Report on Vulnerabilities in Web Applications

Khairnar Chaitali H 1, V. Kala 2

1M. Tech Scholar, 2Associate Professor, Department of Computer Science & Engineering, Maharashtra Institute of Technology (MIT), Aurangabad, Maharashtra, India

Abstract— Nowadays Internet and web applications are more useful in day to day life. Most of the communications and transactions are done with the help of web applications. When the term web application comes in mind, security is a major concern that we have to deal with it. Vulnerabilities in web applications are free solicitations for the attacks and malicious users to perform harmful activities. Most perceptible attacks are SQL Injection and cross site scripting attacks. Crafted user input at the front end side provided by the attacker, performs undesirable changes at the back end side or in a database. SQL Injection and cross site scripting attacks are prominently used by the attacker to gain the user credentials or unauthorized access for performing attacks or future attacks. Various techniques are proposed to address this issue and each one has its own consequences. This paper is a review of all web application vulnerabilities and their proposed methods for the detection and prevention against SQL injection and cross site scripting attacks.

Keywords— Attacks, Web application, Vulnerabilities, Detection, Prevention.

I. INTRODUCTION

Internet and web applications are the key part of daily activities. It provides an easier way to perform tasks like online transaction, Exchange of information, online supermarket run, paying bills etc. Strong security measures are necessary for the web applications and protection against attacks. Vulnerable web applications are targeted by the attackers and code injection attacks are playing a big role in it. Of many kind of attacks against web applications, SQL Injection Attack (SQLIA) is the top most threat against them [7]. SQL Injection is nothing but a crafted structured query fired by an attacker on the front end side which shows the information by performing undesirable actions at the backend side on a database. Structured Query Language Injection (SQLI) includes some keywords, symbols, and semantics as a part of valid structured query which gives a true output, and the attacker successfully enters into the system and also gains unlimited access. Cross site script is also known as XSS. The injected script can redirect the users browser control to the untrusted server without the users permission. Social networking sites like Facebook, Twitter, MySpace are those suffered from such attacks.

The first OSN worm, Samy that hit MySpace in 2005 by exploiting a cross-site scripting (XSS) vulnerability in a MySpace web application infected about one million victims within 24 hours [8].

From the security perspective, Detection and Prevention of such attacks are obligatory. SQL injection vulnerabilities (SQLIVs) accounted for 20% of input validation vulnerabilities and 10% of total cyber vulnerabilities between 2002 and 2007 [9,10]. In 2006, 21.5% of all newly reported vulnerabilities were XSS, making it the most frequently reported vulnerability of the year [11].

Instruction Set Randomization, CIDT (Code Injection Detection Tool), Reverse proxy, SQL Domain Object Model by using concept of Object oriented programing; Model based hybrid approach and stored procedures and hash values etc. are those techniques which made great contribution in preventing web applications against code injection attacks.

This paper is organized as: Section 2 Definition, Section 3 Classification of vulnerabilities Section 4 Assessment on Detection and Prevention approaches and Section 5 Conclusion.

II. DEFINITION

Code Injection Attacks

Code injection attacks are broadly classified as SQL injection attack and Cross site scripting attack.

A. SQL Injection

SQL injection is a crafted illegal form of query to vulnerable web application which performs undesirable activities in the database and provides information to the attacker.

Example [12]:

query = "SELECT * FROM accounts WHERE login="
+ request.getParameter("login")
+ ", AND password="
+ request.getParameter("password") + ";

The above query can access the user name and password stored in the database, concatenates them, if both are matching, query results into a true output and gives account related information.
But in case of SQL injection the attacker fires a query like:

```
SELECT * FROM accounts WHERE name='admin'
AND password='abc' OR '1=1'
```

In the above injected query, the attacker puts substitution for the password field which was written as “password” by using string, logic control keyword and logical control assignment. Below figure shows the exact information that attacker gains using an injected query.

**Figure 1: Typical problem overview [12]**

### B. Cross site scripting attack

When the injected code is in the form of a script, then it is called as a Cross site scripting attack. When malicious script is executed at user’s browser, it redirects the control from the trusted server to the untrusted server under the attacker control. Various scripting languages like JavaScript, HTML and PHP etc. are used for this purpose.

### III. CLASSIFICATION OF VULNERABILITIES

#### A. SQL Injection vulnerabilities

There are various SQL Injection attacks that occur on a web application are mentioned below.

**Tautologies**

This is a SQL injection type of attack. It uses SQL tokens: Conditional query Statements for the purpose of by-pass authentication. Inserting tautology into the query, the attacker gains unauthorized and unlimited access. If a web application is vulnerable to tautologies type of attack, the injected query evaluates true result and the attacker will be successful in getting the access of an account.

Example [13]:

When the user accesses information in his account by inserting the username and password in the related text boxes of username and password, the valid SQL query generated is:

```
"SELECT * FROM Employee WHERE username = 'admin' AND password='12345'"
```

The attacker fires a query in injected form as:

```
"SELECT * FROM Employee WHERE name = ' ' OR 1=1
AND password= ' 12345'.
```

By inserting certain symbols, the query runs successfully without generating error and the attacker can access the employee account details.

**Logically Incorrect queries**

Whenever a query fails in its execution, error messages are provided through backend and those messages contain a meaningful error fixing information. The attacker makes use of that information for accomplishing the attacks or in future attacks. The attacker passes illegal data as input, and at that time if query execution fails, then the error message is shown from the database.

Example [13]:

An attacker enters as input

```
" UNION SELECT SUM (username) from users--".
```

The resulting query formed is shown below:

```
SELECT * FROM users WHERE username=" UNION
SELECT SUM (username) from users--' and password="";
```

The above query shows the name of the database and the column field information in an error message due to invalid type conversions.

**Union Query:**

With the Union keyword, the attacker makes a combination of a valid SQL query with an injected illegal query. The attacker makes use of the keyword called Union in such way that, the attacker obtains information which is not mentioned in valid SQL query.

Example [13]:

```
SELECT username FROM user1 WHERE designation ="%lecturer" UNION
SELECT username FROM dba_users WHERE username
LIKE'%'
```

The above Concatenation of a valid query with an unsafe query not only returns the list of all selected lecturers but also all the database users present in the database.
**Stored Procedure:**

As the name suggests, the stored procedures are intended to execute the procedures stored in backend database.

Example [13]:

```
SELECT Salary FROM employee WHERE Username=''; SHUTDOWN; -- Password=' '; 
```

By adding the symbols and malicious code at the end of a valid query, the system can be abruptly shut down.

**Piggy-Backed Queries**

This is one of the most harmful SQL injection attacks. The attacker basically fires concatenation of a legal query with an illegal query. Data extraction is the main intention behind this attack. Likewise in computer networks, acknowledgment (ACK) of a packet is sent along with the next packet, attacker tries to inject additional queries with a valid SQL query.

Example [13]:

```
SELECT * FROM Employee WHERE eid='e001' AND password='1234'; DROP TABLE Employee; --';
```

Again the above query shows the concatenation of legal and illegal query due to which deletion of table occurs.

**Inference**

This is categorized into two namely Blind Injection and Timing attack.

**Blind Injection**

Sometimes error messages are not shown from the database side because the programmer or the developer hides the debugging messages appearing on the client side as a security concern. It’s a complex scenario for the attacker to inject a malicious query. As an alternative to error messages, the developer provides generic pages that attackers have to face. It’s a quite tricky method, but the attacker still steals information by going through a series of true or false type questions.

**Timing attack**

This attack is similar to blind injection, but it depends on the Timing determination. The attacker observes timing delays from the backend side. Query used in this attack is in the form of if –then statements.

**Alternate Encoding:**

In this attack type, then attacker injects query using alternate encoding. Hexadecimal, ASCII and Unicode encoding methods are used.

**B. Cross site scripting attack types**

The various Cross site scripting attacks that occur on a web application are mentioned below.

**Reflected XSS**

This attack is called as non-persistent attack which means it’s not permanently stored in the system and it is another type of code injection attack. If users click on a malicious link, the injected script gets executed and transfers control from the trusted server to the untrusted server without notifying to the user. The attacker will be successful to hijack the user session.

**Stored attack:**

This attack is called as a persistent attack which is persistently stored in targeted server. An attacker stores the script only once and it is executed as many times as the web page is visited by victim and send the victim’s sensitive information from his site to the attacker’s site [13].

IV. ASSESSMENT ON DETECTION AND PREVENTION APPROACHES

Stephen W. Boyd and Angelos D. Keromytis [4, 13] in 2004 proposed an approach based on the concept known as Instruction Set Randomization method, which concatenates a random integer to a valid SQL query. So every time the database gets a new integer value when the query executes. Recognition of random integer is a difficult task for the attackers. But those random integers cannot be recognized by the backend database too. On solution to this problem, the author proposed a module which decodes the query before sending to the database. This method shows a poor performance in detecting SQL injection vulnerabilities. Only tautology attacks are detected.

William G.J. Halfond and Alessandro Orso [2,13] in 2005 proposed a technique based on the concept, model based hybrid approach in “Analysis and Monitoring for NEutralizing SQL Injection Attacks” (AMNeSIA) which detects injected queries. It works on the static as well as the dynamic approach. Dynamically generated queries are successfully compiled with statically generated queries at the time of valid SQL query. This tool can identify a hotspot in SQL query and construct NFA based query model for each generated hotspot. An inadequacy of Amnesia tool is that, it can only be used in Java Server Pages (JSP) web applications [16]. This tool makes use of the Java String Analyzer (JSA) library to construct a query model which is not available in any other languages[16].
This model totally depends upon the accuracy of static analysis for constructing the query model.

MeiJunjin [3, 13] in 2009 provides extension on AMNESIA by providing automatic testing. The input for the is generated by analyzing and locating input flows as well as identification of hotspot and test case generation are done with respect to their SQLInjectionGen and Jchasher. When using different tools it requires more number of steps. The proposed automated technique is evaluated with the static analysis tool, FindBugs, and resulted to be efficient as regard to the fact that false positive was completely absent in the experiments [13].

Shaukat Ali, Azhar Rauf, Huma Javed [13, 14] in 2009 proposed an approach based on the concept known as stored procedure and hash values. SQLIPA (SQL Injection Protector for Authentication) prototype was developed for more secure authentication. It automatically generates hash values with respect to user name and password. But again, it’s a restrictive form of technique, detects only Tautology attacks.

Pritvi Bisht, P. Madhusudan et al. [15] in 2007 proposed an approach based on the concept known as dynamic candidate evaluation against SQL injection attacks. It provides very a powerful technique against SQLIA.

Russell A. McClure and Ingolf H. Krüger [1] in 2005 proposed an approach based on the concept known as executable sqldomgen which generates DLL as output, contains classes namely SQL Domain Object Model (SQL DOM). It works on object oriented programming concepts. SQL DOM classes are used to construct dynamic SQL statement for every possible legal SQL query input. This technique is unsuccessful in detecting the stored procedures SQL injection attack.

Kiezun et al. in [16]2009 proposed an approach based on the concept known as ardilla tool which detects both SQL injection and cross site scripting attack. This technique is based on the static analysis. It creates a tangible input by scrutinizing the source code of web applications that expose vulnerabilities.

Buehrer et al. [18] in 2005 and Wassermann et al. [17] in 2006 proposed an approach based on the concept known as Parse Tree Validation Approach. SQL Guard and SQL Check both work on runtime environment. SQL Guard is having a large overhead whereas SQL Check is having a low overhead and shows no false positives or false negatives. Both approaches use secret key.

Atul S. Choudhary, M.L.Dhore [13] in 2012 proposed an approach based on the concept known as proxy agent which classifies input request as query structure or script format, and then detects the respective attack. Code Injection Detection Tool (CIDT) acts like a proxy agent contains a separate Query Detector or Script Detector. Input requests are validated by both detectors if gets validated and then request gets forwarded to the Web server. This approach shows efficient performance in detecting both types of vulnerabilities.

S. Fouzul Hidhaya, Angelina Geetha[6] in 2012 proposed an approach based the on concept known as reverse proxy and MD5 algorithm to validate input requests. Grammar expressions rules are used to detect vulnerabilities in URLs. Reverse proxy server is a mediator between client and the server. It validates requests and then forward to intended server. Client http request is first redirected to reverse proxy server before passing to an intended server. Sanitizing application in reverse proxy server checks the input request. If injected URL is found, then the sanitizer application shows that the URL request is malicious.

Pankaj Sharma, Rahul Johari, S.S.Sarma [16] in 2012 proposed an approach which is an extension of Model based Hybrid Approach to Prevent SQL Injection Attack (MHAPSIA) works in production web environment which provides protection from SQL injection and cross site scripting attack. This mechanism is divided into safe mode, which creates a secure query model for SQL injection and sanitizer model for reflected cross site scripting attacks. And another mode is production environment, in this mode dynamic queries are validated against secure query model in safe mode and normal input request is validated against sanitizer model in safe mode.

This paper provides overall information about SQL Injection, cross site scripting vulnerabilities and their respective detection and prevention mechanism.

V. CONCLUSION

In this paper, we have reviewed various Code Injection attacks and the classification of their vulnerabilities in addition to the different approaches, Detection and Prevention against SQL Injection and cross site scripting attacks.
We have presented a survey report on SQL Injection and their related issues as well as the cross site scripting and their related issues.

REFERENCES


