

**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**



**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

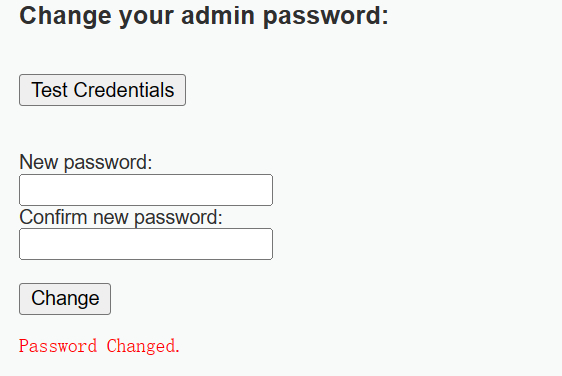
database connection is set and whether it is an object.

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#

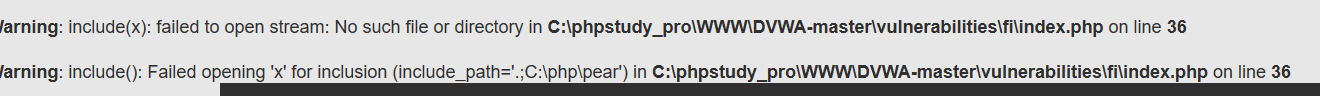


**3. File Inclusion**

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**



Then we get the file folder address of the server:

**C:\phpstudy\_pro\WWW\DVWA-master\vulnerabilities\fi\**

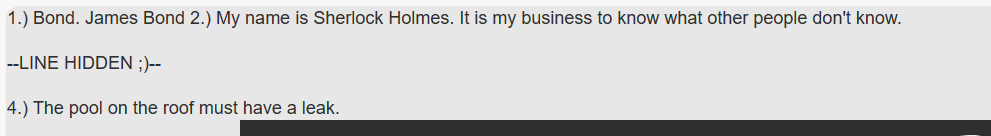
1. we can modify the file include query to get

../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](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. 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](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：

****

1. 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**

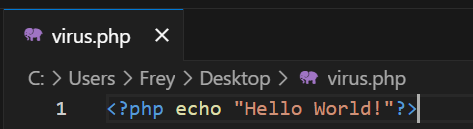
Uploaded files represent a significant risk to web applications. The first

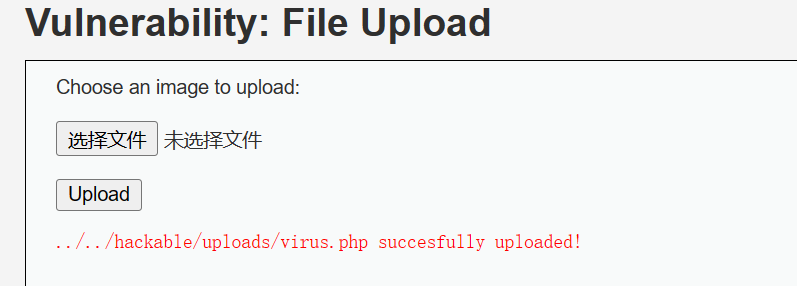
step in many attacks is to get some code to the system to be attacked. Then the

attacker only needs to find a way to get the code executed. Using a file upload

helps the attacker accomplish the first step.

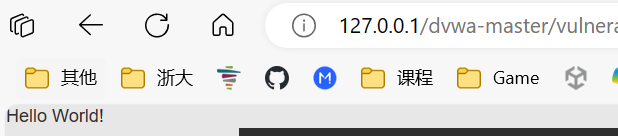
1. View the source We can find that the server does not do any checking and filtering on the type and content of the uploaded files, Upload a php file as follows





(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**

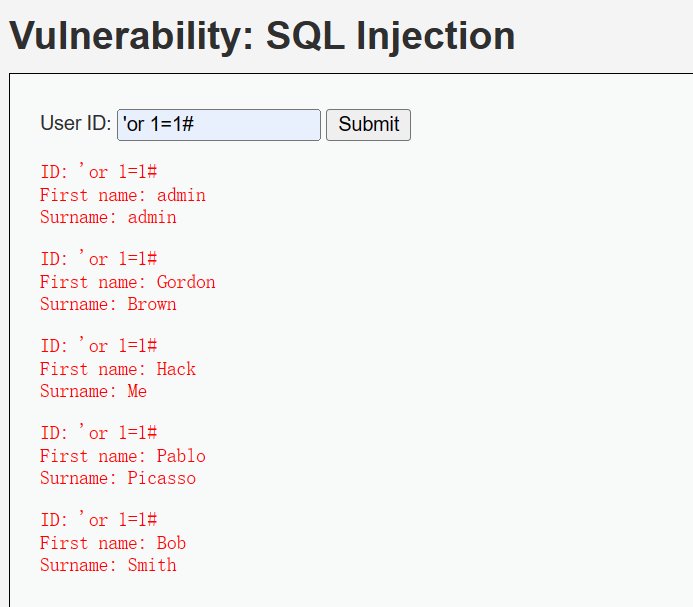


1. **SQL Injection**

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**

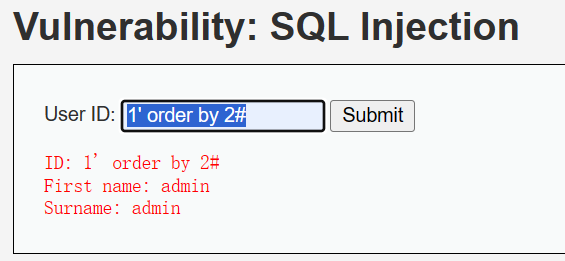
Input: 'or 1=1#



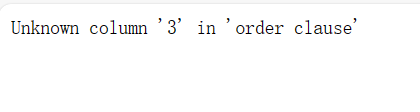
The type of injection is character

1. **Check how many numbers of fields are in SQL query statement**

Input: 1' order by 2#



Input: 1' order by 3#

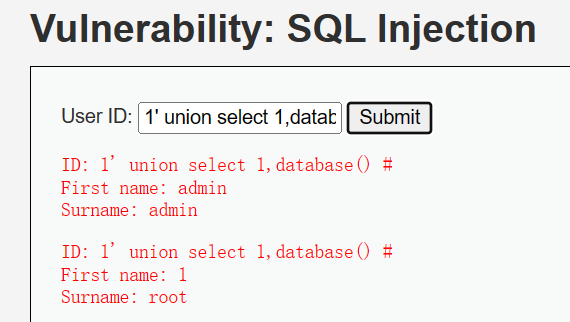


We know that there two fields are in SQL query statement, they are first

name and surname

1. **Get the current table name**

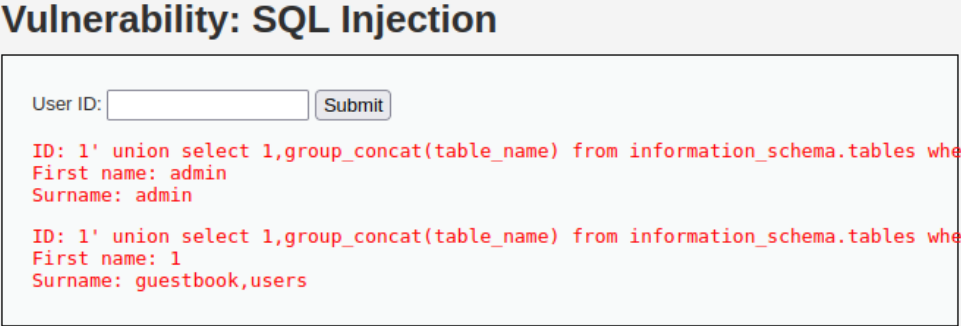
Input:1' union select 1, table\_name from information\_schema.tables where table\_schema=database() #



The name of current database is root

1. **Get the current tables**

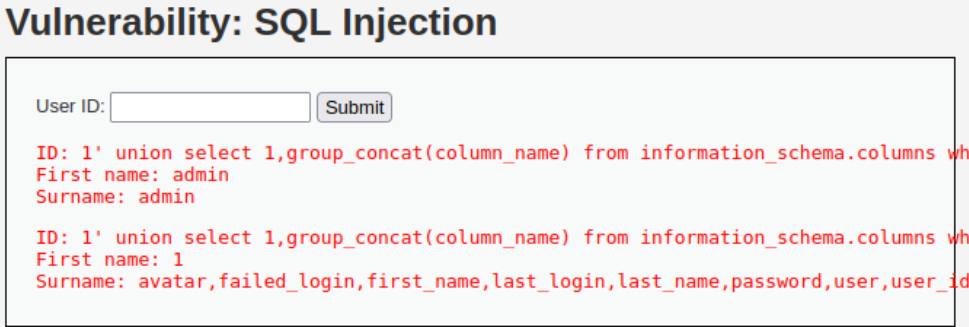
Input :-1' union select 1, group\_concat(table\_name) from information\_schema.tables where table\_schema = 'root' #



We find two tables: gustbook and users .

1. **Get the fields’ name of table users**

Input: 1' union select 1,group\_concat(column\_name) from information\_schema.columns where table\_name='users' #



1. **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 #



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



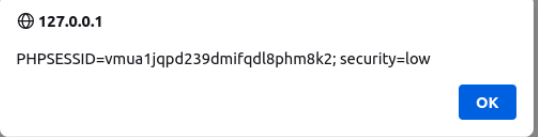
**8. XSS (DOM)**

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>



**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.

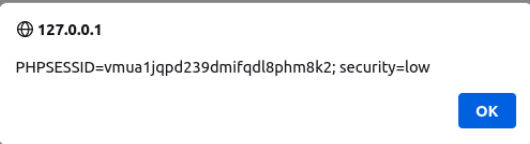
The script is activated through a link, which sends a request to a website with a

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:



**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:

