# 简介

所谓SQL注入，就是通过把SQL命令插入到Web表单提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。

应用程序在向后台数据库传递SQL（Structured Query Language，结构化查询语句）查询时，如果为攻击者提供了影响该查询的能力，就会引发SQL注入。

一个正常的PHP数据查询语句：

$conn = mysql\_connection($host, $username, $password)

$query = “select \* from users where user=’admin’ and passwd=’”

$query = $query.$\_GET[“passwd”].”’”；

$result = mysql\_query($query);

这里查询语句中的passwd是根据用户输入设定的，没有做任何的限定，攻击者可以利用这个修改查询语句。

产生上述问题根源在于我们SQL语句采用拼接的方式，没有引入预编译（prepare，会检查类型），这样如果客户端输入语句为1 or 1=1，最后服务端查询语句为select \* from tb where passwd =1 or 1=1;当然返回成功。因为这里直接将语句拼接了，如果采用预编译，即select \* from tb where passwd = ?，后面会填充在这个？并进行类型判断，就不会出现拼接的错误语句。

# 分类

## SQL注入点

字符型

整数型

搜索型：like ‘’

## SQL注入攻击

没有正确的过滤转义字符

错误的类型处理：比如要求传入整数型，最终传入字符型

数据库服务器中的漏洞

SQL盲注：错误没有回显

条件响应：1=1 AND 1=2

条件性差错（select 1/0 from t1 where id=0;）

时间延迟

# 原理

正常的 SQL 语句：

select \* from users where id = 1;//正常情况不会把id写死，而是根据外部传入

动态的构造过程：

$query = "select \* from users where user='admin' and passwd='"

$query = $query.$\_GET["passwd"]."'";

利用动态构造构造注入语句：

不传入id数字，而是传入字符等非法字符

# 测试

简单的规则：发送以外数据来触发异常

识别Web应用的数据输入：

select \* from users where id =1 and 1=1; //TRUE and TRUE，返回正确结果

select \* from users where id =1 and 1=2;//TRUE and FALSE,返回空，SQL注入

了解哪种类型的请求会触发异常

检测服务器响应中的异常

# SQL注入工具

## BSQL Hacker

## The Mole

## Pangolin

## Sqlmap

## Havij

## Enema SQLi

## SQLninja

## Sqlsus

## Safe3 SQL Injector

## SQL Poizon

# 利用与防御

解决方案：SQL预编译

## 不同数据库的SQL注入

### Access的SQL注入

Access 注入猜解过程

猜解表名

猜解列名

猜解字段值长度

ASCII 逐字解码法猜解字段值

ASP+Access：

### MySQL的SQL注入

### SQLServer的SQL注入

有趣的出错信息

Group By 与 Having 查询

爆表名与字段名

爆数据记录

## 基于不同方法的条件语句的使用

### 基于时间

基于 Web 应用响应时间上的差异

基于时间（SQLServer）

有管理员访问权限

IF (system\_user = 'sa') WAITFOR DELAY '0:0:5' –-

IF (substring((select @@version), 25, 1) = 5) WARITFOR DELAY '0:5:15' –-

EXEC master..xp\_cmdshell 'ping -n 5 127.0.0.1'

有管理员访问权限，没有启动xp\_cmdshell,于 2005和 2008 版本

EXEC sp\_config 'show advanced options', 1;

GO

RECONFIGURE;EXEC sp\_configure 'xp\_cmdshell', 1;

2000版本

exec master..sp\_addextendedproc 'xp\_cmdshell', 'xplog70.dll'

基于时间（MySQL）

select benchmark(1000000),sha1('fooying'));

select sleep(5); (5.0.12 版本以上)

### 基于错误

根据指定位值的不同来触发不同的错误响应并提取位值

基于错误

/is\_srvrolemember('sysadmin')

/case when (system\_user = 'sa') then 1 else 0 end

### 基于内容

基于页面返回的不同 HTML 进行相关的判断

基于内容

%2B case when (system\_user = 'sa') then 1 else 0 end

//%2B相当于+

## 其他SQL注入利用手法

### 万能密码

'or'= 'or'

sql = "select \* from users when username='"& name & "'and passwd='"&password & "'"

sql = "select \* from users when username='' or'='or" and passwd='admin'

假 or 真 or 假 and 假=假 or 真 or （假 and假）=假or真or假=（假or真）or假=真or假=真

### 字符转换与编码的处理

Declare 与 OX6e 编码

二次编码

/\*\*/替换空格

Declare 与 OX6e 编码

16进制编码

0x6e6574207573657220666f6f79696e67202f616464

declare @a sysname

select @a=<16进制代码>

exec master .dbo.xp\_cmdshell @a

## SQL注入的防御

### 基础与二次过滤

输入检查

特殊符号过滤

白名单与黑名单

### 不同语言的SQL防御

使用参数化语句

特殊符号过滤

白名单与黑名单

Java

使用?占位符进行构造查询语句

JSF(Java Server Faces) 的 输入验证类 javax.faces.validator

.NET

@parameter(SqlClient)

:parameter(OracleClient)

?(OleDb/Odbc)

Regular-Expression Validator 控件

Custom Validator 控件

PHP

?占位符(mysqli,MySql)

Pg\_query\_params()(mysqli,PostgreSQL)

冒号字符与?占位符(PEAR:MDB2)

冒号字符与?占位符(PDO)

Preg\_match 使用正则表达式执行匹配验证

Is\_<type>(input) 判断输入类型

Strlen(input) 判断输入长度

### 平台层的防御

Web 应用防火墙

URL 策略与页面层策略

数据库权限控制

额外的部署考虑

## 总结

1. 把应用服务器的数据库权限降至最低，尽可能地减少 SQL 注入攻击带来的危害
2. 避免网站打印出SQL错误信息，比如类型错误、字段不匹配等，把代码里的SQL语句暴露出来，以防止攻击者利用这些错误信息进行SQL注入。
3. 对进入数据库的特殊字符（'"\尖括号&\*;等）进行转义处理，或编码转换。
4. 所有的查询语句建议使用数据库提供的参数化查询接口，参数化的语句使用参数而不是将用户输入变量嵌入到SQL语句中（即prepare），即不要直接拼接SQL语句。
5. 在测试阶段，建议使用专门的 SQL 注入检测工具进行检测。网上有很多这方面的开源工具，例如sqlmap、SQLninja等。
6. 善用数据库操作库，有些库包可能已经做好了相关的防护，我们只需阅读其文档，看是否支持相应的功能即可。