SQL Injection

Introduction

SQL injection is a very serious code defect that can lead to machine compromises, the disclosure of sensitive data, and even spreading of malicious software. Systems affected by the vulnerabilities are often e-commerce applications or applications handling sensitive data or personally identifiable information (PII) or line-of-business database-driven applications having SQL injection bugs.If the applications that communicate with databases has one or more SQL injection vulnerabilities, then all data in the database is at risk.An attacker does not need to assume the *sysadmin* role to steal data.

In some countries, states, and industries, you may be legally liable should this occur. Eg: in the state of California, the Online Privacy Protection Act could land you in legal trouble if your databases are compromised and they contain private or personal data. The damage from a SQL injection attack is not limited to the data in the database; an attack could lead to server, and potentially network, compromise also. For an attacker, a compromised backend database is simply a stepping stone to bigger and better things.

AFFECTED LANGUAGES

Any programming language used to interface with a database can be affected.

But mainly high-level languages such as Perl, Python, Ruby, Java,C#,VB.NET server page technologies such as ASP, ASP.NET, JSP, and PHP, are vulnerable.
Sometimes lower-level languages, such as C and C++ using database libraries or classes can be compromised.
Even the SQL language itself can be sinful.

SIN EXPLAINED

An attacker provides your database application with some malformed data, and your application uses that data to build a SQL statement using string concatenation. This allows the attacker to change the semantics of the SQL

query.Programmer use string concatenation because they dont know there is another, safer method, and string concatenation is easy but wrong.

A less common variant is SQL stored procedures that take a parameter and simply execute the argument or perform the string concatenation with the argument and then execute the result.

Examples

Sinful C#

```
using System.Data.SqlClient;
public static void main(String args[])
{
int id = 0;
string result="";
try {
Connect to database;
Open connection;
id = Get id value from user;
string sqlstring="SELECT custName,Age" +
" FROM cust WHERE id=" + Id;

result = Execute query(sqlstring);

Close connection;

}
catch (exception e)
{
}
}
```

In these examples, the attacker completely controls the Id variable in the querystring, and because he can determine exactly what the querystring is, the results are potentially catastrophic.The classic attack is to simply change the SQL query by adding more clauses to the query and comment out unneeded clauses

For example, if the attacker controls Id, he could provide

```
 1 or 2>1 --
```

, which would create a SQL query like this:

SELECT ccnum FROM cust WHERE id=1 or 2>1 –

Here 2>1 is true for all rows in the table, so the query returns all rows in the cust table. The comment operator

```
(-- )
```

comments out any characters added to the query by the code. Some databases use

```
(-- )
```

, and others use #

Sinful SQL

```
CREATE PROCEDURE dbo.doQuery(@query nchar(128))
AS
exec(@query)
RETURN
```

This stored procedure simply takes a string as a parameter and executes it. Here also the attacker can modify the query to retrieve or manipulate any data

Related Sins

- Connecting using a high-privilege account
- Embedding a password in the code
- Giving the attacker too much error information
- Canonicalization issues

SPOTTING THE SIN PATTERN

DURING CODE REVIEW

When reviewing code for SQL injection attacks, look for code that queries a database in the first place. Scan code looking for the constructs that load the database access code. For example: in C# search for Sql, SqlClient, OracleClient, SqlDataAdapter

Determine where the queries are performed and determine the trustworthiness of the data used in each query. It can be done by looking for all the places where SQL statements are executed, and determine if string concatenation or replacement is used on untrusted data

TESTING TECHNIQUES

In case you do not have access to the code,supplement the code review with testing.

For that, determine all the entry points into the application used to create SQL queries. Next, create a client test harness that sends partially malformed data to those end points. For example, if the code is a web application and it builds a query from one or more form entries, you should inject random SQL reserved symbols and words into each form entry.

Third-party tools are also available, such as IBM Rational AppScan from IBM (was Sanctum, then Watchfire), WebInspect from HP (was SPI Dynamics), and ScanDo from Kavado.

REDEMPTION STEPS

- Do not use string concatenation or string replacement.Never trust input to SQL statements

  Always validate the data being used in the SQL statement as correctly formed. The simplest way is to use a regular expression to parse the input, assuming you are using a relatively high-level language.

- Use prepared or parameterized SQL statements, also known as prepared statements.

  Use prepared, also called parameterized, queries. Some technologies refer to them as placeholders or binding

```
Eg: SqlConnection sqlConn = new SqlConnection(GetConnnection);
string str = "sp_GetCreditCard";
cmd = new SqlCommand(str, sqlConn);
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add("@ID", Id);
cmd.Connection.Open();
SqlDataReader read = myCommand.ExecuteReader();
ccnum = read.GetString(0);
```

- Deny access to underlying database objects such as tables, and grant access only to stored procedures and views
- Encrypt the underlying data such that it cannot be disclosed in the case of a SQL injectionâĂŞinduced breach.
- Use URLScan