in an attempt to make the application execute a well-formed query. The generated response from the server will be tested for successful reply so this will confirm the success of OR test.

The following expression used by our function to complete this exploit test**:**

```
"1%20OR%20'1'%3D'1'--",
"1'%20OR%201%3D1--",
"1\)%20OR%20'1'%3D'1'--",
"1'\)%20OR%201%3D1--",
"1\)\)%20OR%20'1'%3D'1'--",
"1'\)\)%20OR%201%3D1--",
"1\)\)\)%20OR%20'1'%3D'1'--",
"1'\)\)\)%20OR%201%3D1--",
"%20OR%20'1'%3D'1'--",
"'%20OR%201%3D1--",
"1'%20OR%20'1'%3D'1",
"1'%20OR%201%3D1",
"1%20OR%20'1'%3D'1'",
"1'\)%20OR%20\('1'%3D'1",
"1'\)%20OR%20\(1%3D1",
"1\)%20OR%20\('1'%3D'1'",
"1'\)\)%20OR%20\(\('1'%3D'1",
"1'\)\)%20OR%20\(\(1%3D1",
"1\)\)%20OROR%20\(\('1'%3D'1'",
"1'\)\)\)%20OR%20\(\(\('1'%3D'1",
"1'\)\)\)%20OR%20\(\(\(1%3D1",
"1\)\)\)%20OR%20\(\(\('1'%3D'1'"
```

After making the request to the target server, we check for successful reply and by gaining this reply we confirm that the server is vulnerable for applying next tests.

3- UNION-Blind Injection:

If the web application does not respond with error messages from database server and only respond with developer messages, then SQL injection by inference can’t be used and our scanner will attempt to blind injection technique if only we have successfully exploited the server with the OR test. Blind SQL injection is a type of SQL injection vulnerability where the attacker can manipulate an SQL statement and the application returns different values for true and false conditions [book: sql inj].

Our UNION-Blind technique is a merged technique of UNION based injection technique and BLIND injection technique. Our proposed method starts with detecting the database server type since each database server has its own SQL syntax. Secondly, the method starts the BLIND injection in order to determine the number of columns in the target query as well as to determine the columns data types in that query. Thirdly, In case our BLIND technique failed, the UNION technique will start to determine the number of columns in the target query as well as to determine the columns data types in that query.

In order to start BLIND column count of the target query we have to use ORDER BY statement. By appending the ORDER BY followed by the column number into our last stage injection result (OR test) we can count the number of columns found in the query. By incrementing the ORDER BY counter value, we will count the columns until we reach an error response from the server.

After we have determined the number of columns in the target query, we have to determine the data type of each column. This procedure is database specific, so we first have to determine the type of our target database server.

In order to determine the target database server, we have to test for private information that distinct database servers from each other. This information can be a default table name that differs from database server to other, and list of common data types names that also differ in name from database server to other. As an example, ORACLE database server use a default table name (ALL_TABLES) table and a common data types names (CHAR, NUMBER, and DATE). On the other side, MSSQL database server use a default table name (master.sysdatabases) table and a common data types names (VARCHAR, INT).

After determining the database server type, we can start our BLIND columns data type test. This test will be done for each column in the target query. In order to determine the data type, we iterate through columns positions in the target query and for each column we loop through a known data type list until we get a response from the server with no error reply.

In case our BLIND Injection test failed the scanner will start the UNION injection test immediately. UNION Injection is based on return error messages from the server. These error messages are specific for each server and have to be known in our scanner. This procedure has three main methods, and for each method there are specific error message the scanner will search for, table 4.3 list each error message for database servers.

Database server	**Error type**	**Error message**
Microsoft SQL Server	Invalid table in UNION	Invalid object name.
	Incorrect number of columns in UNION	All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target lists.
	Incorrect data type in UNION (three possible messages)	Error converting data type nvarchar to float or Syntax error converting the nvarchar value " to a column of data type into Operand type clash
MySQL Server	Invalid table in UNION	SQLSTATE: 42S02 (ER_BAD_TABLE_ERROR) Unknown table '%s'
	Incorrect number of columns in UNION	SQLSTATE: 21000 (ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT) The used SELECT statements have a different number of columns
	Incorrect data type in UNION (three possible messages)	incorrect date time value in column incorrect string value in column incorrect integer value in column
Oracle	Invalid table in UNION (two possible messages)	Table or view does not exist or Invalid table name
	Incorrect number of columns in UNION	Query block has incorrect number of result columns.
	Incorrect data type in UNION	Expression must have same data type as corresponding expression.

Table 4.3: MySQL, Microsoft SQL Server, and Oracle error messages

First UNION injection test module is to determine if UNION injection is possible for the current target query, this is done by attempting to run a UNION query against a nonexistent table and checks to see if its specific error message is returned. This error message is also used to determine the type of database server we are exploiting because the error messages differ depending on the type of server being queried.

Second UNION injection test module is to determine whether the UNION query contains the correct number of columns. Once we attempt to query a valid table within the UNION query, the database should respond with an error indicating that our query must have the same number of columns as the original query. We attempt to brute-force the number of columns in the original query by continuing to add columns to the UNION query until this error goes away.

Third UNION injection test module is to determine the appropriate data type in each column position. Once we have the right number of columns in our UNION query, the database server should return an error indicating that the data types in each column must match those in the original query. Our scanner precedes to brute-force the correct data type combination by attempting every possible combination of data types within the allotted number of columns.

At the end, our scanner will report its success or fail results in the current query request for the whole SQL injection test modules and continue testing for the XSS vulnerability.

4.3.4 Cross Site Scripting (XSS)

Analyzing cross-site scripting (XSS) vulnerabilities can be a complex and time consuming task. To speed up the location of XSS vulnerabilities, we inject a test string containing JavaScript code into every HTTP parameter request done by our scanner, and then the scanner checks if the injected string is returned in the HTTP response.

Our scanner tests for XSS vulnerabilities in 2 ways. Simple java script alert message test request, and encoded java script alert message test request. XSS figure 4.8 represents the block diagram of XSS proposed methodology.

In simple java script alert message, the scanner simply injects a java script alert message and reads the reply message from the request, if this reply was the same as the injected alert message then this page request with its parameter is marked as vulnerable. The simple alert message is “<script>alert("XSS")</script>”.

In encoded java script alert message request, the scanner encodes the request and reads the reply message from the request, if this reply was the same as the injected alert message then this page request with its parameter is marked as vulnerable. There are two reasons for encoding the injected alert message. A first reason is to Avoid the Intrusion Detection System (IDS). A Second one is to bypass any filtering mechanism engines that may filter the normal java script code in the URL request. The encoded alert messages are divided into normal encoded messages and fully encoded messages. These messages are as follow:

1- Normal encoded messages:

```
%3cscript%3ealert('XSS')%3c/script%3e
%3c%53cript%3ealert('XSS')%3c/%53cript%3e
%3c%53cript%3ealert('XSS')%3c%2f%53cript%3e
%3cscript%3ealert('XSS')%3c/script%3e
%3cscript%3ealert('XSS')%3c%2fscript%3e
%3cscript%3ealert(%27XSS%27)%3c%2fscript%3e
%3cscript%3ealert(%27XSS%27)%3c/script%3e
%3cscript%3ealert("XSS")%3c/script%3e
%3c%53cript%3ealert("XSS")%3c/%53cript%3e
%3c%53cript%3ealert("XSS")%3c%2f%53cript%3e
%3cscript%3ealert("XSS")%3c/script%3e
%3cscript%3ealert("XSS")%3c%2fscript%3e
%3cscript%3ealert(%34XSS%34)%3c%2fscript%3e
%3cscript%3ealert(%34XSS%34)%3c/script%3e
```

2- Fully encoded messages:

```
?%22%3e%3c%73%63%72%69%70%74%3e%64%6f%63%75%
6d%65%6e%74%2e%63%6f%6f%6b%69%65%3c%2f%73%63%
72%69%70%74%3e
```

?%27%3e%3c%73%63%72%69%70%74%3e%64%6f%63%75%6d%65%6e%74%2e%63%6f%6f%6b%69%65%3c%2f%73%63%72%69%70%74%3e

%3e%3c%73%63%72%69%70%74%3e%64%6f%63%75%6d%65%6e%74%2e%63%6f%6f%6b%69%65%3c%2f%73%63%72%69%70%74%3e

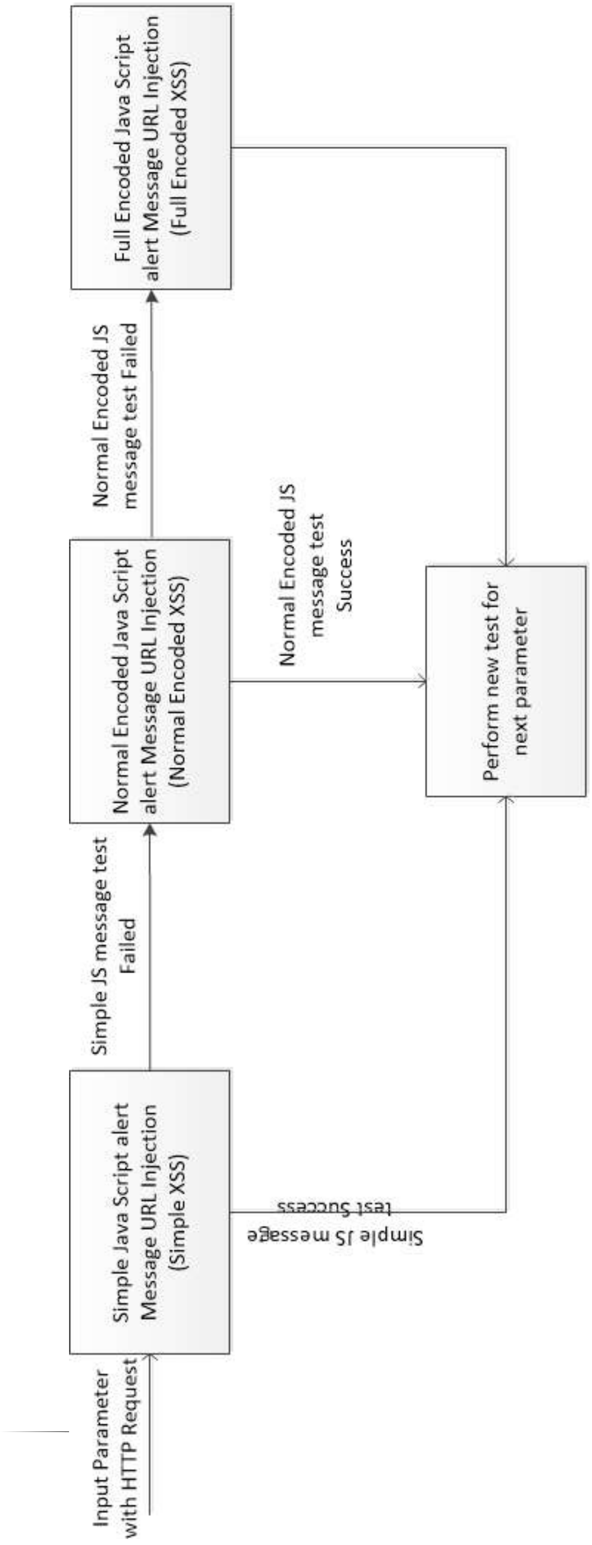

Input Parameter with HTTP Request
Simple Java Script alert Message URL Injection (Simple XSS)
Simple JS message test Failed
Simple JS message test Success
Normal Encoded Java Script alert Message URL Injection (Normal Encoded XSS)
Normal Encoded JS message test Failed
Normal Encoded JS message test Success
Full Encoded Java Script alert Message URL Injection (Full Encoded XSS)
Perform new test for next parameter

Figure 4.8: XSS methodology Block Diagram

Chapter 5
EXPERIMENTAL RESULTS

5.1 Introduction

In this chapter we validate the proposed Vulnerability Scanner. In order to do that we developed experiments to measure the performance of our scanner and used some performance metrics to measure the performance of it. The performance metrics used are:

- **Accuracy:** Accuracy measures the rate of generating correct results. If there are N vulnerabilities being monitored by the scanner and there are x of these vulnerabilities for which the scanner predicts correctly if the path of these packets is hacked or not then:

  Accuracy (%) = (x/N)*100%

  Accuracy is also defined as the number of vulnerabilities detected by the system (True Positive) divided by the total number of vulnerabilities present in the test set.
- **False positive rate:** This is the rate at which the scanner states that the HTTP request has vulnerability while in fact the HTTP request has no vulnerability. If the scanner tests N HTTP request for possible vulnerabilities. In addition, the scanner detects N1 positive vulnerabilities that N1 have vulnerabilities out of N HTTP requests. If x of these N1 are false, then:

  False positive rate (%) = (x/N1)*100 %

- **False negative rate:** This is the rate at which the scanner dose not detects vulnerability while in fact the HTTP request has vulnerability. If the scanner test N URL request for possible vulnerabilities. In addition, the scanner detects N1 negative vulnerabilities that N1 requests have vulnerabilities out of N HTTP requests. If x of these N1 requests are false, then:

  False negative rate (%) = (x/N1)*100%

In order to validate our scanner we compare the performance results of it with performance of similar tools in the literature. The comparison shows how our scanner performs compared to other similar tools.

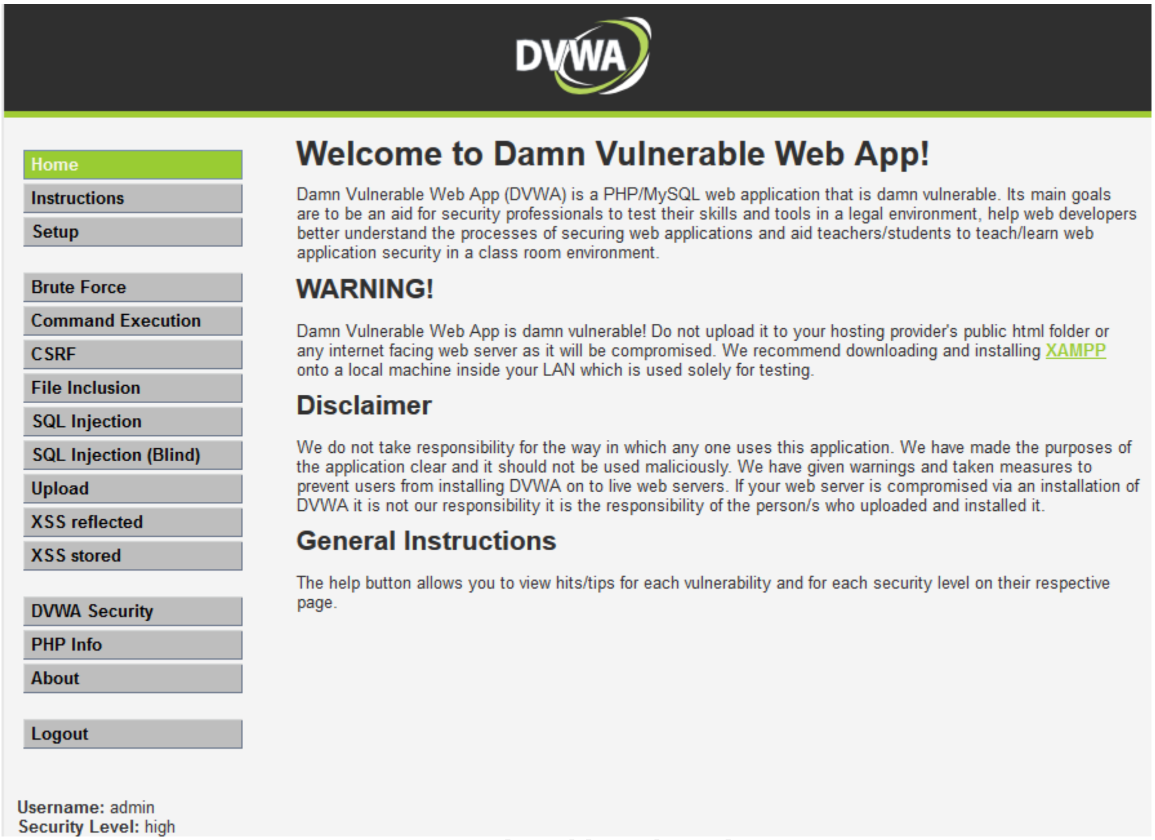
To test our scanner, we used developed test beds. These test beds are common vulnerable web applications that are built by security developers and tested by nature for web penetration testing and hacking. These vulnerable web applications support to be installed and configured under multiple web servers and multiple database servers. The vulnerabilities on these web applications are known so we can apply our accuracy, false negative and false positive metrics as required.

5.2 Test Beds Description

This section describes the used test beds to test our scanner. These test beds are built in PHP and can be configured with multiple database servers.

1- **DVWA (Dam Vulnerable Web Application)** [74]- this vulnerable PHP/MySQL web application is one of the famous web applications used for or testing your skills in web penetration testing and your knowledge in manual SQL Injection, XSS, Blind SQL Injection, etc. DVWA is developed by Ryan Dewhurst a.k.a ethicalhack3r and is part of RandomStorm Open Source project. Figure 5.1 shows the main page of the application.

2- **Mutillidae** [75]- is a free and open source web application for website penetration testing and hacking which was developed by Adrian “Irongeek” Crenshaw and Jeremy “webpwnized” Druin. It is designed to be exploitable and vulnerable and ideal for practicing your Web Fu skills like SQL injection, cross site scripting, HTML injection, Javascript injection, clickjacking, local file inclusion, authentication bypass methods, remote code execution and many more based on OWASP (Open Web Application Security) Top 10 Web Vulnerabilities. Figure 5.2 shows the main page of this web application.



Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Disclaimer

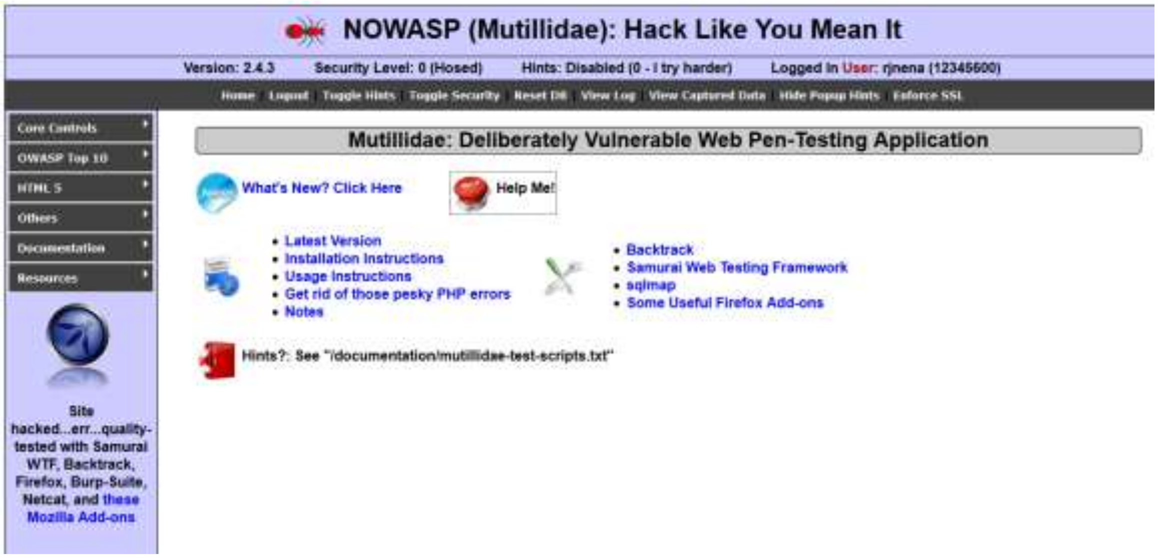
We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

Username: admin
Security Level: high

Figure 5.1: Dam vulnerable web application


NOWASP (Mutillidae): Hack Like You Mean It
Home | Logout | Toggle Hints | Toggle Security | Reset DB | View Log | View Captured Data | Hide Popup Hints | Enforce SSL
Core Controls
OWASP Top 10
HTML 5
Others
Documentation
Resources
What's New? Click Here
Help Me!

Version: 2.4.3 Security Level: 0 (Hosed) Hints: Disabled (0 - I try harder) Logged In User: rjnena (12345600)

Mutillidae: Deliberately Vulnerable Web Pen-Testing Application

- Latest Version
- Installation Instructions
- Usage Instructions
- Get rid of those pesky PHP errors
- Notes

- Backtrack
- Samurai Web Testing Framework
- sqlmap
- Some Useful Firefox Add-ons

Hints?: See "/documentation/mutillidae-test-scripts.txt"

Site hacked...err...quality-tested with Samurai WTF, Backtrack, Firefox, Burp-Suite, Netcat, and these Mozilla Add-ons

Figure 5.2: Mutillidae vulnerable web application

3- **OWASP InsecureWebApp** - is a web application that includes common web application vulnerabilities. It is a target for automated and manual penetration testing, source code analysis, vulnerability assessments and threat modeling.

4- **Web Security Dojo** [76] – is a free open-source self-contained training environment for Web Application Security penetration testing that includes tools and vulnerable web applications targets. Some of included vulnerable targets are OWASP's WebGoat, Google's Gruyere, Damn Vulnerable Web App, Hacme Casino, OWASP InsecureWebApp, and w3af's test website.

5.3 Results of our scanner with test beds

In this section we discuss and describe the output results of our scanner for the selected test beds. Choosing only four test beds for out tests and analysis was sufficient enough because the selected test beds covers all possible vulnerabilities implemented in the insecure version of any web application.

In order to start our tests we have configured the web server and implemented all test beds with their default configurations. This insures that all implemented vulnerabilities are available for test and no changes on the database parameters and data were changed.

Second step of testing procedure is to crawl our vulnerable web applications with our described method in section 4.3. The result of this step will be a list of all HTTP requests in one log file for each vulnerable web application.

Thirdly, we have started our vulnerability scanner with each log file and started to get report and results for analysis. After each successful test, the scanner results were reviewed to determine which vulnerabilities and what kind of each vulnerability was detected.

Results were classified as false-negatives or false-positives by following the classification procedure described in section 5.1.

The results obtained from the tests that targeted the test beds for SQL injection vulnerability are listed in Table 5.1.

	No. of Detected Vulnerabilities	No. of known Vulnerabilities	False Negative	False Positive
Test bed 1	18	20	2	1
Test bed 2	34	35	1	2
Test bed 3	36	40	4	1
Test bed 4	15	15	0	1
Total	103 (93.6%)	110	7 (6.4%)	5 (4.8%)

Table 5.1: SQL injection Results

As shown in Table 5.1, our scanner detected 103 SQL injection vulnerabilities out of 110 known implemented vulnerabilities, so the corresponding false negative is only 7 vulnerabilities. As well as, the false positive vulnerabilities of the detected ones were only 5. The overall detection rate was 93.6 % and the corresponding false negative rate was 6.4 % and the false positive rate was 4.8 %. Figure 5.3 represents the graph of the corresponding result.

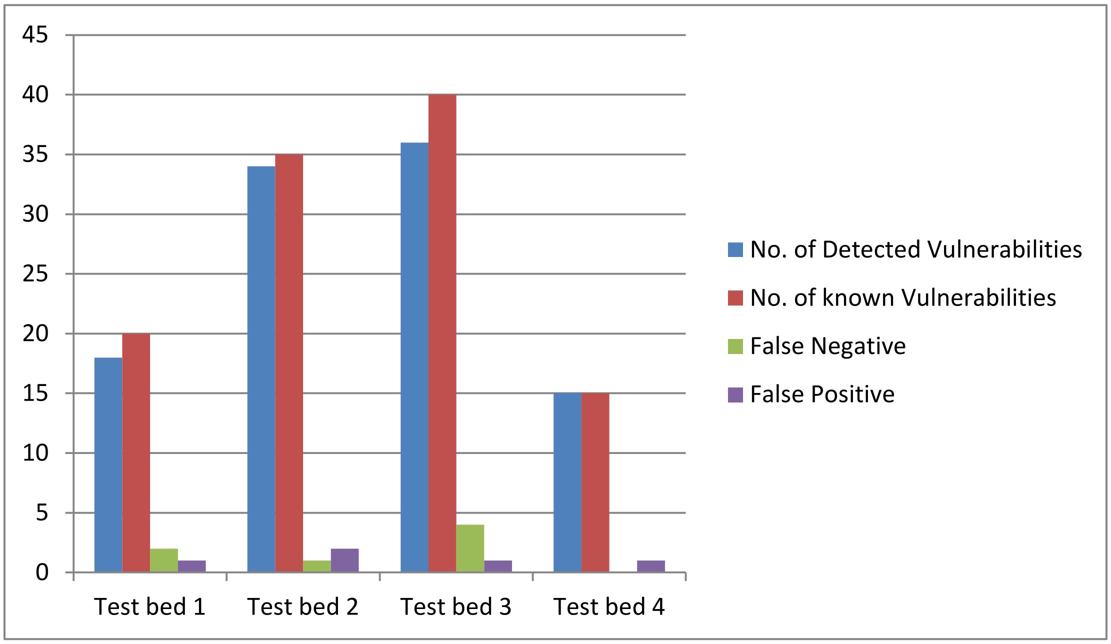


Figure 5.3: SQL injection Results Graph

The results obtained from the tests that targeted the test beds for Cross Site Scripting (XSS) vulnerability are listed in Table 5.2.

	No. of Detected Vulnerabilities	No. of known Vulnerabilities	False Negative	False Positive
Test bed 1	33	35	2	1
Test bed 2	24	25	1	1
Test bed 3	44	45	1	2
Test bed 4	18	20	2	1
Total	119 (95.2%)	125	6 (4.8%)	5 (4.2%)

Table 5.2: Cross Site Scripting (XSS) Results

As shown in Table 5.2, our scanner detected 119 XSS vulnerabilities out of 125 known implemented vulnerabilities, so the corresponding false negative is only 6 vulnerabilities. As well as, the false positive vulnerabilities of the detected ones were only 5. The overall detection rate was 95.2 % and the corresponding false negative rate was 4.8 % and the false positive rate was 4.2 %. Figure 5.4 represents the graph of the corresponding result.

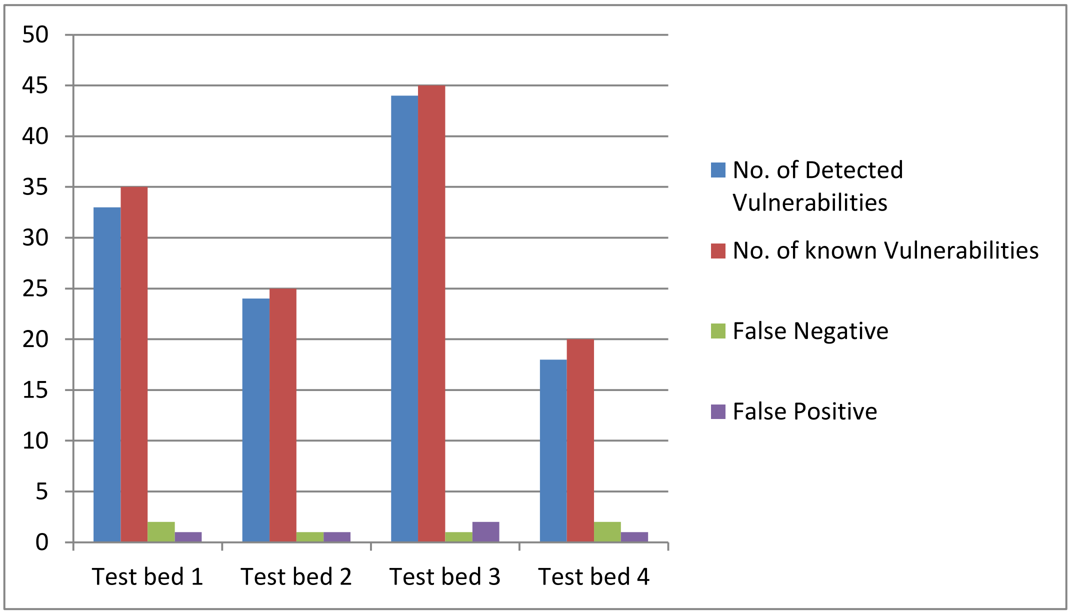


Figure 5.4: Cross Site Scripting (XSS) Results Graph

5.4 Other Web applications vulnerability scanners

In this section we describe the other web applications vulnerability scanners used to compare results with our scanner. These scanners vary from commercial to open source vulnerability scanners. As well as, these scanners supports different criteria as some of them has SQL Injection and XSS scanning ability.

1- Acunetix web vulnerability scanner:

Acunetix scanner [8] divides type of scanning according to the severity of the type of web attack. It divides in four type's high, medium, low and informational severity. Acunetix is used to detect various types of web vulnerabilities as below.

i) SQL injection.

ii) Cross site scripting.

iii) CGI scripting.

iv) Firewalls and SSL.

v) URL redirection.

SQL injection and Cross site scripting scans are comes under the high severity type as they are considered most dangerous attacks in the web security. Other attacks are categorized according to their severity on the web services.

Although this scanner does little bit extra amount of scanning, it is very slow as compare to the other tool available in market and slower than our scanner as well.

2- SQLmap:

Sqlmap [77] is an SQL injection scanner build in Python. The aim of this tool is to detect SQL injection vulnerabilities and take advantage of these vulnerabilities on web application. Sqlmap initially detect the whole loop in the target site and then use variety of option to perform extensive back-end database management, enumerate users, dump entire or specific DBMS, retrieve DBMS session user and database, read specific file on the file system etc. SQLmap is bit faster than acunetix web scanner but still slower than our scanner, and it also make very few URL injection in to the database as compare to our scanner.

3- **Wapiti:**

Wapiti [78] is command line based tool build in python and uses a Python library called lswww. This is the spider library helps to crawl each page on given web site. Wapiti allows us to inspect the security of our web site. This tool also used html Tidy lib to clean the html pages which are not well formatted. This library helps a lot to extract information from bad-coded html web pages. Basically it does black-box scan. Wapiti scans the all Web Pages available on the web site and try to find out scripts and form where it can inject data to check how many types of attack are possible on selected injection point.

Wapiti can detect SQL injection and XSS (Cross Site Scripting) injection. Wapiti has one of the best features that it's able to differentiate temporary and permanent XSS vulnerabilities.

4- **Paros:**

Paros [12] is used for web application security assessment. Paros is written in Java, and people generally used this tool to evaluate the security of their web sites and the applications that they provide on web site. It is free of charge, and using Paros's you can exploit and modified all HTTP and HTTPS data among client and server along with form fields and cookies. In brief the functionality of scanner is as below.

According to web site hierarchy server get scan, it checks for server misconfiguration. They add this feature because some URL paths can't be recognized and found by the crawler. The other automatic scanners are not able to do that. Basically to work this functionality Paros navigates the site and rebuilds the website hierarchy. Presently Paros does three types of server configuration checks. HTTP PUT, Directory index, and obsolete file exist. Paros also provides log file, which is create when all the HTTP request and reply pass through Paros. In log panel Paros shows back as request and reply format.

5- Pixy:

Pixy [2] is the second tool that is written in Java. Pixy does automatic scans for PHP 4 for the detection for SQL injection and XSS attacks. The major disadvantage of Pixy is that it only works for PHP 4 and not for OOPHP 5. Pixy take whole PHP file as an input and produce a report that shows the possible vulnerability section in that PHP file along with some additional information to understand attack.

While SQL injection analysis Pixy divides result in three categories: untainted, weakly tainted, and strongly tainted. It also provide dependence graph and dependence value. Dependent value is nothing but the list of points in program on which the value of variables is depends.

5.5 Performance evaluation of our scanner with other vulnerability scanners

This section states and describes the comparison in performance and accuracy between our scanner and the other web vulnerability scanners described in the above section.

The comparison between scanners has two criteria. First, features comparisons which are the supported database systems, development language, and attack types in each scanner. Second, the time required by each scanner to complete its vulnerability scan test on known number of vulnerabilities and the total number of detected vulnerabilities of these known ones.

1- Features comparisons:

Table 5.3 lists the features comparison between vulnerability scanners.

Vulnerability scanner	Supported Database Systems	Development language	Attack types
Our scanner	ORACLE, MS SQL Server, MySQL	Perl	SQLi, XSS
Acunetix	ORACLE, MS SQL Server, MySQL, PSQL, MS Access	-	5 (section 5.4)
SQLmap	ORACLE, MS SQL Server, MySQL, PSQL	Python	3 (section 5.4)
Wapiti	ORACLE, MS SQL Server, MySQL	Python	SQLi, XSS
Paros	ORACLE, MS SQL Server, MySQL	Java	SQLi, XSS
Pixy	ORACLE, MS SQL Server, MySQL	Java	SQLi, XSS

Table 5.3: Features comparison between vulnerability scanners.

2- Performance test:

In order to test the time performance between the vulnerability scanners, we have tested these scanners on known number of vulnerabilities in vulnerable web application. The number of vulnerabilities was 100 vulnerability that are SQL injection and Cross Site Scripting vulnerabilities.

In this test, we have counted the number of detected vulnerabilities for each scanner from these known vulnerabilities in order to calculate its accuracy in vulnerability detection. Table 5.4 describes this measure test. Figure 5.5 describes the comparison between these measures and scanners.

Vulnerability scanner	**Number of vulnerabilities**	**Vulnerabilities type**	**Execution time (minutes)**	**Number of detected vulnerabilities**
Our scanner	100	SQLi, XSS	2m 20sec	95
Acunetix	100	SQLi, XSS	25m	85
SQLmap	100	SQLi, XSS	2m 35sec	90
Wapiti	100	SQLi, XSS	5m 40sec	70
Paros	100	SQLi, XSS	6m 10sec	50
Pixy	100	SQLi, XSS	4m	45

Table 5.4: Performance test of vulnerability scanners

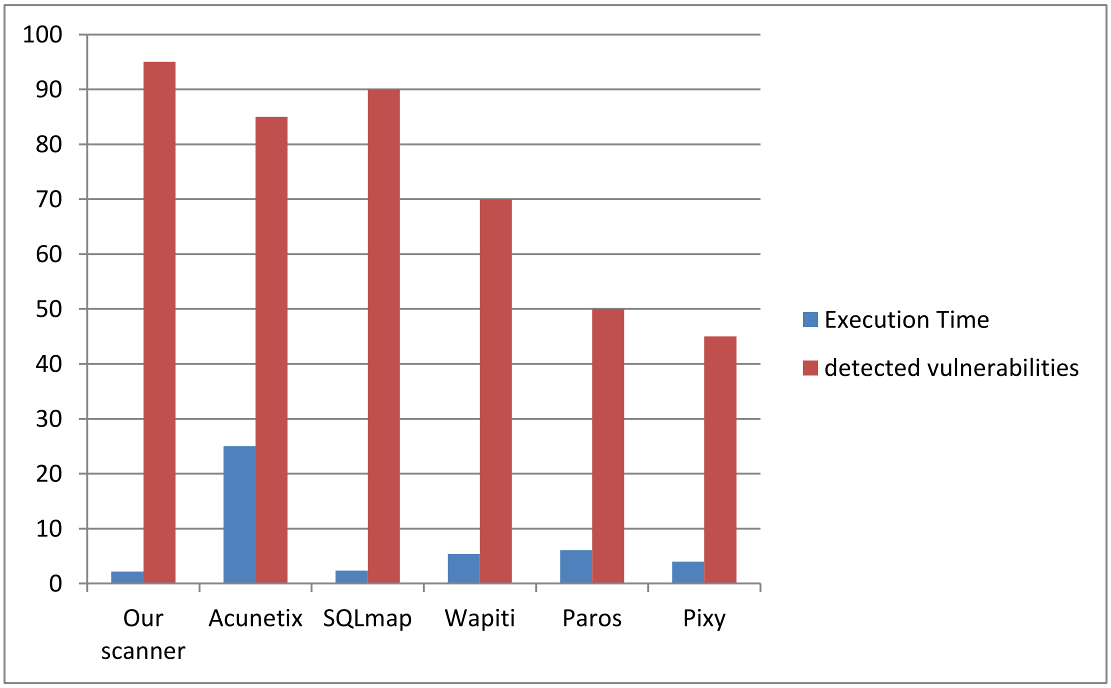


Figure 5.5: Performance test of vulnerability scanners

Table 5.6 lists the false positive rate comparison between vulnerability scanners, as well as the vulnerabilities detection rate of the same vulnerabilities number described above.

Vulnerability scanner	False Positive Rate – FPR (%)	Vulnerabilities detection rate (%)
Our scanner	1.1	95
Acunetix	1.8	85
SQLmap	1.4	90
Wapiti	5.2	70
Paros	5.7	50
Pixy	5.8	45

Table 5.5: false positive rate comparison of vulnerability scanners

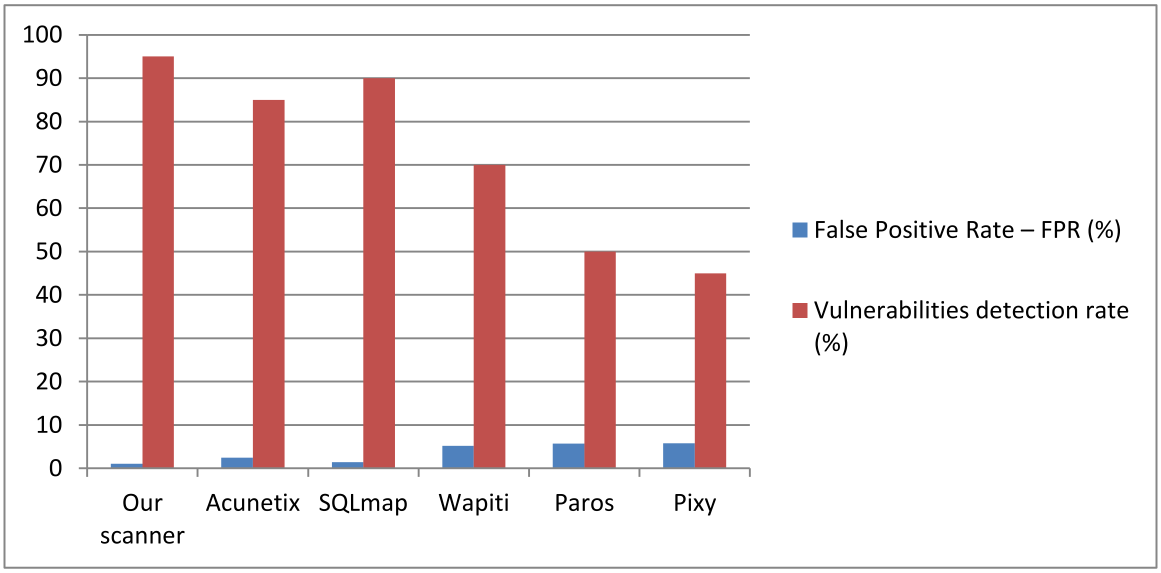


Figure 5.6: false positive rate comparison of vulnerability scanners

Chapter 6
CONCLUSION
AND
FUTURE WORK

Conclusion

There are many web applications vulnerability scanners implemented for analyzing and detecting security holes in web applications. And because security is still one of the most important issues all across the globe in our thesis we have implemented a complete approach that scans for the most important vulnerabilities for web applications, namely SQL Injection and Cross Site Scripting (XSS). Since XSS and SQL injection vulnerabilities in web applications has huge risk not only for the web applications but also for users as well. We studied many existing approaches to detect and prevent these vulnerabilities in an application, giving a brief note on their advantages and disadvantages. All the approaches followed by different authors' leads to a very interesting solution; however some failures are associated with almost each one of them at some point. Furthermore these scanners don't support all web applications, many of them supports only known web applications with known vulnerabilities.

In this thesis we are providing a vulnerability scanning and analyzing tool of various kinds of SQL injection and Cross Site Scripting (XSS) attacks. Our approach can be used with any web application not only the known ones. As well as it supports the most famous Database management servers, namely MS SQL Server, Oracle, and MySQL.

We validate the proposed vulnerability scanner by developing experiments to measure its performance. We used some performance metrics to measure the performance of the scanner which include accuracy, false positive rate, and false negative rate. We also compare the performance results of it with performance of similar tools in the literature.

Future Work

- Develop a GUI for the scanner script, so make it easy for anyone to install and use the scanner.
- Add full support for all known XSS detection techniques.
- Implement a local proxy module to be included in the scanner work, which will make a full vulnerability analysis environment.

REFERENCES

[1] M. Curphey and R. Arawo. Web application security assessment tools. Security & Privacy, IEEE, 4(4):32–41, July-Aug. 2006.

[2] N. Jovanovic, C. Kruegel, and E. Kirda, “Pixy: A static analysis tool for detecting web application vulnerabilities (short paper),” In 2006 IEEE Symposium on Security and Privacy, Oakland, CA: May 2006.

[3] E. Fong, R. Gaucher, V. Okun, P. E. Black, and E. Dalci. Building a test suite for web application scanners. Hawaii International Conference on System Sciences, 0:479, 2008.

[4] Fonseca, J. CISUC, Univ. of Coimbra, Coimbra, Portugal Vieira, M. and Madeira, H. Vulnerability & attack injection for web applications. Dependable Systems & Networks, 2009. DSN '09. IEEE/IFIP International Conference on, 93-102, 2009.

[5] M. Vieira, N. Antunes, and H. Madeira. Using Web Security Scanners to Detect Vulnerabilities in Web Services Using Web Security Scanners to Detect Vulnerabilities in Web Services. IEEE/IFIP Intl Conf. on Dependable Systems and Networks, Lisbon, Portugal, June 2009.

[6] Ounce, http://www.ouncelabs.com/

[7] Pixy, http://pixybox.seclab.tuwien.ac.at/pixy/

[8] Acunetix Web Vulnerability Scanner, 2012, http://www.acunetix.com/vulnerability-scanner/

[9] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic. SecuBat: A Web Vulnerability Scanner. ACM 1-59593-323-9/06/0005 WWW 2006,Edinburgh, Scotland.

[10] Nikto. Web Server Scanner. http://www.cirt.net/code/nikto.shtml.

[11] Tenable Network SecurityTM. Nessus Open Source Vulnerability Scanner Project. http://www.nessus.org/.

[12] Paros proxy vulnerability scanner. Available at: http://www.parosproxy.org/.

[13] Y. Huang, S. Huang, and T. Lin. Web Application Security Assessment by Fault Injection and Behavior Monitoring. 12th ACM International World Wide Web Conference, May 2003.

[14] W. G. Halfond and A. Orso. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. In Proceedings of the IEEE and ACM International Conference on Automated Software Engineering (ASE 2005), Long Beach, CA, USA, Nov 2005.

[15] W. G. Halfond and A. Orso. Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks. In Proceedings of the Third International ICSE Workshop on Dynamic Analysis (WODA 2005), pages 22–28, St. Louis, MO, USA, May 2005.

[16] Z. Su and G. Wassermann. The Essence of Command Injection Attacks in Web Applications. In The 33rd Annual Symposium on Principles of Programming Languages (POPL 2006), Jan. 2006.

[17] G. T. Buehrer, B. W. Weide, and P. A. G. Sivilotti. Using Parse Tree Validation to Prevent SQL Injection Attacks. In International Workshop on Software Engineering and Middleware (SEM), 2005.

[18] Y. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee, and S. Y. Kuo. Securing Web Application Code by Static Analysis and Runtime Protection. In Proceedings of the 12th International World Wide Web Conference (WWW 04), May 2004.

[19] V. B. Livshits and M. S. Lam. Finding Security Errors in Java Programs with Static Analysis. In Proceedings of the 14th Usenix Security Symposium, pages 271–286, Aug. 2005.

[20] S. Saha. Consideration Points: Detecting Cross-Site Scripting. (IJCSIS) International Journal of Computer Science and Information Security, Vol. 4, No. 1 & 2, 2009.

[21] T. Pietraszek, and C. V. Berghe, “Defending against Injection Attacks through Context-Sensitive String Evaluation,” In Proceeding of the 8th International Symposium on Recent Advance in Intrusion Detection (RAID), September 2005.

[22] Z. Su and G. Wassermann, “The essence of command Injection Attacks in Web Applications,” In Proceeding of the 33rd Annual Symposium on Principles of Programming Languages, USA: ACM, January 2006, pp. 372-382.

[23] D. Balzarotti, M. Cova, V. V. Felmetsger, and G. Vigna, “Multi-Module Vulnerability Analysis of Web-based Applications,” In proceeding of 14th ACM Conference on Computer and Communications Security, Alexandria, Virginia, USA: October 2007.

[24] G. Wassermann, Z. Su. Static Detection of Cross-Site Scripting Vulnerabilities. ICSE’08, May 10–18, 2008, Leipzig, Germany.

[25] Fu, X., Lu, X., Peltsverger, B., Chen, S., Qian, K., and Tao, L., A Static Analysis Framework for Detecting SQL Injection Vulnerabilities. Proc. 31st Annual International Computer Software and Applications Conference 2007 (COMPSAC 2007), 24-27 July (2007), 87-96.

[26] T. Haixia, Y. and Zhihong, N., A database security testing scheme of web application. Proc. of 4th International Conference on Computer Science & Education 2009 (ICCSE '09), 25-28 July (2009), 953-955.

[27] M. Ruse, T. Sarkar, and T. Basu. Analysis & Detection of SQL Injection Vulnerabilities via Automatic Test Case Generation of Programs. Proc. 10th Annual International Symposium on Applications and the Internet (2010), 31-37

[28] S. Thomas, L. Williams, and T. Xie. On automated prepared statement generation to remove SQL injection vulnerabilities. Information and Software Technology, Volume 51 Issue 3, March (2009), 589–598 1.

[29] K. Kemalis and T. Tzouramanis. SQL-IDS: A Specification-based Approach for SQLinjection Detection. SAC'08. Fortaleza, Ceará, Brazil, ACM (2008), 2153 2158.

[30] R.A. McClure and I.H. Kruger, SQL DOM: compile time checking of dynamic SQL statements. 27th International Conference on Software Engineering (ICSE 2005), 15-21 May (2005), 88- 96.

[31] S. Ali, S.K. Shahzad, and H. Javed. SQLIPA: An Authentication Mechanism Against SQL Injection. European Journal of Scientific Research, Vol. 38, No. 4 (2009), 604-611.

[32] Z. Su and G. Wassermann, "The essence of command Injection Attacks in Web Applications," In Proceeding of the 33rd Annual Symposium on Principles of Programming Languages, USA: ACM, January 2006, pp. 372-382.

[33] A. Tajpour, M. Masrom, M.Z. Heydari, and S. Ibrahim. SQL injection detection and prevention tools assessment. Proc. 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT'10) 9-11 July (2010), 518-522

[34] P. Bisht, P. Madhusudan and V.N. Venkatakrishnan. CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. ACM Transactions on Information and System Security, Volume 13 Issue 2, (2010).

[35] Y. Huang, S. Huang, T. Lin, and C. Tsai, "Web application security assessment by fault injection and Behavior Monitoring," In Proceeding of the 12th international conference in World Wide Web, ACM, New York, NY, USA: 2003, pp.148-159.

[36] Y. Huang, F. Yu, C. Hang, C. Tsai, D. Lee, and S. Kuo. "Verifying Web Application using BoundedModel Checking," In Proceedings of the International Conference on Dependable Systems and Networks, 2004.

[37] "Web Application Security Testing – AppScan 3.5," Sanctum Inc., http://www.sanctuminc.com.

[38] "Web Application Security Assessment," SPI Dynamics Whitepaper, SPI Dynamics, 2003.

[39] “InterDo Version 3.0,” Kavado Whitepaper, Kavado Inc. , 2003

[40] Y. Huang, F. Yu, C. Hang, C. Tsai, D. Lee, and S. Kuo,“Securing web application code by static analysis and runtime protection,” In Proceedings of the 13 th International World Wide Web Conference, 2004.

[41] D. Scott, and R. Sharp, “Abstracting Application-Level Web Security,” In Proceeding 11thinternational World Wide Web Conference, Honolulu, Hawaii: 2002, pp. 396-407

[42] Fonseca, J. CISUC - Polithecnic Inst. of Guarda, Guarda Vieira, M. ; Madeira, H. Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks. Dependable Computing, 13th Pacific Rim International Symposium. 2007.

[43] D. Balzarotti, M. Cova, V. V. Felmetsger, and G. Vigna, “Multi-Module Vulnerability Analysis of Web-based Applications,” In proceeding of 14th ACM Conference on Computer and Communications Security,Alexandria, Virginia, USA: October 2007.

[44] T. Pietraszek, and C. V. Berghe, “Defending against Injection Attacks through Context-Sensitive String Evaluation,” In Proceeding of the 8th International Symposium on Recent Advance in Intrusion Detection (RAID), September 2005.

[45] Information Assurance Tools Report – Vulnerability Assessment. Sixth Edition, Revision by Karen Mercedes Goertzel, with contributions from Theodore Winograd. 2011.

[46] "The Three Tenants of Cyber Security". U.S. Air Force Software Protection Initiative. Retrieved 2009-12-15.

[47] The OWASP Foundation. OWASP Top 10 - 2010, 2010.

[48] D. Shelly. Using a Web Server Test Bed to Analyze the Limitations of Web Application Vulnerability Scanners. Virginia Polytechnic Institute and State University. 2010.

[49] A. Klein. DOM Based Cross Site Scripting or XSS of the Third Kind. Available at: http://www.webappsec.org/projects/articles/071105.shtml, July 2005.

[50] CodeScan Labs. CodeScan Developer - Security at the Source. Available at: http://www.codescan.com/, 2009.

[51] Y. Huang and D. Lee. Web Application Security-Past, Present, and Future. Pages 183–227. 2005.

[52] Y. W. Huang, C. H. Tsai, D. Lee, and S. Y. Kuo. Non-detrimental web application security scanning. In Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on, pages 219–230, Nov. 2004.

[53] N. Antunes and M. Vieira. Detecting SQL Injection Vulnerabilities in Web Services. In Dependable Computing, 2009. LADC '09. Fourth Latin-American Symposium on, pages 17–24, Sept. 2009.

[54] D. Byrne and E. Duprey. Grendel-Scan. Available at: http://www.grendel-scan.com/.

[55] N. Surribas. Wapiti. Available at: http://www.ict-romulus.eu/web/wapiti/.

[56] A. Riancho. W3AF-Web Application Attack and Audit Framework. Available at: http: //w3af.sourceforge.net/.

[57] G. F. Lyon. NMAP.ORG. Available at: http://nmap.org/.

[58] Cenzic, Inc. Hailstorm Core and Hailstorm Starter. Available at: http://www.cenzic.com, 2010.

[59] N-Stalker. N-Stalker The Web Security Specialists. Available at: http://nstalker.com, 2010.

[60] Mavituna Security Ltd. Netsparker Web Application Security Scanner. Available at: http://www.mavitunasecurity.com, 2010.

[61] PortSwigger. Burp Scanner. Available at: http://portswigger.net/.

[62] IBM. Rational AppScan Standard Edition. Available at: http://www-01.ibm.com, 2010.

[63] BuyServers Ltd. Falcove Web Vulnerability Scanner. Available at: http://www.buyservers.net, 2008.

[64] Carahsoft Technology Corp. HP WebInspect software. Available at: http://www.carahsoft.com/hp/products/webinspect, 2009.

[65] NT OBJECTives. NTOSpider. Available at: http://www.ntobjectives.com, 2010.

[66] Perl Wikipedia page. Available at: http://en.wikipedia.org/wiki/Perl, last modified 2013.

[67] Matt Bishop, “Introduction to Computer Security,” Publisher: Prentice Hall PTR, Pub Date: October 26, 2004, ISBN: 0-321-24744-2.

[68] A. Humphrey, “Network Vulnerability Analysis”; article published: year 2007, month 05.

[69] Search mid-market security, http://searchmidmarketsecurity.techtarget.com/definition/vulnerability-analysis

[70] C. Sample and I. Poynter, “Quantifying Vulnerabilities in the Networked Environment: Methods and Uses,” 2000;

[71] Y. Dong, Y. Hou, and Z. Zhang, “A Server-Based Performance Measurement Tool within Enterprise Networks,’’ J. Performance Evaluation, vols. 36–37, Nov. 15, 1999, pp. 233–247.

[72] S. Hariri, G. Qu, Tushneem Dharmagadda, M. Ramkishore;“Vulnerability Analysis of Faults and Attacks in Large-Scale Network”, IEEE Security and Privacy magazine, October & November Issue 2003.

[73] Denial of Service, tech. report, Malaysian Computer Emergency Response Team (My-CERT) 2003.

[74] DVWA (Dam Vulnerable Web Application). Available at: http://www.dvwa.co.uk/.

[75] Mutillidae vulnerable web application. Available at: http://sourceforge.net/projects/mutillidae/

[76] Web Security Dojo. Available at: http://www.mavensecurity.com/web_security_dojo/

[77] SQLmap vulnerability scanner. Available at: http://sqlmap.org/.

[78] Wapiti vulnerability scanner. Available at: https://launchpad.net/ubuntu/precise/i386/wapiti/1.1.6-3.