SQL Injection Attack as a Threat of Web Portals

Theodoros Tzouramanis
University of the Aegean, Greece

INTRODUCTION

SQL injection attack (CERT, 2002) is one of the most prevalent security problems faced by today's security professionals. It is today the most common technique to indirectly attack Web-powered databases and disassemble effectively the secrecy, integrity and availability of Web portals. The basic idea behind this insidious and pervasive attack is that predefined logical expressions within a pre-defined query can be altered simply by injecting operations that always result in true or false statements. With this simple technique, the attacker can run arbitrary SQL queries and thus s/he can extract sensitive customer and order information from e-commerce applications, or she/he can bypass strong security mechanisms and compromise the back-end databases and the file system of the data server. Despite these threats, a surprisingly high number of systems on the internet are totally vulnerable to this attack.

The article discusses various ways in which SQL can be "injected" into a Web portal. It presents some advanced methods of SQL injection, which can result in the compromise of the system. Techniques for the detection of SQL injection attacks are presented and some database lockdown issues related to this type of attack are discussed. The article concludes by providing secure coding practices and mechanisms that protect Web applications against unexpected data input by users; alteration to the database structure; corruption of data; and disclosure of private and confidential information that are all owed to the susceptibility of these applications to this form of attack.

BACKGROUND

Most organizations that have an "online presence" these days will be protected by some kind of software or hardware firewall solution (Theriault & Newman, 2001). The purpose of the firewall is to filter network traffic that passes into and out of the organization's network, limiting the use of the network to allowed, "legitimate" users. One of the conceptual problems with relying on a firewall for security is that the firewall operates at the level of IP addresses and network ports. Consequently, a firewall does not understand the details of higher-level protocols such as hypertext transfer protocol (HTTP), that is, the protocol that runs the Web portals.

There is a whole class of attacks that operate at the application layer and that, by definition, pass straight through firewalls. SQL injection is one of these attacks. It takes advantage of nonvalidated input vulnerabilities to pass SQL commands through a Web portal for execution by a backend database, that is, the heart of most Web applications. Attackers take advantage of the fact that programmers often chain together SQL commands with user-provided parameters, and can therefore embed SQL commands inside these parameters. Therefore, the attacker can execute malicious SQL queries on the backend database server through the Web portal.

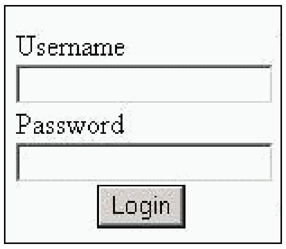
To be able to perform SQL injection hacking, all an attacker needs is a Web browser and some guess work to find important table and field names. This is why SQL injection is one of the most common application layer attacks currently being used on the Internet. The inventor of the attack is the Rain Forest Puppy, a former hacker and, today, a security advisor to international companies of software development.

SQL INJECTION ATTACK

SQL injection is a particularly insidious attack since it transcends all of the good planning that goes into a secure database setup and allows mistrusted individuals to inject code directly into the database management system (DBMS) through a vulnerable application (Litchfield, 2001). The basic idea behind this attack is that the malicious user counterfeits the data that a Web portal sends to the database aiming at the modification of the SQL query that will be executed by the DBMS (Spett, 2002). This falsification seems harmless at first glance but it is actually exceptionally vicious. One of the most worrying aspects of the problem is that successful SQL injection is very easy to perform, even if the developers of the Web portals are aware of this type of attack.

The technologies vulnerable to SQL injection attack are dynamic script languages like ASP, ASP.NET, PHP, JSP, CGI, and so forth (Anupam & Mayer, 1998). Imagine, for example, the typical user and password entry form of a Web portal that appears in Figure 1. When the user provides her/his credentials, an ASP (active server page) code similar to the one that appears in Figure 2 might undertake to produce the SQL query that will certify the user's identity.

Figure 1. A typical user authentication form in a Web portal



In practice, when the user types a combination of valid login name and password, the portal will confirm the elements by submitting a relative SQL query in some table *USERS* with two columns: the column *username* and the column *password*. The most important part of the code of Figure 2 is the line:

*sql = "select * from users where username = ' " + username + " ' and password = ' " + password + " ' ";*

The query is sent for execution into the database. The values of the variables *username* and *password* are provided by the user. For example, if the user types:

username: *george*

password: *45dc&vg3*

the SQL query that is produced is the:

*select * from USERS where username = 'george' and password = '45dc&vg3';*

which means that if this pair of *username* and *password* is stored in the table *USERS*, the authentication is successful and the user is inserted in the private area of the Web portal.

If however the malicious user types in the entry form the following unexpected values:

username: *george*

password: *anything' or '1' = '1*

then the dynamic SQL query is the:

*select * from USERS where username = 'george' and password = 'anything' or '1' = '1';*

The expression *'1'='1'* is always true for every row in the table, and a true expression connected with '*or*' to another expression will always return true. Therefore, the database returns all the tuples of the table *USERS*. Then, provided that the Web portal application received, for an answer, certain tuples, it concludes that the user's password is '*anything*' and permits his/her entry. In the worst case the Web portal application presents on the screen of the malicious user all the tuples of the table *USERS*, which is to say all the *usernames* with their *passwords*.

If the malicious user knows the whole or part of the login name of a user, the malicious user can log on as the user, without knowing the user's *password*, by entering a *username* like in the following form:

username: *' or username like 'admin%'--*

password:

The "—" sequence begins a single-line comment in Transact-SQL, so in a Microsoft SQL Server environment, everything after that point in the query will be ignored. By similar expressions the malicious user can change a user's *password*, drop the *USERS* table, create a new database: the malicious user can effectively do anything possible to express as an SQL query that the Web portal has the privilege of doing, including running arbitrary commands, creating

Figure 2. An ASP code example that manages the users' login requests in a database through a Web portal

```
username = Request.form("username");
            password = Request.form("password");
            var con = Server.CreateObject(ADODB.Connection");
            var rso = Server.CreateObject(ADODB.Recordset");
            var sql = "select * from users where username = ' " + username + " ' and
password = ' " + password + " ' ";
            rso.open(sql,con);
            if not rso.eof () then
                        responsible.while ("Welcome to the database!")
```