

SQL 盲注攻击技术综述

coolswallow of Javaphile (coolswallow@shaolin.org.cn)

Blind SQL Injection Techniques: A Survey

Abstract: This paper gives a survey of current Blind SQL Injection Techniques. It first introduces the definition of SQL Injection and its risk, and reviews several solutions proposed to solve the problem and their each disadvantages. Then, the paper discusses that if detailed error messages are suppressed, how to identify SQL Injections based on minimal reaction of the server, and how to identify SQL Injection vulnerable parameters, to generate valid injection syntax and to build the required exploit. At last, an injection of UNION SELECT statements is described in detail, including how to count the columns and how to identify columns types. Although the provided examples in the paper refer to Microsoft SQL Server and Oracle only, the same techniques can be applied to other Databases as well. By the paper, we hope to make it clear that application level vulnerabilities must be handled by application level solutions, and that relying on suppressed error messages for protection from SQL Injection is eventually useless.

Key words: SQL Injection; Blind Injection; Database Attack; Web Application Security

摘要: 本文对目前 SQL 注入攻击中使用的盲注技术进行了综述。本文首先介绍了普通 SQL 注入技术的定义和危害,回顾了儿种已被提出的针对 SQL 注入的防御手段及其各自的缺点,然后讨论了在错误信息被屏蔽或掩饰的情况下探测 SQL 注入漏洞是否存在所需的服务器最小响应,以及如何确定注入点和确定正确的注入句法并构造利用代码。最后本文还以 UNION SELECT 语句为例,详细介绍了利用该语句在盲注条件下统计数据表的列数和判断列的数据类型的方法和步骤。本文中给出的例子都是针对 Microsoft SQL Server 和 Oracle 的,但同样的技术也可以被应用到其他数据库系统。本文的目的是明确应用程序级别的漏洞只能通过应用程序级别的方案才能解决,仅仅依靠屏蔽错误信息来回避 SQL 注入攻击是无益的。

关键词: SQL 注入; 盲注; 数据库攻击; 网络应用程序安全

1 简介

1.1 普通SQL注入技术概述

目前没有对SQL注入技术的标准定义,微软中国技术中心从2个方面进行了描述^[1]:

- (1) 脚本注入式的攻击
- (2) 恶意用户输入用来影响被执行的 SQL 脚本

根据Chris Anley的定义^[2],当一个攻击者通过在查询语句中插入一系列的SQL语句来将数据写入到应用程序中,这种方法就可以定义成SQL注入。Stephen Kost^[3]给出了这种攻击形式的另一个特征,“从一个数据库获得未经授权的和直接检索”,SQL注入攻击就其本质而言,它利用的工具是SQL的语法,针对的是应用程序开发者编程过程中的漏洞,“当攻击者能够操作数据,往应用程序中插入一些SQL语句时,SQL注入攻击就发生了”。实际上,SQL注入是存在于常见的多连接的应用程序中一种漏洞,攻击者通过在应用程序中预先定义好的查询语句结尾加上额外的SQL语句元素,欺骗数据库服务器执行非授权的任意查询。这类应用程序一般是网络应用程序(Web Application),它允许用户输入查询条件,并将查询条件嵌入SQL请求语句

中，发送到与该应用程序相关联的数据库服务器中去执行。通过构造一些畸形的输入，攻击者能够操作这种请求语句去获取预先未知的结果。

在风险方面，SQL 注入攻击是位居前列的，与缓冲区溢出等漏洞基本相当。而且如果要实施缓冲区溢出攻击，攻击者必须首先能绕过站点的防火墙；而对于 SQL 注入攻击，由于防火墙为了使用户能访问网络应用程序，必须允许从 Internet 到 Web 服务器的正向连接，因此一旦网络应用程序有注入漏洞，攻击者就可以直接访问数据库进而甚至能够获得数据库所在的服务器的访问权，因此在某些情况下，SQL 注入攻击的风险要高于所有其他漏洞。

SQL 注入攻击利用的是 SQL 语法，这使得这种攻击具有广泛性。理论上说，对于所有基于 SQL 语言标准的数据库软件包括 SQL Server, Oracle, MySQL, DB2, Informix 等以及与之连接的网络应用程序包括 Active/Java Server Pages, Cold Fusion Management, PHP 或 Perl 等都是有效的。当然各种软件有自身的特点，实际的攻击代码可能不尽相同。SQL 注入攻击的原理相对简单，且各类基于数据库系统的应用程序被广泛使用，介绍注入漏洞和利用方法的公开出版物也大量问世，造成近年 SQL 注入攻击的数量一直增长，注入攻击的形式也有被滥用的趋势。

关于针对 MS SQL Server 的普通 SQL 注入技术的详细介绍，可以参考 Chris Anley 所撰的“SQL Server 应用程序中的高级 SQL 注入”^[2]一文和其后续“更多的高级 SQL 注入”^[4]，Cesar Cerrundo 所撰的“利用 SQL 注入操纵 Microsoft SQL Server”^[5]一文，以及 SPI 实验室的 Kevin Spett 撰写的白皮书“SQL 注入 - 你的网络应用程序是否会受攻击？”^[6]；而针对 Oracle 的普通 SQL 注入技术介绍，可以参考 Stephen Kost 的“针对 Oracle 开发人员的 SQL 注入攻击简介”^[3]一文。

1.2 SQL 注入攻击的防御手段

由于越来越多的攻击利用了 SQL 注入技术，也随之产生了很多试图解决注入漏洞的方案。目前被提出的方案有：

- (1) 在服务端正式处理之前对提交数据的合法性进行检查；
- (2) 封装客户端提交信息；
- (3) 替换或删除敏感字符/字符串；
- (4) 屏蔽出错信息。

方案(1)被公认为是最根本的解决方案，在确认客户端的输入合法之前，服务端拒绝进行关键性的处理操作，不过这需要开发者能够以一种安全的方式来构建网络应用程序，虽然已有大量针对在网络应用程序开发中如何安全地访问数据库的文档出版，但仍然有很多开发者缺乏足够的安全意识，造成开发出的产品中依旧存在注入漏洞；方案(2)的做法需要 RDBMS 的支持，目前只有 Oracle 采用该技术；方案(3)则是一种不完全的解决措施，例如，当客户端的输入为“...ccmdmcmd...”时，在对敏感字符串“cmd”替换删除以后，剩下的字符正好是“...cmd...”；方案(4)是目前最常被采用的方法，很多安全文档都认为 SQL 注入攻击需要通过错误信息收集信息，有些甚至声称某些特殊的任务若缺乏详细的错误信息则不能完成，这使很多安全专家形成一种观念，即注入攻击在缺乏详细错误的情况下不能实施。

而实际上，屏蔽错误信息是在服务端处理完毕之后进行补救，攻击其实已经发生，只是企图阻止攻击者知道攻击的结果而已。本文所介绍 SQL 盲注技术就是一些攻击者使用的新技术，其在错误信息被屏蔽的情况下使攻击者仍能获得所需的信息，并继续实施注入攻击。

1.3 本文的结构组织

为了理解盲注攻击，我们首先将介绍确定 SQL 注入漏洞所需的服务器的最小响应；其次，我们将构造一个合乎语法的 SQL 请求，并可以将之替换成任何有效的 SQL 请求；最后，我们将讨论在没有详细错误信息的情况下如何利用 UNION SELECT 语句。本文所讨论的盲注攻击的条件是我们在攻击前对网络应用程序、数据库类型、表结构等等信息都一无所知，这些信息都需要在注入的过程中通过探测获得。

2 确定注入漏洞

要进行 SQL 注入攻击，首先当然是确认要攻击的网络应用程序存在注入漏洞，因此攻击者首先必须能确立一些与服务器产生的错误相关的提示类型。尽管错误信息本身已被屏蔽，网络应用程序仍然具有能区分正确请求和错误请求的能力，攻击者只需要学习去识别这些提示，寻找相关错误，并确认其是否和 SQL 相关。

2.1 识别错误

一个网络应用程序主要会产生两种类型的错误，第一种是由 Web 服务器产生的代码异常(exception)，类似于“500:Internal Server Error”，通常如果 SQL 注入语句出现语法错误，比如出现未闭合的引号，就会使服务器抛出这类异常。如果要屏蔽该类错误，一般会采用将默认的错误信息替换成一个事先定制的 HTML 页面，但只要观察到有这种响应出现，就可以确认其实是发生了服务器错误。在其他情况下，为了进一步屏蔽该类错误，有些服务器一出现异常，会简单地跳转到主页面或前一个访问过的页面，或者显示一条简单的错误消息但不提供任何细节。

第二类错误是由应用程序代码产生的，这代表其开发者有较好的编程习惯。这类应用程序考虑到可能会出现一些无效的情况，并分别为之产生了一个特定的错误信息。尽管出现这类错误一般会返回一个请求有效的响应(200 OK)，但页面仍然会跳转到主页面，或者采用某种隐藏信息的办法，类似于“Internal Server Error”。

为了区分这两种错误，我们看一个例子：有两个电子商务的应用程序，A 和 B，两个应用程序都使用同一个叫 proddetails.asp 的页面，该页面期待获得一个参数，叫 ProdID。它获取该参数后，从数据库中提取相应的产品详细信息数据，然后对返回的结果进行一些处理。两个应用程序都是通过一个产品列表页面上的链接调用 proddetails.asp，因此能保证 ProdID 一直都是存在且有效的。应用程序 A 认为这样就不会出现问题，因此对参数不做额外的检查，而如果攻击者篡改了 ProdID，插入了一个在数据表中不存在的 id，数据库就会返回一个空记录。由于应用程序 A 没有料到可能会出现空记录，当它试图去处理该记录中的数据时，就可能会出现异常，产生一个“500:Internal Server Error”。而应用程序 B，会在对记录进行处理前确认记录的大小超过 0，如果是空记录，则会出现一个错误提示“该产品不存在”，或者开发者为了隐藏该错误，会将页面重新定位到产品的列表页面。

因此攻击者为了进行 SQL 盲注，会首先尝试提交一些无效的请求，并观察应用程序如何处理这些错误，以及如果出现 SQL 错误会发生什么情况。

2.2 定位错误

对要攻击的应用程序有了初步的认识后，攻击者会试图定位由人为构造的输入而产生的错误信息。这时，攻击者就会使用标准的 SQL 注入测试技术，比如添加一些 SQL 关键字（如 OR，AND 等）和一些 META 字符（如;或'等）。每一个参数都被独立地进行测试，而获得的响应将被检验用来判断是否产生了错误。通过一个拦截代理服务器(intercepting proxy)或者类似的工具可以方便地识别页面跳转和其他一些可预测的隐藏错误，而任何一个返回错误的参数都有可能存在 SQL 注入漏洞。而在单独测试每个参数过程中，必须保证其他参数都是有效的，因为需要避免除注入以外任何其他可能的原因所导致的错误影响了判断结果。测试的结果一般是一个可疑参数的列表，列表中的一些参数可能的确可以进行注入利用，另外一些参数则可能是由一些 SQL 无关的错误所造成，因此需要被剔除。攻击者接下来就需要从这些参数中挑选真正存在注入漏洞的参数，我们称之为确定注入点。

2.3 确定注入点

SQL 字段可以被划分为三个主要类型：数字、字符串和日期。虽然每个类型都有其特点，但却与注入的过程无关。每一个从网络应用程序提交给 SQL 查询的参数都属于以上三个类型中的一类，其中数字参数被直接提交给服务器，而字符串和日期则需要加上引号才被提交，例如：

```
SELECT * FROM Products WHERE ProdID = 4
```

与

```
SELECT * FROM Products WHERE ProdName = 'Book'
```

而 SQL 服务器，并不关心它接受到的是什么类型的参数表达式，只要该表达式是相关的类型即可。而这个特点则使攻击者能够很容易地确认一个错误是否和 SQL 相关。如果是数字类型，最简单的处理办法是使用基本的算术操作，例如以下请求：

```
/mysite/proddetails.asp?ProdID=4
```

测试该参数的一种办法是插入 4' 作为参数，另一种是使用 3+1 作为参数，假设这两个参数已直接被提交给 SQL 请求语句，则将形成以下两个 SQL 请求语句：

```
(1) SELECT * FROM Products WHERE ProdID = 4'
```

```
(2) SELECT * FROM Products WHERE ProdID = 3 + 1
```

第一个 SQL 语法有问题，将一定会产生一个错误，而第二个如果被顺利地执行，返回和最初的请求（即 ProdID 等于 4）一样的产品信息，这就提示该参数是存在注入漏洞的。

类似的技术可以被应用于用一个符合 SQL 语法的字符串表达式替换该参数，这里有两个区别：第一，字符串表示式是放在引号中的，因此需要阻断引号；第二，不同的 SQL 服务器连结字符串的语法不同，比如 MS SQL Server 使用符号+来连结字符串，而 Oracle 使用符号||来连结。例如以下请求：

```
/mysite/proddetails.asp?ProdName=Book
```

要测试该 ProdName 参数是否有注入漏洞，可以先将其替换成一个无效的字符串比如 Book'，然后再替换成一个可能生成正确字符串的表达式，比如 B'+ook（对于 Oracle，是 B'||ook）。这就会形成以下两个 SQL 请求语句：

```
(1) SELECT * FROM Products WHERE ProdName = 'Book'
```

```
(2) SELECT * FROM Products WHERE ProdID = 'B' + 'ook'
```

则第一个仍然可能产生一个 SQL 错误，而第二个则可能返回和最初的请求一样的值为 Book 的产品。

我们注意到，即使应用程序已经过滤了'和+等 META 字符，我们仍然可以在输入时过把字符转换成 URL 编码（即字符 ASCII 码的 16 进制）来绕过检查，例如：

```
/mysite/proddetails.asp?ProdID=3+1 就等于/mysite/proddetails.asp?ProdID=3%2B1
```

```
/mysite/proddetails.asp?ProdID=B'+ook 就等于/mysite/proddetails.asp?ProdID=B%27%2B%27ook
```

类似的，任何表达式都可以用来替换最初的参数。而特殊的系统函数也可以被用来提交以返回一个数字，一个字符串或一个日期，比如 Oracle 中 sysdate 返回一个日期表达式，而在 SQL Server 中，getdate()会返回日期表达式。其他的技术同样可以被用来判断是否存在 SQL 注入漏洞。

通过以上介绍可以发现，即使没有详细的错误信息，对于攻击者来说，判断是否存在 SQL 注入漏洞仍然是一个非常简单的任务。

3 实施注入攻击

攻击者在确定注入点后，就要尝试进行注入利用，这需要其能确定符合 SQL 语法的注入请求表达式，判断出后台数据库的类型，然后构造出所需的利用代码。

3.1 确定正确的注入句法

这是 SQL 盲注攻击中最难也最有技巧的步骤，如果最初的 SQL 请求语句很简单，那么确定正确的注入语法也相对容易，而如果最初的 SQL 请求语句较复杂，那么要想突破其限制就需要多次的尝试，但进行这些尝试所需要的基本技术却是非常简单。

确定基本的句法的过程即通过标准的 SELECT ... WHERE 语句，被注入的参数（即注入点）就是 WHERE 语句的一部分。为了确定正确的注入句法，攻击者必须能够在最初的 WHERE 语句后添加其他数据，使其能返回非预期的结果。对一些简单的应用程序，仅仅加上 OR 1=1 就可以完成，但在大多数情况下如果想构造出成功的利用代码，这样做当然是不够的。经常需要解决的问题是如何配对插入语符号（parenthesis，比如

成对的括号),使之能与前面的已使用的符号,比如左括号匹配。另外常见的问题是一个被篡改的请求语句可能会导致应用程序产生其他错误,这个错误往往难于和一个 SQL 错误相区分,比如应用程序一次如果只能处理一个记录,在请求语句后添加 `OR 1=1` 可能使数据库返回 1000 条记录,这时就会产生错误。由于 WHERE 语句本质上是一串通过 OR、AND 或插入语符号连接起来的值为 TRUE 或 FALSE 的表达式,因此要想确定正确的注入句法,关键就在于能否成功地突破插入语符号限制并能顺利地结束请求语句,这就需要进行多次组合测试。例如,添加 `AND 1=2` 能将整个表达式的值变为 FALSE,而添加 `OR 1=2` 则不会对整个表达式的值产生影响(除非操作符有优先级)。

对于一些注入利用,仅仅改变 WHERE 语句就足够了,但对于其他情况,比如 UNION SELECT 注入或存储过程(stored procedures)注入,还需要能先顺利地结束整个 SQL 请求语句,然后才能添加其他攻击者所需要的 SQL 语句。在这种情况下,攻击者可以选择使用 SQL 注释符号来结束语句,该符号是两个连续的破折号(--),它要求 SQL Server 忽略其后同一行的所有输入。例如,一个登录页面需要访问者输入用户名和密码,并将其提交给 SQL 请求语句:

```
SELECT Username, UserID, Password FROM Users WHERE Username = 'user' AND Password = 'pass'
```

通过输入 `john'--` 作为用户名,将会构造出以下 WHERE 语句:

```
WHERE Username = 'john' --'AND Password = 'pass'
```

这时,该语句不但符合 SQL 语法,而且还使用户跳过了密码认证。但是如果是另外一种 WHERE 语句:

```
WHERE (Username = 'user' AND Password = 'pass')
```

注意到这里出现了插入语符号,这时再使用 `john'--` 作为用户名,请求语句就会错误:

```
WHERE (Username = 'john' --' AND Password = 'pass')
```

这是因为有未配对的插入语符号,请求语句就不会被执行。

这个例子显示出使用注释符号能够用来判断请求语句是否被顺利地结束了,如果添加了注释符号且没有产生错误,这就意味着注释符号前的语句已经顺利地结束。如果出现了错误,这就需要攻击者进行更多的请求尝试。

3.2 判断数据库类型

攻击者一旦确定了正确的注入句法后,就会开始利用注入去判断后台数据库的类型,这个步骤比确定注入句法要简单得多。攻击者一般会使用以下几种技巧,这些技巧是基于不同类型数据库引擎在具体实现上的差异。下面只介绍如何区分 Oracle 和 MS SQL Server:

最简单的办法,就是前面提到的利用字符串的连结符号,在注入句法已经确定的情况下,攻击者可以对 WHERE 语句自由地添加额外的表达式,那么就可以利用字符串的较来区分数据库,例如:

```
AND 'xxx' = 'x' + 'xx' (或者 AND %27xxx%27+%3D+%27x%27+%2B+%27xx%27)
```

通过将+替换成||,就可以判断出是数据库是 Oracle 还是 MS SQL Server,或者是其他类型。

其他的办法是利用分号字符(即;),在 SQL 中,分号是用来将几个 SQL 语句连接在同一行中。在注入时,也可以在注入代码中使用分号,但 Oracle 驱动程序却不允许这样使用分号。假设在前面使用注释符号时没有出现错误,那么在注释符号前加上分号对 MS SQL Server 是没有影响的,但如果是 Oracle 就会产生错误。另外,还可以使用 COMMIT 语句来确认是否允许在分号后再执行其他语句(例如,注入语句 `xxx'; COMMIT --`),如果没有出现错误就可以认为允许多句执行。

最后,表达式还可以被替换成能返回正确值的系统函数,由于不同类型的数据库使用的系统函数也是不同的,因此也可以通过使用系统函数来确定数据库类型,比如 2.3 节提到的 MS SQL Server 的日期函数 `getdate()` 与 Oracle 的 `sysdate`。

3.3 构造注入利用代码

当所有相关的信息都已获得后,攻击者就可以开始进行注入利用,而且在构造注入利用代码过程中也不再需要详细的错误信息,构造利用代码本身可以参考其他描述标准 SQL 注入攻击的文档。

由于对于普通的 SQL 注入利用，已经有很多其他论文进行了详细的讨论，故本文只会在下一节介绍一种 UNION SELECT 注入。

4 UNION SELECT 注入

尽管通过篡改 SELECT...WHERE 语句来注入对于很多应用程序非常有效，但在盲注情况下，攻击者仍然愿意使用 UNION SELECT 语句，这是因为与 WHERE 语句所进行的操作不同，使用 UNION SELECT 可以让攻击者在没有错误信息的情况下依然能访问数据库中所有表。

进行 UNION SELECT 注入需要预先获知数据库的表中的字段个数和类型，而这些信息一般被认为在没有详细错误信息的提示下是不可能获得的，但本文下面就将给出解决该方法的方法。

另外需要注意的是，进行 UNION SELECT 的前提是攻击者已经确定了正确的注入句法，本文的前面一节已经阐明了这在盲注条件下是可以实现的，而且在使用 UNION SELECT 语句之前，SQL 语句中所有的插入语符号都应该已经完成配对，从而可以自由地使用 UNION 或者其它指令进行注入。UNION SELECT 还要求当前语句和最初的语句查询的信息必须具有相同的数和相同的数据类型，不然就会出错。

4.1 统计列数

当错误信息没有被屏蔽时，要获取列数只需要在进行 UNION SELECT 注入时每次尝试使用不同的字段数即可，当错误信息由“列数不匹配”变成“列的类型不匹配”时，当前尝试的列数就是正确的。但在盲注条件下，由于我们对无法获悉错误信息究竟是哪个，所以该方法也就失去了作用。

新的办法是利用 ORDER BY 语句，在 SELECT 语句最后加上 ORDER BY 能够改变返回的记录集的次序，一般是按一个指定的列名的值进行排序。例如，当通过产品号查询产品时，一个有效的注入语句如下：

```
SELECT ProdNum FROM Products WHERE (ProdID=1234) ORDER BY ProdNum --
AND ProdName='Computer') AND UserName='john'
```

人们往往会忽略的是 ORDER BY 语句后还可以使用数字指代列名，在上例中如果 ProdNum 是查询请求返回的记录中的第一列，则注入 1234) ORDER BY 1--返回的结果是一样的。由于上例查询请求只返回一个字段，注入 1234) ORDER BY 2 --就会出错，即返回的记录无法按指定的第二个字段排序。这样，ORDER BY 就可以被用来对列数进行统计了。由于每个 SELECT 语句都至少返回一个字段，故攻击者可以先在注入句中中添加 ORDER BY 1 来确定语句是否能被正确执行，有时对字段的排序也可能产生错误，这时添加关键字 ASC 或 DESC 可以解决这个问题。一旦确定 ORDER BY 句法是有用的，攻击者就会对排序列号从列 1 到列 100 进行遍历（或者到列 1000，直到列号被确定为无效），理论上当出现第一个错误时，前一个列号就是要统计的列数，但在实际情况中，有些字段可能不允许排序，那么在出现第一次错误时可以再多尝试一到两个数字，以确认列号已遍历完。

4.2 判断列的数据类型

在统计完列数后，攻击者需要再判断列的数据类型，在盲注情况下判断类型也是有技巧的，由于 UNION SELECT 要求前后查询语句查询的字段类型相同，故如果字段数有限，可以简单地利用 UNION SELECT 语句对字段类型进行暴力穷举(brute force)，但如果字段数较多，判断就会出现困难。根据前文，字段的类型只有数字、字符串和日期三种可能的类型，一旦字段数有 10 个，那么就意味着有 3^{10} （约 60,000）种可能的组合，假设每一秒可以自动进行 20 次尝试，穷举一遍也需要近一个小时，如果字段数更多，那么测试所需时间就会令人难以忍受。

一种简单的办法是利用 SQL 的关键字 NULL，与静态字段的注入需要区分是数字类型还是字符类型不同，NULL 可以匹配任何一种数据类型。因此可以注入一个所有查询字段都为 NULL 的 UNION SELECT 语句，那么就不会出现任何类型不匹配的错误。让我们再举一个与前面类似的例子：

```
SELECT ProdNum,ProdType,ProdPrice,ProdProvider FROM Products
WHERE (ProdID=1234 AND ProdName=' Computer') AND UserName='john'
```

假设攻击者已经获得了列数（在该例中为 4），那么就可以很简单地构造一个 UNION SELECT 语句，其中所有查询字段都为 NULL，还需要构造一个不会产生权限问题的 FROM 语句。对于 MS SQL Server，即使忽略 FROM 语句也不会出错，但对于 Oracle，则可以使用一个名叫 dual 的表。最后，还需要一个值一定为 FALSE 的 WHERE 语句（比如 WHERE 1=2），这是为了确保查询不会返回只包含 null 值的记录集，以杜绝产生其他可能的错误。那么针对 MS SQL Server 的注入语句如下：

```
SELECT ProdNum,ProdType,ProdPrice,ProdProvider FROM Products
WHERE (ProdID=1234) UNION SELECT NULL,NULL,NULL,NULL
WHERE 1=2 -- AND ProdName=' Computer') AND UserName='john'
```

这个 NULL 注入语句有两个目的，主要目的是构造一个不会产生任何错误的 UNION SELECT 语句以测试 UNION 语句是否可以被执行，另一个目的是为了对数据库类型的判断进行 100% 确认（可以通过在 FROM 语句里添加一个数据库开发商预置的表名进行测试）。

如果 NULL 注入语句被顺利执行，那么就可以快速地对每个列的类型进行判断。在每一轮尝试中，只对一个字段类型进行测试，由于类型只有三类，所以每个字段最多被测试三次就会有结果，这样尝试的次数最多是列数的三倍，而不是以 3 为底数以列数为指数的次数。假设 ProdNum 属于数字类型，其它三个字段都属于字符串类型，那么以下顺序的注入语句就可以判断出正确的类型：

• 1234) UNION SELECT NULL,NULL,NULL,NULL WHERE 1=2 --

无错 – 句法正确，使用的是 MS SQL Server 数据库

• 1234) UNION SELECT 1,NULL,NULL,NULL WHERE 1=2 --

无错 – 第一个字段是数字类型

• 1234) UNION SELECT 1,2,NULL,NULL WHERE 1=2 --

出错 – 第二个字段不是数字类型

• 1234) UNION SELECT 1,'2',NULL,NULL WHERE 1=2 --

无错 – 第二个字段是字符串类型

• 1234) UNION SELECT 1,'2',3,NULL WHERE 1=2 --

出错 – 第三个字段不是数字类型

• 1234) UNION SELECT 1,'2','3',NULL WHERE 1=2 --

无错 – 第三个字段是字符串类型

• 1234) UNION SELECT 1,'2','3',4 WHERE 1=2 --

出错 – 第四个字段不是数字类型

• 1234) UNION SELECT 1,'2','3','4' WHERE 1=2 --

无错 – 第四个字段是字符串类型

攻击者现在就已经获得了每一列的数据类型，盲注还可以被应用于从数据库的表中获取数据，比如获得数据表的列表以及它们各自的列名，还可以从应用程序中获得数据，而这些技术在其他一些关于 SQL 注入的论文中已经有讨论，故本文不再继续介绍。

5 总结

本文综述了目前在 SQL 盲注攻击方面的技术发展，阐明了在错误信息即使被屏蔽的情况下，SQL 注入漏洞仍然可以被利用。即使已经采取了很多措施来隐藏和掩饰返回给用户的信息，使用本文介绍的技术，很多应用程序仍然可以被注入利用。这就表明应用程序级别的漏洞，仅仅依靠对服务器的基本设置做一些改动是不能够解决的，必须从提高应用程序的开发人员的安全意识入手，加强对代码安全性的控制，在服务端正式处理之前对每个被提交的参数进行合法性检查，以从根本上解决注入问题。

References:

- [1] Microsoft China Technology Center, SQL Server 安全回顾, <http://www.microsoft.com/china/ctc/Newsletter/04/ctc2.htm>, 2004.
- [2] Anley, C., Advanced SQL Injection In SQL Server Applications, http://www.ngssoftware.com/papers/advanced_sql_injection.pdf, An NGSSoftware Insight Security Research (NISR) Publication, 2002
- [3] Kost S., Introduction to SQL Injection Attacks for Oracle Developers, ©Integrigy Corporation, <http://www.net-security.org/dl/articles/IntegrigyIntrotoSQLInjectionAttacks.pdf>, 2004.
- [4] Anley, C., (more) Advanced SQL Injection, http://www.ngssoftware.com/papers/more_advanced_sql_injection.pdf, An NGSSoftware Insight Security Research (NISR) Publication, 2002
- [5] Cerrudo, C., Manipulating Microsoft SQL Server Using SQL Injection, 2002
http://www.appsecinc.com/presentations/Manipulating_SQL_Server_Using_SQL_Injection.pdf.
- [6] Spett, K., SQL Injection - Are your web applications vulnerable?, 2002
<http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf>