TELKOMNIKA, Vol.17, No.2, April 2019, pp.852~856
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v17i2.10484

Security vulnerabilities related to web-based data

Mohammed Awad[1], Muhammed Ali[2], Maen Takruri[3], Shereen Ismail[4]
[1,2,4]Department of Computer Science and Engineering,
American University of Ras Al Khaimah, Ras Al Khaimah, UAE
[3]Department of Electrical, Electronics and Communications Engineering,
American University of Ras Al Khaimah, Ras Al Khaimah, UAE
*Corresponding author, e-mail: mohammed.awad@aurak.ac.ae[1], m.ali@aurak.ac.ae[2],
maen.takruri@aurak.ac.ae[3], shereen.subhi@aurak.ac.ae[4]

Abstract

In this digital era, organizations and industries are moving towards replacing websites with web applications for many obvious reasons. With this transition towards web-based applications, organizations and industries find themselves surrounded by several threats and vulnerabilities. One of the largest concerns is keeping their infrastructure safe from attacks and misuse. Web security entails applying a set of procedures and practices, by applying several security principles at various layers to protect web servers, web users, and their surrounding environment. In this paper, we will discuss several attacks that may affect web-based applications namely: SQL injection attacks, cookie poisoning, cross-site scripting, and buffer overflow. Additionally, we will discuss detection and prevention methods from such attacks.

Keywords*: buffer overflow, cookie poisoning, cross-site scripting, SQL injection, web security*

1. Introduction

Nowadays, every organization supplies an enormous amount of information to clients, employees, and the general public online via the web. While printed information is static, web-based content is dynamic and updated promptly. However, websites and web servers are exposed to various security threats. Any server connected to an online network will not only suffer from internal threats caused by employees as a result of misuse of network resources, but are also vulnerable to a range of outside threats [1, 2]. Web security is extremely important in order to protect sensitive data (stored online) from unauthorized access, alteration, or destruction. Such security is accomplished by applying strict policy measures to safeguard such valuable assets. Security threats can negatively impact organization's business or reputation if attackers manage to gain access to that organization's data.

2. Fundemantels of Web Security

Several cornerstones must be assured to describe a system secure. The most common fundamentals are:

a. Authentication

Authentication is the process of successfully verifying the identity of the users for a certain application [3].

b. Authorization

Authorization means giving a user a specific permission to do certain operations on specific resources [4]. Resources may include files, databases, tables, and rows.

c. Auditing

Auditing is important to determine if any attack or intrusion took place. In addition to effective auditing, logging is important to assure non-repudiation. Non-repudiation means that the sender of a message should not be able to prove that he has not sent that message. It should always be feasible to find out the sender of a message [5].

Received May 4, 2018; Revised August 30, 2018; Accepted September 15, 2018

d. Confidentiality

Confidentiality is the process of keeping the data private, and making sure that only authenticated user can view that specific data. One way to implement confidentiality is by using data encryption.

e. Availability

In our context, availability means that a system is always available only for legitimate users. Attackers succeed to compromise a system's availability by overloading the network so that other users cannot successfully gain access to that system [6].

3. Technical Vulnerabilities

3.1. SQL Injection Attack

SQL injection is a malicious attack where an attacker injects SQL commands into an SQL statement, via web page input to compromise a database. An SQL injection is capable of destroying a database [7]. Typically, on a web form for user authentication, when a user enters personal information such as username and password or other login credentials, these values are inserted into a SELECT query. If these credentials are found in the database, access is granted. Otherwise, the access is denied.

Unfortunately, most of these web forms have no mechanisms in place to block input other than names and passwords. Hence, an attacker might be able to use the input boxes to send malicious requests to the database. Consequently, these attempts could allow the attacker to interact with the database or even download it entirely [8]. There are several threats that may affect online applications. In this paper, we will highlight four of them. Namely: SQL injection attack, cookie poisoning, cross-site scripting, and buffer overflow. For the SQL code below [9]:

```
String SQLQuery = "SELECT Username, Password FROM users WHERE Username=' " + Username + " ' AND Password=' " + Password +" ' ";
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery(SQLQuery);
While (rs.next()) {…}
```

Since user input is needed in order to run the above query. The interpreter will execute the command based on the provided input for the username and password fields [9]. Hence, if an attacker inserts 'or 0=0' as both the username and password, then the query will be constructed as:

```
String SQLQuery ="SELECT Username, Password FROM users WHERE Username=" or 0=0" AND Password=" or 0=0";
```

By running the above query, the attacker managed to compromise the entire database. This is due to the fact that zero will always be equal to zero. Thus, the query will return all records in the database [9]. SQL injection attack is considered a common threat that targets database-driven applications and exposes their vulnerabilities [10]. Generally, there are two ways to carry SQL injection attacks. The first way is known as SQL Manipulation. In SQL Manipulation the attacker alters the SQL statement with the aid of built-in operation or built-in function such as UNION. The other method is known as Code Injection where an attacker manages to insert vulnerable SQL statement.

Defensive coding is one of the proposed solutions to prevent SQL injection attack. However, defensive coding is considered a challenging task. On one hand developers continuously keep adding controls in their source code. On the other hand, attackers never stop finding ways to overcome these added controls [11]. Thus, it is difficult for developers to keep up in this race. Additionally, implementing the best defensive coding techniques is very challenging and needs special set of skills. Table 1 explains some of the possible SQL injection attacks [12].

Nonetheless, SQL injection can still be prevented in several ways. Enforcing the faultlessness of the user's input value at the programming level can prevent SQL injection attacks. This means enforcing the user to provide correct information and prevent irregular values. This technique blocks unsafe values, which could be harmful to the database server.

Such approach can be implemented on either database side (the client's or the server's) [12]. The following are some of the techniques for preventing SQL injection attacks:

a. Static Query Statement

To achieve this prevention method, the database uses parameterized statements where the server encodes the parameters as needed.

b. Least Privilege

The least privilege techinque may restrict users access by eliminating database connection with root access for all users, restricting the privileges (such as read and write) on the database's tables and views, rejecting access to stored procedures and special system utilities, or by utilizing stored procedures with fine-grained control. Fine-grain control will set control over the users granted roles. For example, a better alternative for granting the user access to an entire table is that a stored procedure returns only one row of data [13].

c. Web Application Gateway

Web Application Gateway (WAG) works as a secure reverse web proxy. WAG can interpret more information in the HTTP protocol, and HTML content compared to a web proxy. WAG can check whether form inputs are within expectations and verifies whether hidden fields and cookies were modified. Additionally, WAG verifies whether URL flow follows the original design or not. Hence, it provides protection against other threats as well [13].

d. SQL driver proxy

A SQL driver proxy functions in a similar way to web application gateway. Unlike the web application gateway, it intercepts API calls and monitors database function calls rather than intercepting network connections and monitoring HTTP requests [14]. Additionally, the proxy sanitizes error message sent from database server back to the client application [14]. This is an essential step since such messages usually reveal information about the database schema.

Tabel 1. SQL Injection Possible attacks [12]

Types of Attacks	Description
Tautologies	This attack belongs to the SQL Manipulation category. Anonymous user can inject vulnerable code into SQL statement. This based upon conditional statement
Logically Incorrect Queries	This type of attack also belong to the SQL Manipulation category with the help of this attack anonymous user can get help message from error message which will produced by data base server.
Union Query	This attack belongs to the code injection category and SQL manipulation category in which inject infected queries with safe queries to get table or database information.
Stored Procedure	This type of attack belongs to the function call injection category. In this type attacks anonymous user execute built in procedure for getting table information or database information.
Piggy Back Query Attack	This type of attack represent to the code injection category.
Alternate Encoding	In this type of attacks, SQL queries will be manipulate using alternate encoding, Hexadecimal, ACSII and Unicode.
Blind Attacks	In this type of attacks anonymous user can theft useful information from database asking true and false a question through SQL statement.

3.2. Cookie Poisoning

Cookies are personal information stored on a user's machine. Modifying cookies by an attacker to gain unauthorized information about the user for malicious purposes is known as Cookie Poisoning. The attacker may use the hacked identity information to open new accounts or to obtain access to the user's existing accounts [15]. If security methods are not used, an attacker can observe a cookie to know its purpose and make some changes to it to allow the attacker accessing user information from the website that sent the cookie.

Many web applications store tracking information within cookies saved in the user computer. The tracking information usually include user IDs and time stamps. When a user logs into a website, a login web script validates his username and password and sets a cookie with his numerical identifier [16]. In many cases, cookies can be modified by attackers as they are cryptographically insecure, thus, the application is fooled and the hacker gains full access to a

system. Encrypting cookie contents is a useful prevention method against cookie poisoning. Additionally, avoiding storing session ID on the server is another helpful technique.

3.3. Cross-site Scripting

Cross-site Scripting (XSS) refers to client-side code injection attack, wherein an attacker can execute malicious scripts (also commonly known as a malicious payload) into a legitimate website or web application [17]. XSS is amongst the most rampant of web application vulnerabilities and occurs when a web application makes use of not validated or not encoded user input within the output it generates [18]. For example, the script below is not harmful but will open a thousand windows:

```
<script>
for (i = 0; i < 1000; i++)
 window open ("http://goingzoo.com ");
</script>
```

A solution to this could be to disable script language capability in web browsers. However, some trusted websites also use a script to provide a lot of functionalities. Hence, disabling script language could prevent users from accessing those features. Another solution would be using an anti-XSS HTML parser like safeHTML. For example, a hacker could upload the following message:

```
"Hello message board. This is a message.
 <SCRIPT>malicious code</SCRIPT>
This is the end of my message."
```

If the victim's script is enabled in their browser, the malicious code will execute unexpectedly. Some of the scripting tags that can be embedded in a similar fashion include <SCRIPT>, <OBJECT>, <APPLET>, and <EMBED>. When a server mediates client-to-client communications, site developers clearly conclude that data input is untrustworthy when it is presented to other users. Such an input will be considered unacceptable and will be filtered by the server before being sent to other readers [19].

To carry a cross-site script attack, the attacker will need to find a web page with unchecked input and embed a "poisoned link" within that web page. Once the victim clicks on the poisoned link, the script executes whatever the purpose behind it is. Note that the victim does not have to click the link; it is possible for the scripts to execute upon loading the page. Cross-site script attack can be prevented if the input data on the web server was sanitized and the cookie data encrypted. Additionally, users should turn off client-side scripting.

3.4. Buffer Overflow

Buffer Overflow occurs when an attacker attempts to send more data purposely than a program or process can handle. This kind of attack can force a system to crash [20]. The code below is an example of buffer overflow attack.

```
void Function( char *Input )
{
char Buffer[17];
// Input is copied straight into the buffer
// when no type checking is performed
strcpy(Buffer, Input);}
```

Some of the solutions to prevent buffer overflow attack is bounds checking, use of safe libraries, and running static code analysis. Developers use HTML and JavaScript to limit the number of characters to be submitted as input. Nevertheless, the HTML can still be changed by the hacker, the JavaScript can be turned off as well and then a buffer overflow attack can be submitted. To ensure ultimately safeguarding the application from this attack, all input from the client has to be carefully checked on the server. Also, a firewall or security gateway application could be installed to scan out requests with unusual length.

4. Conclusion

Over the past decade, web applications have evolved significantly. Such an upraise introduced new and serious security-related challenges. On one hand security researchers are putting substantial efforts to eliminate these threats, and on the other hand, attackers are still finding ways to attack and invade victims' security and privacy. This paper presented some of the fundamentals of web security, in addition to several kinds of web attacks, and discussed some prevention methods.

References

[1] service delivery models of cloud computing. *Journal of network and computer applications*. 2001; 34(1), pp.1-11

[2] Teodoro N, Serrão C. Web application security: Improving critical web-based applications quality through in-depth security analysis. In Information Society (i-Society). International Conference on IEEE. 2011; 457-462.

[3] Zissis D, Lekkas D. Addressing cloud computing security issues. Future Generation computer systems. 2012; 28(3): 583-592.

[4] Li X, Xue Y. A survey on web application security. Nashville, TN USA. 2011.

[5] Meier JD, Farre C, Taylor J, Bansode P, Gregersen S, Sundararajan M, Boucher R. Improving Web Services Security: Scenarios and Implementation Guidance for WCF. Microsoft Developer Network. 2009.

[6] Kadam K, Paikrao R, Pawar A. Survey on Cloud Computing Security. International Journal of Emerging Technology and Advanced Engineering. 2013; 3(12): 239-249.

[7] Sadeghian A, Zamani M, Abdullah SM. *September. A taxonomy of SQL injection attacks. In Informatics and Creative Multimedia (ICICM).* International Conference on IEEE. 2013; 269-273.

[8] Lee I, Jeong S, Yeo S, Moon J. A novel method for SQL injection attack detection based on removing SQL query attribute values. *Mathematical and Computer Modelling*. 2012; 55(1): 58-68.

[9] Shrivastava S, Tripathi RRK. Attacks Due to SQL injection & their Prevention Method for Web-Application. *International Journal of Computer Science and information technologies*. 2012; 3(2): 3615-3618.

[10] Clarke-Salt J. 2009. SQL injection attacks and defense. Elsevier.

[11] Sharma P, Johari R, Sarma SS. Integrated approach to prevent SQL injection attack and reflected cross site scripting attack. *International Journal of System Assurance Engineering and Management*. 2012; 3(4): 343-351.

[12] Raghuvanshi K, Dixit P. Prevention and Detection Techniques for SQL Injection Attacks. *International Journal of Computer Trends and Technology (IJCTT)*. 2014; 12(3): 107-110.

[13] Krishnamurthy A, Mettler A, Wagner D. *Fine-grained privilege separation for web applications.* In Proceedings of the 19th international conference on World wide web. ACM. 2010; 551-560.

[14] Johns M, Winter J. *Request Rodeo: Client side protection against session riding*. In Proceedings of the OWASP Europe Conference. 2006.

[15] Bhadauria R, Chaki R, Chaki N, Sanyal S. *A survey on security issues in cloud computing*. IEEE Communications Surveys and Tutorials. 2011; 1-15.

[16] Reshef E, Bar-Gad I. Web Application Security. Retrieved from http://www.cgisecurity.com/lib/reschef.pdf. 2000.

[17] Tajpour A, Ibrahim S, Sharifi M. Web application security by sql injection detectiontools. *International Journal of Computer Science Issues*. 2012; 9(2): 332-339.

[18] Manaa ME, Hussein R. Preventing Cross Site Scripting Attacks in Websites. Asisan. *Journal of Information Technology*. 2016; 15(16): 2297-2804.

[19] CERT Advisory CA-2000-02. Malicious HTML Tags Embedded in Client Web Requests. February 2, 2000. http://www.cert.org/advisories/CA-2000-02.html

[20] Grobauer B, Walloschek T, Stocker E. Understanding cloud computing vulnerabilities. *IEEE Security & Privacy*. 2011; 9(2): 50-57