IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 6, Ver. V (Nov.-Dec. 2016), PP 67-71
www.iosrjournals.org

"Web Application Security- SQL Injection – Modes of Attacks, Defense and why it Matters"

Monali Sachin Kawalkar
Mumbai, India

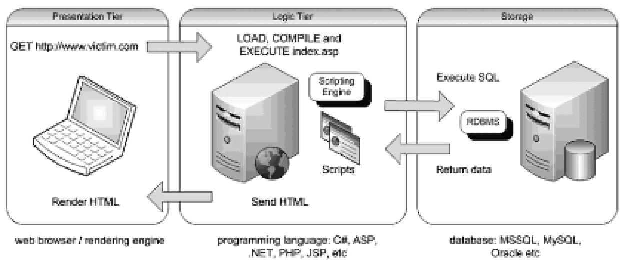
Abstract: *SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks an incredible number of systems on the internet are susceptible to this form of attack.Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can be almost totally prevented. This paper will look at a selection of the methods available to a SQL injection attacker and how they are best defended against.*

I. Introduction

SQL Injection (SQLi) refers to an Injection attack wherein an attacker can execute malicious SQL statements that control a web application's database server (also commonly referred to as a relational database Management System-RDBMS). Since an SQL injection vulnerability could possibly affect any website or web application that makes use of an SQL-based database, the vulnerability is one of the oldest, most prevalent and most dangerous of web application vulnerabilities.

By leveraging an SQL injection vulnerability, given the right circumstances, an attacker can use it to bypass a web application's authentication and authorization mechanisms and retrives the contents of an entire database. SQL injection can also be used to add, modify and delete records in a database, affecting data integrity.

To such an extent, SQL injection vulnerability can provide an attacker with unauthorized access to sensitive data including customer data, personally identifiable information (PII), trade secrets, intellectual property and other sensitive information. The information might be very confidential from Business perspective.


Presentation Tier
Logic Tier
Storage
GET http://www.victim.com
LOAD, COMPILE and EXECUTE index.asp
Scripting Engine
Scripts
Execute SQL
RDBMS
Return data
Render HTML
Send HTML
web browser / rendering engine
programming language: C#, ASP, .NET, PHP, JSP, etc
database: MSSQL, MySQL, Oracle etc

Simple Three-Tier Architecture

As shown in above figure, The Web browser (presentation) sends a request to the middle tier (logic), which services them by making queries and updates against the database (storage).A fundamental rule in three-tier architecture is that the presentation tier never communicates directly with the data tier; in a three-tier model, all communication must pass through the middle wear tier. Conceptually, the three-tier architecture is linear.

The user fires up his web browser and connects to http://www.victim.com. The Web server that resides in the logic tier loads the script from the file system and passes it through its scripting engine, where it is parsed and executed. The script opens a connection to the storage tier using a database connector and executes an SQL statement against the database. The database returns the data to the database connector, which is passed to the scripting engine within the logic tier. The logic tier then implements any application or business logic rules before returning a Web page in HTML format to the user's Web browser within the presentation tier. The user's Web browser renders the HTML and presents the user with a graphical representation of the code. All of this happens in a matter of seconds and its transparent ot the user.

Threat Modeling

a) SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

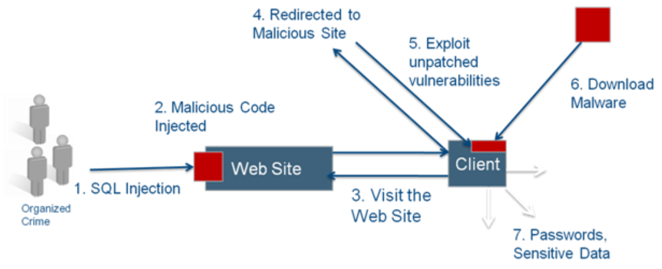
b) SQL Injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces. Due to the nature of programmatic interfaces available, J2EE and ASP.NET applications are less likely to have easily exploited SQL injections.
c) The severity of SQL Injection attacks is limited by the attacker's skill and imagination, and to a lesser extent, defense in depth countermeasures, such as low privilege connections to the database server and so on. In general, consider SQL Injection a high impact severity.

This paper consist of 4 parts-
Part I-Injection Principles (How sql injection works)
Part II-Protection
Part III-Conclusion
Part IV-Reference

II. Injection Principle (How SQL Injection Works)

The principle of basic SQL injection is to take advantage of insecure code on a system connected to the internet in order to pass commands directly to a database and to then take advantage of a poorly secured system to leverage an attacker's access . In order to run malicious SQL queries against a database server, an attacker must first find an input within the web application that is included inside of an SQL query.

In order for an SQL injection attack to take place, the vulnerable website needs to directly include user input within an SQL statement. An attacker can then insert a data that will be included as a part of the SQL query and run against the database server.
The following diagram shows how SQL injection done.


4. Redirected to Malicious Site
5. Exploit unpatched vulnerabilities
6. Download Malware
2. Malicious Code Injected
Web Site
Client
1. SQL Injection
Organized Crime
3. Visit the Web Site
7. Passwords, Sensitive Data

Step 1: The attacker attacks the Web Site using SQL injection. This involves information gathering queries followed by the specific attack that injects the link. The link is injected into string fields of the database so that dynamic web page content generated from the database will contain the link.
Step 2: Once the attack is successful, the attacker injects malicious code to the Web Site.
Step 3, 4 and 5: Users visit the website and are redirected to a malicious site which attacks the desktop using known vulnerabilities in operating system, browser and ActiveX plugins.
Step 6: Once successful, the attacker connects to various other sites to download malware (keylogger, password stealer etc)
Step 7: The attacker has sensitive information and has complete control of the desktop

The following server-side pseudo-code is used to authenticate users to the web applications.
# Define POST variables
uname = request.POST['username']
passwd = request.POST['password']
# SQL query vulnerable to SQLi
Sql = "**SELECT** id **FROM** users **WHERE** username=' " +**uname** + " ' **AND password**='" + passwd + " '"
# **Execute** the **SQL** statement
database.execute(sql)

The above script is a simple example of authenticating a user with a username and a password against a database with a table named users, and a username and password column. The above script is vulnerable to SQL injection because an attacker could submit malicious input in such a way that would alter the SQL statement being executed by the database server. A simple example of an SQL injection payload could be something as simple as setting the password field to password' OR 1=1
This would result in the following SQL query being run against the database server.
SELECT id **FROM** users **WHERE** username='username' **AND password='password' OR** 1=1"
An attacker can also comment out the rest of the SQL statement to control the execution of the SQL query further.

-- MySQl, MSSQL, Oracle, PostgneSQL, SQLite
OR '1' ='1' –
OR '1' = '1' /*
-- MySQL
OR '1' = '1' #
--Access (using null characters)
OR '1' ='1' %00
OR '1'= '1' %16

Once the query executes, the result is returned to the application to be processed, resulting in an authentication bypass. In the event of authentication bypass being possible, the applications will most likely log the attacker in with the first account from the query result – the first account in a database is usually of administrative users. Such many methods likes above example are included in modes of attack. It is likes hacking of the query string, breaking the query string, database foot printing, adding unauthorized data.

Why SQL injection is a problem?
SQL is a programming language designed for managing data stored in RDBMS, therefore SQL can be used to access, modify and delete data. Furthermore, in specific cases, an RDBMS could also run commands on the operating system from an SQL statement.
Keeping the above in mind, when considering the following, it's easier to understand how lucrative a successful SQL injection attack can be for an attacker.

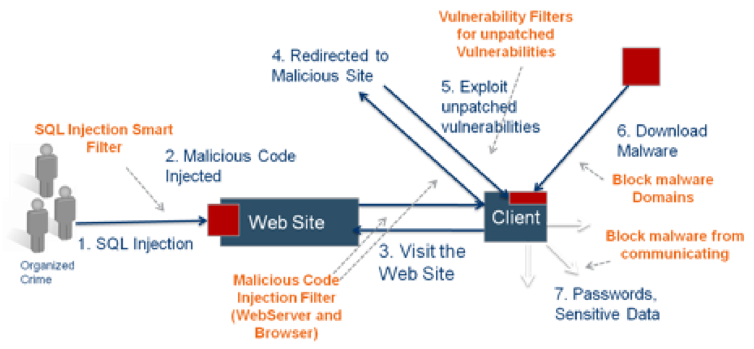
- An attacker can use SQL injection to bypass authentication or even impersonate specific users.
- One of SQL's primary functions is to select data based on a query and output the result of that query. An SQL injection vulnerability could allow the complete disclosure of data residing on a database server.
- Since web adding applications use SQL to alter data within a database, an attacker could use SQL injection to alter data stored in database. Altering data affects data integrity and could cause repudiation issues, for instance such as voiding transactions, altering balances and other records.

SQL is used to delete records from a database. An attacker could use an SQL injection vulnerability to delete data from database. Even if an appropriate backup strategy is employed, deletion of data could affect an application's availability until the database is restored.

Some database servers are configured (intentional or otherwise) to allow arbitrary execution of operating system commands on the database server. Given the right conditions, an attacker could use SQL injection as the initial vector in an attack of an internal network that sits behind a firewall. Defense against this type of attacks will be covered in PART II-PROTECTION.

III. Protection

Protecting your site against SQL injection is like most security related topics –no one solution solves everything. Instead it is a series of small walls that will together form a formidable – but perhaps not impenetrable – wall that will at least deter novices or those attackers just 'passing by' looking for dangling apples.
The following diagram shows how to protect against it.


Vulnerability Filters for unpatched Vulnerabilities
4. Redirected to Malicious Site
5. Exploit unpatched vulnerabilities
6. Download Malware
SQL Injection Smart Filter
2. Malicious Code Injected
Block malware Domains
Web Site
Client
1. SQL Injection
Block malware from communicating
Organized Crime
3. Visit the Web Site
Malicious Code Injection Filter (WebServer and Browser)
7. Passwords, Sensitive Data

*Install filters for known vulnerabilities in Browsers, Operating Systems and ActiveX Plugins
* Install filters which prevent the user from accessing sites serving malicious pages. In this case, we released a specific protection which detects if the user visits a site that has malicious javascript in it.
* Install filters which block domains which download the malware on the target machine.
* Install filters detecting existence of known malware on the machine.

The following small walls are required at a minimum – some a skilful attacker will be able to climb over or around, but hopefully they will not bypass all.

a) Use original and difficult to guess table and column names
b) Use aliases
c) Set length limits on any form fields on your site and don’t use real column names
d) Validate all your data on the server side at a minimum for content, length and format
e) If you are using a product where the source code is available – for example a message board – keep up-to-date on patches and advisories about vulnerabilities.
f) In relation to (e) try to make your schema unique – if it allows you to generate the table names, be sure to differentiate your site from the default setting
g) Where possible use stored procedures
h) Audit your code

i) Lockdown your server

In addition the developer can also avoid SQL injection attacks by following-

i) Secure Coding practices –Secure coding practices are the preferred method to avoid SQL injection attacks.
ii) Security development Lifecycle – Organizations must implement security in their software developments process. The paradigm that exists today does not incorporate these practices. This has lead to countless security flaws.
iii) Web application scanning – Web application scanning is a great way for organizations to assess their web applications. Scanning should be performed on production applications and incorporated in to the software development process. There are several open source and commercial scanners available.
iv) Web application firewall – according to WASC, ‘Web application firewalls (WAF) are a new breed of information security technology designed to protect web sites from attck.WAF are capable of preventing attacks that network firewalls and intrusion detection systems can’t, and they do not require modification of application source code” (WASC)

IV. Conclusion

SQL injection is a threat to any system connected to the internet with a database backend. An intelligent attacker will find any holes in your code and/or your server security and leverage them to their maximum advantage.

An SQL injection is relatively easy to guard against, and should be a consideration of any project before a line of code is written, or a network card plugged it. Numerous papers show how to launch a comprehensive SQL injection attack. Not checking your code and system set-up at the outset is tantamount to waiting for an attacker to do it for you.

With the advent of SQL Injection, possibly the first automated script for checking upon SQL injection vulnerabilities, unprotected sites days of waiting may well be numbered. SQL injection is straightforward to protect against, and with the wealth of susceptible systems on the internet, you really need to take care to avoid your system being the lowest apple on the tree.

A. Abbreviations and Acronyms

OWASP – *Open Web Application Security Project (OWASP) – The Open Web Application Security Project (OWASP) is a worldwide free and open community focused on improving the security of application software. Our mission is to make application security "visible," so that people and organizations can make informed decisions about application security risks. See: http://www.owasp.org/*
SQL – Structure Query language
RDBMS – Relational Database Management System
OUTPUT VALIDATION – The canonicalization and validation of application output to Web browsers and to external systems.
WAF – Web application Firewall
WASC – Web application Source Code

References

SQL Injection is not a dark art. A wealth of information is available on the internet regarding how to use it and how to protect against it.

[1] "SQL Injection Are Your Web Applications Vulnerable?". SPI Dynamics.2002.http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf
[2] Www.OWASP.org/index.php/SQL_Injection_Prevention_Cheat_sheet
[3] Anley, Chris. "Advanced SQL Injection in SQL Server Applications". NGSSoftware Insight Security Research (NISR) Publication. 2002.
[4] OWASP Top Ten Project - http://www.owasp.org/index.php/Top_10
[5] OWASP Code Review Guide - http://www.owasp.org/index.php/Category:OWASP_Code_Review_Project

[6] OWASP Testing Guide - http://www.owasp.org/index.php/Testing_Guide
[7] OWASP Enterprise Security API (ESAPI) Project - http://www.owasp.org/index.php/ESAPI
[8] OWASP Legal Project - http://www.owasp.org/index.php/Category:OWASP_Legal_Project
[9] http://www.ngssoftware.com/papers/advanced_sql_injection.pdf
[10] http://www.asptoday.com/content/articles/20020225.asp 25 Feb
[11] http://www.wiretrip.net/rfp/p/doc.asp?id=42&iface=6 27 Feb. 2001.
[12] "SQL Injection/Insertion Attacks". insecure.org.
[13] http://lists.insecure.org/pen-test/2002/Jan/att-0031/01-mh-sql.txt
[14] SQL Injection Attacks and Defense(Book) - Justin Clarke
[15] M.Young,TheTechnicalWriter'sHandbook.MillValley,CA:UniversityScience,1989.