SQL Injection Defense Mechanisms for IIS+ASP+MSSQL Web Applications

Beihua Wu*

East China University of Political Science and Law,
555 Longyuan Road, Shanghai, China, 201620
wubeihua@ecupl.edu.cn

Abstract. With the sharp increase of hacking attacks over the last couple of years, web application security has become a key concern. SQL injection is one of the most common types of web hacking and has been widely written and used in the wild. This paper analyzes the principle of SQL injection attacks on Web sites, presents methods available to prevent IIS+ASP+MSSQL web applications from these kinds of attacks, including secure coding within the web application, proper database configuration, deployment of IIS and other security techniques. The result is verified by WVS report.

Keywords: SQL Injection, Web sites, Security, Cybercrime.

1 Introduction

Together with the development of computer network and the advent of e-business (such as E-trade, cyber-banks, etc.) cybercrime continues to soar. The number of cyber attacks is doubling each year, aided by more and more skilled hackers and increasing easy-to-use hacking tools, as well as the fact that system and network administrators are exhausted and have inadequately trained. SQL injection is one of the most common types of web hacking and has been widely written and used in the wild. SQL injection attacks represent a serious threat to any database-driven sites and result in a great number of losses. This paper analyzes the principle of SQL injection attacks on Web sites, presents methods available to prevent IIS+ASP+MSSQL web applications from the attacks and implement those in practice. Finally, we draw the conclusions.

2 The Principle of SQL Injection

SQL injection is a code injection technique that exploits a security vulnerability occurring in the database layer of an application [1]. If user input, which is embedded in SQL statements, is incorrectly filtered for escape characters, attackers will take advantage of the present vulnerability. SQL injection exploit can allow attackers to obtain unrestricted access to the database, read sensitive data from the database, modify database data, and in some cases issue commands to the operating system.

* Academic Field: Network Security, Information Technology.

X. Lai et al. (Eds.): E-Forensics 2010, LNICST 56, pp. 271–276, 2011.

The multistep of SQL injection attack is as follows:

2.1 Finding Vulnerable Pages

In the first, try to look for pages that allow you to submit data, such as login pages with authentication forms, pages with search engines, feedback pages, etc. In general, Web pages use *post* or *get* command to send parameters to another ASP page. These pages include <*Form*> tag, and everything between the <*Form*> and <*/Form*> has potential parameters that might be vulnerable [2]. You may find something like this in codes:

```
    <Form action="search.asp" method="post" id="search">
      <input type="text" size="12" name="t_name" />
      <input type="submit" name="Submit" value="search"
/>
    </Form>
```

Sometimes, you may not see the input box on the page directly, as the type of <input> can be set to hidden. However, the vulnerability is still present.

On the other hand, if you can't find any <*Form*> tag in HTML code, you should look for pages like ASP, PHP, or JSP web pages, especially for URL that takes parameters, such as: http://www.sqlinjection.com/news.asp?id=1020505.

2.2 SQL Injection Detection

How do you test if the web page is vulnerable? A simple test is to start with single quotation marks (‘) trick. Just enter an ‘ in a form that is vulnerable to SQL injection, or input it in the URL with parameters, such as: http:// www.sqlinjection.com/ news.asp ?id=1020505’, trying to interfere with the query and generate an error. If we get back an ODBC error, chances are that we are in the game.

Another usual method to be used is *Logic Judgement Method*. In others words, some SQL keywords like *and* and *or* can be used to try to modify the query and to detect whether it is vulnerable or not. Consider the following SQL query:

```
SELECT * FROM Admin WHERE Username='username' AND
Password='password'
```

A similar query is generally used in the login page for authenticating a user. However, if the *Username* and *Password* variable is crafted in a specific way by a malicious user, the SQL statement may do more than the programmer intended. For example, setting the *Username* and *Password* variables as *1' or '1' = '1* renders this SQL statement by the parent language:

```
SELECT * FROM Admin WHERE Username = '1' OR '1' = '1'
AND Password = '1' OR '1' = '1'
```

As a result, this query returns a value because the evaluation of '1'='1' is always true [3]. In this way, the system has authenticated the user without knowing the username and password.

2.3 SQL Injection Attacks Execution

Without user input sanitization, an attacker now has the ability to add/inject SQL commands, as mentioned in the source code snippet above. As default installation of MS SQL Server is running as SYSTEM, which is equivalent to administrator access in Windows, the attacker has the ability to use stored procedures like *master..xp_cmdshell* to perform remote execution:

```
exec master..xp-cmdshell "net user user1 psd1 /add"
exec master..xp-cmdshell "net localgroup administrators
user1 /add"
```

These inputs render the final SQL statements as follows:

```
SELECT * FROM Admin WHERE Username = '1' ; exec
master..xp_cmdshell "net user user1 psd1 /add"
SELECT * FROM Admin WHERE Username = '1' ; exec
master..xp_cmdshell "net localgroup administrators
user1 /add"
```

The semicolon will end the current SQL query and thus start a new SQL command. These above statements can create a new user named *user1* and add *user1* to the local *Administrators* group. In the result, SQL injection attacks succeed.

3 SQL Injection Defense

The major issue of web application security is SQL injection, which can give the attackers unrestricted access to the database that underlie web applications and has become increasingly frequent and serious. In this section, we present some methods available to prevent from SQL injection attacks and implement on the IIS+ASP+MSSQL web applications practically.

3.1 Secure Coding within the Web Application

Attackers take advantage of non-validated input vulnerabilities to inject SQL commands as an input via Web pages, thus execute arbitrary SQL queries on the backend database server. A straight-forward way to prevent injections is to enhance the reliability of program code.

Use Parameterized Statements. On most development platforms, parameterized statements can be used that work with parameters (sometimes called *placeholders* or *bind variables*) instead of embedding user input in the statement directly. For example, we construct the code as fallow:

```
searchid = request.querystring("id")
searched = checkStr(searchid)
sql = "SELECT Id, Title FROM News WHERE Id= '" &
searchid &"'"
```

Here, *checkStr* is a function for input validation. It is seen that the user input is assigned to a parameter, and then the SQL statement is fixed.

To protect against SQL injection, user input must not be embedded in SQL statements directly. Instead, parameterized statements are preferred to use.

Enhance Input Validation. It is imperative that we should use a standard input validation mechanism to validate all input data for length, type, syntax and business rules before accepting the data to be displayed or stored [4].

Firstly, limit the input length because most attacks depend on query strings. For instance, the length of I.D. card is limited to 15 or 18 in China.

Secondly, a crude defense is to restrict particular keywords used in SQL. It means that we should draw up a black list, which includes keywords such as *drop*, *insert*, *drop*, *exec*, *execute*, *truncate*, *xp_cmdshell* and *shutdown*. Also, ban SQL code such as single quotes, semicolon, --, %, =.

After checking the existence of normalized statement in the ready-sorted allowable list, we will be able to determine whether a SQL statement is legal or not. If the input data consists of illegal characters, the URL will redirect to a custom error page.

3.2 Proper Database Configuration

Enforce Least Privilege when Accessing the Database. Connecting to the database using the database's administrator account has the potential for attackers to execute almost unconfined commands with the database [5]. For instance, A system administrator account in MSSQL(sometimes called *sa*) is available to exploit *xp_cmdshell* command to perform remote execution.

To minimize the risk of attacks, we enforce the least privileges that are necessary to perform the functions of the application. Even though a malicious user is able to embed SQL commands inside the parameters, he will be confined by the permission set needed to run SQL Server.

Use Stored Procedures Carefully. As mentioned above, it is important to validate input data to ensure that no illegal characters are present. However, it is doubly important to restrict the application database user to execute the specified stored procedures. Validate the data if the stored procedure is going to use *exec(some_string)* where *some_string* is built up from data and string literals to form a new command [5].

Moreover, remove the extended stored procedure as follow:

```
use master
sp_dropextendedproc 'xp_cmdshell'
```

Visit the registry to store, and delete *Xp_regaddmultistring*, *Xp_regdeletekey*, *Xp_regdeletevalue*, *Xp_regenumvalues*, *Xp_regread*, *Xp_regwrite*, *Xp_regremovemultistring* extended procedures.

Release Security Patches. Last but not least, deploy database patches as they are released. It is an essential part in the defense against external threats.

3.3 Deployment of IIS (Internet Information Services)

Avoid Detailed Error Messages. Error messages are useful to an attacker because they give some additional information about the database. It is helpful for the technical

supporter to get some useful information when the application has something wrong. However, it tells the hacker much more. A better solution is that just display a generic error message instead, which does not compromise security.

To resolve this problem, we set a generic error page for individual pages, for a whole application, or for the whole Web site or Web server. Additionally, select *Send the following text error message to client* to enable IIS to send a default error message to the browser when any error prevents the Web server from processing the ASP page.

Improved File-System Access Controls. To ensure each Web site has a different anonymous impersonation account identity configured, we create a new user to be used as an anonymous Internet User Guest Account and grant the appropriate permissions for each site, and disable the built-in IIS anonymous user. Moreover, deny *write* access to any file or directory in the web root directory to the anonymous user unless it is necessary.

In addition, FTP users should be isolated in their own home directories. FTP provides a means for transferring data between a client and the web host's server. While the protocol is quite useful, FTP also presents many security risks. Such attacks may include Web site defacement by uploading files to the web document root and remote command execution via the execution of malicious executables that may be uploaded to the scripts directory [6]. So we configure the *Isolation mode* for an FTP site when creating the site through the FTP Site Creation Wizard.The limitation prevents a user from uploading malicious files to other parts of the server's file system.

3.4 Other Security Techniques

We can improve the security of our Web servers and applications by using the tools, such as URLScan Security Tool, IIS Lockdown Tool, IIS Security Planning Tool. Here, we use URLScan 2.5 on IIS in practice.

URLScan is a security tool that restricts the types of HTTP requests that Internet Information Services (IIS) will process. By blocking specific HTTP requests, URLScan helps to prevent potentially harmful requests from being processed by web applications on the server [7].

All configuration of URLScan is performed through the *URLScan.ini* file, which is located in the *%WINDIR%\System32\Inetsrv\URLscan* folder. Define the *AllowVerbs* section as *get*, *post*, *head*. And permit the requests that use the verbs which are listed in the *AllowVerbs* section. Furthermore, configure URLScan to reject requests for *.exe*, *.asa*, *.bat*, *.log*, *.shtml*, *.printer* files to prevent Web users from executing applications on the system. In addition, we configure it to block requests that contain certain sequences of characters in the URL, Such as *'..'*, *'./'*, *'\'*, *':'*, *'%'*, *'&'*. It is seen that URLScan includes the ability to filter based on query strings, which can help reduce the effect of SQL injection attacks.

4 Conclusion

Scanning our Web site with Acunetix WVS6.5, three low-severity vulnerabilities have been discovered by the scanner. The result is given in Table 1. It is seen that

possible sensitive directories have been found, and these directories are not directly linked from the Web site. To fix the vulnerabilities, we restrict access to these directories. For instance, *admin* directory is confined to access only for appointed IP address, and deny write access to *cms* and *data* directory.

Table 1. Web vulnerability scanning report with Acunetix WVS6.5

Severity level	Quantity	Vulnerability description	Detail
High	0		
Medium	0		
Low	3	possible sensitive directory	/admin /cms /data

SQL injection has been one of the most widely used attack vectors for cyber attacks in recent years. In this paper, we pose SQL Injection Defense Mechanisms available to prevent IIS+ASP+MSSQL web applications, including secure coding within the web application, proper database configuration, deployment of IIS and other security techniques.

In the end, we must emphasize that each prevention technique cannot provide complete protection against SQL Injection Attacks, but a combination of the presented mechanisms will cover a wide range of these attacks.

References

1. Watson, C.: Beginning C# 2005, databases. Wrox, 201–205 (2005)
2. SQL Injection Walkthrough, http://www.securiteam.com/securityreviews/5DP0N1P76E.html
3. Pan, Q., Pan, J., Shi, Y., Peng, Z.: The Theory and Prevention Strategy of SQL Injection Attacks. Computer Knowledge and Technology 5(30), 8368–8370 (2009) (in Chinese)
4. Data Validation, http://www.owasp.org/index.php/Data_Validation
5. SQL Injection Attacks and Some Tips on How to Prevent Them, http://www.codeproject.com/KB/database/SqlInjectionAttacks.aspx
6. Belani, R., Muckin, M.: IIS 6.0 Security, http://www.securityfocus.com/print/infocus/1765
7. How to configure the URLScan Tool, http://support.microsoft.com/kb/326444/en-us