SQL Injection

More than half a million of legitimate websites worldwide have been hacked and are serving up malware in first half of 2008, including United Nations, UK Government sites and the US Department of Homeland Security. (*Ref. #1, #2*)

Visiting legitimate websites is no longer safe, let alone malicious ones. Apart from taking potential risk of getting malware infection, sensitive information leakage and theft might unnoticeably happen. It may sound exaggerative but it is the fact that all web citizens are facing these under-estimated threats. We have prepared this brief article and give you some ideas on potential catastrophic impacts of these threats and measures to mitigate these risks.

Maybe you have made great efforts in updating system patches, installing anti-virus software, building firewalls and intrusion detection systems in order to protect your information assets. However, you may have created your own vulnerabilities to get through these defense systems due to negligence in secure application coding practices.

ALL user inputs are *EVIL!!! NEVER TRUST* them.

Nowadays, lots of websites are interactive, dynamic and database-driven, which run various web applications in servers with data stored in back-end database. Web 2.0 technologies allow users to do more than just retrieve information. They can access and modify the content and distribute their information in websites such as social networking sites, wikis and blogs. In other words, they can control the database information via a web browser. However, web application flaws offer the vulnerabilities of unauthorized database control and malicious code injection which attackers can take advantage of. Attackers are trying to get valuable information held in database. This hack is a kind of application attack called SQL injection.

#1 - Hackers jack thousands of sites, including U.N. domains

http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9079961

#2 - Hackers hijack a half-million sites in latest attack

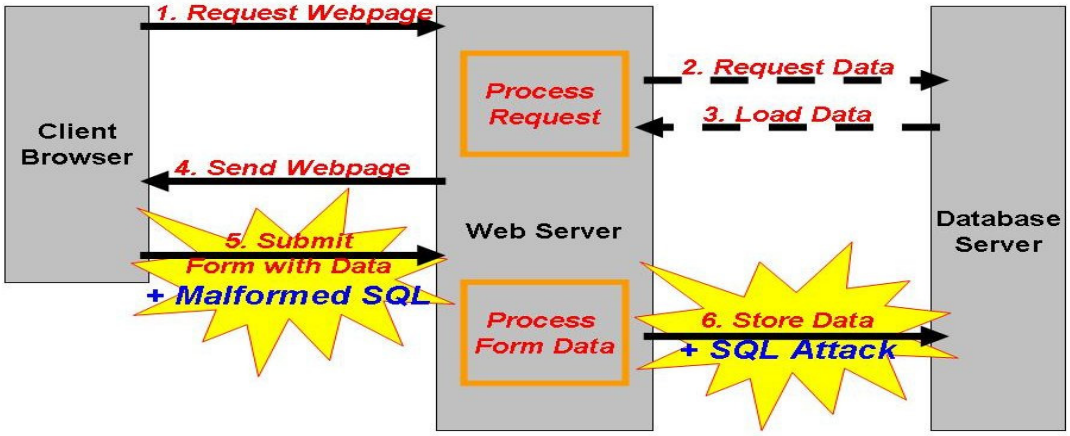
http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9084991

A. What is SQL?

SQL (Structured Query Language) is a database computer language designed for the retrieval and management of data in relational database management systems (RDBMS), database schema creation and modification, and database object access control management. It is used for querying and modifying data, accessing and manipulating database.

B. What is SQL injection and how it works?

SQL injection consists of insertion of an SQL query via the input data from the clients to the application. It works in a similar way when you submit a web form. (Steps 1-4) Text is entered into textboxes in a web form which are used to execute a query against a database. Hackers can enter a malformed SQL statement into the textbox (Step 5) and changes the nature of the query (Step 6) so that it can be used to break into, alter, or damage the back-end database.



The following is an example of a SELECT query that returns the password of a user account with particular login name for authentication procedure of a system.

Normal query

SELECT *passwd* **FROM** USERS **WHERE** *uname*=`**$username**`

e.g. Extracting the password of a user with login name `Robert` from a database table USERS, the SQL query would become

SELECT *passwd* **FROM** USERS **WHERE** *uname*=`**Robert**`

If we just make a tiny modification in the input field `$username` with malformed SQL in a way shown as follows, we will have surprising results.

Example 1:

- By crafting the parameter value, additional queries can be added and executed.

Query	SELECT *passwd* FROM USERS WHERE *uname*=`**Robert`; DROP TABLE USERS;--**`
Result	A second query [DROP TABLE USERS] is executed and remove the database table USERS. All user accounts are eliminated.

Example 2:

- By crafting the query criteria to be always logical TRUE, all records in the query table can be extracted.

Query	SELECT *passwd* FROM USERS WHERE *uname*=`` **OR 1=1**`
Result	All account information in database table USERS can be extracted.

Example 3:

- By crafting the query criteria in first query to be logical FALSE and combining the results from two queries, empty result from first query with confidential information from second query will be returned.

Query	SELECT *password* FROM USERS WHERE *uname*=`` **AND 1=0 UNION SELECT *card_holder_name*, *card_number*, *expiry_date* FROM CREDITCARD**`
Result	Sensitive information in database table CREDITCARD from second query can be extracted.

C. Possible risks

Websites and web applications are often available for public access and provide the required services to end-users round-the-clock. Firewalls and anti-virus tools usually offer little protection against web application hacking which may lead to direct access to valuable backend data such as customer databases. Most web applications are tailor-made and thus the significance of software security testing and quality assurance is sometimes overlooked. As a result, custom applications are more susceptible to attack.

A successful SQL injection exploit can read confidential data from the database, modify database data (INSERT/UPDATE/DELETE), execute administration operations on the database [such as shutdown of database management system (DBMS)], recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. It may also lead to many potential attacks in other forms.

	Possible Risks
Confidentiality	Loss of confidentiality is an obvious risk of SQL injection vulnerabilities as databases are generally used to hold sensitive data. Data leakage or theft may unnoticeably happen.
Integrity	Apart from having ability to read sensitive information, it is possible to make changes or even delete information in the database.
Availability	Data removal or loss of DBMS control (e.g. DMBS shutdown) would cause information unavailable.
Authorization	If authorization privileges are stored in database, it is possible to change this information through successful SQL attacks. As a result, unauthorized access is feasible for further attacks.
Authentication	It is possible to connect and access the system even with no previous knowledge of login credentials (e.g. passwords)

- Web Defacement

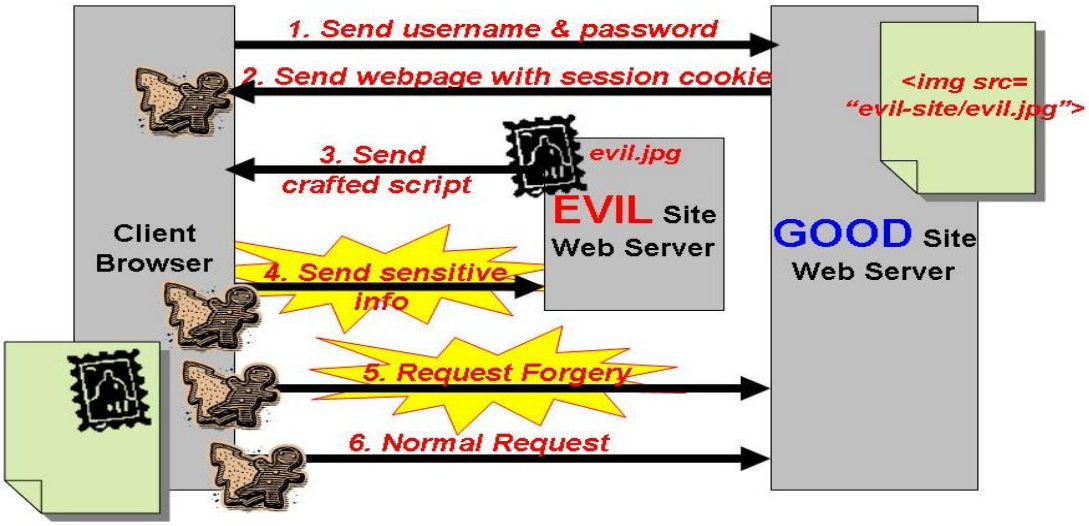
Normal content would be replaced or erased from the site entirely.

- Malicious Script / Malware Hosting

Once a hacker injected malicious script in a vulnerable website, it then loads an external script that can compromise other computers. All its visitors and online customers have potential risk of getting malware infected and becoming zombie hosts. Infected bots become the accomplice of attackers to attack other vulnerable websites (e.g. Asprox trojan) and further propagate to new hosts.

- Cross-Site Request Forgery (XSRF)

When a user accesses a site in which he is supposed to have authenticated (Step 1), authentication information are usually stored in a cookie until the cookie has expired.(Step 2) Hacker injected scripts [such as a crafted image or zero-size iframe object] in webpage will be loaded and executed.(Step 3) The attempt by browser to load the injected scripts (image/iframe) may send sensitive information [such as authentication cookie] to evil site (Step 4) or submit a crafted request with user's cookie which authorizes a transaction with approval on behalf of the user (Step 5). Such request is carried out from the client browser and thus untraceable.



- Cross-Site Script Inclusion (XSSI)

Similar to XSRF, third party scripts from malicious sites can be dynamically included. (Step 3) Apart from the leakage of browser authentication cookies, hacker could redefine the callback method [e.g. in asynchronous requests (AJAX)] with full access to client data.

D. Defense Guideline

As an END-USER, you should make your contribution in the following areas.

- DO follow HKCERT Security Guideline "Home PC Baseline Security Self-Assessment Checklist" and check your own computer health
- DO report immediately to site owners/administrators and request them to check and remove any suspicious injected code or malware hosted in the websites

As a SITE OWNER / ADMINISTRATOR, the best approach to defend against SQL injection is multi-faceted, and should include the following steps:

Prevention & Detection

Design and Development

- DO validate and sanitize ALL user inputs at the server-side
  - Identify allowable characters and white-list only valid characters
  - Allow well-defined set of safe values via regular expression (*e.g. [A-Za-z0-9]*)
  - Limit the length of each entry
  - Include appropriate data sanitization routines in all application components and auxiliary services
- DO utilize parameterized statements with bind variables
  - Use strongly typed parameterized queries which separate command from data by substitution of input parameters with bind variable
    - *Java EE - use strongly typed PreparedStatement, or ORMs such as Hibernate or Spring*
    - *.NET - use strongly typed parameterized queries, such as SqlCommand with SqlParameter or an ORM like Hibernate.*
    - *PHP - use PDO with strongly typed parameterized queries (using bindParam())*
- AVOID dynamic SQL
  - SQL string concatenation with user-entered values may pose threat in creating malformed queries through a web application.
- AVOID displaying detailed error message
  - Exception handling should always offers minimal information which may offer the details to attackers to diagnose and refine hacking attempts
- DO code scanning
  - Utilize source code scanning tools to look for SQL injection flaws and then fix the vulnerabilities
    - *MS Source Code Analyzer for SQL Injection [http://support.microsoft.com/?kbid=954476]*

Implementation

- DO execute with least DB user privilege
  - Create a new login/user specifically for each application and deny access to all objects that are unnecessary to be used by the applications
  - *AVOID using "root" or "dbo" accounts to access the database*
  - *AVOID information leakage of database connection string (e.g. password)*

Assessment

- DO runtime vulnerability scanning
  - Utilize scanning tools to look for SQL injection vulnerabilities on a running website.
    - *Scrawler [https://download.spidynamics.com/products/scrawlr/]*

Production and Maintenance

- DO utilize application firewall and filtering tools
  - Use application firewall and filtering solution to block unwanted requests.
  - This should only be used as a proactive measure or as emergency fix (short term) for SQL injection vulnerabilities. The best defense should be user-input validation and source code scanning.
    - *mod-security of Apache [http://sourceforge.net/projects/mod-security/]*
    - *URLScan [http://www.microsoft.com/technet/security/tools/urlscan.mspx]*
    - *WebKnight [http://www.aqtronix.com/webknight]*
- DO check and monitor web application for errors
  - Check suspicious access / errors in web server log
  - Evaluate errors which are generated by the database system

Containment

- DO prohibit further access to database if applicable
- DO prohibit access to compromised web and database servers if applicable
- DO remove all injected code immediately

Eradication

- DO code scanning and vulnerability scanning to identify possible flaws
- DO fix code flaws and apply all necessary patches
- DO correct system misconfiguration
- DO utilize application firewall and filter tools
- DO change all user credentials (e.g. password) and verify access privileges of databases in DBMS immediately
- DO test thoroughly before system is restored to normal operation
- *Please see "Prevention and Detection" for steps and tools*

Recovery

- DO restore database from a clean backup copy
- DO pre-production security assessment

Follow-up

- DO regular review, otherwise you will get hit again