Secure query processing by blocking SQL injection attack

Dissertation

Submitted to:
Central department of Computer Science and Information Technology
Tribhuvan University
Kirtipur, Nepal

In Partial Fulfillment of the Requirements for the Degree of

Master Of Science

In

Computer Science and Information Technology

Submitted by:
Renuka Chitrakar
December, 2012

Tribhuvan University

Institute of Science and Technology

Central department of Computer Science and Information Technology

Date : ……………………

LETTER OF RECOMMENDATION

Mrs. Renuka Chitrakar has carried out this thesis work entitle "Secure query processing by blocking SQL injection attack" under my supervision and guidance. In my best knowledge this thesis successfully completed which fulfills the requirements for the aware of the Degree of Master's in Computer Science and Information Technology, therefore I recommended for further evaluation.

....

Prof. Dr. Subarna Shakya
Department of Electronics and Computer Engineering
Pulchowk Campus, Pulchowk
(Supervisor)

Tribhuvan University

Institute of Science and Technology

Central department of Computer Science and Information Technology

We certify that we have read this dissertation work and in our opinion it is satisfactory in the scope and quality as a dissertation as the partial fulfillment of the requirement of Master of Computer Science and Information technology from Tribhuvan University, Nepal.

Evaluation Committee

………………………	………………………….
Mr Nawraj Paudel	Prof. Dr. Subarna Shakya
Head, Central Department of Computer	Department of Electronics and
Science and Information Technology	Computer Engineering
TU, Kritipur	Pulchowk Campus, Pulchowk
	(Supervisor)
……………………….	…………………………..
(External Examiner)	(Internal Examiner)

Abstract

With the rise of the Internet, web applications, such as online banking and web-based email the web services as an instant means of information dissemination and various other transactions has essentially made them a key component of today's Internet infrastructure. Web-based systems consist of both infrastructure components and of the application specific code. But there are many reports on intrusion from external hacker which compromised the back end database system so here introduce briefly the key concepts and problems of information security and here present the major role that SQL Injection is playing in this scenario. SQL Injection Attacks are a class of attacks that many of these systems are highly vulnerable to, and there is no known fool-proof defense against such attacks. Based on the above analysis and on today's computer security state-of-the-art, focus on the research especially on the SQLIA's, which are still one of the most exploited and dangerous intrusion techniques used to access web applications.

In this research work, here propose a method for determining allowability of a structured query language (SQL) statement, the method comprising the steps of normalizing the SQL statement with a predetermined set of allowable statements. The most available solution of that problem either requires source code modification, which is an overhead to an existing system as well as which can increase the possibilities of new injection points, or required a computational overhead at runtime which increase the minimum response time. But in normalization technique eliminates the need of source code modification along with an improved overall efficiency.

Acknowledgements

This thesis work would not exist without help, advice and encouragement of many people. I thank my Supervisor, Professor Dr. Subarna Shakya, who taught me much about teaching and research methodology.

I am grateful to the professor and lecturer, Prof. Dr. Shahidhar Ram Joshi, Dr. Onkar pd. Sharma. Mr Sudarshan Karanjit, Mr Min Bahadur Khati, Mr Samujjal Bhandari, Mr Hemanta G.C., Mr Nawraj Paudel, Mr Arjun Saud and Mr Jagdish Bhatta of Central Department of Computer Science and Information Technology who, while not being directly involved in my thesis work, nevertheless influenced me greatly.

Many thanks go to my colleagues, who helped me directly or indirectly to accomplish my work. I am especially grateful Mr Nawraj Paudel, Head, Central Department of Computer Science and Information Technology, has also been a positive and encouraging influence on my research efforts.

I am also indebted to my friends Mr Bikash Balami and Mr Prakash Saud for their interest, cooperation, worries and complain.

Finally, I thank to my parents and family, who were ultimately the people, who prepared me for this endeavor. I own you all my success.

Renuka Chitrakar

ABBREVIATIONS

SSL	Secure Socket Layer
SQL	Structure Query Language
SQLIAs	SQL Injection Attacks
RDBMS	Relational Database Management System
CIA	Confidentiality, Integrity, Availability
AAA	Authentication, Authorization and Accounting
CGI	Common Gateway Interface
DML	Data manipulation Language
DDL	Data Definition Language

Table of Content

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Background

Now days our dependence on web application has increased and we continue to integrate them in our everyday routine activities. Web applications are reliable in online shopping, reading newspaper, paying bills etc. Many services are provided via the world wide web, efforts from both academic and industry are striving to create techniques and standards that meets the sophisticated requirements of today's web application and users. From the hacker's perspective, web application is divided in to the several layers. Current technologies such as antivirus software program and network firewalls offer some security at the host and network levels but not at the application level. Although the application level firewalls offer immediate assurance of web application security, they have at least two drawbacks: they do not identify errors.

1.2 Problem

Different researchers have published different papers in the field of web application and its security mechanism. According to Spett [21] from the hacker's perspective, web application consists of five layers, ie desktop layer, transport layer, access layer, network layer and application layer. Current technologies such as antivirus software program and network firewall offer comparatively secure protection at the host and network levels, but not at the application level. However, testing processes cannot guarantee identification of all bugs. Scott and Sharp [18] proposed the use of a gateway that filters invalid and malicious inputs at the application level.

1.3 Objective

Objective of this thesis work is to study the SQL injection attack in the web application with their prevention technique and then implement them in the ASP.NET. Especially objective of this thesis is categorized as follows points.

- To show how SQL injection statement can be injected and attack
- To show how the injected query can be prevented from executing on server

1.4 Literature review

Use of web application is increasing rapidly everywhere. In the field of web application, numerous research works is published whether it may be paper or internet. Companies and organizations use web applications to provide a broad range of service to users. Database of web application contain the important information of that organization like customer and financial record, these applications are frequent target for attacks. There are numerous attacking technique such as cross site scripting, content spoofing etc, one of them is SQL injection, can give attackers a way to gain access to the database underlying web application and with that power to leak, modify and in some time delete information that is stored in the database. In the other word, SQL -Injection attacks can occur when a web application receive user input and use it to build a database query without validating it. Conceptually, SQLIAs could be prevented by a rigorous application of defensive techniques.

If we closely analyze the code structure of an SQLIA then we found that hacker inject the user string in a way that could alter the structure of the original query and query injection is done either at the middle of the query or at the end; means hacker usually append the query. So if we store all the information of structure of the valid query and cross check it with the dynamically generated query then we can determine that the dynamically generated query is a SQLIA or a valid query.

Different researchers have published papers in the field of web applications and its security mechanism. Wasserman and Su[8] proposed an approach that uses a static analysis combined with automated reasoning. This technique verified that SQL queries generated in the application usually do not contain a tautology. This technique is effective only for SQL injections that insert a tautology in the SQL queries, but cannot detect other types of SQL injections attacks.

Su and Wasserman[27] present grammar-based approach to detect and stop queries having SQLIAs by implementing SQLCHECK tool. They mark user supplied portions in queries with a special symbol and augment the standard SQL grammar with production rule. A parser is generated based on the augmented grammar. The parser successfully parses the generated query at runtime, if there are no SQLIAs in the generated queries after adding user inputs. This approach uses a secret key to discover user inputs in the SQL queries. Thus, the security of the approach relies on attackers not being able to discover the key. Additionally, this approach requires the application developer to rewrite code to manually insert the secret keys into dynamically generated SQL queries.

Buehrer[7] secure vulnerable SQL statements by comparing the parse tree of a SQL statement before and after input and only allowing SQL statements to execute if the parse trees match. They conducted a study using one real world web application and applied their SQLGUARD solution to each application. They found that their solution stopped all of the SQLIAs in their test set without generating any false positives. While it stopped all of the SQLIAs in their solution required the developer to rewrite all of their SQL code to use their custom libraries, which is an overhead we are trying to eliminate. However, the approach is ineffective, if the user supplied input does not appears at the leaf of the tree.

Chapter 2

Computer and Information Security: an Overview

2.1 Terminologies and Formal Definitions

Computer security is a branch of technology known as information security, applied to computers. Information security is based on the general concept of the protection of data against unauthorized access. The objective of computer security varies and can include protection of information from theft or corruption, or the preservation of availability, as defined in the security policy.

Computer security is the process of preventing and detecting unauthorized use of your computer. Prevention measures help you prevent unauthorized users, also known as “intruders", from accessing any part of your computer system. Detection helps you to determine whether or not someone attempted to break into your system, whether or not the breach was successful, and the extent of the damage that may have been done. This makes computer security particularly challenging because it is difficult enough just to ensure that computer programs do everything they are designed to do correctly [23]. Nowadays most information in the world is processed through computer systems, so it is common to use the term information security to also denote computer security. This is quite a common mistake: in fact, academically, the definition of information security includes all the processes of handling and storing information. Information can be printed on paper, stored electronically, transmitted by post or by using electronic means, shown on films, or spoken in conversation. The U.S. National Information Systems Security Glossary [14] defines Information systems security (INFOSEC) as:

“the protection of information systems against unauthorized access to or modification of information, whether in storage, processing or transit, and against the denial of service to authorized users or the provision of service to unauthorized users, including those measures necessary to detect, document, and counter such threats."

It defines computer security as:

“Measures and controls that ensure confidentiality, integrity, and availability of the information processed and stored by a computer"

This observation on information pervasiveness is especially important in today’s increasingly interconnected business environment. As a result of important, information is exposed to a growing number and a wider variety of threats and vulnerabilities, which often have nothing to do with computer systems at all. In this work, however, we will deal mostly with computer security and not information systems in general.

2.2 The C.I.A. Paradigm

Information security has held that confidentiality, integrity and availability, known as the C.I.A. paradigm [23], are the core principles of information security.

Confidentiality is the property of preventing disclosure of information to unauthorized individuals or systems. For example, a credit card transaction on the Internet requires the credit card number to be transmitted from the buyer to the merchant and from the merchant to a transaction processing network. The system attempts to enforce confidentiality by encrypting the card number during transmission, by limiting the places where it might appear (in databases, log files, backups, printed receipts, and so on), and by restricting access to the places where it is stored. If an unauthorized party obtains the card number in any way, a breach of confidentiality has occurred. Confidentiality is necessary, but not sufficient for maintaining the privacy of the people whose personal information a system holds.

Integrity means that data cannot be modified without authorization. Integrity is violated when a message is actively modified in transit. For example, when someone accidentally or with malicious intent deletes important data files, when a computer virus infects a computer, when an employee is able to modify his own salary on a payroll database, when an unauthorized user vandalizes a web site, when someone is able to cast a very large number of votes in an online poll, and so on.

Availability is the important property that a rightful request to access information must never be denied, and must be satisfied in a timely manner. In other words, for any information system to serve its purpose, the information must be available when it is needed. Ensuring availability also involves preventing denial-of-service attacks.

Authenticity important for authenticity to validate that both parties involved are who they claim they are. In computing, e-Business and information security it is necessary to ensure that the data, transactions, communications or documents (electronic or physical) are genuine.

Non-repudiation implies that one party of a transaction cannot deny having received a transaction nor can the other party deny having sent a transaction. In law, non-repudiation implies one's intention to fulfill their obligations to a contract.

Electronic commerce uses technology such as digital signatures and encryption to establish authenticity and non-repudiation.

Sometimes other goals have been added to the C.I.A. paradigm, such as authenticity, accountability, non-repudiation, safety and reliability. However, the general consensus is that these are either a consequence of the three core concepts defined above, or a means to attain them.

2.3 The A.A.A. Architecture

The A.A.A. architecture and components are specifications of a software and hardware system architecture which strives to implement those requirements. Then, of course, security systems are the real world implementations of these specifications. In computer security A.A.A. stands for Authentication, Authorization and Accounting [24]. These are the three basic issues that are encountered frequently in many network services where their functionality is frequently needed. Examples of these services are dial-in access to the Internet, electronic commerce, Internet printing, and Mobile IP. Typically, authentication, authorization, and accounting are more or less dependent on each other. However, separate protocols are used to achieve the A.A.A. functionality.

Authentication: Authentication refers to the process where an entity's identify is authenticated, typically by providing evidence that it holds a specific digital identity such as an identifier and the corresponding credentials. Examples of types of credentials are passwords, one-time tokens and digital certificates.

Authorization: Authorization refers to the granting of specific types of privileges (or not privilege) to an entity or a user, based on their authentication, what privileges they are requesting, and the current system state. Authorization may be based on restrictions, for example time-of-day restrictions or physical location restrictions. Most of the time the granting of a privilege constitutes the ability to use a certain type of service. Examples of types of service include, but are not limited to: IP address filtering, address assignment, route assignment and encryption.

Accounting: Accounting refers to the tracking of the consumption of network resources by users. This information may be used for management, planning, billing, or other purposes. Real-time accounting refers to accounting information that is delivered concurrently with the consumption of the resources. Batch accounting refers to accounting information that is saved until it is delivered at a later time. Typical information that is gathered in accounting is the identity of the user, the nature of the service delivered, when the service began, and when it ended.

Chapter 3

Web Applications

Web application, or webapp, is the general term that is normally used to refer to all distributed web-based applications. According to the more technical software engineering definition, a web application is described as an application accessible by the web through a network. Many companies are converting their computer programs into web-based applications. Web Applications are similar to computer based programs but differ only in that they are accessible through the web, allowing the creation of dynamic websites and providing complete interaction with the end-user. Web Applications are placed on the Internet and all processing is done on the server, the computer which hosts the application [24][25].

Web applications are sets of web pages, files and programs that reside on a company's web server, which any authorized user can access over a network such as the World Wide Web or a local intranet. A web application is usually a three-tiered construction. Normally, the first tier is a Web browser on the client side, the second is the real engine on the server-side where the applications core runs, and the third layer is a database as showed in figure 3.1. The Web browser makes the initial request to the middle layer, which, in turn, accesses the database to perform the requested task, either by retrieving information from the database, or by updating it and generating a user interface. A server processes all user transactions and usually the end-user simply accesses the web application by a Web browser, interacting with it. Since web applications reside on a server, they are easy to manage. In fact, they can be updated and modified at any time by the web applications owner with minimal effort and without any distribution or installation of software on the clients machines. This is the main reason for the widespread adoption of Web applications in today's organizations [25].

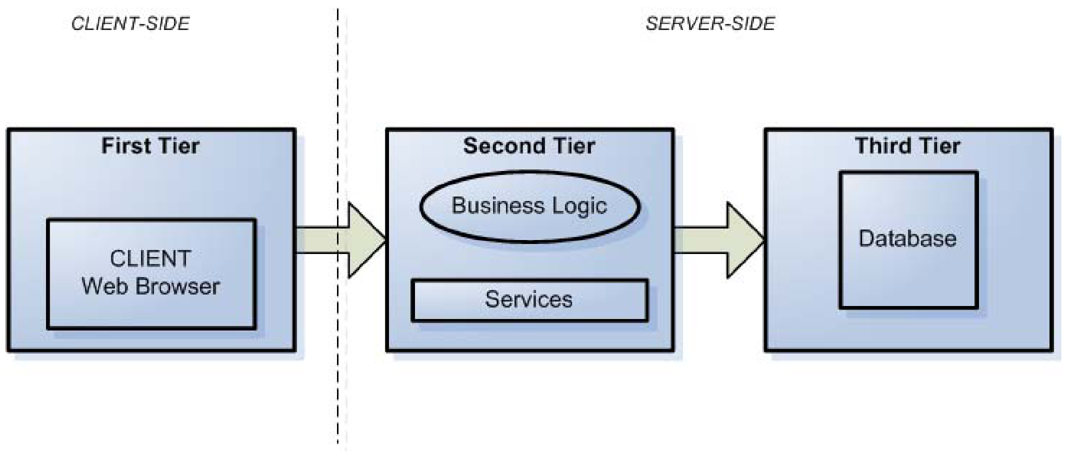

Figure 3.1 Three tired J2EE Web Application Model

Nowadays, web applications are becoming increasingly popular and are poised to become a major player in the overall software market due to the benefits they afford, such as visibility and worldwide access. They are, without a doubt, essential to the current and next generation of businesses and they have become part of our everyday online lives. In fact, a web application is a worldwide gate accessible not only through standard personal computers but also though different communication devices such as mobile phones and PDAs (fig. 3.2). The use of web applications is especially beneficial for a company: with just a little investment, a company can open up a marketing channel that will allow potential clients easy global access to its business 24 hours a day. A typical example of a web application is an online questionnaire or user survey. The end-user client simply completes the online questions by filling in a form that is accessible worldwide through any kind of network device and submits the responses to the application that then collects and stores the data in a database on the server side [3].

Web applications are present in all aspects of our daily internet use. Common examples are those applications used for searching the internet such as “Google"; for collaborative open source projects as “Source Forge"; for public auctions as “eBay" and many others as well as blogs, web mail, web-forums, shopping carts, e-commerce, dynamic contents, discussion boards and social networks.

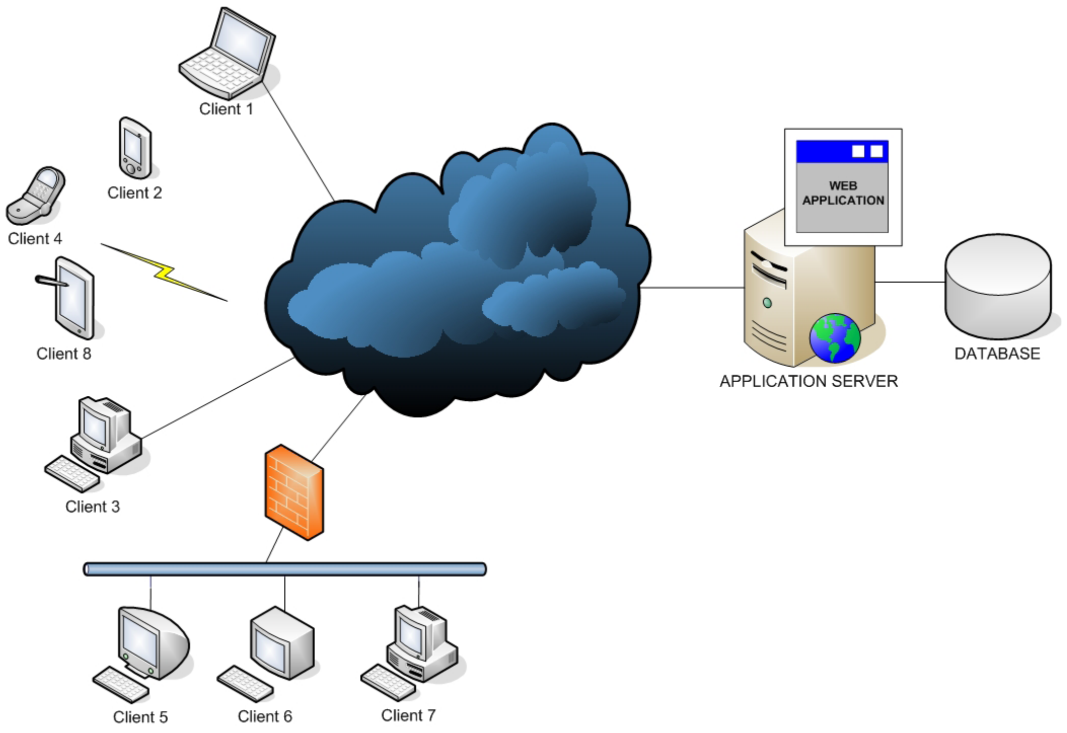

Figure 3.2 Typical Internal World Wide Web Network

The core part of a web application, as stated above, is stored on the server-side within the application server. This core consists of a real computer software program coded in a browser supported programming language such as PHP, ASP, CGI, Perl, Java/JSP, J2EE. Generally, to run the application, you must deploy it in a server and configure it properly. However, the way you install web applications depends on server machine you are using and also the particular application used.

3.1 Architecture

From a hacker's perspective, a corporation's web application can be viewed as a horizontal value chain of layers [21].

3.1.1 Client-Server

According to Connolly et al. [10], the web itself is comprised of a network of computers, and each computer acts in different roles: as a client, a server or both. In order to accommodate an increasingly decentralized business environment, web applications operating on the web use the client-server architecture. The term client-server, as mentioned by Connolly et al., refers to the processes with which software components interact to form a system, i.e. client processes require resources provided by server processes. Combinations of the client-server architecture, or topology, include: (a) single client, single server; (b) multiple clients, single server; (c) multiple clients, multiple servers. The client in a web application is usually represented by a web browser like Internet Explorer or Netscape Navigator. Servers typically include web servers, e.g. Microsoft Internet Information Server, Apache and Tomcat [3][19][20].

3.1.2 The Client-Server Architecture and Layers

Before proceeding, we stress that the layers we will refer to throughout this thesis concern responsibilities and task processing in web applications, and not how communication in networks are organized into abstraction levels. Therefore, we do not consider the layered approach taken in models such as the Open Systems Interconnection (OSI) reference model to be relevant in our discussions. In a client-server architecture, applications can be modeled as consisting of logical layers. While there exist different conventions for naming those layers, we conclude that the following three different layers are included [6][11][12][16][18][19][20]:

Presentation: The layer where information is being presented to users and which constitutes the interaction point between users and the application. This layer is actually constituted of two parts, where one part is dedicated to the client-side and the other part concerns the server side. While this layer generates and decodes web pages, it can also be responsible for presentation logic, meaning that components of this layer can reside both on the client-side and server-side. Distributed logic needed to connect to a proxy layer on the server-side along with a proxy tier in order to make use of middle-ware, e.g. CORBA and RMI, could also reside here.

Application logic: The layer where application logic and business logic and rules are implemented. This layer processes user input, makes decisions, performs data manipulation and translation into information, including calculations and validations, manages work flow, e.g. keeping track of session data, and handles data access for the presentation layer.

Data management: The layer responsible for managing both temporary and permanent data storage, including database operations.

3.1.3 The Client-Server Architecture and Tiers

We have found several different models which describe how web applications are composed using the logical layers mentioned in section 3.1.2 [6][11][12][16][18][19][20]. One main characteristic shared by those models constitutes the combination of logical layers into a 2, 3 or n-tier architecture [Connolly et al. [19]] in order to provide for a separation of tasks, where a tier is defined as one of two or more rows, levels and ranks arranged one above another.

Two-Tier Architecture

In the two-tier client-server architecture, mentioned by Connolly et al. [19] as the basic model for separating tasks, clients constitute the first tier and servers the second tier. A client is primarily responsible for presentation services, including handling user interface actions, performing application logic and presentation of data to the user and performing the main business application logic. The server is primarily concerned with supplying data services to the client. Data services provide limited business application logic, typically validation of the client and access control to data. Typically, the client would run on end-user desktops and interact with a centralized DBMS over a network.

Three-Tier Architecture

In three-tier architecture, the first tier still constitutes the client which is now considered a thin client, i.e. is only responsible for the application’s user interface and possibly simple logic processing, such as input validation. The core business logic of the application now resides in its own tier, the middle tier that runs on a server and is often called the application server. The third tier constitutes an RDBMS, which stores the data required by the middle tier, and may run on a separate server called the database server.

N-Tier Architecture

This type of architecture simply implies any number of tiers. One example of this is when the web server and database server reside in separate computers. Another example is when several database servers are used and one computer is dedicated responsible of managing access to each database server, running on separate computers. [16][19]

3.2 Input Validation Based Vulnerabilities

The most prominent class of input validation errors are SQL injections. SQL injections are the classes of vulnerabilities in which an attacker causes the web application server to produce HTML documents and database queries, respectively, that the application programmer did not intend.

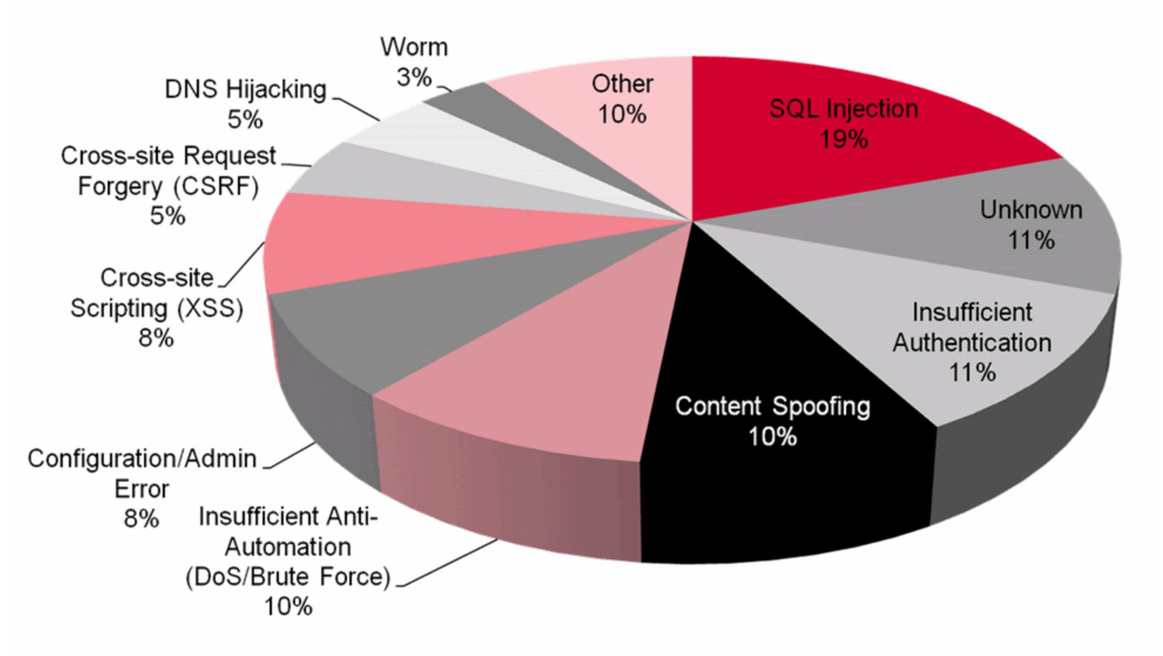

Figure 3.2.1 : Reported Web Attacks in 2009 [15]

Figure 3.2.1 shows percentages of reported web attacks for the year 2009(this data comes from the Web Hacking Incidents Database)[15].Although many attacks go unreported or even undetected, this chart shows that 19% of the web-based attacks that made the press in 2009 were SQL injection.

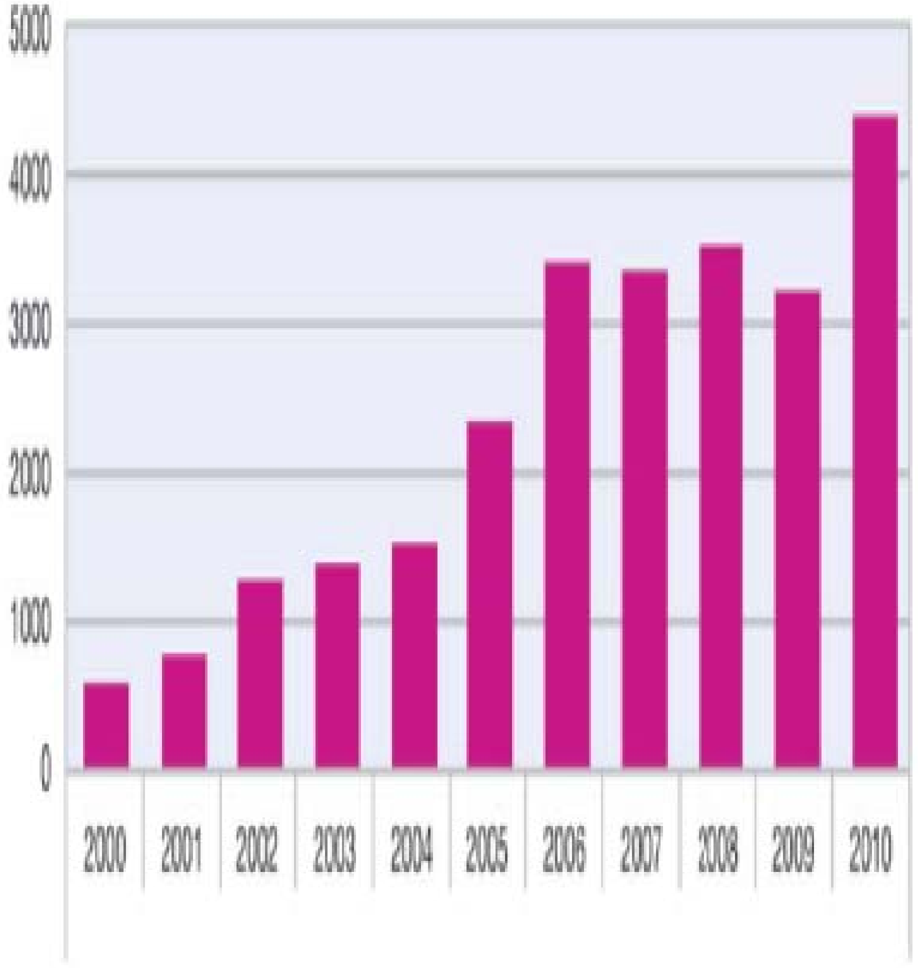

Figure 3.2.2 : Vulnerability Disclosures in the First Half of Each Year 2000-2010 [10]

Vulnerability disclosures up 36% where web applications continue to be the largest category of disclosure. Increase in vulnerability disclosures due to significant increases in public exploit releases and to efforts by several vendors to identify and mitigate security vulnerabilities. The most critical two vulnerabilities disclosed in the first half of 2010 were remote code execution vulnerabilities in Java Web Start and Microsoft Windows Help and Support Center. Both were publicly disclosed before patches were available from the respective vendors.(this is from IBM X-Force®)[10].

Most Attacked Organization

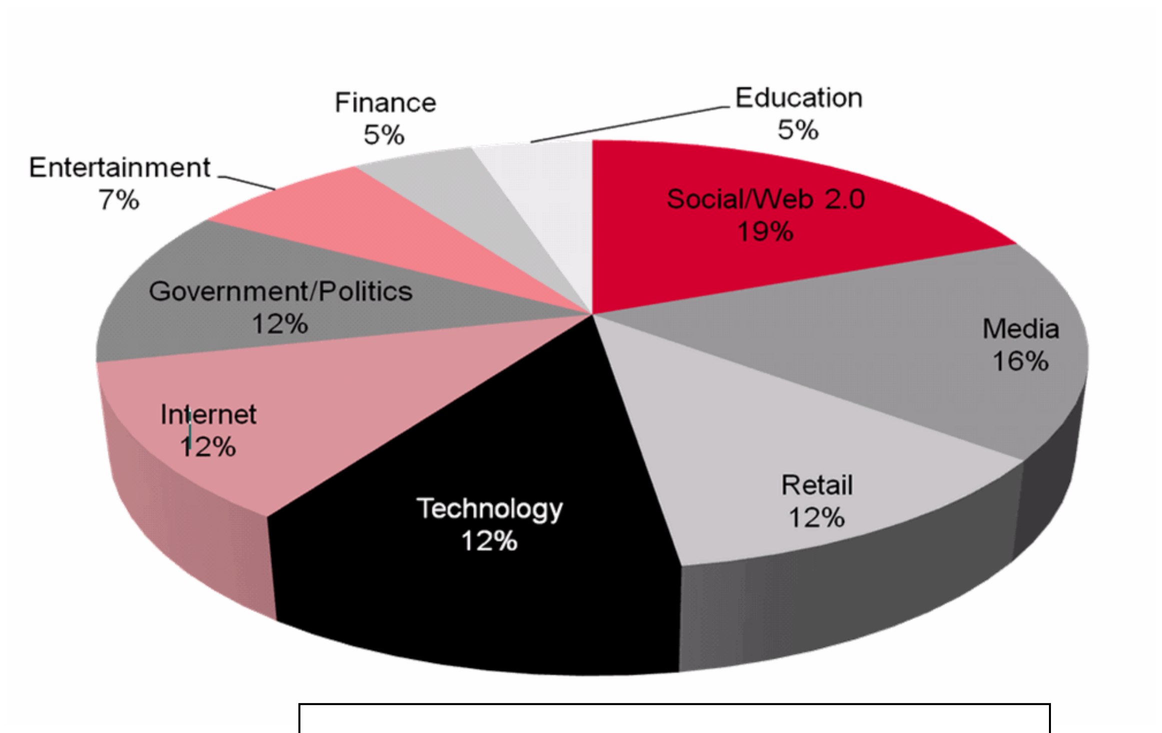

Figure 3.2.3 : Most Attacked Organization [15]

Another aspect we looked into is the type of organizations attackers chose as targets. We found that the largest category of hacked organizations is social/web .Besides this government is prime target due to ideological reasons.(this data comes from the web hacking incidents database[15]).These statistics, however, are biased, to a degree, as the public disclosure requirements of government and other public organizations are much broader than those of commercial organizations.

On the commercial side, Internet-related organizations top the list. This group includes retail shops, comprising mostly e-commerce sites, media companies and pure internet services such as search engines and service providers. It seems that these companies do not compensate for the higher exposure they incur, with the proper security procedures.

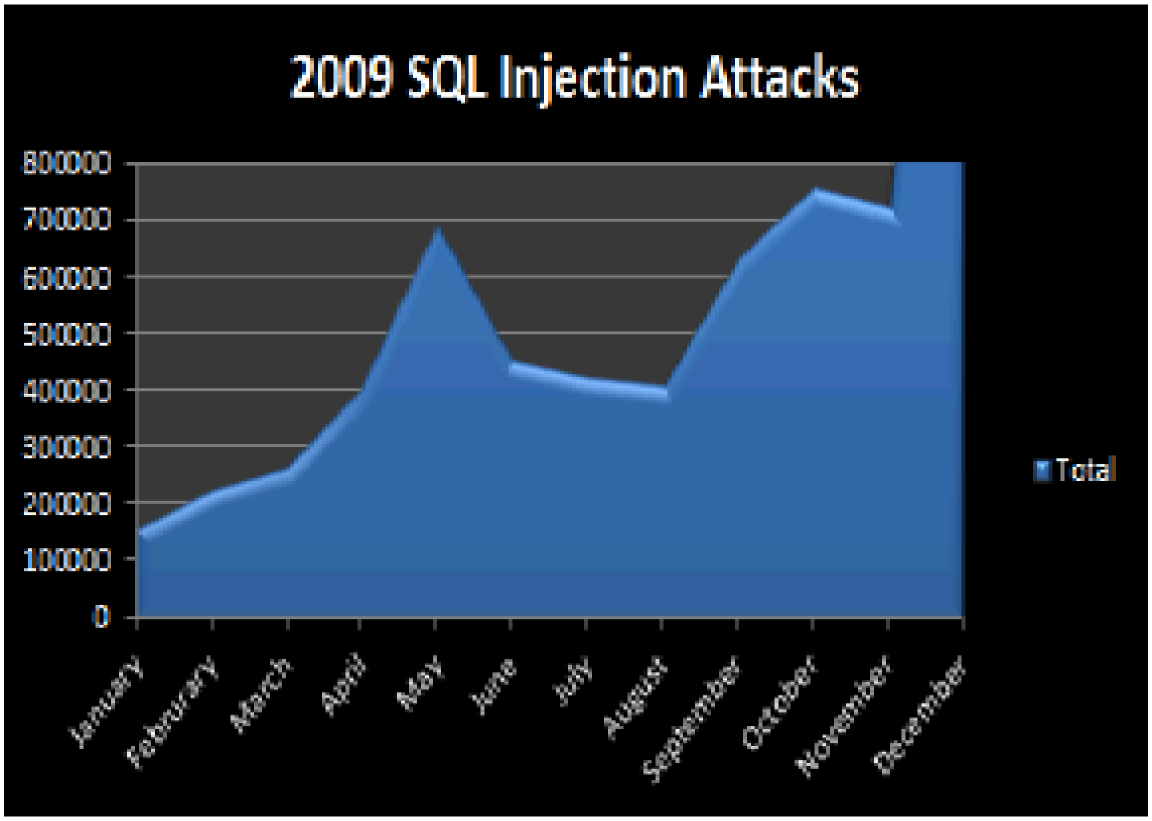

Figure 3.2.4: SQL Injection attacks Monitored by IBM ISS Managed Security Services [10]

3.3 Communication

A typical communication exchange in a business web application, according to Connolly et al., is initiated by users that request information. The client takes a user’s request, checks the syntax and generates database requests in e.g. SQL. Then, the client transmits the message to the server, waits for a response, and formats the response for the end-user. The server accepts and processes the database requests, then transmits the results back to the end-user.

3.3.1 Information

Information on the web is stored in documents and the formatting language, or system, most commonly used is the HTML. Using HTML, documents are marked up, or tagged, to allow for publishing on the web in a platform independent manner. HTML documents are displayed in web browsers that understand and interpret HTML. [19]

3.3.2 Content

HTML documents stored in files constitute static content, i.e. the content of the document does not change unless the file itself is changed. However, documents resulting from requests such as queries to databases need to be generated by the web servers. These documents are dynamic content and as databases are dynamic, changing as users create, insert, update, and delete data, the generation of dynamic web pages is a more appropriate approach than static content, particularly in web applications. [19]

3.3.3 Protocol

The exchange of information in web applications is mainly governed by protocols such as HTTP or HTTPS, which define how clients, i.e. web browsers, and servers, i.e. web servers, communicate. HTTP relies on a request-response paradigm and a transaction consists of the following stages [4][19][20]:

Connection The client establishes a connection with the web server.
Request The client sends a request message to the web server
Response The web server sends a response, i.e. a HTML document, back to the client.
Close The connection is closed by the web server.

Basically, a request in a HTTP connection constitutes an object containing, e.g. a requested resource. Consequently, a response is the result to be presented in the web browser. When a user visits a page, web pages, client-side scripts and formatting components are sent back to the client for rendering and presentation. In the case a user requests data contained in a relational database, user input parameters are typically embedded in the request. Those parameters can be included as arguments to methods in application processing components that dynamically build SQL queries.

The response object will contain data for presentation in the web browser. That data may have been parsed and prepared by either application processing components, server-side scripts or both for rendering purposes: either for tailoring the graphical design or ease the rendering process in the web browser.

HTTP brings up several security weaknesses. A HTTP request is composed of different parts and attackers can manipulate those parts in order to try to bypass security mechanisms. The web server listens on an open port for incoming requests from clients. For general web traffic, i.e. HTTP, port 80 is often used as the default port and for encrypted traffic, i.e HTTPS, port 443 is normally chosen. However, each web server requires a unique port to listen to and since corporations can have several web servers, the port of each server has to be configured. Moreover, application servers require open ports as well. While this means that an attacker cannot always assume that the web server of choice listens to port 80, the important issue is that there exists an open port through security mechanisms such as firewalls into a corporation's web server. [21]

3.3.4 URL Encoding

According to OWASP [20], a server can receive input from a client in two basic ways: either data is passed in HTTP headers or it can be included in the query portion of the requested Uniform Resource Locator (URL), which uniquely defines where resources can be found on the Internet. Both methods correspond to two methods used when including input in client requests: GET and POST. Manipulation of a URL or a form is simply two sides of the same issue. However, when data is included in a URL, it must be specially encoded to conform to proper URL syntax. Unfortunately, as OWASP notes, the URL encoding mechanism allows virtually any data to be passed from a client to the server.

Chapter 4

RDBMS and SQL

In this section we give an introduction to Relational Database Management Systems (RDBMS) and the Structured Query Language (SQL), its syntax and usage. We do not intend to a give a complete review of these subjects, as we consider this to be outside this thesis' boundaries. This section relays heavily on the book written by Connolly et al. [19] and its comprehensive explanation of SQL.

4.1 RDBMS

Connolly et al. [19] defines a database as "A shared collection of logically related data (and a description of this data), designed to meet the information needs of an organization." A database management system, DBMS, is used to allow user interaction with the database. Such DBMSs are defined by Connolly et al. as "A software system that enables users to define, create, and maintain the database and provide controlled access to this database."

According to Gollman [4], the most widely used database model in databases today is the relational model, which is used to organize relational databases. A DBMS which relies on the relational model (an RDBMS) is a system through which users can administrate a database which is perceived as a collection of tables. These tables (or relations) are organized as a two dimensional array containing rows and columns. A table's rows (or tuples) are the elements of the table, and its columns (or attributes) are the names of data that is represented.

As Connolly et al. state, the Structured Query Language (SQL) has become the standard language used in relational databases and is the only database language to gain wide acceptance. This language allows users to administrate databases using an RDBMS as well as communicating with them.

4.2 SQL

The SQL standard, according to Connolly et al. [19], was defined by the American National Standards Institute (ANSI) and was later adopted by the International Standards Organization (ISO). Its objectives are to allow users to create database and relation structures, managing tables by inserting, modifying, and deleting data as well as retrieve information from the database through queries. SQL queries are commands that are passed to the RDBMS, and specify which data is to be gathered from one or more tables and how it should be arranged. We intend to follow the ISO SQL standard used by Connolly et al.[ISO 9075:1992(E)], and will be using it throughout this thesis unless stated otherwise.

SQL consists of two major components: the Data Manipulation Language (DML) and the Data Definition Language (DDL). Using the DML, users can manipulate data stored inside tables in the database, while the DDL allows creating and destroying database objects such as schemas, domains, tables, views and indices.

4.2.1 DML

The Data Manipulation Language has four available statements, namely SELECT, INSERT, UPDATE and DELETE. We describe each of these statements according to Connolly et al. [10] using the syntax described in table 4.2.1.

Symbol	Represents
SELECT, INSERT, . . .	reserved words
table name, column list, . . .	user-defined words
|	choice among alternatives
{}	required element, for example {a}
[]	optional element, for example [a]
. . .	optional repetition (zero or more times)

Table 4.2.1: SQL syntax

SELECT used for retrieving information from one or more tables in the database and displaying it.

The syntax of the SELECT statement is given below:

```
SELECT [DISTINCT|ALL]

{*|column_expression[AS new_name]][,...]}

FROM table_name [alias][,...]

[WHERE condition]

[GROUP BY column_list][HAVING condition]

[ORDER BY column_list]
```

where column expression represents a column name or expression, newname is a new temporary name to use for the column expression, table name represents the name of the database table or view table to select from, alias represents an optional name for the table name, condition is the condition upon which selection is made or a condition for display (see HAVING), and column list represents the list of table columns to group or order the result upon.

The sequence of the SELECT statement processing, and the meaning of the reserved words are:

FROM specifies which tables to choose from

WHERE filters the selected data rows due to a condition

GROUP BY groups together rows with same column value

HAVING filters the selected groups due to a condition

SELECT specifies which column should appear in the result

ORDER BY specifies the order to sort the output upon

INSERT used for adding new data rows in a table.

The syntax of the INSERT statement is given below:

INSERT INTO table_name[(column_list)]

VALUES(data_value_list)

where table name represents the name of the database table or view table, column list represents the list of table columns to update, and data valuelist represents the list of values to enter into each column in the new row. The number, position, and type of data values must correspond to the table’s column list.

UPDATE used for modifying data rows in a table.

The syntax of the UPDATE statement is given below:

UPDATE table_name

SET column_name1 = data_value1

[, column_name2 = data_value2...]

[WHERE search_condition]

where table name represents the name of the database table or view table, column name represents the column name to modify, and data value represents the new value to enter into the column. The new given value must correspond to the table’s column. The WHERE clause specifies which row is to be modified, according to the search condition. If omitted, the whole table will be affected.

DELETE used for removing data rows from a table.

The syntax of the DELETE statement is given below:

DELETE FROM table_name

[WHERE search_condition]

where table name represents the name of the database table or view table, and the WHERE clause specifies which row is to be modified, according to the search condition. If omitted, all data in the table will be deleted.

SELECT statements can be used to retrieve data in many different ways. In order to explain this, we give here a list of different query formulations and the way they are commonly used according to Connolly et al.

• **Simple Queries** can be used to retrieve either all or a selection of columns and rows from one or more tables. A condition can also be specified to minimize the selection.

• **Sorting Results** can be achieved by using the ORDER BY clause.

• **Aggregate Functions** are used to retrieve numeric information about the data. The clauses COUNT, SUM, AVG, MIN and MAX are used to retrieve number of rows, sum of values, values average, minimum value and maximum value, respectively.

• **Grouping Results** can be achieved using the GROUP BY clause.

• **Sub queries** can help creating complex queries wherein result from a secondary query can used for instance as a condition for the primary query.

• **The ANY and ALL Clauses** can be used to compare results of a primary query with all or any of the results of a secondary query.

• **Multi-Table Queries** are used to combine columns from different tables through usage of different JOIN clauses.

• **The EXISTS and NOT EXISTS Clauses** can be used to check if a value exists or not in a table or in a result from a secondary query.

• **Combining Result Tables** can be made using the UNION, INTERSECT and EXCEPT clauses.

4.2.2 DDL

Connolly et al. [19] defines the Data Definition Language (DDL) as “A descriptive language that allows the DBA or user to describe and name the entities required for the application and the relationships that may exist between the different entities.” Thus, the DDL is used when manipulating the database’s meta-data, which describes the objects contained in the database and allows access to them. The DDL does not allow users to manipulate data stored in the database.

4.3 Query Techniques

An SQL query to be executed in a RDBMS can be constructed using two techniques. Either the query is allowed to be dynamically tailored with respect of both SQL keywords and query arguments, or the query syntax is unchangeable, only allowing arguments to be passed [19].

4.3.1 Dynamic SQL

Dynamic SQL refers to the concept of allowing an SQL query to be dynamically built by concatenating statements and using variables that supply the query with dynamic values. According to Connolly et al. [19] and Khatri [9], the query is typically stored in a variable and the query builders consist of application logic components that adds SQL syntax and arguments to the variable in a process governed by specified conditions. Such queries are interpreted and compiled at run-time by the RDBMS, meaning that the query will be compiled every time it is executed. Since dynamic SQL allows SQL syntax to be added, both SQL keywords and values may be passed as arguments to queries.

4.3.2 Static SQL

Static SQL refers to the concept of using fixed and unchangeable SQL queries. Such queries are predefined and compiled and are not permitted to add SQL keywords, defined in DDL or DML. Only arguments to clauses, e.g. WHERE, may be allowed to be passed to the queries. Either the query is embedded in application logic code in form of prepared statements or it resides in RDBMS as stored procedures.

Stored procedures are pre-compiled collections of SQL statements, or sub-routines that reside in the RDBMS. Either they are supplied by the database vendor, i.e. system stored procedures, or additionally constructed by system developers, database administrators or application programmers. They allow a developer to access and manipulate databases quickly and efficiently. Since they are compiled in advance, they possess the property of being executed faster than dynamic SQL. Another property is that once created, stored procedures cannot be modified via dynamic SQL. Stored procedures are executed by invoking a command that includes the procedure identifier. This can be done either from a command prompt or from application programs written in languages such as C or Visual Basic. Several RDBMS supports this feature, but the set of stored procedures that follow with the installation and the syntax for invoking them varies. [9][17][19]

4.4 Error Messages

RDBMSs have in-built error handling mechanisms that may generate error messages when for example an SQL query could not be executed. The error message format used and degree of details embedded in generated messages vary from RDBMS to RDBMS. Nevertheless, if you run a query and accidentally make a mistake by entering e.g. a table that does not exist in the database, the RDBMS may return an error message containing information about the error. Some RDBMSs even react to all errors in the same manner, whether those errors are generated by users, databases, objects, or the system. [5][13][21]

Error messages are typically propagated back to the source that caused the error. The web server or application server will propagate an error page that displays the message to the client. Web application developers can take advantage of these messages for debugging purposes, as noted by Spett [21]. However, as we shall see in section 5, it is not a wise error-handling approach to let web servers display these error messages in error pages to users of web applications.

4.5 Security

Database security, according to Connolly et al. [19], concerns “The protection of the database against intentional or unintentional threats using computer-based or non-computer-based controls.” Besides the effect that poor database security can have on the database, it may also threaten other parts of a system and thus an entire organization. The risks related to database security are:

• **Theft and fraud** which are activities made intentionally by people. This risk may result in loss of confidentiality or privacy.

• **Loss of confidentiality** which refers to loss of organizational secrets.

• **Loss of privacy** which refers to exposure of personal information.

• **Loss of integrity** which refers to invalid or corrupt data.

• **Loss of availability** which means that data or system cannot be reached.

Threats that correspond to those risks are such situations or events in which it is likely that an action, event or person will harm an organization. Threats can be tangible, that is, cause loss of hardware or software, or intangible, as in with loss of credibility or confidence. In order to be able to face threats, a risk analysis should be conducted, in which a group of people in an organization tries to identify and gather information about the organization’s assets, the risks and threats that may harm the organization and the countermeasures that can be used to face those risks. Decisions made using such risk analysis are thereafter used to implement security measures in the system. These security measures can be computer-based controls or non-computer-based controls.

4.5.1 Computer-Based Controls

According to Connolly et al. [19], computer-based controls are used for protecting DBMS through means of authorization, views, backup and recovery, integrity, encryption and associated procedures.

Authorization

Authorization is used to define which activities (or privileges) are granted to different users (or subjects), which allows them to manipulate or retrieve information from different database objects. In order to ensure that the user is who she claims, authentication is used. Usually, a simple mechanism of usernames and passwords is used, whether in the DBMS or in combination with the operating system where the DBMS resides. A user is asked to fill her name and password, and the authentication mechanism confirms that the user is who she claims to be by comparing the password with the corresponding password in a list it maintains.

The DBMS usually maintains a list of privileges that subjects have on certain database objects. A DBMS that operates as a closed system, maintains a privileges list in which users are not allowed to operate on any objects except the ones in the list. A DBMS that operates as an open system, on the other hand, allows users to operate on all objects except those that are explicitly removed and listed in the privileges list.

Privileges may also be group-based or role-based. Both users and objects may be joined in a group and privileges may be given to a group of users or objects. Certain roles can also be given privileges on objects, and a number of different users may undertake a certain role.

Views

Views are virtual tables that are created through some operations on database objects. By removing certain columns or rows and combining certain tables, such views can be used to limit the scope of objects that users can manipulate or retrieve information from.

Backup and Recovery

In order to be able to recover from a failure, a DBMS must regularly make a copy of the database and log files. Log files are a list of activities made in the database that can be used to recover the database after a failure. Checkpoints made in certain time intervals can assure that the backup and log files are synchronized. This allows for safe recovery since operations that are listed in the log file need only be carried out from the point in time when the last backup was made.

Integrity

Integrity controls can be used to see to that data in database does not get corrupt. Such controls are called relational integrity controls, and are rules that some databases implement internally to maintain data validity. Other database does not implement those controls and it is up to the application programmer that uses the database to see to that data validity is being maintained.

Encryption

Encryption is a method that is used for encoding the data so that other programs cannot read it. Some DBMSs contain an internal encryption mechanism, while other relays on the operating system or third-party programs.

Associated Procedures

Connolley et al. describe some associated procedures that should be used to further protect the database:

• **Authorization and authentication:** in order for these mechanisms to work properly, a password policy should be maintained, which regulates matters like minimum passwords length, how often they should be replaced, as well as revoking old passwords.

• **Backup:** procedures should regulate how often backups should be made as well as what parts of the database backups should include. Furthermore, backups should be kept in a safe place.

• **Recovery:** recovery mechanisms should regulate how backups and logs can be used in case of failure. These procedures should also be tested regularly.

• **Audit:** audits should be carried out regularly to control the security and to see to that all mechanisms are adequately functioning.

• **Installation of new software:** before any new software is to be installed, it should be properly tested so that it would not harm any data or mechanisms.

• **Installation/upgrading of system software:** any system upgrades should by documented and reviewed. Before such upgrades take place, the risks of such an act should be considered and plans should be made for possible failures and changes.

4.5.2 Non-Computer-Based Controls

The most important countermeasure among non-computer-based controls is the security policy and the contingency plan. A security policy concerns security maintenance in an organization, and contains ". . . a set of rules that state which actions are permitted and which actions are prohibited." [21] .A contingency plan is a detailed description of the actions that should be taken in order to deal with unusual events, such as sabotage, fire or flood.

Chapter 5

SQL Injection

SQL injection is a code injection technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed. It is an instance of a more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another.

5.1 Scope

We will view SQL injection as the majority of the authors do: a technique used for manipulating server-side scripts that send SQL queries to an RDBMS. This is done by manipulating client-side data, including changing SQL values and concatenations of SQL statements, which are sent to a web server embedded in HTTP requests. Once the web server receives a request, it forwards the information in it to a script which uses that information to build SQL queries. The goal of the attacker who uses SQL injection is to manipulate with the SQL query used by the script so that it would yield unwanted results, such as fetching, inserting, manipulating or deleting protected rows or tables in the database. [21].

Before proceeding, we think that a discussion of the scope of SQL injection is necessary. Attack methods of SQL injection have by some authors been classified into direct and indirect attacks.

Using direct attacks, an attacker tries to take control of an RDBMS. The purpose of such attacks is to further take control of other host computers and compromise a network. First, attackers scan for open ports that database servers are listening to. If such ports are found, they continue with executing system commands through a command console, communicating with the RDBMS directly. [21]

Indirect attacks, on the other hand, are performed through web applications. True, it is possible to execute commands by embedding calls to stored procedures in dynamic SQL and that may cause devastating results if successful [2][14][21]. However, the main purpose is to directly attack the RDBMS in general and its stored data in particular [2][21].

A majority of the authors do not mention or discuss direct attacks. This may stem from the fact that they either are not aware of such flaws or that they do not consider them as falling into the scope of SQL injection. Regardless of the reason, direct attacks are conducted through the RDBMS and aims at the network infrastructure. When discussing direct attacks, authors refer to direct communication with the RDBMS and not attacks on the RDBMS itself. It seems to be true that attackers can take advantage of some aspects of SQL injection when performing such attacks. However, we consider the concept of direct attacks to be somewhat misleading since it does not relate to web applications. Furthermore, from a security perspective, we think that direct attacks relate to network security rather than application security. Flaws like open ports that allow attackers to communicate with the RDBMS using arbitrary protocols from command consoles can be prevented by existing network security countermeasures as well as database security configuration, e.g securing the system administrator account. Therefore, we consider direct attacks to be outside the scope of this thesis.

5.2 Basics

As noted by Spett [21], a web application can, from a hacker's perspective, be viewed as consisting of the following layers: desktop layer, transport layer, access layer, network layer and application layer. At the desktop layer, computers with web browsers acting as clients are used for accessing a system. The transport layer represents the web, and the access layer constitutes the entrance point into a corporation’s internal system from the web. The network layer consists of the corporation's internal network infrastructure and finally, the application layer includes web servers, application servers, application logic and data storage. Every layer may have its own implemented countermeasures in order to detect, prevent and recover from attacks, as

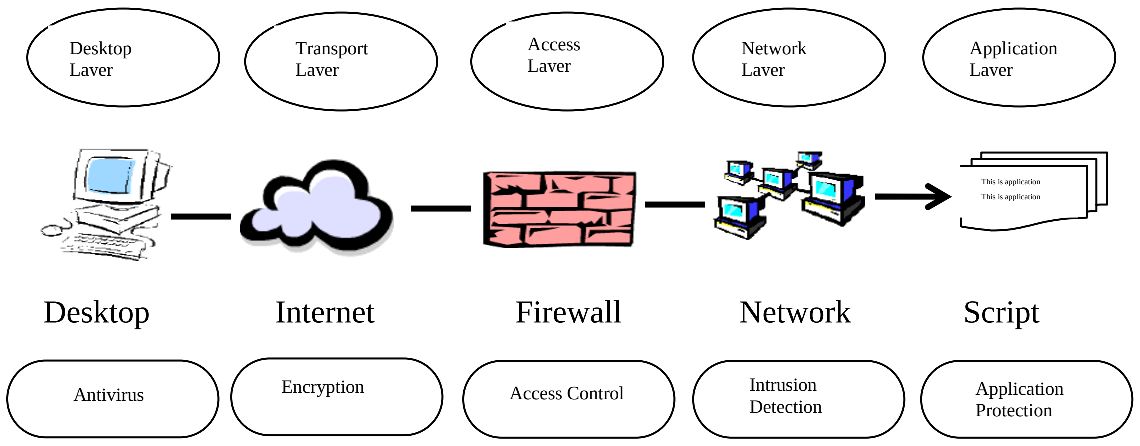

Figure 5.2.1: Security layers in web applications [21]

Unfortunately, SQL injection attacks can only be prevented in application logic components such as scripts and programs in the application layer. No matter how many resources and how much effort a corporation spends in the other layers, if application security has not been properly applied in the application layer, their web applications may contain vulnerabilities that SQL injection attackers can exploit. We do not say that countermeasures like encryption, firewalls, intrusion detection and database security are not important. They are effective when dealing with other types of attacks. However, they have been shown insufficient and ineffective regarding SQL injection and therefore we will not consider them [21]. Encryption for example, only protects stored data or data during transport in and between lower layers. In the context of web applications, user input may be encrypted between the client-side and server-side. Furthermore, SQL queries may be encrypted during transport between components such as scripts and programs on the server-side. But in order for the server-side to construct SQL queries and for RDBMS to execute them, they must first be decrypted. The data may still be encrypted but the SQL queries could have been manipulated through SQL injection.

Basically, most web servers are protected by firewalls. However, from a security perspective, web applications offer users legitimate channels through firewalls into corporations systems. The reason for this is that when clients request services from servers on the web, the underlying communication takes place through HTTP, and web applications are no exceptions. HTTP is firewall-friendly, i.e. it is one of the few protocols most firewalls allow through. This stems from the fact that HTTP requests are considered legal, since traffic between clients and servers must be allowed in order for the web applications to be of any use. SQL injection takes advantage of this property by embedding attacks in HTTP requests.

5.3 How SQL injection Attacks (SQLIAs) Work

SQL injection refers to a class of code-injection attacks in which data provided by the user is included in the SQL query in such a way that part of the user's input is treated as SQL code. It is a trick to inject SQL query or command as an input possibly via the web pages. They occur when data provided by user is not properly validated and is included directly in a SQL query. By leveraging these vulnerabilities, an attacker can submit SQL commands directly access to the database. There are two major SQL injection techniques: i) access through login page or user input and ii) access through URL.

The first technique is the easiest in which it bypasses the login forms where users are authenticated by using passwords. This kind of technique can be performed by the attackers through: 'or' condition, 'having' clause, multiple queries and extended stored procedure.

This kind of vulnerability represents serious threats.

Select * from users where username = ' " & username & " ' and password = ' " & password & " ' "

If the username and password as provided by the user are used, the query to be submitted to the database takes the form:

Select * from users where username = 'guest' and password = 'password'

If the user were to enter [' or 1=1 --] and [ ] instead of [guest] and [password], the query would taken form:

Select * from users where username = ' ' or   1=1 --' and password = ' '

The query now checks for the conditional equation of [1=1] or an empty password, then a valid row has been found in the users table. The first [‘] quote is used to terminate the string and the characters [--] mark the beginning of a SQL comment, and anything beyond is ignored. The query as interpreted by the database now has a tautology and is always satisfied. Thus an attackers can bypass all authentication modules gaining unrestricted access to critical information on the server. This attack can be used to gain confidential information, to bypass authentication mechanisms, to modify the database, and to execute arbitrary code.

The second technique can be performed by the attackers through: manipulating the query string in URL and using the SELECT and UNION statements.

When a user enters the following URL:

http://www.mydoman.com/products/products.asp?productid=123

The corresponding SQL query is executed:

Select productName, productDescription from products where productNumber = 123

An attacker may abuse the fact that the productId parameter is passed to the database without sufficient validation. The attacker can manipulate the parameter’s value to build malicious SQL statements. For example, setting the value [123 or 1=1] to the productId variable results in the following URL:

http://www.mydoman.com/products/products.asp?productid=123 or 1= 1

The corresponding SQL statement is:

Select productName, productDescription from products where productNumber = 123 or 1=1

This condition would always be true and all productName and productDescription pairs are returned. The attacker can manipulate the application evan furthere by inserting malicious commands. For example, an attacker can request the following URL:

http://www.mydoman.com/products/products.asp?productid=123; drop table products

In this example the semicolon is used to pass the database server multiple statements in a single execution. The second statement is “drop table products” which causes SQL server to delete the entire products table .

An attacker may use SQL injection to retrieve data from other tables as well. This can be done using the SQL UNION SELECT statement. The UNION SELECT statement allows the chaining of two separate SQL SELECT queries that have nothing in common. For example, consider the following SQL query:

Select productName, productDescription from products where productNumber = 123 union select username , password from users;

The result of this query is a table with two columns, containing the results of the first and second queries, respectively. An attacker may use this type of SQL injection by requesting the following URL:

http://www.mydoman.com/products/products.asp?productid=123 union select username , password from users;

5.4 Classification of SQLIA Techniques

An SQL injection attack has a set of properties, such as assets under threat, vulnerabilities being exploited and attack techniques utilized by threat agents. The detail feature of every property in the SQL injection attack model is identified in this section.

5.4.1. Attack Intent

Attacks can also be characterized based on the goal, or *intent*, of the attacker. Therefore, each of the attack type definitions that we provide in Section 4 includes a list of one or more of the attack intents defined in this section.[22]

Identifying injectable parameters: The attacker wants to probe a Web application to discover which parameters and user-input fields are vulnerable to SQLIA.

Performing database finger-printing: The attacker wants to discover the type and version of database that a Web application is using. Certain types of databases respond differently to different queries and attacks, and this information can be used to “fingerprint” the database. Knowing the type and version of the database used by a Web application allows an attacker to craft database specific attacks.

Determining database schema: To correctly extract data from a database, the attacker often needs to know database schema information, such as table names, column names, and column data types. Attacks with this intent are created to collect or infer this kind of information.

Extracting data: These types of attacks employ techniques that will extract data values from the database. Depending on the type of the Web application, this information could be sensitive and highly desirable to the attacker. Attacks with this intent are the most common type of SQLIA.

Adding or modifying data: The goal of these attacks is to add or change information in a database.

Performing denial of service: These attacks are performed to shut down the database of a Web application, thus denying service to other users. Attacks involving locking or dropping database tables also fall under this category.

Evading detection: This category refers to certain attack techniques that are employed to avoid auditing and detection by system protection mechanisms.

Bypassing authentication: The goal of these types of attacks is to allow the attacker to bypass database and application authentication mechanisms. Bypassing such mechanisms could allow the attacker to assume the rights and privileges associated with another application user.

Executing remote commands: These types of attacks attempt to execute arbitrary commands on the database. These commands can be stored procedures or functions available to database users.

Performing privilege escalation: These attacks take advantage of implementation errors or logical flaws in the database in order to escalate the privileges of the attacker. As opposed to bypassing authentication attacks, these attacks focus on exploiting the database user privileges.

5.4.2 Assets

Assets are information or data an unauthorized threat agent attempt to gain.

Database Server Fingerprint: The database server fingerprint contains information about the database system in use. It identifies the specific type and version of the database, as well as the corresponding SQL language dialect. A compromise of this asset may allow attackers to construct malicious code specially for the SQL language dialect in question.

Database schema: The database scheme describes the server’s internal architecture. Database structure information such as table names, size, and relationships are defined in the database schema. Keeping this asset private is essential in keeping the confidentiality and integrity of the database data. A compromise in the database, schema may allow attackers to know the exact structure of the database, including table, row, and column headings.

Database data: The database data is the most crucial asset in any database system. It contains the information in the tables described in the database schema, such as prices in an online store, personal information of clients, administrator passwords, etc. A compromise in the database data will usually result in failure of the system’s intended functionality, thus, its confidentiality and integrity must be protected.

Host: A host is a discrete node in any network, usually uniquely defined with an IP address. It may have various privileges in a network and may be a database server or a regular computer terminal.

Network: A network interconnects numerous hosts together and allows communication between them. A compromise in a network will most likely compromise every host in the network. Some networks may also be interconnected with other networks, furthering the potential damage, should an attack be successful.

5.5 Methodology for a Successful SQLIA

Attack techniques are the specific means by which a threat agent carries out attacks using malicious code. Threat agents may use many different methods to achieve their goals, often combining of these sequentially or employing them in different varieties.[22]

5.5.1 Attacks Techniques

Tautologies

Attack Intent: Bypassing authentication, identifying injectable parameters, extracting data.

Description: The general goal of a tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. The consequences of this attack depend on how the results of the query are used within the application. The most common usages are to bypass authentication pages and extract data. In this type of injection, an attacker exploits an injectable field that is used in a query's **WHERE** conditional. Transforming the conditional into a tautology causes all of the rows in the database table targeted by the query to be returned. In general, for a tautology-based attack to work, an attacker must consider not only the inject-able/vulnerable parameters, but also the coding constructs that evaluate the query results. Typically, the attack is successful when the code either displays all of the returned records or performs some action if at least one record is returned.

Example: In this example attack, an attacker submits " ' or 1=1 - - " for the *login* input field (the input submitted for the other fields is irrelevant). The resulting query is:

SELECT accounts FROM users WHERE login='' or 1=1 -- AND pass='' AND pin=

The code injected in the conditional **(OR 1=1)** transforms the entire `WHERE` clause into a tautology. The database uses the conditional as the basis for evaluating each row and deciding which ones to return to the application. Because the conditional is a tautology, the query evaluates to true for each row in the table and returns all of them.

Illegal/Logically Incorrect Queries

Attack Intent: Identifying injectable parameters, performing database finger-printing, extracting data.

Description: This attack lets an attacker gather important information about the type and structure of the back-end database of a Web application. The attack is considered a preliminary, information gathering step for other attacks. The vulnerability leveraged by this attack is that the default error page returned by application servers is often overly descriptive. In fact, the simple fact that an error messages is generated can often reveal vulnerable/inject-able parameters to an attacker. Additional error information, originally intended to help programmers debug their applications, further helps attackers gain information about the schema of the back-end database. When performing this attack, an attacker tries to inject statements that cause a syntax, type conversion, or logical error into the database. Syntax errors can be used to identify injectable parameters. Type errors can be used to deduce the data types of certain columns or to extract data. Logical errors often reveal the names of the tables and columns that caused the error.

Example: This example attack's goal is to cause a type conversion error that can reveal relevant data. To do this, the attacker injects the following text into input field *pin*: "convert(int,(select top 1 name from sysobjects where xtype='u'))". The resulting query is:

SELECT accounts FROM users WHERE login='' AND pass='' AND pin= convert (int,(select top 1 name from sysobjects where xtype='u'))

In the attack string, the injected select query attempts to extract the first user table (xtype='u') from the database's metadata table (assume the application is using Microsoft SQL Server, for which the metadata table is called sysobjects). The query then tries to convert this table name into an integer. Because this is not a legal type conversion, the database throws an error. For Microsoft SQL Server, the error would be: *”Microsoft OLE DB Provider for SQL Server (0x80040E07) Error converting nvarchar value ’CreditCards’ to a column of data type int.”*

There are two useful pieces of information in this message that aid an attacker. First, the attacker can see that the database is an SQL Server database, as the error message explicitly states this fact. Second, the error message reveals the value of the string that caused the type conversion to occur. In this case, this value is also the name of the first user-defined table in the database: “CreditCards.” A similar strategy can be used to systematically extract the name and type of each column in the database. Using this information about the schema of the database, an attacker can then create further attacks that target specific pieces of information.

Union Query

Attack Intent: Bypassing Authentication, extracting data.

Description: In union-query attacks, an attacker exploits a vulnerable parameter to change the data set returned for a given query. With this technique, an attacker can trick the application into returning data from a table different from the one that was intended by the developer. Attackers do this by injecting a statement of the form: **UNION SELECT <rest of injected query>.** Because the attackers completely control the second/injected query, they can use that query to retrieve information from a specified table. The result of this attack is that the database returns a dataset that is the union of the results of the original first query and the results of the injected second query.

Example: Referring to the running example, an attacker could inject the text “’ UNION SELECT cardNo from CreditCards where acctNo=10032 - -” into the login field, which produces the following query:

SELECT accounts FROM users WHERE login=" UNION SELECT cardNo from CreditCards where acctNo=10032 -- AND pass="AND pin=

Assuming that there is no login equal to “”, the original first query returns the null set, whereas the second query returns data from the “CreditCards” table. In this case, the database would return column “cardNo” for account “10032.” The database takes the results of these two queries, unions them, and returns them to the application. In many applications, the effect of this operation is that the value for “cardNo” is displayed along with the account information.

PiggyBacked

Queries

Attack Intent: Extracting data, adding or modifying data, performing denial of service, executing remote commands.

Description: In this attack type, an attacker tries to inject additional queries into the original query. We distinguish this type from others because, in this case, attackers are not trying to modify the original intended query; instead, they are trying to include new and distinct queries that “piggy-back” on the original query. As a result, the database receives multiple SQL queries. The first is the intended query which is executed as normal; the subsequent ones are the injected queries, which are executed in addition to the first. This type of attack can be extremely harmful. If successful, attackers can insert virtually any type of SQL command, including stored procedures,[1] into the additional queries and have them executed along with the original query. Vulnerability to this type of attack is often dependent on having a database configuration that allows multiple statements to be contained in a single string.

Example: If the attacker inputs “’; drop table users - -” into the *pass* field, the application generates the query:

SELECT accounts FROM users WHERE login='doe' AND pass=''; drop table users – ' AND pin=123

After completing the first query, the database would recognize the query delimiter (“;”) and execute the injected second query. The result of executing the second query would be to drop table users, which would likely destroy valuable information. Other types of queries could insert new users into the database or execute stored procedures. Note that many databases do not require a special character to separate distinct queries, so simply scanning for a query separator is not an effective way to prevent this type of attack.

Stored Procedures

Attack Intent: Performing privilege escalation, performing denial of service, executing remote commands.

Description: SQLIAs of this type try to execute stored procedures present in the database. Today, most database vendors ship databases with a standard set of stored procedures that extend the functionality of the database and allow for interaction with the operating system. Therefore, once an attacker determines which backend database is in use, SQLIAs can be crafted to execute stored procedures provided by that specific database, including procedures that interact with the operating system.

It is a common misconception that using stored procedures to write Web applications renders them invulnerable to SQLIAs. Developers are often surprised to find that their stored procedures can be just as vulnerable to attacks as their normal applications [22]. Additionally, because stored procedures are often written in special scripting languages, they can contain other types of vulnerabilities, such as buffer overflows, that allow attackers to run arbitrary code on the server or escalate their privileges [16].

```
CREATE PROCEDURE DBO.isAuthenticated

        @userName varchar2, @pass varchar2, @pin int

AS

        EXEC("SELECT accounts FROM users

        WHERE login=’" +@userName+ "’ and pass=’" +@password+ "’ and pin="
        +@pin);

GO
```

Example: This example demonstrates how a parameterized stored procedure can be exploited via an SQLIA. In the example, we assume that the query string constructed at lines 5, 6 and 7 of our example has been replaced by a call to the stored procedure defined in Figure 2. The stored procedure returns a true/false value to indicate whether the user’s credentials authenticated correctly. To launch an SQLIA, the attacker simply injects “ ’ ; SHUTDOWN; - -” into either the userName or password fields. This injection causes the stored procedure to generate the following query:

```
SELECT accounts FROM users WHERE login=’doe’ AND pass=’ ’; SHUTDOWN;
-- AND pin=
```

At this point, this attack works like a piggy-back attack. The first query is executed normally, and then the second, malicious query is executed, which results in a database shut down. This example shows that stored procedures can be vulnerable to the same range of attacks as traditional application code.

Inference

Attack Intent: Identifying injectable parameters, extracting data, determining database schema.

Description: In this attack, the query is modified to recast it in the form of an action that is executed based on the answer to a true/false question about data values in the database. In this type of injection, attackers are generally trying to attack a site that has been secured enough so that, when an injection has succeeded, there is no usable feedback via database error messages. Since database error messages are unavailable to provide the attacker with feedback, attackers must use a different method of obtaining a response from the database. In this situation, the attacker injects commands into the site and then observes how the function/response of the website changes. By carefully noting when the site behaves the same and when its behavior changes, the attacker can deduce not only whether certain parameters are vulnerable, but also additional information about the values in the database. There are two well known attack techniques that are based on inference. They allow an attacker to extract data from a database and detect vulnerable parameters. Researchers have reported that with these techniques they have been able to achieve a data extraction rate of 1B/s .

Blind Injection: In this technique, the information must be inferred from the behavior of the page by asking the server true/false questions. If the injected statement evaluates to true, the site continues to function normally. If the statement evaluates to false, although there is no descriptive error message, the page differs significantly from the normally-functioning page.

Timing Attacks: A timing attack allows an attacker to gain information from a database by observing timing delays in the response of the database. This attack is very similar to blind injection, but uses a different method of inference. To perform a timing attack, attackers structure their injected query in the form of an if/then statement, whose branch predicate corresponds to an unknown about the contents of the database. Along one of the branches, the attacker uses a SQL construct that takes a known amount of time to execute, (e.g. the WAITFOR keyword, which causes the database to delay its response by a specified time). By measuring the increase or decrease in response time of the database, the attacker can infer which branch was taken in his injection and therefore the answer to the injected question.

Example: Using the code from our running example, we illustrate two ways in which Inference based attacks can be used. The first of these is identifying injectable parameters using blind injection.

Consider two possible injections into the *login* field. The first being “legalUser’ and 1=0 - -” and the second, “legalUser’ and 1=1 - -”.

These injections result in the following two queries:

SELECT accounts FROM users WHERE login='legalUser' and 1=0 – ' AND pass=''AND pin=0

SELECT accounts FROM users WHERE login='legalUser' and 1=1 – ' AND pass='' AND pin=0

Now, let us consider two scenarios. In the first scenario, we have a secure application, and the input for *login* is validated correctly. In this case, both injections would return login error messages, and the attacker would know that the *login* parameter is not vulnerable. In the second scenario, we have an insecure application and the *login* parameter is vulnerable to injection. The attacker submits the first injection and, because it always evaluates to false, the application returns a login error message. At this point however, the attacker does not know if this is because the application validated the input correctly and blocked the attack attempt or because the attack itself caused the login error. The attacker then submits the second query, which always evaluates to true. If in this case there is no login error message, then the attacker knows that the attack went through and that the *login* parameter is vulnerable to injection.

The second way inference based attacks can be used is to perform data extraction. Here we illustrate how to use a Timing based inference attack to extract a table name from the database. In this attack, the following is injected into the *login* parameter: ‘‘legalUser’ and ASCII(SUBSTRING((select top 1 name from sysobjects),1,1)) > *X* WAITFOR 5 --’’. This produces the following query:

SELECT accounts FROM users WHERE login='legalUser' and ASCII(SUBSTRING((select top 1 name from sysobjects),1,1)) > *X* WAITFOR 5 – 'AND pass='' AND pin=0

In this attack the `SUBSTRING` function is used to extract the first character of the first table’s name. Using a binary search strategy, the attacker can then ask a series of questions about this character. In this case, the attacker is asking if the ASCII value of the character is greater-than or less-than-or-equal-to the value of *X*. If the value is greater, the attacker knows this by observing an additional 5 second delay in the response of the database. The attacker can then use a binary search by varying the value of *X* to identify the value of the first character.

Alternate Encodings

Attack Intent: Evading detection.

Description: In this attack, the injected text is modified so as to avoid detection by defensive coding practices and also many automated prevention techniques. This attack type is used in conjunction with other attacks. In other words, alternate encodings do not provide any unique way to attack an application; they are simply an enabling technique that allows attackers to evade detection and prevention techniques and exploit vulnerabilities that might not otherwise be exploitable. These evasion techniques are often necessary because a common defensive coding practice is to scan for certain known "bad characters," such as single quotes and comment operators.

To evade this defense, attackers have employed alternate methods of encoding their attack strings (e.g., using hexadecimal, ASCII, and Unicode character encoding). Common scanning and detection techniques do not try to evaluate all specially encoded strings, thus allowing these attacks to go undetected. Contributing to the problem is that different layers in an application have different ways of handling alternate encodings. The application may scan for certain types of escape characters that represent alternate encodings in its language domain. Another layer (e.g., the database) may use different escape characters or even completely different ways of encoding. For example, a database could use the expression char (120) to represent an alternately-encoded character "x", but char(120) has no special meaning in the application language's context. An effective code-based defense against alternate encodings is difficult to implement in practice because it requires developers to consider of all of the possible encodings that could affect a given query string as it passes through the different application layers. Therefore, attackers have been very successful in using alternate encodings to conceal their attack strings.

Example: Because every type of attack could be represented using an alternate encoding, here we simply provide an example of how esoteric an alternatively-encoded attack could appear. In this attack, the following text is injected into the *login* field: "legalUser'; exec(0x73687574646f776e) - - ". The resulting query generated by the application is:

SELECT accounts FROM users WHERE login='legalUser'; exec(char(0x73687574646f776e)) -- AND pass='' AND pin=

This example makes use of the char() function and of ASCII hexadecimal encoding. The char() function takes as a parameter an integer or hexadecimal encoding of a character and returns an instance of that character. The stream of numbers in the second part of the injection is the ASCII hexadecimal encoding of the string

“SHUTDOWN.” Therefore, when the query is interpreted by the database, it would result in the execution, by the database, of the SHUTDOWN command.

5.6 Proposed Methodology

In this method we use "variable normalization" to extract the basic structure of a SQL statement so that we could use that information to determine if a SQL statement is allowed to be executed. The methods can be used in most scenarios and does not require changing the source code of database applications (ie CGI web application). The presented method can be used for the allowable list of SQL statements, which makes the system very easy to setup. And since the decision of whether a SQL statement is allowed is to check if the normalized statement exists in our ready allowable list, the overhead of the system is very minimal.

Variable Normalization

The variable normalization is tried to strip away the variables and get the basic structure of the SQL statement, so that although the supplied variables differ every time, the basic structure remains the same. If SQL injection happens, the injection code will change the structure of the SQL statement, and we should be able to detect it.

The variable normalization is method of determining allowability of a SQL statement, including normalization the SQL statement and comparing the normalized SQL statement with a predetermined set of allowable statements. In the normalization process each single-quoted string within the SQL statement to a single character, converting all numbers within the SQL statement to a single character, storing the converted SQL statement, storing a position of each variable of the converted SQL statement, and storing a value of each variable of the converted SQL statement.

The predetermined sets of allowable statements contain a set of normalized SQL statements along with corresponding variable positions, variable types and variable requirements. The set of allowable statements include variable length, allowable characters, regular expression patterns, minimum values and maximum values. The comparing includes searching for the SQL statement in the set of allowable statements. When the allowable list contains the SQL statement, verification of each variable value in the SQL statement may be determined by checking it against the variable requirements located in the set of allowable statements. The SQL statement is allowed when each variable value in the SQL statement is verified.

Example 1

SELECT * FROM customer WHERE customer_name = 'ram'

Normalized SQL	SELECT * FROM customer WHERE customer_name = 'a'				
In statements table	Id = 1	Statement = select * from where customer_name = 'ram'			
In parameters table	Id=1	Position=51	String type	StatementId=1	Original value = "ram"

Example 2

SELECT * FROM customer WHERE customer_id = 101 and customer_city = 'kath'

Normalized SQL	SELECT * FROM customer WHERE customer_id= 'a' and customer_city = 'a'				
In statements table	Id =2	SELECT * FROM customer WHERE customer_id = 101 and customer_city = 'kath'			
In parameter table	Id=2 Id=3	Pos=47 Pos = 73	Integer type String type	StatementId=2 StatementId=2	Original value = 101 Original value = "kath"

Example 3

SELECT * FROM loan WHERE amount > 1000

Normalized SQL	SELECT * FROM loan WHERE amount > 'a'				
In statements table	Id = 3	Statement = SELECT * FROM loan WHERE amont > 1000			
In parameters table	Id=4	Position=38	Integer type	StatementId=3	Allowable range 0-1000000

SELECT loan_number FROM account WHERE branch_name = 'kath' and amount > 10000

Normalized SQL	SELECT loan_number FROM account WHERE branch_name = 'a' and amount > 'a'				
In statements table	Id = 4	SELECT loan_number FROM account WHERE branch_name = 'kath' and amount > 10000			
In parameters table	Id = 5	Position=62 Position= 86	String type Integer type	StatementId=4 StatementId=4	Allowable character set[a-z A-Z] max length8 Allowable list 0-1000000

In performing the normalization procedure, as in example 1, example 2, variables of the received SQL statement may be modified. For example, as in example 1, in an embodiment all single-quoted strings in the received SQL statement may be converted to a single character, letter "a". Similarly, all integers or floating point numbers contained in the received SQL statement may be converted to a single character, letter "a" as in example 2.

Upon conversion of the variables as described above, the normalized SQL statement stored in the data structure call a rule. The normalized SQL statement stored in the rule along with corresponding variable information, including variable type and variable position after normalization as shown in example1 and example 2. The non-variable elements of the received SQL statement including SQL comments, carriage returns, white spaces, and character cases are not modified.

Verification of the normalized SQL statement is performed through comparison of the normalized SQL statement with a pre-defined allowable list. The allowable list include set of rules and stored variable requirement as shown in example 3. The allowable list may be defining each normalized SQL statement along with requirement of the variables of the normalized SQL statement.

The normalized SQL statement verified by searching the allowable list to determine if the normalized SQL statement exists in the allowable list as shown in example 4. When the normalized SQL statement is found to exist in the allowable list, it allowed when the variables of the normalized SQL statement are within the expected values. When the variables of the normalized SQL statement are not within the expected values the SQL statement will be blocked.

Example 4

SELECT * FROM customer WHERE customer_name = 'ram'

$\Downarrow$ Normalize

SELECT * FROM customer WHERE customer_name = 'a'

Variable 1 is string type, value = ram

$\Downarrow$ String Search

ALLOWABLE LIST
SELECT * FROM customer WHERE customer_name = …….
SELECT * FROM loanWHERE customer_name = …….
SELECT * FROM customer WHERE customer_id = …….
SELECT * FROM branch__nameWHERE account = …….
SELECT * FROM customer_name WHERE borrower = …….

Input SQL statement	Rule found in allowable list

SELECT * FROM customer
WHERE customer_name = 'ram'
Variable1, Integer, value ram
	SELECT * FROM customer WHERE
	customer_name = 'a'
	Variable1, Integer, character set[a-z A-Z] max length8

Chapter 6

Analysis and Testing

During testing the model with different types of valid and invalid queries, the following results came to be found.[1]

Testing valid query

Number of SQL query	SQL statements
1	Select account_number from account
2	Select distinct branch_name from account
3	Select all branch_name from account
4	Select account_number from account where branch_name = ?
5	Select account_number from account where branch_name = ? and balance > ?
6	Select branch_name from account as a, depositor as d where a.account_number = d.account_number
7	Select account_number from account as a, loan as l where a.branch_name = ?
8	Select loan_number from account as a, loan as l where a.branch_name = l.branch_name and amount > ?
9	Select branch_name from loan as l, borrower as b where b.customer_name = l.customer_name
10	Select customer_name ,borrower.loan_number,amount from borrower, loan where borrower.loan_number = loan.loan_number
11	Select loan_number, branch_name,amount *100 from loan
12	select * from users where username = (SELECT username FROM users WHERE username = 'gita')"
13	Select * from laon order by amount desc, loan_number asc
14	Select customer_name from depositor
15	(Select customer_name from depositor) UNION (Select customer_name from borrower)
16	Select distinct customer_name from borrower .loan_number =loan.loan_number and branch_name = ? order by customer_name
17	Select customer_name from borrower

18	Select avg(balance) from account where branch_name = ?
19	Select branch_name, avg(balance) from account group by branch_name
20	Select avg(balance) from account
21	Select count(*) from customer
22	Select loan_number from loan where amount is null
23	Select distinct t.branch_name from branch as t, branch as s where t.assets > s.assets and s.branch_city = ?

Table 6.1 : List of valid query detected by model

List of invalid query which the model detected as valid = 0

Table 6.2 : List of invalid query which the model detected as valid

Testing invalid query

Number of SQL	Injected SQL Statement
1	Select loan_number from loan where branch_name = '' or 1 =1 and amount = 1400
2	Select loan_number from loan where branch_name = 'admin' --' and amount = 3456
3	Select loan_number from loan where branch_name = 'admin' /*' and amount > 67888
4	Select loan_number from loan where branch_name = 'brookyln' and amount > 6788 ; select * from loan
5	Select loan_number from loan where branch_name = 'brookyln' and amount > 6785 ; drop table loan
6	insert into emp values('app','kath',2);delete from loan--
7	Select loan_number from loan where branch_name = 'brookyln' and amuont > 6778 ; select laon_file (0X633A5C626F6F742E696E69)
8	Select laon_number from loan where branch_name = ' or 'easy' = 'easy' and amount >4567

Table 6.3 : List of invalid query detected by model

Number of SQL	SQL Statement
1	Select * from emp where sal in (Select max(sal) from emp group by deptno)
2	Select * from emp where sal > any(select max(sal) from emp group by deptno)
3	Select empno, ename, emp.deptno from emp, dept where emp.deptno(+) = dept.deptno

4	Select empno, ename, sal from emp where sal> all(select sal from emp where job = ‘manager’)
5	Select empno, ename, emp.deptno from emp, dept where emp.deptno = dept.deptno(+)
6	Select loan_number from loan where branch_name = 'freek's'

Table 6.4: List of valid query which the model detected as invalid

Precision, Recall and F-Measure

In a collection of SQL statements S, the tested SQL are grouped into four groups.[26]

True Positive is the number of valid queries that are detected by model as a valid query.

False positive is the number of queries that are recognized as valid even though they are invalid.

True Negative is the number of invalid queries that are detected as invalid.

False Negative is the number of queries that are recognized as invalid even though they are valid.[26]

True positive (tp) = 23

False positive (fp) = 0

True negative (tn) = 8

False negative (fn) = 6

Precision = $\frac{tp}{tp + fp} = \frac{23}{23} = 1$

Recall = $\frac{tp}{tp + fn} = \frac{23}{29} = 0.79$

F – measure = $\frac{2 \times recall \times precision}{recall + precision} = \frac{2 \text{ X } 0.79 \text{ X } 1}{0.79 + 1} = \frac{1.58}{1.79} = 0.88$

While talking about complexity, the normalization is in fact, very simple, we just need to replace all quoted string with ‘a’ and all numbers also with ‘a’. So normalization process should be negligible. As for searching the normalized SQL statements, the searching is O(log n) operation as we will sort the allowable list before use using binary search algorithm. However, the performance would be greatly affected if there were many SQL statements need to be authorized by regular expressions.

Conclusion

Most web applications employ a middleware technology designed to request information from a relational database in SQL parlance. SQL injection is a common technique attackers employ to attack these web-based applications. These attacks reshape the SQL queries, thus altering the behavior of the program for the benefit of the hacker. Here presented “variable normalization” for SQL statements, which can extract the basic structure of a SQL statement. If SQL injection happens, the structure of the SQL statement will be altered and hence normalized SQL statement will also be altered and we will be able to detect it.

Limitation and Future Work

There are some types of SQL statements that cannot be handled by variable normalization effectively. For example, consider a web page multi-line selection box allowing user to select the deptno by using a multiline selection box

SELECT * FROM emp WHERE ename LIKE ‘%he%’ AND deptno
in (‘10’, ‘20’, ‘30’)
And the normalized SQL statement will be
SELECT * FROM emp WHERE ename LIKE ‘a’ AND deptno in (‘a’,‘a’, ‘a’)
As there is a limitation of this technique so need to improve our solution which can eliminate all those problems.

References :

[1] A. K. Silberschtz, H.F and S. Sudarshan, “Database system concept”, McGraw-Hill, 4th edition

[2] C. Anley, “Advanced sql injection in sql server application. Technical report, NGSSoftware Insight Security Research (NISR)”, 2002. http://www.nextgenss.com/papers/advanced_sql_injection.pdf

[3] CNET.com “2009 Webware 100 winners”, 2009. http://www.webware.com/100/

[4] D. Gollmann, “Computer Security”, John Wiley & Sons, 2001.

[5] D. Litchfield, “Web application disassembly with odbc error messages. In windows security 2001”, Las Vegas, USA, jul 2001. Black Hat. http://www.blackhat.com/presentations/win-usa=01/Litchfield/bh-win-01-litchfield.doc

[6] Distributed Technologies GmbH. 3- and n-tier architectures. Online Documentation, 1998. http://corba.ch/e/3tier.html

[7] G.Buehrer, B.W. Weide and P.A.G. Sivilotti, “Using Parse Tree Validation to Prevent SQL Injection Attacks”, 5th international Workshop on Software Engineering and Middleware, pages 106-113, 2005

[8] G.Wassermann and Z. Su., “An Analysis Framework for security in Web Applications. In Proceedings of the FSE Workshop on Specification and Verification of Component-Based Systems (SAVCBS)”, pages 70-78, 2004.

[9] H. Khatri, “Sql server stored procedures 101. Web advisory”, jun 2002. http://www.devarticles.com/printpage.php?articleId=142.

[10] IBM, “IBM Internet Security SystemsX-Force 2008 Trend and Risk report Jan 2009”,http://www-935.ibm.com/services/us/iss/xforce/trendreports/xforce-2008-annual-report.pdf

[11] I. Sommerville, “Software Engineering”, Addison – Welsey, 2001

[12] J. Wooder, General web architecture. Web advisory, jan 2002. http://www.wooder.ca/archweb.html

[13] M. Spenik and O. Sledge, “Microsoft sql server 2000 error messages”, Web advisory, 2002. http://developer.com/db/article.php/10920_724711_1

[14] National Information System Security(INFOSEC), www.dtic.mil/cgi-bin/GetTRDOC?Location=U2&doc=GetTRDOC.pdf

[15] R. Barnett, “The Web Hacking Incidents Database (WHID): Bi-Annual Report 2009 (January – June)” https://www.owasp.org/images/e/e5/ The_Web_Hacking_Incidents_Database_-_2009_Bi_Annual_Report.pdf

[16] R. Chartier. “Application architecture : An n-tier approach – part1. Online Documentation”, 2001. http://www.15seconds.com/issue/011023

[17] R. Farrow, “Databases under fire”, Web advisory, may 2002. http://www.dirscanner.com/pubs/sql.pdf

[18] Scott, D.Sharp, R(2002): “Abstracting Application- level Security”, in processing of the 11th International Conference on the World Wide Web(WWW) pages 396-407

[19] T. Connolly, C. Begg, and A. Strachan, “Database Systems – A pratical Approach to Design, Implementation, and Management”, Addison – Wesley.

[20] The Open Web Application Security Project, “A guide to building secure web applications, Version 1.1.1 online Documentation”, sep 2002. http://www.oawsp

[21] U. B. Landsmann and D. Stromberg “Web Application Security: A Survey of Prevention Techniques Against SQL Injection", www.auto.tuwien.ac.at/~chris/teaching/papers/sqlinject.pdf, 2003

[22] W. G. J. Halfond, J. Viegas, and A. Orso “A Classification of SQL Injection Attacks and Countermeasures”, 2006

[23] Wikipedia, “Information Security” http://en.wikipedia.org/wiki/Information_security

[24] Wikipedia, “AAA protocol”, http://en.wikipedia.org/wiki//AAA_protocol

[25] Wikipedia, “Web Application”, http://en.wikipedia.org/wiki/Web_appication

[26] Wikipedia, “ Precision and recall”, http://en.wikipedia.org/wiki/Precision_and_recall

[27] Z.Su and G.Wassermann., “The Essence of Command Injection Attacks in Web Applications. Annual Symposium on Principles of Programming Languages (POPL)”, pages 372-382, 2006.

Bibliography:

[1] C. Anley ,”Advanced SQL Injection In SQL Server Applications”, 2002

[2] Huang, Y-W., Hang, C., Tsal, C-H., Lee, D and Yu F (May 2004): “Securing Web Application Code by Static Analysis and runtime Protection”, in processing of the 12th international World Wide Web Conference (WWW), Pages 40-52

[3] M. Dhakal, “Prevention of Web Application Against SQL_Injection Attack”, 2008

[4] Network Working Group, “ Hypertext transfer protocol – http/1.0, request for comments: 1945”, Online Documentation, may 1996. http://www.w3.org/Protocols/rfc1945/rfc1945.

[5] S. Friedl, “SQL Injection Attacks by Example”, 2005

[6] Secerno.com, “SQL Injection Attack: A Security Threat”, http://www.secerno.com/?pg=SQl-Injection#2