Volume 5, Issue 12, December 2015 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Vulnerability Assessment for Web Application

Archana N*
Assistant Professor,
IT Department, PSG College of Technology,
Coimbatore, Tamilnadu, India

Iswariya M
PG Student,
IT Department, PSG College of Technology,
Coimbatore, Tamilnadu, India

Abstract: ***Web applications are often vulnerable to attacks, which can give attackers easily access to the application's underlying database. SQL injection attack occurs when a malicious user, through specifically crafted input, causes a web application to generate and send a query that functions differently than the programmer intended .SQL Injection Attacks (SQLIAs) have known as one of the most common threats to the security of database-driven applications. So there is not enough assurance for confidentiality and integrity of this information. SQLIA is a class of code injection attacks that take advantage of lack of user input validation. In fact, attackers can shape their illegitimate input as parts of final query string which operate by databases. Financial web applications or secret information systems could be the victims of this vulnerability because attackers by abusing this vulnerability can threat their authority, integrity and confidentiality. So, developers addressed some defensive coding practices to eliminate this vulnerability but they are not sufficient .For preventing the SQLIAs, defensive coding has been offered as a solution but it is very difficult. Not only developers try to put some controls in their source code but also attackers continue to bring some new ways to bypass these controls. Hence it is difficult to keep developers up to date, according the last and the best defensive coding practices. On the other hand, implementing of best practice of defensive coding is very difficult and need to special skills. These problems motivate the need for a solution to the SQL injection problem. To overcome these, a testing mechanism will be implemented to test the code for SQL injection. An alert or report containing these vulnerabilities will be generated for the application owner. This will help the developer to eradicate these vulnerable scripts from the web application thereby protecting the user's privacy and data integrity which will in turn result in saving time and cost. Hence, the safety and security of the web applications will be enhanced.***

Keywords: Vulnerability Assessment, Web Application, Sql Injection, Detection, Security.

I. INTRODUCTION

Web applications are computer programs allowing website visitors to submit and retrieve data to/from a database over the Internet using their preferred web browser. The data is then presented to the user within their browser as information is generated dynamically (in a specific format, e.g. in HTML using CSS) by the web application through a web server. Web applications query the content server (essentially a content repository database) and dynamically generate web documents to serve to the client (people surfing the website). The documents are generated in a standard format to allow support by all browsers (e.g., HTML or XHTML).

Web applications raise a number of security concerns stemming from improper coding. Serious weaknesses or vulnerabilities, allow hackers to gain direct and public access to databases in order to churn sensitive data. Many of these databases contain valuable information (e.g., personal and financial details) making them a frequent target of hackers.Web applications often have direct access to backend data
such as customer databases and, hence, control valuable data and are much more difficult to secure.

Even the most secure network is likely to have some unknown vulnerabilities. Vulnerability scanners are useful tools for identifying hidden network and host vulnerabilities. However, for many organizations, vulnerability assessments are highly technical and are carried out mostly for compliance purposes, with little connection to the organization's business risks and executive security budget decisions. Inorder to detect Sql Injection Vulnerability, this tool has been developed.

II. RELATED WORKS

The various approaches developed for the purpose of identification of specific vulnerabilities existing in the web applications have been discussed in the following.

A. Taint Analysis

Huang et al. (2004) developed a static code analysis tool for PHP called WebSSARI based on a CQual-like type system. It has certain limitations. That means that WebSSARI is able to handle the program flow through user-defined functions, but it does not consider the context from where the function is called. Xie and Aiken (2006) presented a static analysis algorithm for detecting SQL injection vulnerabilities in PHP applications using block and function summaries. It is both intra-procedural and inter-procedural and handles more dynamic features of PHP. Their implementation only supports SQLi vulnerabilities in a context-insensitive way and does not model built-in functions.

Jovanovic et al.(2013) developed Pixy, an open source, static code analyzer for PHP written in Java . The down-side of Pixy is that it only supports Cross-Site Scripting and SQL injection vulnerabilities and has only 29 built-in functions configured leading to False Negatives. False positives occur due to missing or imprecise modelling of built-in functions and mark-up context analysis.An extended version of Pixy called Saner was created by Balzarotti et al. (2012) to improve the detection of user-defined sanitization. It uses manually created, predefined test-cases to check sanitization routines with dynamic analysis. Apollo combines symbolic and concrete execution techniques together with explicit-state model checking. The authors tested their tool with phpBB2 version 2.0.21 and detected several vulnerabilities.

B. String Analysis

Yu et al. (2010) extended Pixy to perform automata-based string analysis for checking the correctness of sanitization routines without dynamic analysis. The drawback of STRANGER and Saner is that they are only as good as their test-cases. Even if all configured attack patterns for one vulnerability type are filtered correctly, other attack patterns could exist that bypass the sanitization undetected. Wasserman and Su (2008) presented an efficient approach to detect SQL injection vulnerabilities. The key idea is to first generate an approximation of the query strings a program may generate using a context free grammar, and then analyze if all potential strings are safe with the help of information flow analysis. In another paper, Wasserman and Su presented a static code analysis approach to detect XSS vulnerabilities caused by weak or absent input validation. In this work, the authors combine work on taint-based information flow analysis with string analysis.

III. SQL INJECTION

SQL injection (SQLi) is a technique where the attacker injects an input in the query in order to change the structure of the query intended by the programmer and gaining the access of the database which results modification or deletion of the user's data. In the injection it exploits a security vulnerability occurring in database layer of an application. SQL injection attack is the most common attack in websites. Some malicious codes get injected to the database by unauthorized users and get the access of the database due to lack of input validation. Input validation is the most critical part of software security that is not properly covered in the design phase of software development life-cycle resulting in many security vulnerabilities. Here the various techniques for detection and prevention of SQL injection attack is being discussed. There are no any known full proof defenses available against such type of attacks.

When an application is communicating with the backend database, it does so in the form of queries with the help of an underlying database driver. This driver is dependent on the application platform being used and the type of backend database, such as MYSQL, MSSQL, DB2, or ORACLE.

A generic login query looks like this:

SELECT Column1, Column2, Column3 FROM table_name WHERE username='$variable1′ AND password='$variable2′;

The query can be split into two parts, code section and the data section. The data section is the $variable1 and $variable2 and quotes are being used around the variable to define the string boundary.Ex: The login form, the username entered is Admin and password is p@ssw0rd which is collected by application and values of $variable1 and $variable2 are placed at their respective locations in the query, making it something like this.

SELECT Column1, column2, Column3 FROM table_name WHERE username='Admin' AND password='p@ssw0rd';

Now the developer assumes that users of his application will always put a username and password combination to get a valid query for evaluation by database backend. If the user is malicious and enters some characters which have some special meaning in the query. For example a single quote. So, instead of putting Admin, he puts Admin', thereby causes an error thrown by the DB driver. Whenever an attacker is able to escape the data boundaries, he can append data which then gets interpreted as code by the DB Driver and is executed on the SQL backend, thereby causing SQL injection.

A. Sql injection Attack Process

SQLIA is a hacking technique which the attacker adds SQL statements through a web application's input fields or hidden parameters to access to resources. Lack of input validation in web applications causes hacker to be successful.

For the following example, a web application receives a HTTP request from a client as input and generates a SQL statement as output for the backend database server.

For example an administrator will be authenticated after typing: employee id=112 and password=admin.

Fig 1 describes a login by a malicious user exploiting SQL

Cause of SQL Injection

Web application vulnerabilities are the main causes of any kind of attack.Some of the cause are Invalidated Input,Generous privileges,Uncontrolled Variable size,Error message,dynamic SQL,Client side only control,sub-selects,Multiple statements etc.

B. Security Measures in Practice

Defensive coding

Developers have approached a range of code based development practices to counter SQLi. These techniques are generally based on proper input filtering, potentially harmful character and rigorous type checking of inputs.

Parameterized queries or stored procedures

The attacker takes advantage of dynamic SQL by replacing the original queries and creates some parameterized query in database. These attacks force to developer for first define the SQL code structure before including parameters in query. Because parameters are bound to the defined SQL structure, thereafter it is not possible to inject additional SQL code.

Escaping

If dynamic queries cannot be avoided, escaping all user-supplied parameters is the best option. Then the developer should identify the all input sources to define the parameter that need escaping, follow database-specific escaping procedures, and use standard defining libraries instead of the custom escaping methods.

Data type validation

After following the steps for the parameterized query and escaping the developer must properly validate the input data type. The developer must define the input data type is string or numeric or any other type and input data given by user is incorrect then it could easily reject.

White list filtering

Some of the special character which is normally used during injection .so the developer should characterize such special character as the black list filtering. The filtering approach is suitable for the well structured data. Such as email address, dates, etc. and developer should keep a list of legitimate data patterns and accept only matching input data.

SQL DOM

The manual defensive coding is the best way to avoid the SQLIA. The approach SQL DOM is introduced by Russell McClure and Ingolf Kruger. In the SQL DOM uses the encapsulation of database queries to provide a safe way to avoid the SQLIA problem by changing the query building process from one that uses string concatenation to a systematic one that uses a type checked API. In the process a set of classes that enables automated data validation and escaping. Developers provide their own database schema and construct SQL statement using its API's. It is especially useful when the developer needs to be using the dynamic SQL in place of the parameterized queries for getting flexibility.

Runtime prevention

Runtime prevention may be more complex than the defensive coding because some of the approaches require code instrumentation to enable runtime checking. But it is able to prevent from all SQLi.

Randomization

The approach is proposed by Boyd and Keromytis in which randomized SQL query language is used, pointing a particular CGI in an application, where a proxy server used in between the SQL server and Web server. It sends SQL query with a randomized value to the proxy server, which is received by the client and de-randomized and sends it to the server. This technique has two main advantages is security and portability. But if the random value is predicted then it is not useful. This approach is based on a runtime monitoring system deployed between the application server and database server, it intercept all queries and check SQL keywords to determine whether the queries syntactic structure are legitimate before the application sends them to the database.

IV. APPLIED METHODOLOGY

Even though there are so many security measures applied for the web application by the developer still the web applications are prone to SQL injection attack. Also all these security measures are dynamic in nature which checks the syntax of the query or the connection. None of the existing methods are used to properly check the flow of user input data into the query. In this case the SQLi is very much possible. In order to reduce the overhead of checking all the lines manually, the taint analysis is performed to check the flow of tainted data.

- In this approach all the SQL statements which are used to retrieve the data from the database by the application is listed.
- Next these statements are compared with the input parameters which are used to get the data from the user or some other field which fetch the data from the user.
- In either case if a match is found then it will be identified as vulnerable to SQLi attack. Few false positives may occur if the same name is used in different places by the developer.

V. VULNERABLE WEB APPLICATION

Fig 1 shows an SQL injection in which the attacker tries to inject arbitrary pieces of malicious data into the input fields of an application, which, when processed by the application, causes that data to be executed as a piece of code by the back end SQL server, thereby giving undesired results which the developer of the application did not anticipate. The backend server can be any SQL server (MySQL, MSSQL, ORACLE, POSTGRESS, etc.)The ability of the attacker to execute code (SQL statements) through vulnerable input parameters empowers him to directly interact with the back end SQL server, thereby leveraging almost a complete compromise of system in most cases.

- Many web applications take user input from a form
- Often this user input is used literally in the construction of a SQL query submitted to a database.
- For example:

SELECT product data FROM table WHERE product name = 'user input product name';

- A SQL injection attack involves placing SQL statements in the user input.

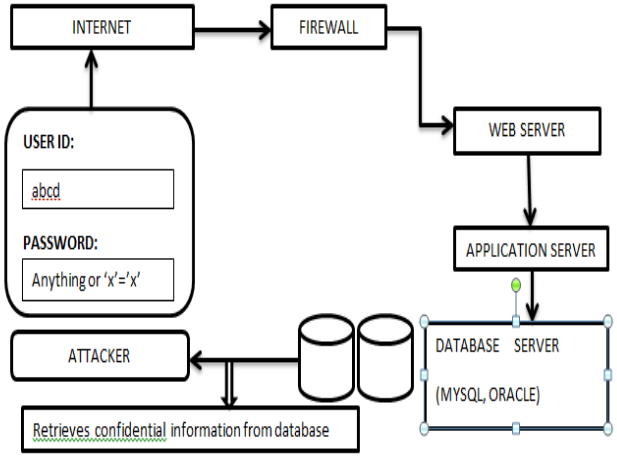

Fig.1. SQL Injection Attack

SELECT * FROM users WHERE username = ' ' or 1=1 – – AND password = 'anything'

Generally when an application is communicating with the backend database, it does so in the form of queries with the help of an underlying database driver. This driver is dependent on the application platform being used and the type of backend database, such as MYSQL, MSSQL, DB2, or ORACLE.

A generic login query would look something like this:

`SELECT Column1, Column2,Column3 FROM table name WHERE username='$variable1′ AND password='$variable2′;`

This query can be split into two parts, code section and the data section. The data section is the $variable1 and $variable2 and quotes are being used around the variable to define the string boundary.

For Example, at the login form, the username entered is Admin and password is p@ssw0rd which is collected by application and values of $variable1 and $variable2 are placed at their respective locations in the query, making it something like this.

`SELECT Column1, column2, Column3 FROM table name WHERE username='Admin' AND password='p@ssw0rd';

Now the developer assumes that users of his application will always put a username and password combination to get a valid query for evaluation by database backend. But if the user is malicious and enters some characters which have some special meaning in the query, then it may lead to an exploit. For example a single quote. So, instead of putting Admin, he puts Admin', thereby causing an error thrown by the DB driver. It is because of the unpaired quote entered by the user breaking the application logic.

Whenever an attacker is able to escape the data boundaries, he can append data which then gets interpreted as code by the DB Driver and is executed on the SQL backend, thereby causing SQL injection.

Fig. 2. Vulnerable website

Since the page is vulnerable to SQL injection, the attacker can retrieve the data from the database using the SQL injection characters listed in table 1.

Table 1. SQL injection characters

Character	Function
' or "	character String Indicators
-- or #	single-line comment
/*…*/	multiple-line comment
+	addition, concatenate (or space in URL)
||	(double pipe) concatenate
%	wildcard attributeindicator

Figure 3 shows the employee details page of the vulnerable website. The user can enter the employee id to view the details of the employee like employee name, hired date, designation, salary etc.

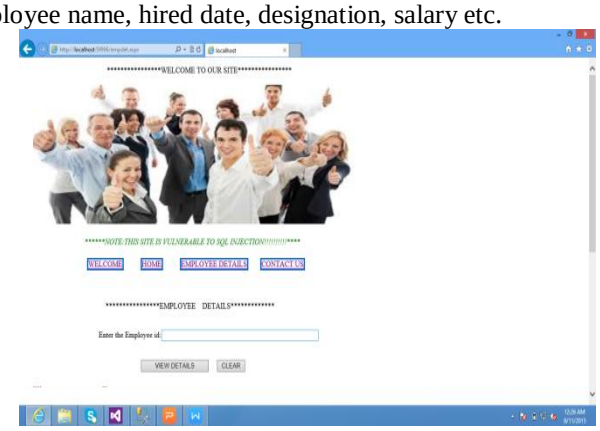

Fig. 3. Employee Details page

Fig. 4. View Details

Figure 4 shows the details of the employee. When the user enters the correct employee id and clicks view details button to view the details, then the details of that particular employee is displayed.

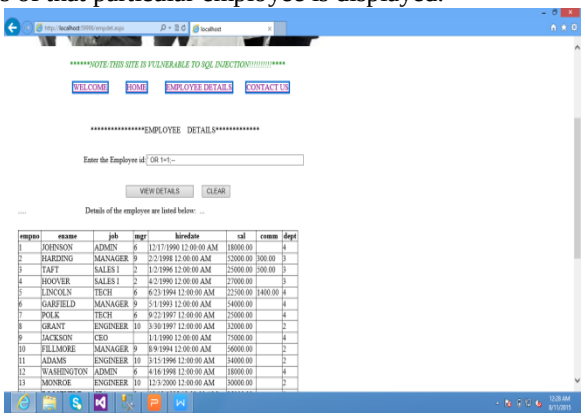

Fig.5. All values Retrieved from database

Figure 5 shows all the values retrieved from the database using SQL injection characters since the page is vulnerable. Now the attacker can use these credentials to perform further attack.

CODE ANALYSIS

Code analysis should be performed after writing the code for any web application in order to detect any vulnerable code if present. The code analysis can be done using regular expression matching where each line of code is checked against the vulnerable code. Some of the vulnerable code would be like this

SELECT * FROM users WHERE login = 'victor' AND password = '123'

(If it returns something then login)

The above code is vulnerable to SQL injection so it should be detected so that it can be modified.

VI. IMPLEMENTATION

A. Uploading the Code for Testing

The code used for designing the website is uploaded in the SQL detector.

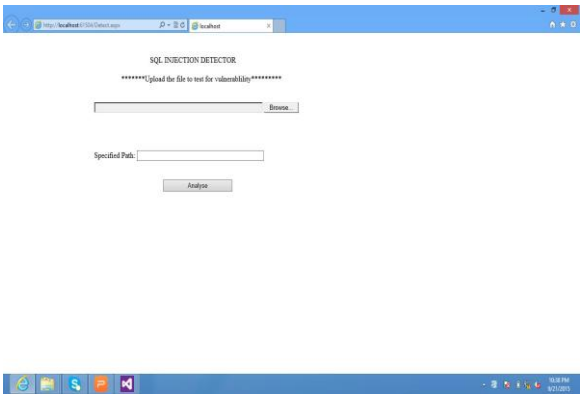

Fig. 6. SQL injection Detector

Figure 6 shows the design of SQL injection detector. The user can upload the code and analyse to detect the vulnerability. The path of the uploaded file will be displayed in the specified path text box. If suppose analyse button is clicked without uploading a file then the user will be intimated to upload a file.
The overall process is as follows,

1. The code should be copied in a text document and stored as text file.
2. Then to upload, the file is browsed from the local system and it is uploaded.
3. Once it is successfully uploaded, the analyse button is clicked to analyse the code uploaded.

Figure 7 shows the path of the file uploaded and after this the code can be analysed to detect vulnerability.

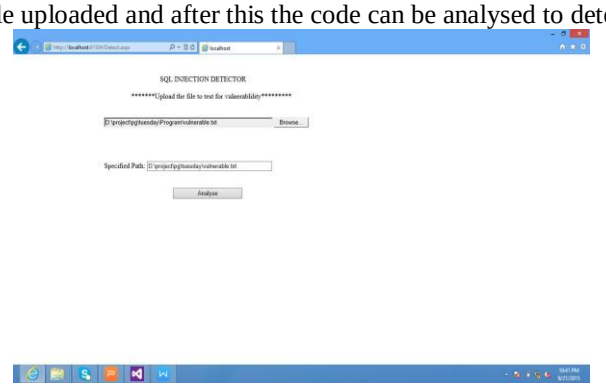

Fig. 7. Uploading the File

B. Detecting the Vulnerable Line of Code

The code is analysed to detect the line which is vulnerable to SQL injection. The overall process is as follows,

- If the code is not vulnerable then the detector will detect it as not vulnerable to SQL injection and the developer can proceed with further steps.
- If the code is vulnerable then an alert showing the code is vulnerable will be displayed and also the line which is vulnerable is also displayed so that it can be rectified. The particular line of code which is vulnerable to SQL injection is also displayed. The particular line can be identified and modified to prevent the application from SQL injection,

VII. CONCLUSION

Web applications are the interfaces between users and web servers that handle critical data such as credit card numbers, email addresses, and passwords. An attacker can abuse security vulnerabilities in these applications to access and steal this sensitive data. To avoid attacks, the source code has to be analysed to identify security vulnerabilities. This static source code analysis application can be used to efficiently detect the most common web application vulnerability namely SQL injection. This resulted in the increased security of web applications and reduced time and effort for reviewing the source code manually. This will in turn result in the enhanced safety of the user's data and security of the web applications.
The future work in this area includes enhancing the scope of this application to include more vulnerability. There are various other vulnerabilities like cross site scripting, File inclusion and file disclosure which can be included in this application so that it can be considered as a single point of testing for almost all the vulnerabilities there by reducing the complexity and resource spent on using various methods to detect various vulnerabilities and error in the source code.

REFERENCES

[1] http://www.webappsec.org/projects/statistics/
[2] http://www.xssed.com/archive
[3] "A Case Study in the Hacking of Web Applications", by Kevin Beaver from Hacking for Dummies, 4th Edition

[4] "Cybercrime in nowadays businesses-a real case study of targeted attack", by Frederic Bourla, Head of ethical Hacking dept, High-Tech Bridge SA.

[5] Sangita Roy, Avinash Kumar Singh and Ashok Singh Sairam, "Detecting and Defeating SQL Injection Attacks ", International Journal of Information and Electronics Engineering, vol. 1,No. 1, July 2011

[6] Jose Fonseca, Marco Vieira, and Henrique Madeira, "Evaluation of Web Security Mechanisms Using Vulnerability & Attack Injection ", IEEE Transactions on Dependable and Secure Computing, vol.11, no.5, september/october2014

[7] D. Hauzar and J. Kofron. On Security Analysis of PHP Web Applica-tions.In IEEE Workshop on Security, Trust, and Privacy for Software Applications (STPSA), 2012.

[8] www.owasp.org

[9] MITRE. Common Vulnerabilities and Exposures (CVE). http://cve:mitre:org/, as of July 2013.

[10] MyBB. Open Source Discussion Board. http://www:mybb:com/, as of July 2013.

[11] myWebland Group. myBloggie Weblog System. http: //mybloggie:mywebland:com/, as of July 2013

