International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-10

AN EMPIRICAL REVIEW OF SQL INJECTION ATTACK IN WEB APPLICATIONS

Shikha Garg
Computer Science & Engineering Department
Haryana Engineering College
Jagadhri, Yamunanagar

Pooja Narula
Computer Science & Engineering Deparment
Haryana Engineering College
Jagadhri, Yamunanagar

Abstract—SQL injection is an attack methodology that targets the data residing in a database through the firewall that shields it. The attack takes advantage of poor input validation in code and website administration. SQL Injection Attacks occur when an attacker is able to insert a series of SQL statements in to a 'query' by manipulating user input data in to a web-based application, attacker can take advantages of web application programming security flaws and pass unexpected malicious SQL statements through a web application for execution by the backend Database. The aim of this research is to study about SQL injection attacks process.

Keywords—SQL Injection,Forms of SQL Injection

(I)Introduction

SQL infusion is a code infusion system, used to strike information driven requisitions, in which noxious SQL explanations are embedded into a passage field for execution (e.g. to dump the database substance to the agressor). SQL infusion must abuse a security helplessness in a provision's product, for instance, when client data is either mistakenly sifted for string exacting break characters installed in SQL articulations or client information is not determinedly written and startlingly executed. SQL infusion is generally known as an ambush vector for sites however might be utilized to assault any kind of SQL database.

A standard SQL query is composed of one or more SQL commands, such as SELECT, UPDATE, or INSERT. SQL injection is one of the most common application layer attacks. According to NIST SQL injection amounted to 14% of the total web application vulnerabilities in 2006 [2]. SQL injection is the act of passing a SQL query or command as input into a web application. It exploits web applications that use client-side data in a SQL query without proper input validation. SQL injection attacks usually target data residing in a database.

A. SQL Injection Attacks

SQL injection attack occurs on database-driven websites when unauthorized SQL queries are executed on vulnerable sites. This attack can bypass a firewall and can affect a fully patched system. For this to happen port 80, the default web port, is the only thing required. SQL injection attacks target a specific

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-10

web application where the vulnerability of the relational database is either know or discovered by the attacker. Figure 1 shows a SQL attack methodology [4].

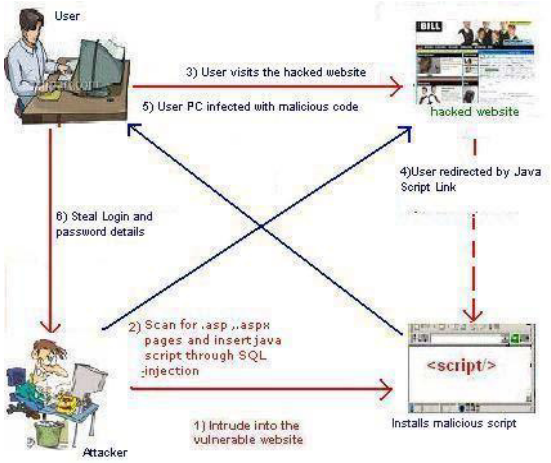

Figure 1: Attack Methodology

In this example students will be separated into individual groups and they will proceed to assigning roles. Some students will portray the role as the web application developer and others will portray the role as the attacker. The developers will create an application that includes a relational database. The attackers will try to hack the application. This case study uses the examples developed by Mitchell Horper to let students get hands-on experience [3].

1)Create a HTML form

In this section how to create a simple login form named frmLogin will be illustrated.

```
<formname="frmLogin"action="login.asp"
method="post">
Username:<inputtype="text"name="userName">
Password:<inputtype="text"name="password">
<input="submit">
</form>
```

When this form is submitted, the username and password are passed to the login.asp script. They are available to the script through the Request.Form collection. A user will be authenticated by providing correct user name and password. The log in process is done by building a SQL query and comparing the user name and password to the login records in the database. Let us write the login.asp script:

```
<%dimuserName,password,querydimconn,rS
userName=Request.Form("userName")
password=Request.Form("password")
setconn=server.createObject("ADODB.Connection")
/*connecttothe database
setrs=server.createObject("ADODB.Recordset")
query = "select count(*) from userswhere userName="&
userName& " " and userPass='" & password & " ' "
/*querycommnad

conn.Open "Provider=SQLOLEDB;
 DataSource=(local);
InitialCatalog=myDB;
UserId=sa;
Password="rs.active"
Connection=connrs.openquery
ifnot rs.eof then
/*checkloginformation
response.write"LoggedIn"
elseresponse.write"BadCredentials"
endif
%>
```

If the user name and password match a record in the database, "Logged In" will be displayed. Otherwise, "Bad Credentials" will be displayed.

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-10

How a SQL Injection Works

In general Web applications use data read from a client to construct SQL queries. This can lead to vulnerability where an attacker can execute SQL queries to cause SQL injection attacks. Several SQL injection attacks such as manipulating the contents of a query command, forcing login and modify information in a database will be discussed in this section.

(B)Create a database

Let us create a database *myDB* that includes user name and password information in a *users* table with some dummy records:

Create databasemyDBusemyDB Go Create Tableuser(user Idintidentity(1,1)notnullusername varchar(50)Not null,userPassvarchar(20)notnull)

insert into users(userName, userPass) values('john', 'doe')

insert into users(userName, userPass) values('admin', 'wwz04ff')

insert into users(userName, userPass) values('fsmith', 'mypassword')

If a user tries to login and provide the username of john and password of doe, the message "Logged In" will be displayed. The query would look like:

select count(*) from users where userName='john' and userPass='doe'

SQL Injection: Manipulate the Contents of a Query

A hacker can manipulate the contents of a query to create a SQL injection attack. For example

Change the userPass into **' ' or 1=1 --' to create a select command** like this:

select count(*) from users where userName='john' and userPass=' '

or 1=1 --'

Therefore the query only checks for the username of john. Instead of checking for a matching password, it checks for an empty password, or the conditional equation of 1=1. In this case if the password field is empty or 1 equals 1(which is always true), a valid row will be found in the users table with username john. The single line delimeter (--) that comments out the last quote stops ASP returning an error about any unclosed quotations. As the result one row will be returned and the message "Logged In" will be displayed.

This method can be used for the username field. If changing the username is **' or 1=1 --- and password is empty such as:**

Username: ' or 1=1 --- Password: [Empty]

And execute a select query:

select count(*) from users where userName=' ' or 1=1 -and userPass=' '

A count of all rows in the users table will be return. This is an example of SQL injection attack that is implemented by adding code that manipulates the contents of a query to get an undesired result.

(C)SQL Injection: Force Login

The following example demonstrates how force login SQL injection works. Consider the following query that is based on the users table.

select userName from users where userName=' ' having 1=1

A page call login.asp can easily be developed to query the database by using these login credentials:

Username: ' having 1=1 --- Password: [Anything]

When a user clicks on the submit button to start the login process, the SQL query causes ASP to send the following error message to the browser:

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-10

Microsoft OLE DB Provider for SQL Server (0x80040E14)
Column **'users.userName'** is invalid in the select list because it is not contained in an aggregate function and there is no **GROUP BY** clause.

This error message tells the unauthorized user the name of one field from the database: **users.userName**. Using the name of this field, a user can use SQL Server's **LIKE** keyword to login with the following credentials:

Username: ' or users.userName like 'a%' --- Password: [Anything]

Once again, this performs an injected SQL query against the users table:

select userName from users where userName=' ' or users.userName like 'a%' --' and userPass=' '

When the users table was created, a user whose userName field was admin and userPass field was wwz04ff was also created. Logging in with the username and password shown above uses SQL's like keyword to get the username. The query grabs the userName field of the first row whose userName field starts with a, which in this case is admin:

Logged In As admin SQL Injection: Modify the Content of a Database

Let us create a *products* table and rows on the SQL server as following:

Createtableproducts(Idintidentity(1,1)not null, prodName varchar(50)notnull) insert into products(prodName) values('PinkHoolaHoop')
insert into products(prodName)values('GreenSoccer Ball')
insert into products(prodName) values('Orange Rocking Chair')
response.write "Gotproduct"&rs.fields("prodName") .value

Although this may seem more secure it is not. By manipulating the database a SQL injection can occur because the WHERE clause of the query is based on a numerical value:

query = "select prodName from products where id = " & prodId

The products.asp page requires a numerical product Id passed as the productId querystring variable.

Consider the following URL to products.asp:

http://localhost/products.asp?productId=0%20or%201=1

Each %20 in the URL represents a URL-encoded space character, so the URL looks like:

http://localhost/products.asp?productId=0 or 1=1

When used in conjunction with products.asp, the query looks like:

select prodName from products where id = 0 or 1=1

From the above select command we know how to use some URL-encoding, the names of the products can be pulled from the product table with the following url:

http://localhost/products.asp?productId=0%20having 1=1

This would generate the following error in the browser:

Microsoft OLE DB Provider for SQL Server (0x80040E14)
Column **'products.prodName'** is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.
/products.asp, line 13

Take the name of the products field (**products.prodName**) and call up the following URL in the browser:

http://localhost/products.asp?productId=0;insert%20into %20products (prodName)%20values(left(@@version,50))

Here is the query without the URL-encoded spaces:

http://localhost/products.asp?productId=0;insert into products(prodName) values(left(@@version,50))

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-10

It returns "No product found". However it also runs an **INSERT** query on the products table, adding the first 50 characters of SQL server's @@version variable (which contains the details of SQL Server's version, build, etc.) as a new record in the products table.

To get to the SQL server's version, a user must call up the products.asp page with the value of the latest entry in the products table such as:

http://localhost/products.asp?productId=(select%20max()%20from%20products)

(II) LITERATURE REVIEW

Fu et al., in [12] propose a Static Analysis Framework in order to detect SQL Injection Vulnerabilities. SAFELI framework aims at identifying the SQL Injection attacks during the compile-time. This static analysis tool has two main advantages. Firstly, it does a White-box Static Analysis and secondly, it uses a Hybrid-Constraint Solver. For the Whitebox Static Analysis, the proposed approach considers the byte-code and deals mainly with strings. For the Hybrid-Constraint Solver, the method implements an efficient string analysis tool which is able to deal with Boolean, integer and string variables.

Thomas et al., in [13] suggest an automated prepared statement generation algorithm to remove SQL Injection Vulnerabilities (SQLIVs). They implement their research work using four open source projects namely: (i) Net-trust, (ii) ITrust, (iii) WebGoat, and (iv) Roller. Based on the experimental results, their prepared statement code was able to successfully replace 94% of the SQLIVs in four open source projects. However, the experiment was conducted using only Java with a limited number of projects. Hence, the wide application of the same approach and tool for different settings still remains an open research issue to investigate.

In [14], Haixia and Zhihong propose a secure database testing design for Web applications. They suggest a few things; firstly, detection of potential input points of SQL Injection; secondly, generation of test cases automatically, then finally finding the database vulnerability by running the test cases to make a simulation attack to an application. The proposed methodology is shown to be efficient as it was able to detect the input points of SQL Injection exactly and on time as the authors expected. However, after analyzing the scheme, we find that the approach is not a complete solution but rather it needs additional improvements in two main aspects: the detection capability and the development of the attack rule library

In [15] Ruse et al. propose a technique that uses automatic test case generation to detect SQL Injection Vulnerabilities. The main idea behind this framework is based on creating a specific model that deals with SQL queries automatically. In addition, the approach identifies the relationship (dependency) between sub-queries. Based on the results, the methodology is shown to be able to specifically identify the causal set and obtain 85% and 69% reduction respectively while experimenting on few sample examples. Moreover, it does not produce any false positive or false negative and it is able to detect the real cause of the injection.

In [16], Roichman and Gudes, in order to secure Web application databases, suggest using a fine-grained access control to Web databases. They develop a new method based on fine-grained access control mechanism. The access to the database is supervised and monitored by the built-in database access control. This approach is efficient in the fact that the

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-10

security and access control of the database is transferred from the application layer to the database layer.

In [17], Shin et al. suggest SQLUnitGen, a Static-analysisbased tool that automate testing for identifying input manipulation vulnerabilities. They apply SQLUnitGen tool which is compared with FindBugs, a static analysis tool. The proposed mechanism is shown to be efficient (483 attack test cases) as regard to the fact that false positive was completely absent in the experiments. However for different scenarios, false negatives at a small number were noticed. In addition to that, it was found that due to some shortcomings, a more significant rate of false negatives may occur "for other applications". Hence, the authors talk about concentrating on getting rid of those significant false negatives and further improvement of the approach to cover input manipulation vulnerabilities as their future works.

In [4] Gary McGraw explains the types of security as Software Security with the term Application Security. The main difference between them is that Application Security is the process of protecting the software after it has been completed and deployed by finding and fixing the security problems after they have occurred, while Software security is the process of building a secure software by designing, planning, coding and implementing taking in consideration common security threats [4].

It is important to understand that there is no way to guarantee that software is 100% secured. The main idea behind Software Security is to integrate the more level of security possible in software in order to diminish the possibilities of an attack [7].

A lot of software developments do not provide proper security because they were created with wrong suppositions in mind [7] such as that all users are friendly and will not be perform an attack, that requiring a password to login will prevent unwanted users to try to hack the application, and that a firewall is enough to protect a software from threats. There have been identified several approaches often used in software development which do not provide a valid solution to security issues in the final version of the product [5].

As it can be inferred, there is not a simple solution or task that can guarantee the security of software once it is installed and in use. However, by applying a set of recommendations and best practices it is possible to achieve acceptable levels of security quality. Furthermore, security should be integrated within the companies' strategies. In this case, the role of managers is to establish policies, measurable goals, support research and training [7] that will situate security quality in a major position in software position. An important strategy to apply in order to attain secure software is to include security tasks within the software life cycle, which is described in the following section.

Approach	Description
Bolt-on	This approach let the analysis of security until the product is coded. Issues found in the security test must be patched in the source code which produce higher costs
Do-it-all-up-front	This approach try to identify all possible threats before starting coding the application which lead to ignore new potential threats that appear through the evolution of the software
Big-bang	This approach focus on dedicating efforts to

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 4 Issue 2 March 2014
International Manuscript ID : 2249054XV4I2032014-10

	secure the application in a single time. It fails in giving the software a continuous tracking on security risks
Buckshot	This approach tries a bunch of security techniques with the hope that this will cover the basic security risks
All-or-nothing	This approach do nothing until a security problem occurs, and then overreact by saturating the software of security precautions which ends being counterproductive

Table 1. Approaches that not provide secure software [7].

In [6] J.D. Meier, explains at constitute the threat modeling process, a brief description of these components are [6]:

- Identify assets: Determine what the most valuable parts of the application that should be protected are.
- Create an architecture overview: Define and document the structure of the application including all its components, their properties and the relations among them.
- Decompose the application: Break the structure of the application into small components in order to understand their comportment to discover possible points of attacks, and in that way, being able to create a security profile for the application
- Identify the threats: Define all plausible security threats that an application can be subject of.
- Document the threats: Document a detailed description of all threats identified in the previous step.
- Rate the threats: Give a weight at each identified threat by analyzing the probability that it occurs and the damage that it could incur.

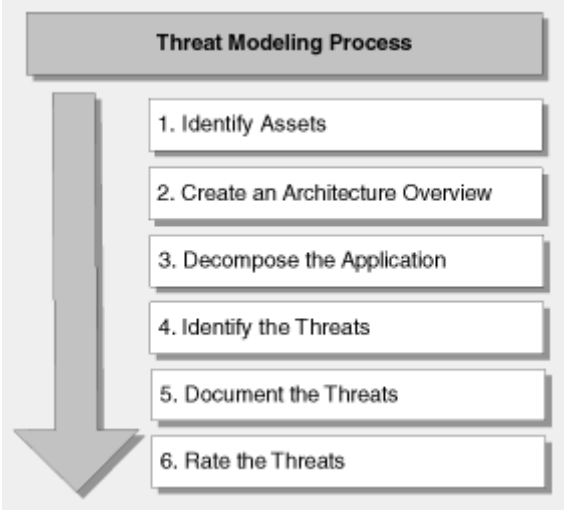

Figure 1. Threat Modeling Process [6].

(III)CONCLUSION

SQL Injection attacks are one of the most dangerous types of threats to web applications. Many solutions to these attacks have been proposed over years. But almost none of them provide security to the full extent of this attack. Also very little emphasis is laid on preventing SQLIA in stored procedures. We have proposed a technique that provides security to both application layer as well as database layer via frontend phase and backend phase. Researchers have provided this two phase security because if security is compromised at one phase, the second phase can still provide security from

attacks. The technique currently works with MYSQL server and used the concept of fuzzy logic also

(IV)REFERENCES

[1] Bhavani M. Thuraisingham, Chris Clifton, Amar Gupta, Elisa Bertino, Elena Ferrari. "Directions for Web and E-Commerce Applications Security." Proceedings of the 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (pp.200-204). 2001.

[2] Bojan Jovicic, Dejan Simic. "Common Web Application Attack Types and Security Using ASP.NET. " ComSIS Consortium. 2006.

[3] Cosmin Striletchi, Mircea-Florin Vaida. "Enhancing the Security of Web Applications." Proceedings of the 25th International Conference on Information Technology Interfaces (pp. 463-468) . 2003.

[4] Gary McGraw. "Software Security." IEEE Security & Privacy (vol. 2, pp. 80-83). 2004.

[5] J.D. Meier. "Web Application Security Engineering." IEEE Security &Privacy (vol. 4, no. 4, pp. 16–24). 2006.

[6] J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan. "Improving Web Application Security: Threats and Countermeasures." Microsoft Corporation. 2003.

[7] Jeff Zadeh, Dennis DeVolder. "Software Development and Related Security Issues." Proceedings of IEEE Southeastcon. 2007.

[8] John R. Maguire, Gilbert Miller. "Web-Application Security: From reactive to proactive." IT Professional (vol. 12, no. 4, pp. 7-9). 2010.

[9] Mark Curphey, Rudolph Araujo. "Web application security assessment tools." IEEE Security & Privacy (vol. 4, no. 4, pp. 32-41). 2006.

[10] Myat Myat Min, Khin Haymar Saw Hla. "Security on Software Life Cycle using Intrusion Detection System." 6th AsiaPacific Symposium on Information and Telecommunication Technologies APSITT 2005 Proceedings. 2005.

[11] Shin-Jer Yang, Jia-Shin Chen. "A study of security and performance issues in designing web-based applications." IEEE International Conference on e-Business Engineering. 2007.

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

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

[14] Haixia, Y. and Zhihong, N., A database security testing scheme of web application. Proc. of 4th International Conference on Computer Science & Education 2009 (ICCSE '09), 25-28 July 2009, pp. 953-955.

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

[16] Roichman, A., Gudes, E., Fine-grained Access Control to Web Databases. Proceedings of 12th SACMAT Symposium, France 2007.