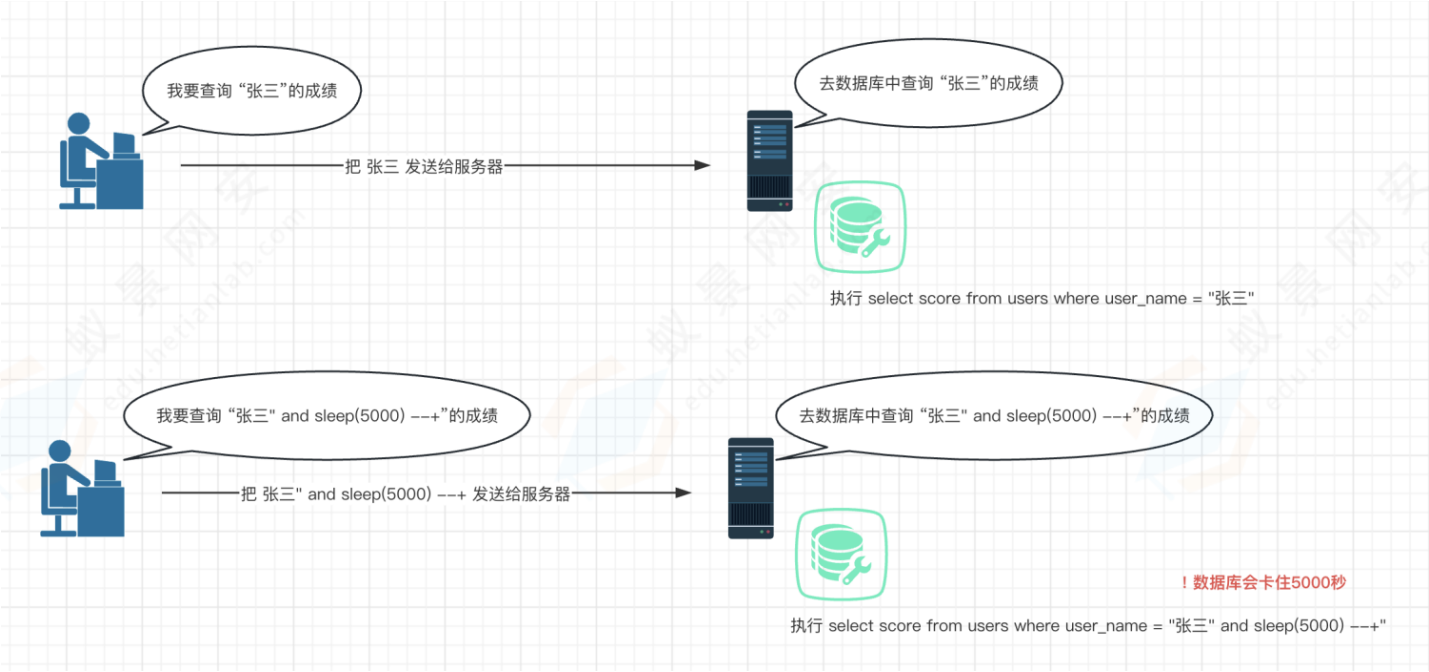
一、注入漏洞简介

- 什么是S注入漏洞
  - S注入是将Web页面的原URL、表单域或数据包输入的参数，修改拼接成S语句，传递给Web服务器，进而传给数据库服务器以执行数据库命令。如Web应用程序的开发人员对用户所输入的数据或cookie等内容不进行过滤或验证(即存在注入点)就直接传输给数据库，就可能导致拼接的S被执行，获取对数据库的信息以及提权，发生S注入攻击，它目前是黑客对数据库进行攻击的最常用手段之一。



1. SQL注入漏洞常见攻击手法

- 获取数据
  - 泄露数据库敏感信息，甚至脱库

```
Database: challenges
Table: KW5SBDQNLL
[1 entry]
+----+----------------------------------+------+--------------------------+
| id | sessid                           | tryy | secret_BDF1              |
+----+----------------------------------+------+--------------------------+
| 1  | cf76c4cd2c8f67fea31b8c15d16ca3bc | 0    | g5KBzvonhtMNPGkOm11eX4ly |
+----+----------------------------------+------+--------------------------+
```

- 植入木马
  - 通过数据库服务相关语法与函数写入网站后门，控制目标服务器

```
[09:28:14] [INFO] trying to upload the file stager on '/var/www/html/upload/' via LIMIT 'LINES TERMINATED BY' method
[09:28:14] [INFO] the file stager has been successfully uploaded on '/var/www/html/upload/' - http://127.0.0.1:10005/upload/tmpuqvxa.php
[09:28:14] [INFO] the backdoor has been successfully uploaded on '/var/www/html/upload/' - http://127.0.0.1:10005/upload/tmpbxzgw.php
[09:28:14] [INFO] calling OS shell. To quit type 'x' or 'q' and press ENTER
os-shell> id
do you want to retrieve the command standard output? [Y/n/a]
command standard output: 'uid=33(www-data) gid=33(www-data) groups=33(www-data)'
os-shell> cat /etc/passwd
do you want to retrieve the command standard output? [Y/n/a]
command standard output:
---
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101::/var/lib/libuuid:
syslog:x:101:104::/home/syslog:/bin/false
mysql:x:102:105:MySQL Server,,,:/nonexistent:/bin/false
---
os-shell>
```

2. SQL注入防御

(1) 代码角度

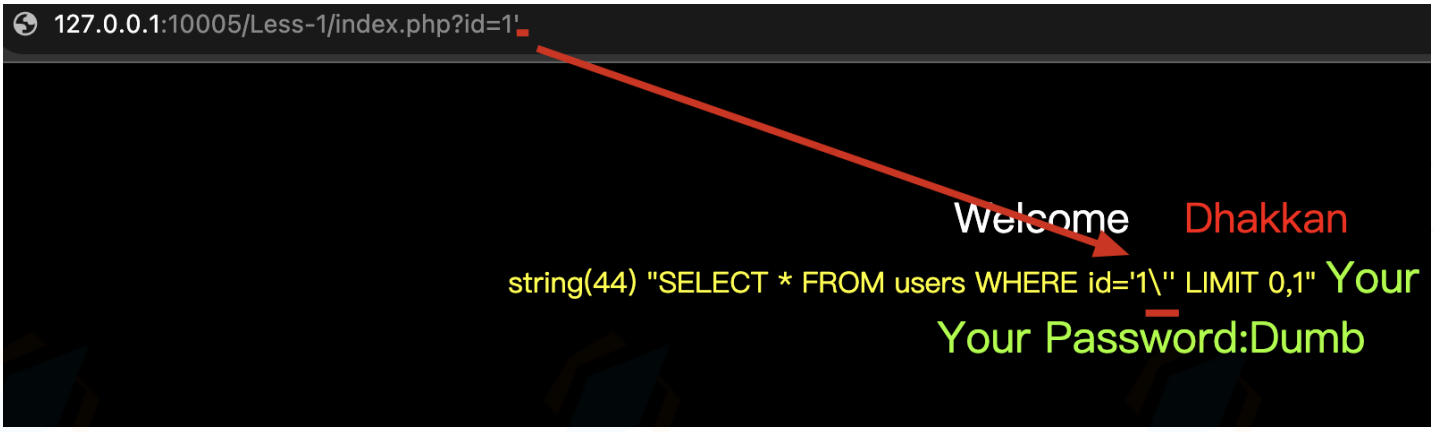
-

> 不允许 - 拦截某些敏感字符, 比如判断用户输入了例如 order 、 union 、select、updatexml、schema、information_schema 等字符串，就进行拦截

- mysqli_real_escape_string： 转义特殊字符
  - 会被进行转义的字符包括：NUL（ASCII 0）、\n、\r、\、'、" 和 Control-Z。

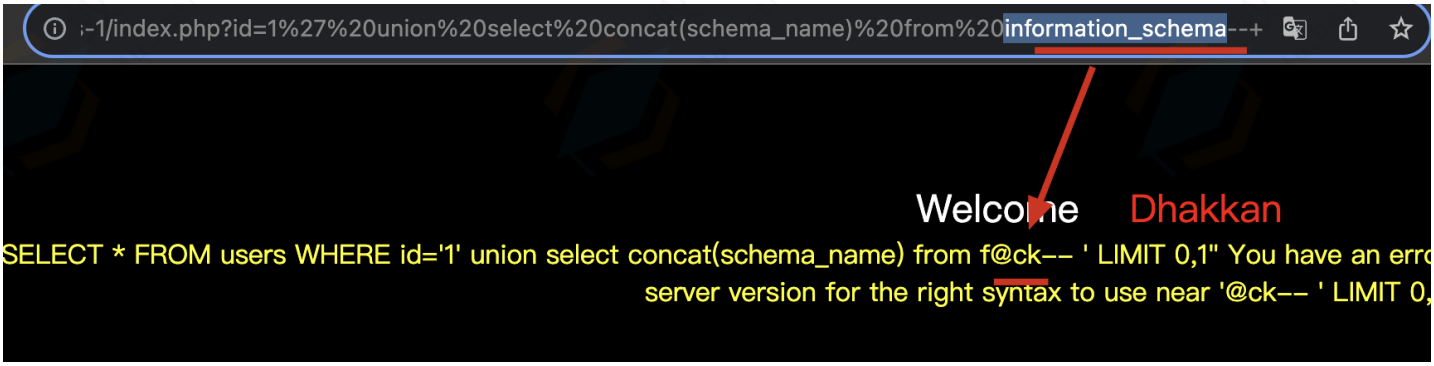
```
mysqli_real_escape_string([]);
;
(,

// exploit 利用
// 1' order by 1 --+    =>  1\' order by 1 --+
// 1' union select 1 --+  => 1\' union select 1 --+
```



- str_replace： 替换字符串

```
[
把用户输入的 information_schema 替换成 f@ck
str_replace("information_schema","f@ck",
str_replace("order","f@ck",
str_replace("database","f@ck",
;
(,
```



- strstr：判断字符串是否在另一个字符串内

```
[
如果用户输入的id中有information_schema ，则终止网站
(strstr(,"information_schema")){
      exit("f@ck"
}
;
(,
```

- 白名单

> -处标记的字符或类型外均不允许

- is_numeric : 检测变量是否为数字或数字字符串

```
[
(!is_numeric()){
    如果用户输入的不是数字
    exit("f@ck"
```

```
}
"select * from users where user_id = ";
(,
```

- 参数化查询

> PDO-S　预处理

- sql注入产生的原因就是因为用户传入的参数没有经过严格过滤直接拼接到了sql语句中，改变了原有的sql执行逻辑
- 通过将用户输入的数据与S语句分离，并使用参数绑定功能，S语句预处理可以有效地防御S注入攻击。因为输入数据没有被直接嵌入到S语句中，攻击者无法通过在输入中插入恶意的S代码来破坏原始的S语句结构或执行意外的查询。

```
假定数据库连接参数
"your_servername";
;
;
"your_dbname";

try
    创建一个新的PDO实例
    $dbh PDO("mysql:host=;dbname="

    使用命名占位符":id"准备语句
    $stmt$dbhprepare("SELECT * FROM users WHERE user_id = :id"

    将参数与命名占位符":id"绑定
    $stmtbindParam(":id"$userId

    假设$userId是用户提供的输入
    $userId$_GET['user_id'

    执行查询
    $stmt

    将所有结果作为关联数组获取
    $result$stmtfetchAll(PDO::FETCH_ASSOC

    根据需要处理结果
    // ...

    关闭数据库连接
    $dbhnull;
catch (PDOException $e
    处理数据库连接或查询执行过程中发生的任何错误
     "Error: "$egetMessage
```

```
}
?>
```

(2) 服务器或配置角度

> 资产数量多的时候，防御很难做到面面俱到，站在服务器安全的角度，让S注入的危害降到最低，和基线排查有一定关联性，但这里是针对于特定漏洞的防御

- 数据库安全
  - 数据库加密存储
  - md5 + salt

```
连接到 MySQL 数据库服务器
"localhost";
;
;
"your_database";

$conn mysqli(

$conn->connect_error) {
    die("连接数据库失败: "$conn->connect_error);
}

获取用户提交的表单数据
$user$_POST['username'
$pass$_POST['password'

对密码进行 MD5 加盐处理
$salt'yijingsec';
$hashedPasswordmd5($pass$salt

构造 SQL 插入语句并执行
"INSERT INTO users (username, password) VALUES ('$user', '$hashedPassword')";

$connquery() === TRUE
    "数据插入成功";

    "插入数据时出错: "$conn->error;
}

$connclose
?>
```

上面代码在将用户密码插入到数据库时使用了加密加盐，如果服务器遭受注入攻击，攻击者看到的数据库如下

```
+----+--------------+----------------------------------+
| id |  username    | password                         |
+----+--------------+----------------------------------+
| 1  | johndoe      | 5f4dcc3b5aa765d61d8327deb882cf99 |
| 2  | janedoe      | 098f6bcd4621d373cade4e832627b4f6 |
| 3  | alice        | c4ca4238a0b923820dcc509a6f75849b |
| 4  | bobsmith     | c81e728d9d4c2f636f067f89cc14862c |
| 5  | emilyjones   | eccbc87e4b5ce2fe28308fd9f2a7baf3 |
+----+--------------+----------------------------------+
```

提高盐值复杂度、或使用双重md5加密及其他类型的加密算法，能够保障数据泄漏后能最大程度降低影响

- 数据库安全配置
  - 配置文件限制读写路径
    - MyS　配置项：secure_file_priv secure_file_priv是用来限制LOAD DATA, SELECT … OUTFILE, LOAD_FILE()读写路径的配置项
      - secure_file_priv的值为null，表示限制mysql不允许导入导出
      - secure_file_priv的值为/tmp/，表示限制mysql的导入导出只能发生在/tmp/目录下
      - secure_file_priv的值没有具体值时，表示不对mysql的导入导出做限制

```
# Set it to a directory to allow importing and exporting data only from that directory.
# If secure-file-priv is set to NULL operations related to importing and exporting data are disabled.
# Setting secure-file-priv to "" is insecure.
secure_file_priv=""
```

  - 禁止外连
    - MyS　查看外连用户: SELECT host, user FROM mysql.user;
      - 如果某个用户的主机地址为 '%'，表示该用户被授权通过任何主机进行连接，即允许外部连接。
      - 如果某个用户的主机地址为其他特定IP地址或主机名，表示该用户只能通过指定的主机进行连接，不允许外部连接。

```
mysql> SELECT host,  FROM mysql.user;
+-----------+---------------+
| host      |               |
+-----------+---------------+
| %         | root          |
| 127.0.0.1 | root          |
| ::1       | root          |
| localhost | mamp          |
| localhost | mysql.session |
| localhost | mysql.sys     |
| localhost | root          |
+-----------+---------------+
7 rows  0.00 sec)
```

```
    - 限制用户外连
```

```
use mysql;
update   host = "localhost" where  = "root"  host = "%";
flush privileges;
```

- 指定某个用户[如 yijing] 47.234.11.2 的IP连接

```
UPDATE mysql.user  Host = '47.234.11.2' where User = 'yijing';
```

- 根据需求设置服务器监听地址

```
[client]
port=3306
[mysql]
default-character-set=utf8

[mysqld]
bind-address = 127.0.0.1
port=3306
basedir="C:/phpstudy/PHPTutorial/MySQL/"
datadir="C:/phpstudy/PHPTutorial/MySQL/data/"
character-set-server=utf8
default-storage-engine=MyISAM
#支持 INNODB 引擎模式。修改为  default-storage-engine=INNODB  即可。
#如果 INNODB 模式如果不能启动，删除data目录下ib开头的日志文件重新启动。

sql-mode="NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION"
max_connections=512
```

- 严格限制用户权限
  - root密码
    - 查看密码 Password from mysql. where = "root";
    - 修改密码 ALTER USER 'root'@'localhost' IDENTIFIED BY 'new_password';
    - 修改完成后要刷新权限表 FLUSH PRIVILEGES;

- 数据库备份
  - mysqldump -u username -p database_name > /path/to/backup/file.sql
    - 服务器安全
    - 站库分离
    - 自己搭建数据库服务器
    - Linux | Windows Server 搭建 MyS RDS
    - https://www.runoob.com/mysql/mysql-install.html
    - Docker 搭建 MyS

- https://www.runoob.com/docker/docker-install-mysql.html
- 使用云数据库
- 腾讯云 RDS https://cloud.tencent.com/product/cdb
- 阿里云 RDS https://www.aliyun.com/product/rds/mysql
- AWS RDS https://aws.amazon.com/cn/rds/

3. WAF是怎么做的

-
  - https://github.com/loveshell/ngx_lua_waf

> nginx + lua 是很多waf的实现方式，性能好，兼容强，不需要考虑后端语言
>
> - openresty https://openresty.org/cn/
>
> OpenResty® 是一个基于 Nginx 与 Lua 的高性能 Web 平台，其内部集成了大量精良的 Lua 库、第三方模块以及大多数的依赖项。用于方便地搭建能够处理超高并发、扩展性极高的动态 Web 应用、Web 服务和动态网关。

- waf 一般都内置了黑名单列表进行漏洞的防御

```
\.\./
\:\$
\$\{
.+(from|limit)
(?:(union(.*?)))
having|rongjitest
sleep\((\s*)(\d*)(\s*)\)
benchmark\((.*)\,(.*)\)
base64_decode\(
(?:from\W+information_schema\W)
(?:(?:current_)user|database|schema|connection_id)\s*\(
(?:etc\/\W*passwd)
into(\s+)+(?:dump|out)file\s*
group\s+by.+\(
xwork.MethodAccessor
(?:define|eval|file_get_contents|include|require|require_once|shell_exec|phpinfo|system|
passthru|preg_\w+||echo|print|print_r|var_dump|(fp)open|alert|showmodaldialog)\(
xwork\.MethodAccessor
(gopher|doc|php|glob|file|phar|zlib|ftp|ldap|dict|ogg|data)\:\/
java\.lang
\$_(GET|post|cookie|files|session|env|phplib|GLOBALS|SERVER)\[
\<(iframe|script|body|img|layer|div|meta|style|base|object|input)
(onmouseover|onerror|onload)\=
```

```
--deny url args
function url_args_attack_check()
    config_url_args_check ==
```

```
        local ARGS_RULES = get_rule('args.rule')
        for _,rule  pairs(ARGS_RULES) do
            local REQ_ARGS = ngx.req.get_uri_args()
            for key, val  pairs(REQ_ARGS) do
                 type(val) == 'table'
                    ARGS_DATA = table.concat(val, " ")

                    ARGS_DATA = val
                end
                 ARGS_DATA  type(ARGS_DATA) ~= "boolean"  rule ~=""  rulematc
h(unescape(ARGS_DATA),rule,"jo")
                    log_record('Deny_URL_Args',ngx.var.request_uri,"-",rule)
                     config_waf_enable ==
                        waf_output()
                        return true
                    end
                end
            end
        end
    end
    return false
end
```

这里可以看到，如果请求的url参数中有符合args.rule规则的字符串，就会被拦截并响应HTTP_FORBIDDEN [403]

> 思考：为什么waf不采用白名单?