Sql Injection

SQL

The acronym SQL (pronounced *sequel*) stands for Structured Query, a language for specifying database queries.

Attacks that use SQL target websites or web applications that are powered by a back-end database.

The attack relies on the strategic insertion of malicious code or statements into existing queries with the intention of viewing or manipulating data that is stored in the tables within the database.

Introducing SQL Injection

The attack works by inserting code into an existing line of code prior to its being executed by a database. If SQL injection is successful, attackers can cause their own code to run.

In the real world this attack has proven dangerous because many developers are either not aware of the threat or don’t understand its seriousness and in some cases don’t even know how to defend against it.

Developers should be aware of the following:

- SQL injection is typically a result of flaws in the web application not an issue with the database.
- SQL injection is at the source of many of the high-level or well-known attacks.
- The goal of attacks of this type is to submit commands through a web application to a

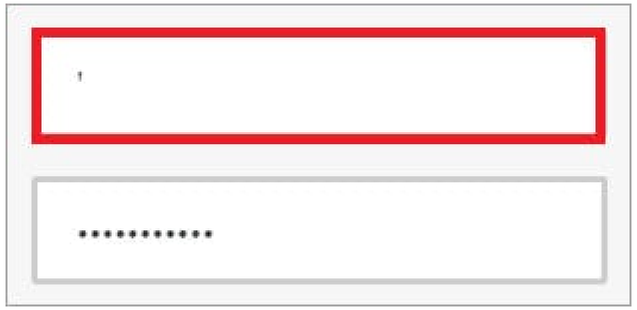
  Database.
- The usual cause of this type of flaw is improper or absent input validation, thus allowing code to pass unimpeded to the database without being verified.

From the attacker’s side

vulnerability to SQL injections is very easy to detect. Visiting a suspect site and getting it to generate error messages can indicate a potential vulnerability to this type of attack.

Example

<u>For Example</u>, Let's test if an appropriate login window is vulnerable for SQL Injection. **For this purpose, in the email address or password field, we just type sign as shown below.**



Internal Server Error

if such input returns result like error message ‘Internal Server Error‘ or any other listed inappropriate result, then we can almost be sure, that this attack is possible for that field.

Cont.

Other results, that can notify possible attack include:

- Blank page loaded.
- No error–and page do not react to the input.
- Success message for malicious code.

Example 2:

A typical log in process would include some one to enter username and password.

If a user inters a name followed by (--) *double hyphens* which in some databases means that the following code is considered a comment and not executed.

if the database executes a query like the following for someone logging with user_name John:

```
SELECT * FROM Users WHERE User_Name = 'John' AND Password =
'Smith’;
```

And the hacker inserts in the user_name field the following text (John’–)

The query then becomes :

```
SELECT * FROM Users WHERE User_Name = 'John'-- AND Password =
'Smith’;
```

If there were any user with the username of John in the Users table, the application could allow the tester to log in as the user John.

Results of SQL Injection

- Identity spoofing through manipulating databases to insert bogus or misleading information.
- Alteration of prices in e-commerce applications.
- Alteration of data or outright replacement of data in existing databases with information created by the attacker.
- Escalation of privileges.
- Denial of service, performed by flooding the server with requests designed to overwhelm the system.
- Data extraction and disclosure of all data on the system.
- Destruction or corruption of data through rewriting, altering, or other means.
- Eliminating or altering transactions that have been or will be committed.

Locating Databases on the Network

tool that is effective at locating rogue or unknown database installations is SQLPing 3.0, as described on the vendor's website; see http://www.vulnerabilityassessment.co.uk/

- SQLPing 3.0 performs both active and passive scans of your network in order to identify all the SQL Server/MSDE installations in your enterprise.

SQLRecon is very similar to SQLPing, but it provides additional techniques to discover SQL Server installations that may be hidden (http://www.vulnerabilityassessment.co.uk/):

Running a scan with either of these tools will give you information about where you may have SQL Server installations that you are unaware of.

Anatomy of a SQL Injection Attack

Acquiring a Target for Attack:

Google hacking is the use of advanced search query commands to uncover better results. Through a little trial and effort, you can find a website that is vulnerable to an attack. There are numerous search queries you can use, but some of the ones that can yield results include the following:

inurl:index.php?id=

inurl:trainers.php?id=

inurl:buy.php?category=

inurl:article.php?ID=

inurl:pageid=

inurl:games.php?id=

inurl:page.php?file=

Cont.

Once you've identified your target, your next step is to look for vulnerabilities. One easy way to determine if a site is vulnerable to SQL injection is to add a single quote to the end

of the URL like so:

http://www.somesite.com/default.php?id=1' //variable to store GET page id is the number of the page content and it is likely in a database and pulled out

For ex: http://ufl.facebook.com/profile.php?id=1234567

Type this URL and press Enter, and then observe the results. If an error is returned, the web application or site located at the URL is vulnerable to SQL injection, though you don't know to what degree.

As a general rule, if the website returns any SQL errors, it may be vulnerable to SQL injection techniques.

Initiating an Attack

One of the first steps you can take to uncover information about a vulnerable site is to learn the structure of the database. To do this you can append a simple order by statement to the URL like so:

http://www.somesite.com/default.php?id=1 order by 1

If this code returns any result other than an error, then increment the number after the order by statement by 1 (or some other amount if desired) until an error is returned.

When an error is encountered, it indicates that the last entry that *did not return an error* is the number of columns in the database.

Altering Data with a SQL Injection Attack

Another way to alter data using SQL injection involves using the forms that appear on many websites. Forms that collect login or other information can be vulnerable to attack depending on their design and any flaws that are present.

Any form that solicits data and is somehow connected to a database of any type may be vulnerable to SQL injection.

Cont.

Let’s consider a form that simply requires the user to enter an email address and then click OK. Once this is done, the application searches the database for the provided email address. It then sends an email to that address.

you input an email address into the form but with a single quote appended to it, like so:

link@hyrule.com’

If the application does not have protection in place and accepts the input without sanitizing it from malicious special characters and proceeds to execute it.

```
SELECT data

FROM table

WHERE Emailinput = 'link@hyrule.com'';
```

Cont.

When the application executes the SQL code shown here, an error message should appear. The content and context of this error message are vital in determining the next step in the process.

If the application is designed well and is validating input and sanitizing it, you probably will not see any type of message in return. However, if the application is not performing any sort of cleanup or sanitization on input, then an error message may result. The presence of these errors indicates that there may be enough of a flaw present to exploit in some manner.

```
SELECT data
FROM table
WHERE Emailinput = 'Y';
UPDATE table
SET email = 'farore@hyrule.com'
WHERE email = 'din@hyrule.com';
```

Note

Many of the common database products such as Microsoft’s SQL Server and Oracle’s Siebel allow several SQL statements separated by semicolons to be executed at once.

;

Injecting Blind

What if the target you are trying to penetrate does not return messages no matter what actions you take?

In this situation you are flying blind, so it makes sense to attempt a *blind SQL injection*. This type of attack is not dependent on the presence of error messages.

Much like any other SQL injection, a blind SQL injection can be used to manipulate information, destroy information, or extract data.

Example:

Trying this code to test the existence of table users:

```
:; IF EXISTS(SELECT * FROM users) WAITFOR DELAY '0 :0 :10 '-
```

This code first checks whether the database users exists.

If it doesn’t, the code displays, “We are unable to process your request. Please try back later.” If the database does exist, it will pause for 10 seconds. After 10 seconds, it displays, “We are unable to process your request. Please try back later.”

Here as the system doesn’t return message by itself, you try to send queries that will force messages out of the database.