Systems and Network Analysis Center
Information Assurance Directorate

Minimize the Effectiveness of SQL Injection Attacks

Any application that queries a database using user-supplied data is a potential target for SQL injection attacks. The Common Weakness Enumeration website, http://capec.mitre.org/data/definitions/66.html, provides an explanation/definition of SQL injection, along with various attack examples. DBAs may not be able to stop all SQL injection attacks against their database servers; however, there are some things they and application programmers can do to minimize the effectiveness of these attacks.

What can a Database Administrator (DBA) do?

- Follow security recommendations provided in the various database server security benchmarks. Download benchmarks from www.nsa.gov/snac or www.cisecurity.org
- Do not host the database and web server on the same box.
- Block internet access to and from the database using a firewall or non-routable IP address. Once configured, packets from the database server will not be able to route to the Internet. A route will need to be added to the web server so it will be able to find the database server.
- Configure trusted IP access (e.g., IPSEC) to control which machines are able to communicate with the database server.
- Remove all sample scripts and applications from the database server.
- Use a limited privilege OS account as the database service account on the server.
- Use a dedicated, low-privileged account for each application's database connection account. Do not use sa, dba or admin.
- Do not grant users or applications direct access to database tables. Make use of application roles that have limited access to the database. If the application only needs read access, limit database access to read only.
- Create views and grant only required access to these views to the application roles. This will prevent the application from having direct access to the underlying tables in the database.
- Remove unused stored procedures from production databases.
- Grant applications access to user-created stored procedures only.
- Do not "grant" applications _ANY_ access to OS commands or system stored procedures.

***For special security environments with highly sensitive data stored in databases, an available option is to place the database server on a backnet, protecting it from access by outside networks. Do this by placing 2 NICs in your web server and have one connected to the outside network and the other connected to a small internal network (such as an administrative network) containing the database server. Prevent the web server from routing packets from the outside network NIC to the backnet NIC and use non-routable addresses (such as 192.168.x.x) in the backnet.*

What can an Application Programmer do?

A well-respected source of information on web application security, to include SQL injection issues, is the Open Web Application Security Project (OWASP). At a minimum, implement the following OWASP recommendations:

- Create custom, general error messages generated by the application. Attackers can gain valuable information, such as table and column names and data types through default error messages generated by the database during a SQL injection attack.
- Validate user input prior to forwarding it to the database. Only accept expected user input and limit input length. **Note:** Where possible, use white-list style checking on all user input. (Application Server Firewalls can be used to check input and accept only those that fit the pattern of acceptable input for the server.)
- Use web application scanning tools to discover code vulnerabilities during development.
- Isolate the web application from the SQL. Place all SQL required by the application in stored procedures on the database server.
- Use static queries. If dynamic queries are required, use prepared statements.
- Have the application execute the stored procedure using a safe interface, e.g., JDBC's CallableStatement or ADO's Command Object.

The use of user-created stored procedures and prepared statements (or parameterized queries) makes it nearly impossible for a user's input to modify SQL statements because they are compiled prior to adding the input. ***Note:*** *Make sure the stored procedures are not themselves susceptible to SQL injection by having the application sanitize all user input.*

Detecting SQL Injection Vulnerabilities and Attacks

Detecting applications that are vulnerable to SQL injection is tough because these vulnerabilities can exist in any of the application interfaces exposed to the user. Although not all SQL injection attack techniques can be easily detected, there are a few things DBAs/developers can do:

- ➢ Read the web server logs. SQL injection attacks can sometimes be easily spotted in these logs because of the larger than normal amount of entries written to the logs.
- ➢ Look for HTTP 404 and HTTP 500 error log entries, as well as other error log entries generated by programs written to check user input.
- ➢ Use a web application scanning tool. These tools can be used to alert DBAs to places in applications that are susceptible to SQL injection attacks.
- ➢ Detecting SQL injection vectors in applications prior to deployment is crucial. Information on how to go about testing for SQL injection vulnerabilities can be found on the OWASP website at http://www.owasp.org/index.php/Testing_for_SQL_Injection.

Several third-party options exist for protecting web applications and their backend databases. These include Intrusion Prevention Systems, Web Application Firewalls, and Database Gateways. More information on these protection options can be found through Internet research.