Structured Query Language Injection (SQLI) Attacks: Detection and Prevention Techniques in Web Application Technologies

Wisdom Kwawu Torgby
Computer Science Department
School of Applied Sciences, Accra Polytechnic,
Accra, Ghana

Nana Yaw Asabere
Computer Science Department
School of Applied Sciences, Accra Polytechnic,
Accra, Ghana

ABSTRACT
This paper investigates and reports on web application vulnerabilities with a specific focus on Structured Query Language Injection (SQLI) attacks and measures and how to counter such threats. SQLI attacks cause very serious dangers to web applications, they make it possible for attackers to get unhindered access to the primary source of data which is in the database and possibly the very sensitive information that the database contains. Even though practitioners and researchers in the web application security field have proposed a range of techniques to get to the bottom of the SQLI attack challenge, presently adopted approaches have either resolved the problem to some extent or have inadequacies that prevent their use and adoption. To help address this challenge, this paper presents a broad review of SQL injection attacks. An appraisal of current detection and prevention techniques against SQL injection attacks are also presented. Furthermore, a vulnerability assessment was conducted on the Centre for Computational Intelligence (CCI) Website as a case study. A snippet code that can be used to redesign the CCI website as a protective measure to counter threats of SQLI was proposed. An examination of this paper indicates that current solutions being promoted may not address the problem, and that web application firewalls provides the answer to SQLI attacks.

General Terms
Attacks, Threats

Keywords
Web Application, Website, Security, Structured Query Language Injection (SQLI), Vulnerabilities

1. INTRODUCTION
Currently, web applications are playing a magnificent role in providing vital information to all manner of users on a global scale. Web applications usually consist of a three tier architecture. The database, which is the most valuable assets in any organization, is found in the third tier. Due to the rapid adaptation of web applications, various attacks against web applications are possible. Most organizations are mapping their business from a room to the world with the help of these web apps [1]. More and more web applications suffer the presence of cross-site scripting vulnerabilities that could be exploited by attackers to access sensitive information (such as credentials or credit card numbers). Hence proper tests are required to assess the security of web applications [2].

The Internet, and in particular the World Wide Web (WWW), have become one of the most common communication media in the world. Millions of users connect every day to different web-based applications such as Google and Yahoo to perform financial operations, search for information, interact with each other, conduct business, pay taxes, exchange messages and many more. Some of these critical web-based services are targeted by several malicious users intending to exploit possible weaknesses and vulnerabilities, which could cause not only the disruption of the service, but also compromise the information belonging to users and organizations. Most of the time, these malicious users succeed in exploiting different types of vulnerabilities and the consequences can be disastrous. Most of these vulnerabilities are directly related with the web-based application's lack of quality as a result from a poorly implemented Software Development Life Cycle (SDLC) [3].

According to Tian *et al.* [4] web application software security becomes more and more important as a result information access through web applications. Recent investigations show that web application vulnerabilities have become the largest security threat. The Web Sense Security Report shows that in the first half of year 2008, the most popular websites that have been utilized by various hackers to run malicious code were above 75%. Detecting and solving vulnerability is the effective way to enhance web security [4].

Much effort in the past decade has been spent on mitigating web application vulnerabilities. According to Scholte *et al*. [5] current techniques focus mainly on sanitization: either on automated sanitization, the detection of missing sanitizers, the correctness of sanitizers, or the correct placement of sanitizers. However, these techniques lack precision and require significant modifications to the client and/or server infrastructure. Additionally, these techniques are either not able to prevent new forms of input validation vulnerabilities such as HTTP Parameter Pollutions that's are accompanied with large runtime overhead [5]. Generally, people are unaware of the wide variety of techniques at the disposal of attackers who are always making efforts to take advantage of SQLI vulnerabilities.

This paper presents a research analysis of SQLI attacks and countermeasures and the attack techniques that are used to take advantage of the security design flaws. The purpose of this paper is to specifically focus on SQLI attacks. The simple reason that web application attacks are directed at websites almost on daily basis makes this research interesting, especially because it appears not many people are well informed of the SQLI attack, which is the most devastating attack in web application technologies. Consequently, this paper proposes a snippet of suggested codes for redesigning the web firewall of the CCI Website as a case study for protection against SQL injection attack. The paper is presented in five sections. After the Introduction in Section 1,

Section 2 presents a Literature Review of the SQLI Attacks, Section 3 discusses the Importance and Benefits of Web Firewalls in Prevention of SQLI Attacks, Section 4 elaborates on Related Work, Section 5 presents the Vulnerability Assessment of the CCI Website as a Case Study. The paper is finally concluded in Section 6.

2. SQLI ATTACKS

Basically SQLI attacks are deceptive ploys used to introduce input provided by the user. The term SQL stands for Structured Query Language. SQL is an exceptional coding language which is used to send queries to databases. In most cases web applications regularly make use of input provided by the user to produce SQL statements for dynamic web page demands. In situations where an application fails to accurately cleanse input provided by the user it is likely that a web attacker can change the database statements at the background [6][7].

SQLI attack can also be described as the process of sending SQL code to a web application that is not planned by the developer of the application with the express purpose of causing harm. This attack technique takes advantage of the security flaw occurring in the database level of a web application. The attacker takes advantage of the weakness inherent in the application and executes illegal SQL commands by exploiting insecure codes. SQL injection makes it feasible for the attacker to transmit crafted user name and password fields which will modify the query [6][7]. This vulnerability has been described largely as one of the most grave security dangers for web applications. When an application becomes susceptible to SQLI attack, there is the possibility for a wicked attacker to get total control and to obtain access to the database. Usually these databases hold in storage sensitive customer or user information. Consequently, the security breaches may include losing confidential information, stealing of identity and situations where people can commit fraud [1][2][4][6].

The following are some of the features of web application that may be liable to SQL injection attacks:

Login pages, search pages, support and product request forms, shopping carts, feedback forms etc.

Input is taken from the user in a web form and sent to a server-side script through Hypertext Transfer Protocol (HTTP) methods such as Power On Self-Test (POST) and GET. A process is initiated that establishes connection to the database. A query is generated for the database and the result is retrieved. The outcome is then transmitted back to the attacker.

Example: A web based authentication form may have a code which will appear similar to the following:

SQLQuery = "SELECT Username FROM Users WHERE

Username = ' " & strUsername & " ' AND Password =

& srPassword & " ' " strAuthCheck =

GetQueryResult(SQLQuery)

In this script, the person who developed the code is taking user input from the form and inserting it straight into an SQL query. Assuming an attacker presents a login and password that seem to be like the following:

Login: 'OR "="

Password: 'OR "="

This will make the ensuing SQL query turn out to be:

SELECT Username FROM Users WHERE Username = OR "=" AND Password = " OR "="

2.1 Types of SQLI Attacks

Basically there are two main types of SQL injection attacks, these are:

- Normal SQL injection attack
- Blind SQL injection attack

2.1.1 Normal SQL Injection Attack

By adding a union select statement to the perimeter, the attacker can then attempt to see if it is possible to get access to the database.

2.1.2 Blind SQL Injection Attack

With this type of SQL injection attack, rather than displaying a database error, the server presents a user friendly error page notifying the user that an error has occurred. In this case, SQL injection attack is possible even though it is not simple to discover. A simple way to discover Blind SQL injection attack is to insert a TRUE and FALSE statement into the parameter value.

2.2 SQLI Attack Objectives

SQLI attacks can be categorized according to their aim, goal or the intentions of the attacker. Some SQLI attack objectives are as follows [6]:

2.2.1 Identifying Injectable Parameters

In this instance the attacker intends to investigate the web application to determine which of the parameters and user input fields are susceptible to SQL injection attack.

2.2.2 Performing Database Finger-Printing

In this instance the aim is to ascertain the type and version of the database that the web application is utilizing. Some brands of databases react in a different way to other different queries and attacks, this vital information can be used to pinpoint the database. Being aware of the type and version of the database used by a web application makes it possible for an attacker to design exact and precise attacks.

2.2.3 Determining Database Schema

In order to accurately dig out data from a database, the attacker will need to have some knowledge of the database schema, for instance table name, column names, and column data types. An attacker with the intention of getting the database schema crafts such attacks to gather these sorts of information.

2.2.4 Extracting Data

This attack mechanism makes use of methods that help to take out data values from the database. Based on the kind of the web application, the information extracted can be very confidential and very much attractive to the attacker. These objective are the most common form of SQLI attacks.

2.2.5 Adding or Modifying Data

The main aim of this type of attack is to put into operation a technique whereby information can be appended to the database or transformed.

2.2.6 Performing Denial of Service
This attacking technique is implemented by employing mechanisms to shut down the database of the web application. As a result of such an attack, service is denied to potential users of the application. Attacks that entail locking or dropping database tables come under this grouping.

2.2.7 Evading Detection
This class of attack mechanism puts into operation certain techniques to steer clear of auditing and detection by system protection measures that have been instituted to ensure that the application is safe.

2.2.8 Bypassing Authentication
The purpose of this kind of attack is to permit the attacker to evade the database and mechanisms implemented to verify authentication for the application. Employing bypassing techniques can make it possible for the malicious attacker to take control of the rights and privileges related to the application and the system.

2.2.9 Executing Remote Commands
The goal of this kind of attack is to use techniques that tries to perform arbitrary commands on the database. The commands employed to achieve this type of attack could be functions or stored procedures which is accessible to users of the database.

2.2.10 Performing Privilege Escalation
The main objective of this type of attack is to exploit the implementation mistakes or logical errors in the database with the express purpose of escalating the privileges of the attacker. In contrast to bypassing authentication attacks, performing privilege escalation attacks centre attention on taking advantage of user privileges of the database.

2.3 SQL Injection Attack Input Methods
Harmful SQL codes can be presented to a web application that is susceptible to errors by way of making use of assortment of input mechanisms. This section of the paper generally explains in some of the different input mechanisms available.

2.3.1 Insertion of Data via User Input
In this instance, the malicious attacker inserts SQL commands by supplying a well-crafted input by the user. The applications can interpret input by the user in many different ways depending on the setting in which the application has been deployed. Generally attackers, who aim to use SQL injection attacks in web applications, usually make use of user input which comes from form submissions transmitted to the web application through HTTP, GET or POST demands. In general web applications have the capability to access user input [6].

2.3.2 Insertion via Cookies
Cookies can be described as files that include the state of the information created by the web application which is held in storage of the client's computer. As soon as a client goes back to the application, the cookies are then used to bring back the state information of the client. In view of the fact that the client exercises control in terms of the storage of the cookie, a client with criminal intent can possibly corrupt the cookie's content to construct SQL queries. This could result in a situation where an attacker can very simply present an attack by implanting it in the cookie [6].

2.3.3 Injection Through Server Variables
Server variables can be described as a set of items that are flexible and changeable. They include HTTP, network headers, and environmental variables. Usually web applications make use of server variables in many different ways. For instance taking down statistics for using the application, and making out and classifying browsing trends. When it happens that these variables are logged to a database with no cleansing mechanism in place, there could be resulting situation in which SQL weakness is generated. Consequently, attackers can create fake values that are positioned in HTTP and network headers, the attacker can then take advantage of the weakness by the insertion of SQL injection attacks straight into the headers. As a result of such scenarios, whenever the query to log the server variable is presented to the database, the attack is activated due to the fake values positioned in the headers [6].

2.3.4 Second-Order Injection
In regard to second-order injections, the criminal attacker plants wicked and harmful inputs to the database to indirectly activate an attack in situations where that particular input is used at a later time. It is important to note that the main purpose of this type of attack differs considerably from first-order injection attack. As a matter of fact, second-order injections usually do not try to make the attack occur when the harmful input at the outset gets to the database. In this scenario, attackers depend on the information of where the input will be consequently used and design the attack so that it gets activated when it is being used [6].

2.4 How SQL Injection Works
The process of checking if a web application is susceptible to SQL injection attack can be started with a simple input ploy.

Example

hi' or 1=1 --

Login: hi' or 1=1 --

Password: hi' or 1=1 --

The above depiction is an example of how SQLI is seeking to gain access through the login page of the web application. This example below is to attempt to get access to the web application through the Uniform Resource Locator (URL) http://duck/index.asp?id=hi' or 1=1 --

Another method is to attack through the HTML source code:

<FORM action=http://duck/Search/search.asp method=post> <input type=hidden name=A value="hi' or 1=1 --"> </FORM>. To do this, the attacker must download the HTML source code from the website and then save it on a hard dive, make changes to the URL and hidden field after which he/she can reload.

2.5 Consequences of SQLI Attacks

- **Loss of Confidentiality:** In view of the fact that SQL databases usually store data of sensitive nature, failure to maintain confidentiality of the database is a recurrent problem associated with SQLI weaknesses.

- **Problems of Authentication:** When it happens that poor SQL commands are utilized to verify names of users and password, it then becomes feasible to link up the system to another user without any prior knowledge of the password.
- **Authorization:** When information about authorization is stored in an SQL database, it becomes feasible to modify the information by means of successfully taking advantage of the SQL weakness.
- **Integrity:** Even though it is very likely to read and interpret sensitive information, it is possible to alter or alternatively erase the information through an SQL injection attack.

2.6 SQLI Countermeasures

The following guidelines are the recommended countermeasures to incorporate when designing web applications. It is expected that these recommended measures will be implemented at initial stage of developing the web application to ensure that it is secure [6][8]

2.6.1 Validate Input and Output Data

Majority of the most common attacks on web applications can be prevented or the chances of this looming danger happening can be considerably reduced, by incorporating the proper data validation techniques. Data validation is one of the very vital aspects of building a secure web application. Validation in this context refers to both input and output of a web application [8].

The input and output data, respectively coming into and out of a system is the route of the malicious attacks of the system. Every user input and output must be examined to guarantee it is suitable and anticipated. The most appropriate strategy for handling system input and output is to permit only clearly defined features and drop all other data. For instance, if an input field must accept a Social Security Number, then any data that is not a string of nine (9) digits is not legal and must be rejected. There are three (3) key models to consider when designing a data validation strategy [8][9]

- Accept Only Known Valid Data
- Reject Known Bad Data
- Sanitize Bad Data

The point must be emphasized that "Accept Only Known Valid Data" is the best strategy. Nevertheless, the fact must be recognized that this is always not possible for financial and other technical reasons. All these three methods should be examined:

- Data Type
- Syntax
- Length

Examining the data type is extremely vital. The web application must be examined to make sure a string is submitted and not a different object.

- **Accept Only Known Valid Data**: This is the ideal way to validate data that has been inputted. Web applications must allow only input that is known to be secure and anticipated. Assuming that a password reset system allows usernames as input, the valid and applicable usernames would be defined as ASCII A-Z and 0-9. Checks must be done by the web application to ensure that the input is of the right type of string, is composed of A-Z and 0-9 and is of a valid length [8].

- **Reject Known Bad Data**: The idea of disallowing the bad data approach depends on the application knowing about particular malicious payloads. Although it is true that this approach can limit exposure, it is very hard for any application to maintain an update database of web application attack signatures [9].

- **Sanitize All Data**: Cleaning the data is a key solution. Every data that is supplied to the client requires that it should be cleaned of any characters that may be exploited by the malicious attacker. The most excellent way to disinfect data is to make use of default deny and regular expressions to get rid of characters that may be causing problems. Below are some of the very important measures to incorporate during input validation [9].

- Escape Quotes: Substitute every apostrophe with two apostrophes (one back slash with two back slashes for MySQL).
- Ensure that numeric fields in actual fact appear like numbers.
- The steps above should not only be implemented on users direct input, but also on all variables that are not constant.
- Verify to ensure that inputs are within expectation (e.g. 0<age <120, login id with no space etc.).
- Avoid making use of SQL power characters in user data with no proper encoding. Example of SQLI power characters.

- **Never Depend on Client-Side Data Validation**: Data Validation on the client-side can at all times be evaded. Every data validation should be done on the server which is trusted or under control of the web application. Through client-side processing an attacker can very simply watch the return value and change it as and when he wants to. This appears amazingly clear, yet many still validate users, including login and using only client-side code like JavaScript. Validation of data on the client-side, for reasons of ease of use or user friendliness, is acceptable, but must not be considered a true validation process. Every validation must be done on the server-side, even if it is necessary to hasty validation performed on the client-side [9].

- **Fail Securely (Closed)**: The designing of security mechanism employed by the web application should be done in a way that, in an event that it fails, that failure is closed. Meaning that it should fail to state where all successive security requests are rejected rather than allowing them. An example could be a user authentication system. If it is not able to process a request to authenticate a user or entry and the process crashes, further authentication requests should not return negative or null authentication criteria. Another example is a firewall, if a firewall fails it should drop all successive packets [9].

2.6.2 Encoding of Inputs

The introduction of malicious characters into a string parameter is time and again achieved via the use of meta-characters that normally outwits the SQL parser into interpreting user input as SQL tokens. Despite the fact that it is feasible to disallow the utilization of these meta-characters, implementation of such a measure would limit and make it difficult for a user who has no malicious intentions to be able to specify legitimate inputs that include such characters. A good way for a solution to these problems is to make use of functions that encode a string in a way that every meta-character is encoded in a special way and then interpreted by the database as standard characters [9].

2.6.3 Positive Pattern Matching

When developing web applications, the person doing the development should institute input validation mechanisms which will spot and single out good input as opposed to bad input. This method of approach is normally referred to as positive validation. Since designers of the application may not be able to foresee every kind of attack that may well be initiated against web applications being developed, it is only proper to specify all forms of authorized input. Positive validation is a secure approach to confirm and ensure that inputs are checked [6].

2.6.4 Identification of All Input Sources

During the process of development, the developer must examine every input to the application. The fact must be noted that there are many potential sources of input to an application. If used to create a query, these input sources could be a very important means of launching an SQL injection attack. For these reasons, every source of input must be critically examined [6].

2.6.5 Enforce Less Privileges

System Administrators must **p**ut into force measures that will give users the precise lowest amount of the right of access whenever establishing connection between databases and other backend systems [9].

2.6.6 Use Stored Procedures

Web application users must make use of stored procedures in view of the fact that they are usually protected by SQLI attacks. Nevertheless, care must be taken as they can be injected through the use of exec () or linking together arguments used inside the stored procedure. In order to protect an application against SQLI attacks, the person who develops the application should in no way permit data supplied by the client to alter the syntax of SQL statements. As a matter of fact, the best means of providing defense is to separate the web application from SQL altogether. Every SQL statement that the application needs should be in stored procedures and reserved on the database server [6].

2.6.7 Use of Prepared Statement

A prepared statement must be used in an event where stored procedures cannot be used for whatever motive and dynamic SQL commands have to be used. Usually, prepared statements are used to transmit precompiled SQL statements by means of one or more parameters. In general prepared statements are resistant to SQL injection attacks as the database will use the worth of the bind variable exclusively and not interpret the contents of the variable in any way [6].

2.6.8 Avoid Detailed Error Messages

Try as much as possible to keep away from giving comprehensive and exhaustive error messages that might be of much use for an attacker who has the intention of compromising an application.

2.6.9 Additional Recommended Measures to Guarantee Security

- Create or operate the target infrastructure in such a way that reverse proxy servers are used and direct sockets cannot be established with the real server hosts from the public internet. This will in addition take into account the implementation of some tightly configured web application firewall. In situations where web services are involved, they have to be uniquely safeguarded.
- Separate the web servers: It is always likely that something is either wrongly or deliberately left open on a web server. Architectures must be planned assuming that the bad guys will have full access to web servers through violations. With that guess, one must segregate the web servers in using a very rigid approach.
- The system must be made tight, and customized for error handling used. This is due to the fact that the default error handling for the majority of web structures involves the needless disclosure of sensitive data. Implementing such measures will prevent needlessly exposing data. For instance, picture how easy it is going to be for an attacker in a situation where a full SQL query displayed a result as an error.

3. WHAT IS A WEB APPLICATION FIREWALL?

A Web Application Firewall can be described as an intermediary device, which sits between a client and a web server. The main central role of a web application firewall is analyzing and examining network message transmissions to determine whether there has been a breach in the security policy of the application. It is a tool which is used to secure the web server as well as the applications from harmful attacks [12]. Web application firewall can also be referred to as 'Deep Packet Inspection Firewall' for the simple reason that this security technology examines every request and response inside the HTTP. Several web application firewalls seek to look for particular attack signatures and try to identify the nature of attack being initiated by those intending to harm the application, whereas others search for odd behavior patterns that do not fall in line with a website's usual traffic flow. This security technology can either be software or hardware based, under normal circumstances it is installed at the front of a web server.

The purpose for setting up the firewall is to try and protect the application from incoming attacks launched by wicked and dangerous attackers. The inherent features of web application firewalls have the capability of stopping attacks that network firewalls and intrusion detection and prevention systems cannot, without having to modify the source code of the application. The web application firewall performs its role independently of the application in use [12]. Essentially the web application firewall protects very important business applications and servers from both known and unknown attacks. The firewall also provides protection against the harm

a security violation could cause to a company either through loss of name and status as well as loss in cost terms. For web applications already in production, the applications firewall in this case is very good in making the site secure [12].

3.1 Importance and Necessity of Developing Web Application Firewalls

One significant subject bothering many business organizations and attracting the attention of top management on a daily basis is the issue of developing secure web applications. The leadership of most corporate business organizations are generally doing everything feasible to give confidence and assurance to customers and suppliers so that proper security procedures are employed to deal with the problem of gaining unlawful access to confidential information on the web. The economic cost of violating the security of the web applications is unbelievable, and is a matter that has serious economic implications for businesses doing business on the web. As a result of the coming into being of the internet, many businesses are setting-up and making use of web applications to help increase revenues, reduce the cost of conducting business, and ensuring that customer services improves [10].

To guarantee security of web applications, businesses would have to take steps to put into operations secure coding practices, engage dealers to scan vulnerabilities and assess their web applications, set up a network intrusion detection and intrusion prevention systems or implement other technologies that guarantees the security of the applications. The greatest dilemma confronting web applications presently is the issue of vulnerabilities which is widespread. These faults can make it feasible for attackers to execute queries in the application's database, and most probably assume control of the web server. The danger confronting web applications are real and occur daily, this is coming about basically due to errors in the applications. An additional difficulty confronting the applications is the cost of late detection of the vulnerabilities. Majority of businesses leave the detection and exposure of web application faults to a group of devoted security staff who examine the application for any errors before putting it into production [11]. Rectifying the error, demands that the code is analyzed line by line from beginning to the end by those who developed the web application as well as quality assurance testing employees. Detecting and correcting an error of a web application already in production can be very expensive. The longer it takes for errors to go undetected, the higher the cost of developing the application. According to a Report by the National Institute of Standards and Technology (NIST) [11], when a design error is discovered or detected the expenditure level is 30 times what it would have been used to correct the error during the period of design. A point must be noted that this cost is only for the rectification of the error.

Additional expenses like losing customers as well as suppliers, lost market share, and name and status of the business must be taken into account. When the web applications are tested early enough, it will go a long way to reduce the cost significantly, but the real concern is how to expose the faults without bringing development to a crushing stop. Due to the critical nature of business transactions going on the internet, and for customers to have access to constant information, the web application must always be available day and night and throughout the course of the year. Business organizations simply do not have the luxury of taking down web applications to rectify errors uncovered after addition of new features. Clearly businesses cannot endure a situation where vulnerabilities keep on plaguing their applications, this will not create confidence in customers as well as partners.

Table 1: Relative Costs Based on Time Lapse between Error Creation and Discovery

	Found in Design	Found in Coding	Found in Integration	Found in Beta	Found in GA
Design Errors	1x	5x	10x	15x	30x
Coding Errors		1x	10x	20x	30x
Integration Errors			1x	10x	20x

Source: [11]

3.2 Benefits of Web Application Firewalls

The drawbacks of secure web programming as well as the inability of vulnerability scanners, intrusion detection and prevention systems to provide the required protection for web applications, several corporate business institutions have incorporated web application firewalls as balancing procedures to help in their efforts to achieve a better security level for their applications. The applications firewall is built with the expressed purpose of safeguarding a business's web application from malicious attacks and disclosing very private information.

In order to achieve these objectives, the web application firewall employs two security mechanisms. The first which is positive security mechanism makes it possible for genuine and lawful traffic to come into the web application. The second is negative security mechanism, which disallows codes or attacks which are harmful from getting to the application, and as a result stops the application from leaking any sensitive information. These two security techniques makes it possible for businesses to secure important data from known and unknown attacks, thereby enhancing the security level of the applications. This application security technology presents a number of different benefits to business organizations. Below is a brief summary of the benefits that will accrue to businesses owing to the deployment of a web application firewall [13].

- **Good Return on Investment (RoI)**: A web application firewall is a tool that helps to reduce cost appreciably in an efficient way once it is deployed by the business organization. For example, with the web application firewall installed there will be no need to conduct regular secure code reviews as well as scanning for vulnerabilities; this will result in major savings for the business. In cost terms it is particularly advantageous to business entities with several applications in operation with each having its own peculiar security needs.
- **Cost of Maintenance is Low**: One of the most important elements for the implementation of any technological security initiative is the cost of maintenance. A web application firewall has certain in-built mechanisms which ensure that maintenance effort reduces significantly. As a result of such

initiatives the cost of deployment of the web application firewall becomes lower in the long run.

- **Shut Down Access on Web Attacks**: One of the very significant advantages of a web application firewall is getting rid or reducing the dangers of malicious attacks usually by means of a business organization's web application. Essentially web application firewalls are able to secure web applications already installed and running live. Normally these applications are made of several parts and in events where errors are discovered, and changes cannot be made quickly by the designers of the application, or in situation where the application has been poorly documented, or applications developed by third parties, a web application firewall can be employed to quickly shield and shut down the application from web attacks.
- **Speed of Developing Applications Improves**: Apart from stopping malicious attacks, web application firewalls can as a matter of fact speed up the development of new web applications. Currently the deployments of fresh applications are hindered by this idea of scan and repair mentality. This means a code is written, scanned to identify errors, and then returned to those who developed it, to rewrite it again. This approach is not only slow, expensive and time-consuming, but is also not useful in identifying unknown vulnerabilities. It only discovers a limited number of known security violations. With web application firewall the team developing the application can concentrate on quick and fast building of new applications, with full knowledge and assurance that the code will be safeguarded from any wicked attacks.
- **No Need to Rewrite Code**: Where there is no sufficient safeguard for an application, those who do the development work are always under pressure to hunt for faults and take steps to rectify these faults. Scanners and code review can work to some extent. As soon as a web application firewall is in place, application designers can now pay attention to fast deployment of new applications as well as the addition of new features.
- **Significantly Reducing Reactive Patching**: Due to the security holes normally resident in web applications yet to be protected, developers from time to time install patches as and when the holes are detected in order to make the application secure. With a web application firewall in place, it reduces the need for the application to rely on patches to seal the holes. This is accomplished by simply keeping harmful traffic out, and letting in good traffic.
- **Elimination of Dangers to Businesses**: Because malicious attackers are fully aware that many unprotected web applications are vulnerable to attacks, they try to get access to classified information by transmitting malicious code, to website of companies with the intention of compromising their system. The outcome of a breach in security can be loss of customers, injury to the brand name, and financial loss etc. Web application firewall considerably reduces the danger of getting the system compromised by incorporating practical security protection mechanisms to guard against dangerous attacks.

3.3 Evaluation of Web Application Firewalls

There are three (3) major questions to consider when evaluating web application firewalls**.** In situations where a business organization makes a decision to deploy a web application firewall, potential vendors may have to answer the following questions:

- **Has the technology been proven in real-world environments?** This answers the question of whether the web application firewall has been deployed in another business organization without problems and that they will be willing to bear witness of what the vendors have said about the web application firewall. The web application firewall as deployed should be able to clearly show that it is capable of addressing the same security difficulties that confront the new businesses.
- **Does the product support or impede business operations?** This describes a situation in which it is expected that the web application firewall would be visible to users of the application and not interrupt or obstruct them when conducting business transactions online. Furthermore, it is expected that the web application firewall must function efficiently without having to bring in a new point of failure which can render the applications that have been secured inoperative.
- **Does the web application firewall give actionable information which can be used by the security, development and testing teams?** The product must offer detailed snapshots of each secured application that provides the security team's comprehensive appreciation of transactions and the background in which they function. In addition, the web application firewall must give sufficient details about vulnerabilities and application security deficiencies so that the development and testing group can without difficulty appreciate the measures they need to employ to rectify the errors.

3.3.1 Reasons Why Secure Coding Initiatives is Not the Answer to the Prevention of SQLI Attacks

The desire of many business institutions seeking to develop an entirely flawless secure code is a very good aspiration indeed, but the snag is that, it is really a difficult challenge. This is because studies have suggested that so far it has not been possible to successfully deploy a secure code. Basically there are two types of code, that is code developed in-house and the one contracted out to be developed by companies outside a business. Secure coding as a matter of fact is an option that is best recommended, and it brings up the implementation of good security measures early in the code development process. Taking the case of applications developed in-house, most of the time many business organizations employ people in-house to develop their web applications. Such people who design these applications are not infallible.

They are also subject to make mistakes due to pressure of work or the enormity of the task at hand. It is possible that they may have too many web applications to secure at the same time or that they took charge of a website whose original developers may have left the organization. Those there at the time of the web application development may not have sufficient information due to incomplete documentation

and for that reason cannot be in a position to effect the required changes to the application. Another problem could be that getting people with secure coding skills is not going to be an easy issue, since secure coding is an emerging area in application security and is generally not taught in school [10]. Another method employed by companies doing business on the web is to give the job of code development to firms that are outside the business organization, what is commonly referred to as service providers, outsourcing or third parties. Even this approach has its own unique benefits of ensuring quicker and speedy development of applications, it brings with its own peculiar set of difficulties. One problem with this approach is that it is too expensive to contract other organizations to do this, also even though the work is done outside when it comes to testing and correcting errors it has to be brought in-house to be done. Anytime a newer version of the application is released the same process is repeated. In circumstances where a business organization has several web applications deployed, each new version released for the application could lead to incurring and additional expenses as well as utilizing more resources and time [10].

3.3.2 Reasons Why Vulnerability Scanning is Not the Answer to the Prevention of SQLI Attacks

In the attempt to look for a way to secure web applications, several corporate businesses have resorted to vulnerability scanners as their saviours. The belief is that the vulnerability scanner has the capability of scrutinizing the web application and exposing all the holes that are contained therein for the necessary corrections to be made. It is one of the best recommended practices; it also discovers faults faster than line by line code review [5][14]. Even though vulnerability scanners are very useful tool in the fight against web attacks, they go through the same difficulties that majority of security technologies experience. The difficulties start with getting the person with requisite skills to appropriately set-up and implement the tool, at the same time analyze and interpret the findings of the scanners.

Another difficulty is that they act in response to the actions of attackers and are not proactive at all. In a situation where there is a modification in the code, the danger is the website may become insecure; it does not have the ability to examine every line in the code in order to identify faults for rectification. With vulnerability scanners there is the danger of inaccuracy, also findings are not consistent. In events where errors are discovered they do not have any mechanisms in place to correct those errors. For instance, taking a case where a business entity has several web applications deployed, it would be very costly for such businesses to employ vulnerability scanners to secure their applications. In the case of new and unforeseen vulnerabilities it is not possible for vulnerability scanners to detect them since no new signatures would have been created for that purpose [14].

3.3.3 Reasons Why Deploying Network IDS/IPS is Not the Answer to the Prevention of SQLI Attacks

The quest for business organizations to put in place suitable and acceptable defense for their network security is a laudable objective that needs to be pursued with all strength they can command. These defensive techniques employed for protection of network security does not take into consideration the safeguarding of the web applications that runs on the network. Many business organizations may be under the mistaken impression that once they deploy intrusion detection and prevention technologies then everything is secure and foolproof. Nevertheless, network security does not have the capacity to offer the sort of protection needed by web applications. The kind of security technology used to provide security solutions for protecting networks differs from that of web applications. The network security solutions cannot offer any safeguards to the applications, because each and every application has its own peculiar vulnerability. Network security technologies do not have the ability to scrutinize secure socket layer traffic which is what most web applications use to send private information over the web. It is also imperative to appreciate the fact that a good number of intrusion detection and prevention security technologies are not configured in the correct way and for that reason do not work as expected. They do not provide any defense against web application security defects as a result of poor coding. To address this situation, a web application firewall as described above, is the answer to solving this problem. With the deployment of web application firewall, businesses can quickly put their applications into production, because they have the confidence and assurance that there will be instant and uninterrupted security.

4. RELATED WORK

In this section we present some related work involving various techniques used by the researchers to prevent SQLI vulnerabilities and attacks.

Singh and Roy [1] presented a network based vulnerability scanner approach for SQLI which provides a better coverage and with no false positive within a short span of time.

Avancini and Ceccato [2] took advantage of static analysis to detect candidate cross-site scripting vulnerabilities. Input values that expose these vulnerabilities were searched by a genetic algorithm and, to help the genetic algorithm escape local optima, symbolic constraints were collected at run-time and passed to a solver.

Teodoro and C. Serrao [3] discussed the direct implication of the lack of security and the importance of quality of the Software Development Life Cycle (SDLC), and the major factors that influence them. Furthermore, they proposed a set of security automated tools and methodologies that can be used throughout the SDLC as a means to improve critical web-based applications security and quality.

Tian *et al.* [4] focused on the regression test in web vulnerability detection, and presented a strong-association rule based algorithm to make the detection more efficient.

Scholte *et al.* [5] presented *IPAAS*, a novel technique for preventing the exploitation of cross-site scripting and SQL injection vulnerabilities based on automated data type detection of input parameters. *IPAAS* automatically and transparently augments otherwise insecure web application development environments with input validators that result in significant and tangible security improvements for real systems. They implemented *IPAAS* for PHP and evaluated it on five real-world web applications with known cross-site scripting and SQL injection vulnerabilities.

Table 2. Discovered Vulnerability in the CCI Website

Assessment Criticality	SQL Injection Possible
Assessment URL	http://www.cci.dmu.ac.uk/
Assessment Summary	A possible SQLI flaw was spotted in the CCI web application. When it happens that the SQL injection is successful, this will enable the attacker to gain access to contents of the database. The potential to implement system commands in a remote fashion or in certain conditions paves the way to assume control of the Windows server that hosts the database. SQLI is possible. User parameters submitted will be formulated into a SQL query for database processing. If the query is built by simple 'string concatenation', it is possible to modify the meaning of the query by carefully crafting the parameters. Depending on the access rights and type of database used, tampered query can be used to retrieve sensitive information from the database or execute arbitrary code.
Assessment Implication	SQL injection weaknesses can make it possible for an attacker to modify, alter and amend information from the database. Based on how severe the disclosure, data may be harmed, lost, or in other instances, system level executables may be uncovered unnecessarily.
Assessment Recommendations	Recommendations consist of implementing secure and safe coding mechanisms to appropriately cleanse input and to make sure that only data that is anticipated is allowed by an application. The server for the database should as matter of urgency be toughened in order to put a stop to data from being accessed improperly. SQLI attacks can be averted by making use of secure and safe coding mechanisms that foils client supplied values from meddling with SQL statement syntax. Mechanisms for input validation should be employed on input to check inappropriate characters. Instituting measures to make sure that the web application gives as little information as possible to users in events where database error takes place is necessary. The complete error message must not be revealed. Furthermore, it is very much recommended that database permissions be evaluated and that the SQL script be moved to stored procedures wherever feasible.
Assessment Solution	Do not trust client side input even if there is client side validation. In general, • If the input string is numeric, type check it. • If the application used JDBC, use PreparedStatement or CallableStatement with parameters passed by. • If the application used ASP, use ADO Command Objects with strong type checking and parameterized query. • If stored procedure or bind variables can be used, use it for parameter passing into query. Do not just concatenate the string into a query in the stored procedure. • Do not create dynamic SQL query by simple string concatenation. Use minimum database user privilege for the application. This does not eliminate SQL injection but minimizes its damage. For example, if the application requires reading one table, access should be granted only to such an application. Avoid using 'sa' or 'db-owner'.
Assessment References	• The OWASP Developer guide at *www.owasp.org/index.php/Category:OWASP_Guide_Project*

5. VULNERABILITY ASSESSMENT ON WEBSITE

Table 2 depicts information about the discovered vulnerability in the CCI website. The findings of the vulnerability assessment on the CCI are depicted in Tables 3 and 4 and Figure 1.

Table 3. Findings of the Assessment

Severity	Findings
Critical	2
High	2
Medium	3
Low	0
Informational	0

Table 4: Interpretations of Findings in Percentages Terms

Severity	Findings
Critical	**29%**
High	**29%**
Medium	**42%**
Low	**0%**
Informational	**0%**
Total	**100%**

5.1 Proposed Snippet of Code That Can be Used to Redesign the CCI Website

Based on the above vulnerability assessment this paper proposes a suggested code for redesigning of the CCI website to protect itself against SQL injection attack as follows:

BETTER process_form.php Preventing SQL Injection with PHP[1]

```
<?php

$name = mysql real escape string( $_POST[‘name’] );

$pwd  = mysql real escape string( $_POST[‘pwd’] );

$str_sql = “SELECT * from ‘tbl_users’ WHERE “ .

        “usr_name=’” . $name . “’ AND “ .

        “usr_pwd=’” . $pwd . “’”;

$result = mysql query( $str_sql ) or die ( mysql error() );

?>
```

The mysql_real_escape_string() function breaks out any characters that could possibly modify the query.

[mysql_real_escape_string…] breaks out special characters in the unescaped_string, this takes into consideration the current character set of the connection so that it is harmless to put in a mysql_query(). When it happens that binary data is to be included, then this function must be used.

mysql_real_escape_string() calls MySQL’s library function mysql_real_escape_string, which prepends backslahes to the following characters: \x00, \n, \r, \, ‘, and \xla.

This function must constantly be used to make data safe prior to transmitting a query to MySQL. The graphical representation of the findings of the SQLI attacks and scans performed using web vulnerability assessment tools such “Paros Proxy” and “WebScarab” is depicted in Figure 1 below.

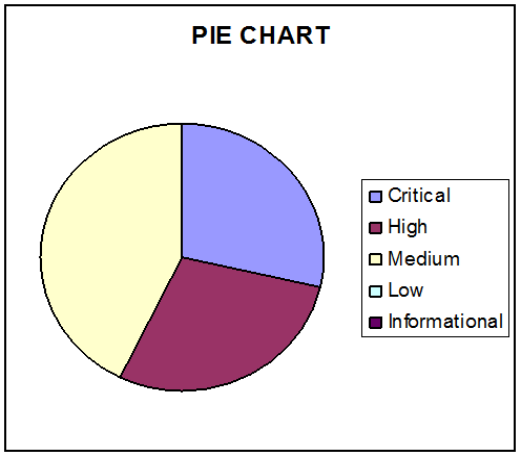

Fig 1: Graphical Representation of Assessment Findings

6. CONCLUSION

The proliferation of ICT coupled with the internet has paved the way for enormous amount of information for individuals/organizations in both developing and developed countries worldwide. The resultant use of the World Wide Web to access information through web applications has introduced various vulnerabilities and attacks. This paper focused on of SQLI, in terms of types, detection and prevention techniques. Furthermore, importance, benefits and evaluation of web application firewalls, efficient methods of preventing SQLI attacks in web applications and other solutions of the related work were discussed and elaborated. Through the vulnerability assessment and analysis conducted in this paper, we are confident that, although web application firewalls may have their disadvantages, they are more reliable as compared to other methods such as secure coding, vulnerability scanners and network intrusion detection techniques and as such should be implemented by organizations in order to prevent SQLI attacks on their web applications.

[1] http://www.digifuzz.net/archives/2007/07/preventing-sql-injection-with-php/

7. REFERENCES

[1] A.K. Singh and S. Roy, "A Network Based Vulnerability Scanner for Detecting SQLI Attacks in Web Applications," in *Proceedings of the IEEE International Conference on Recent Advances in Information Technology*, pp. 585-590, 15-17 March, 2012.

[2] A. Avancini and M. Ceccato, "Security Testing of Web Applications: A Search-Based Approach for Cross-Site Scripting Vulnerabilities," in *Proceedings of the 11th IEEE International Working Conference on* Source Code Analysis and Manipulation (SCAM), pp. 85-94, 25-25 Sept. 2011.

[3] N. Teodoro and C. Serrao, "Web Application Security: Improving Critical Web-based Applications Quality Through In-depth Security Analysis," in *Proceedings of the IEEE International Conference on Information Society (i-Society)*, pp. 457-462, 27-29, June, 2011.

[4] H. Tian, J. Xu, K. Lian and Y. Ying, "Research on Strong-Association Rule Based Web Application Vulnerability Detection," in *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology, (ICCSIT)*, pp. 237-241, 2009.

[5] T. Sccolte, W. Robertson. D. Balzarotti and E. Kirda, "Preventing Input Validation Vulnerabilities in Web Analysis Applications Through Automated Type Analysis," in *Proceedings of the 36th IEEE Annual Computer Software and Applications Conference (COMPSAC)*, pp. 233-243, 16-20 July, 2012.

[6] W.G.J. Halfond, J. Viegas, and A. Orso, "A Classification of SQL Injection Attacks and Countermeasures", in *IEEE Proceedings*, 2006, Available [Online] *http://www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf* (Accessed 29/04/2013).

[7] A. Ciampa, C.A. Visaggio and M.D. Penta, " A Heuristic-based Approach for Detecting SQL-Injection Vulnerabilities in Web Applications," in *ACM Proceedings of the ICSE Workshop on Software Engineering for Secure Systems*, pp. 43-49, 2010.

[8] J. Scambray, M. Shema and C. Sima, "Hacking Web Applications Exposed - 2nd ed. San Francisco", *McGraw-Hill*, 2006.

[9] M. Curphey, D. Endler, W. Hau, S. Taylor, T. Smith, A. Russel, G. Mckenna, R. Parke, K. Mclaughlin, N. Tranter, A. Klein, D. Grooves, I. By-Gad, S. Huseby, M. Eizner, M. Hill and R. McNamara", A Guide to Building Secure Web Applications: *The Open Web Application Security Project*, 2002 Available [Online] *http://www.rootsecure.net/content/downloads/pdf/owasp_guide.pdf* (Accessed 09/05/2013).

[10] M. Dermann, M. Dziadzka, B. Hemkemeier, A. Hoffmann, A. Miesel, M. Rohr and T. Schreiber, "Best Practices: Use of Web Application Firewalls", *The Open Web Security Application Project, OWASP Papers Program*, 2008, Available [Online] *http://www.owasp.org/images/a/a6/Best_Practices_Guide_WAF_v104.en.pdf* (Accessed 07/05/2013).

[11] Planning Report 02-03 - The Economic Impacts of Inadequate Infrastructures for Software Testing", *National Institute of Standards & Technology, US Department of Commerce, 2002*, Available [Online] *http://www.nist.gov/director/planning/upload/report02-3.pdf* (Accessed 05/05/2013).

[12] J.D. Meier, A. Mackman, M. Dunner, S. Vasireddy, R. Escamilla and A. Murukan, "Improving Web Application Security: Threats and Countermeasures*," Microsoft Corporation*, 2003.

[13] K. Tyminski, "The Business Case for Web Application Firewalls," 2008, Available [Online] *www.scanarmor.dk/UserFiles/File/WP_BusinessCaseForWAF_FINAL_092408.pdf* (Accessed 05/05/2013).

[14] R. Barnet, "Why Organizations Need Web Application Firewalls," 2007, Available [Online] *www.scanarmor.dk/UserFiles/File/WP_Why_WAF.pdf* (Accessed 03/05/2013).