**SQL Injection Primer**

**Abstract:** SQL injection is one of the issues making the Open Web Application Security Project (OWASP) Ten Most Critical Web Application Security Risks for 2010 [1]. While there are hundreds of vulnerabilities on the internet, injection, more specifically SQL injection for the purpose of this paper, has taken the lead over Cross Site Scripting (number one in 2007), Cross Site Forgery, Malicious File Execution, and Information Leakage. SQL injection has become an enormous problem today as developers rushed to field web based applications with online databases to support the growing demand for online sales and rapid access to information, making web sites an easy target for hackers to steal critical information with very little knowledge or effort. The purpose of this project is to explore SQL injection, identify how this problem came into existence, define the types of database and web applications that are vulnerable, identify the methods and tools used to exploit SQL injection vulnerabilities and to provide some guidance on how to protect yourself from becoming a victim of this ever growing problem.

# Overview of SQL Injection

A Structured Query Language (SQL) injection is a vulnerability in which an adversary exploits an SQL processor. The SQL processor is used to execute injected commands the owner did not intend, usually leaking or releasing unintended information.

## Availability

SQL vulnerabilities can be made available to an adversary through many mechanisms.

A web site may ask for information such as name and e-mail address. A merchandising company may allow the public to search for an article to buy. A library may allow patrons to search for a book title. These are examples of public access to an SQL service.

Examples of insider or employee based access are just as numerous. A company may have an on-line phone number lookup. If this SQL service is hosted on a Human Resources computer, a breach of the phone number search application could release employee records. A Purchase Request application hosted by the Purchasing and Accounting department could give access to employee payroll information.

A basic premise of Information Assurance is to trust no one. Generally, any input that can be entered to a computer is an avenue for attack. Once an adversary has access to the host system, the entire system and its records are at the command of the adversary.

## SQL and Databases

SQL uses key words and phrases to interact with a relational data base. Relational databases link together data tables. See .



Figure 1. Basic Database Relationships

Records for people include a person's full name, and address. A person could have a home address, a business address, and a different mailing address such as a box office. This represents a one-to-many relationship because one person is linked to many addresses. In another direction, several different addresses can include the same state name. This represents a many-to-one relationship because many addresses are linked with one state name.

Table relationships can be more complicated such as in Figure 2.



Figure 2. More Involved Data Relationships.

To create a simple mailing list, you can use a simple Select SQL statement or query.

SELECT FirstName, LastName, Address, City, StateName, ZipCode

FROM <*table linkage description*>;

This SQL statement would retrieve data such as shown in Figure 3.

Figure 3. Simple Mailing List

Another SQL query would retrieve payroll data such as the following.

SELECT FirstName, LastName, Pay, SSnumber

FROM <*table linkage description*>;

This will produce output such as in Figure 4.

Figure 4. Simple Pay List

A Union query is a more complicated SQL statement that joins the data of two separate queries such as the following.

SELECT FirstName, LastName, Address, City, StateName, ZipCode FROM Simple\_mailing\_list

UNION SELECT FirstName, LastName, Pay, SSnumber, 0, 1 FROM Simple\_pay\_list;

This particular Union query doesn't make sense in general, but it is something that an adversary might inject to a vulnerable web site. This Union query will produce the output shown in Figure 5.

Figure 5. Simple Union Query.

Note that the first Select statement has six field entries. The first time we saw the second query, it only had four field entries. To Union the second Select statement with the first, the number of fields must match and the data types of the fields must be compatible. In this case, the numbers 0 and 1 are taken as text data types and the Pay number-type is printed as text.

## SQL Demonstrator

An SQL based database is available on this project's web site. It can be used to investigate SQL injection. This database form shows the actual SQL statement used to retrieve the data.

Enter a President's name to search for, such as Johnson. Figure 6 shows the result.

Figure 6. Search for Johnson

Exploit the database by entering the following:

\*' UNION SELECT FirstName, LastName, Pay, SSnumber, 0, 1 FROM Simple\_pay\_list

Figure 7 shows the result.

Figure 7. Union Exploit

Select the Parameter SQL type with the same input and Figure 8 is the result.

Figure 8. Parameter SQL

Normally, specific error messages should not be passed to the user.

Note that this database form is extremely vulnerable to exploit. By entering an SQL statement in the SQL window, and clicking the Execute button, the input script will be executed.

# Elaboration

## Google Hacking

Google Hacking is a technique that uses the complex and powerful search engine provide by Google to identify information or web sites that may be vulnerable due to poor security practices. These vulnerabilities may include hidden fields, password files, log files containing usernames and passwords, error logs, administrative pages, hidden pages, and pages that use PHP, ASP or GCI scripts. This type of hacking is one method a hacker will use to gather information to identify a soft spot or weakness that they could exploit. With a few simple advanced Google commands a hacker can narrow his search to a few sites and within minutes start probing for vulnerabilities. A hacker uses advanced operators within a query string to refine the results of a Google search. The general format is operator:string. Below is a list of the most commonly used operators;

 intitle: - restricts the search to text in the title of the page

 Ex. intitle: SQL

 allintitle: - similar to intitle operator, allows concatenation of key words in title search

 Ex. allintitle: SQL Password (is the same as intitle: SQL intitle: Password)

 inurl:, allinurl: - will search for keywords in the URL

 Ex. inurl: login.aspx

 site: - will narrow the search to a specific site or domain like uccs.edu or .gov respectively

 Ex. site:.uccs.edu

 filetype: - used to search for a specific file like doc, php, cgi, or aspx

 Ex. filetype:aspx (do not use dot operator to identify the file type, like .doc)

 intext: - will identify keywords in the text of the webpage

 Ex. intext: SQL Injection

While these commands are quite simple, combining them into a more complex search string provides an inexperienced hacker with a wealth of information. For example combining the commands above into a search string "inurl:phpMyAdmin filetype:php site:.edu" provided results of about 1,840,000 hits, the first hit (see right) provided access to a PHP administration page that could potentially be exploited. This search string simply looks for information related to php administration using the php file extension in the .edu domain. A more refined search could easily provide access to administration pages for other sites providing financial services, online shopping, or access to organizational databases containing a wealth of information.

Google hacking is such a powerful tool a book "Google Hacking for Penetration Testers" by Johnny Long, is dedicated to the subject. Johnny Long also runs a web site known as the Google Hacking Database (GHDB), a site that provides a wealth of information on vulnerabilities that can be identified by the techniques discussed above. This site can be found at the following link: **http://www.hackersforcharity.org/ghdb/.**

For example the GHDB identified information on searching for Advisories, Error Messages, Files that typically contain Usernames, Login Portals, and Online Shopping Information to name a few. After reviewing the site a simple Google search for "filetype:mysql\_history" provides information of the commands performed against a mySQL database. This search provided a link to the following information;

 **update user set Password= password('myburden1982') where User= 'root';**

Google also offers another great service, know as Google Code Search, where one can search public source code, **http://www.google.com/codesearch** . A search for the keyword "SELECT" yielded 12 million hits with the ability to refine this search to a specific language such as javascript, perl, c, c++, and php. A smart or even not so smart hacker with access to vulnerable source code, using Google and some basic Google Hacking techniques could potentially identify sites that are using that vulnerable source code creating a vulnerability that could easily be exploited.

# Detection

SQL injection is notoriously difficult to detect. The most common detection mechanisms include application layer firewalls and intrusion detection systems. These systems maintain a list of signatures for known SQL injection attacks and compare live traffic to this list of patterns. If a match is found then a number of actions can be taken which include sending an alert (via email, etc) and/or blocking the traffic altogether.

An application layer firewall acts as a proxy between the network and a specific application, such as a web program. It inspects each packet and can be configured to either actively reject packets which fail inspection, or just drop them (known as passive mode). While still a fairly new approach, there are a number of commercially available application firewalls, as either hardware appliances or software plugins. This can be especially valuable as it is often easier and cheaper to update firewall rules than to update application program code.

An Intrusion Detection System (IDS) is similar to an application layer firewall in that it scans traffic, but it throws a much wider net and can be an excellent complement. The three most common types of IDS are: network-based, systems, and host-based. A network-based IDS typically exists at a network’s access point, such as just inside the firewall. It captures and examines all packets and compares them to a list of known-malicious signatures. A systems IDS is usually installed on a front-end server and monitors specific protocols for dangerous or uncommon traffic. A host-based IDS is installed directly on the host it is protecting and monitors all activity on that host. An intrusion detection system is either passive (just logging activities and sending alerts) or reactive. A reactive IDS may actually reset the connections and/or reconfigure the firewall, and is technically known as an Intrusion Detection Prevention System (IDPS) or even just Intrusion Prevention System (IPS). This leads to our next focus: how to best prevent an SQL injection.

# Prevention

While many argue that it is not possible to eliminate all risk of SQL injection, there are many options to mitigate the risk. As previously discussed, regularly audited and updated application layer firewalls and intrusion detection / prevention systems can go a long way by rejecting traffic which matches known SQL injection attacks. Other prevention methods focus on the web applications, code, and back-end system design.

There are two basic types of SQL queries: dynamic and parameterized. A dynamic SQL query injects the user input directly into the query that is sent onto the database. This is very dangerous as the input is usually not validated and thus the system may execute any number of unintended commands. A parameterized query binds the user input to pre-defined variables and then those variables are used in the SQL query generated by the web application. This can screen out many unintended queries.

Another preventative step involves using stored procedures. A stored procedure is a query with variables that is stored in the database. The user input is bound to those variables, and the query is executed from within the database. This is a different approach than parameterized statements, but achieves the same thing: a defined and approved query that is not easily altered by an outsider.

A third prevention option is to validate the user input using escape characters. This simple approach within the web application escapes all user input so that any special characters which may have been inserted are regarded as data by the database, instead of executable code.

A fourth defense involves account privileges within the database. In order for the web application to query and store data in the database it must be able to authenticate. This is usually accomplished with a user account. The web application should have its own account (not one used by any other user, and especially not the system administrator’s account) to authenticate. This account should be restricted to only the information it absolutely needs access to. In addition, there could be multiple accounts, such as one for reading data and one for writing data. And what data can be read or written can be limited to individual tables or even just a pre-defined view (only certain columns or rows) of a table.

Another method for preventing SQL injection is to use a whitelist when validating the user input. As discussed previously, many systems keep a list of signatures (patterns) for recognized SQL injection. These systems then compare queries to the signatures blacklist and passes the query onto the database only if it does not find a match. This may block many SQL injections but is far from fool-proof. So, a remedy is to maintain a whitelist of signatures for approved, safe queries, and only allow queries which match a signature on the whitelist.

There is no one perfect way to stop SQL injection, but by using a combination of preventative methods the risk can be greatly reduced.

# Wrap-up

SQL Injection is becoming more and more prevalent and is climbing lists, such as OWASP’s, as one of the most dangerous vulnerabilities. Most web applications now employ backend databases to store and retrieve active content; static content delivery is no longer the norm. SQL injection is a malicious method for manipulating these web apps to compromise the backend databases.

Due to the complexity inherent to SQL injection it may not be possible to absolutely defend against all possible attacks. However, the risk of the attacks can be greatly reduced by employing a number of defensive methodologies. These include maintaining firewalls and intrusion detection / prevention systems, managing access through parameterized queries and stored procedures and user input validation, and utilizing restricted accounts and whitelists.

Security is an ever-growing and ever-present concern, and while much research has been done into the existence of SQL injection, there is a large void in the detection and prevention for this vulnerability. As it gets more publicity we feel that more white papers will tackle this important area of research.

# References

1. “SQL Injection,” The Open Web Application Security Project, 2009. [Online]. Available: http://www.owasp.org/index.php/SQL\_Injection [Accessed: Dec. 6, 2009].
2. “SQL Injection,” in *SQL Server 2008 Books Online*. Microsoft: MSDN, 2009. [Online]. Available: SQL Server Developer Center, http://msdn.microsoft.com/en-us/library/ms161953.aspx [Accessed: Dec. 6, 2009].
3. S. Friedl, “SQL Injection Attacks by Example,” Oct. 2007. [Online]. Available: http://unixwiz.net/techtips/sql-injection.html [Accessed: Dec. 6, 2009].
4. SecuriTeam, “SQL Injection Walkthrough,” *Beyond Security*, 2008. [Online]. Available: http://www.securiteam.com/securityreviews/5DP0N1P76E.html [Accessed: Dec. 6, 2009].
5. D. Litchfield, “SQL Injection and Data Mining through Inference,” *NGS Consulting*, 2004. [PowerPoint]. Available: http://www.databasesecurity.com/webapps/sql.ppt [Accessed: Dec. 6, 2009].
6. K.K. Mookhey, N. Burghate, “Detection of SQL Injection and Cross-site Scripting Attacks,” *SecurityFocus*, 2004. [Online]. Available: http://www.securityfocus.com/infocus/1768 [Accessed: Dec. 6, 2009].
7. M. Cobb, “SQL Injection Detection Tools and Prevention Strategies,” *Search Security*, 2009. [Online]. Available: http://searchsecurity.techtarget.co.uk/tip/0,289483,sid180\_gci1375425,00.html [Accessed: Dec. 6, 2009].