Web Security and Malware Analysis

Answers for Assignment X

SILVIA LUCIA SANNA – 70/90/00053

Task 1 – PHP Reading

This code is a php code for the HTML web page with title "Combined Feedback Form". From line 7 to line 12 there is the declaration of 6 variables: 5 of them (self, username, useraddr, comments, sent) take values from global variables (respectively SERVER, POST['username'], POST['useraddr'], POST['comments'], POST['sent']) which are very important because they are always accessible from every function in the program.

```
 1 <html> #root of the page
 2 <head><title>Combined Feedback Form</title></head> #title of html page
 3 <body> #start the body of the page
 4
 5 <?php #starts php section
 6
 7 $self = $_SERVER['PHP_SELF']; #var self has the name of the script that is running on server
 8 $username = $_POST['username']; #username has the username written in post action
 9 $useraddr = $_POST['useraddr']; #useraddr has the e-mail written in post action
10 $comments = $_POST['comments']; #comments has the text written in post action
11 $sent = $_POST['sent']; #sent is the button to send the data to post
12 $errmsg = ""; #define empty var
```

Figure 1: variable initialization on php code

From line 14 to line 23 there is the creation of the "form" section where the user will put his/her name, e-mail and comment. To submit use the button "sent".

```
14 #Form creation, stored it in a var called "form" and each field of the form is concatenated to the same
   var form
15 $form ="<form action=\"$self\" method=\"post\">"; #form creation
16 $form.="Name:<input type=\"text\" name=\"username\""; #name field in form
17 $form.=" size=\"30\" value=\"$username\" >"; #
18 $form.="Email:<input type=\"text\" name=\"useraddr\""; #email field in form
19 $form.=" size=\"30\" value=\"$useraddr\">"; #
20 $form.="Comments:<textarea name=\"comments\" >"; #comments field in form
21 $form.="$comments</textarea><br/>"; #
22 $form.="<input type=\"submit\" name=\"sent\" value=\"Send Form\">"; #submit button
23 $form.="</form>"; #end form
```

Figure 2: form creation on php code

When the user sends the form, there are 3 checks for username, e-mail and comments (lines from 30 to 37). If everything is ok, 3 private variables (name, host, tlds) will be initialized with their respectively RegExr.

```
25 if($sent) #if var sent exists, so if the user has pressed the button sent to send something
26 #errmsg empty var defined earlier
27 {
28   $valid=true; #var valid set to true
29   #check if the var are right, if not concatenates an error message and set valid to false
30   if( !$username ) #if username not put
31   { $errmsg.="Enter your name...<br />"; $valid = false; } #messaggio errore
32
33   if( !$useraddr ) #if useraddress is empty
34   { $errmsg.="Enter your email address...<br />"; $valid = false; } #error message for empty email field
35
36   if( !$comments ) #if comments is empty
37   { $errmsg.="Enter your comments...<br />"; $valid = false; } #error message for empty field comments
```

Figure 3: check for correct inputs

After that on line 39 there is the check on the correct e-mail address using the preg_match function. This function performs a regular expression match, meaning that checks if the first input is the model that it is looking for in the second input: indeed in our example the first input made by the concatenation between all possible characters forming a name in e-mail, the symbol at, the possible characters contained in the host and last the characters for the tlds (last layer of DNS); again in our example the second input is the e-mail address written by the user. So, the function is checking for the correct syntax of the e-mail address: if it is not correct there will be an error message with “Incorrect format for e-mail address”.

```
39   $useraddr = trim($useraddr); #update useraddr: deletes white spaces from begin and end of the string
40   # underscore before variable means that the variable is private or protected
41   $_name = "/^[-!#$%&\'*+\\.\/0-9=?A-Z^_'{|}~]+"; #regex for name variable
42   $_host = "([-0-9A-Z]+\.)+"; #regex for host variable
43   $_tlds = "([0-9A-Z]){2,4}$/i"; #regex for tlds var; linked to highest level of DNS> .com, .it, .org
44   #regex tells us the characters pool forming a specific variable, all possible characters to be
   inserted for that var
45   if( !preg_match( $_name."@".$_host .$_tlds,$useraddr ) ) #regular expression match: check if email
   inserted has right format
46   #right format is made by the concatenation of name, @, host and tlds
47   {
48     $errmsg.="Email address has incorrect format!<br />"; #error message for incorrect format
49     $valid=false;
50   }
51 }
```

Figure 4: check for e-mail format

If the user has not got error messages and so if the value of the variable “valid” is true, the final part of the code will be executed. From line 59 to 61 there is the definition of 3 useful variables: $to contains the e-mail address to whom is the e-mail addressed; $re contains the e-mail object; $msg contains the message written by the user in the form. From line 64 to 68 we have the definition of the HTTP header to be sent.

```
53 if($valid != true) #if there is some error (valid=false)
54 {
55   echo( $errmsg.$form ); #print error message
56 }
57 else #if everything is correct
58 {
59   $to = "php@h.com"; #e-mail address to who is the e-mail addressed
60   $re = "Feedback from $username"; #object of the e-mail to be sent
61   $msg = $comments; #msg of the e-mail to be sent
62
63   #send HTTP header. In function mail the header is the person that sends the e-mail
64   $headers  = "MIME-Version: 1.0\r\n";
65   $headers .= "Content-type: text/html;";
66   $headers .= "charset=\"iso-8859-1\"\r\n";
67
68   $headers .= "From: $useraddr \r\n";
```

Figure 5: header creation and variable initialization for feedback e-mail

On line 70 the function mail will send the e-mail: the first argument is the receiver, the second is the subject, the third is the content/message. Our addition parameter is the header. After sending the e-mail, the user will see a message of the correct sending.

```
70   if(mail($to,$re,$msg, $headers)) #if the e-mail is sent
71   { echo("Your comments have been sent - thanks $username");} #the user has a message that everything is
   ok
72    #What if mail is wrong?
73 }
74 ?>
75 </body></html>
```

Figure 6: feedback mail sent

So briefly recap: what does this code do? I connect to a specific web page, where there is running a certain php code. I put my username, my e-mail and some comments. I send this all and if everything is ok the server receives it. When the server receives it, it sends an e-mail to an address with a specific object and the message with the comments of the clients. If the e-mail was ok, I will have a message that everything is ok. If the form is wrong, I get specific error message.

Task 2 – More NATAS

- Natas6: when I browsed to natas 6 there was a button to view the source code and on line 17 there was a file inclusion. I thought that this one could be a file to which I can access from the web page, so I paste it in the URL after the domain name, so as first path. When I pressed

enter, Burp was not able to catch the request. So, I tried to analyse the page and in fact there was the secret. I came back to the home page and I put the found secret, clicked on “Invia richiesta” and I found the password for next level.

Figure 7: code inspection of natas6

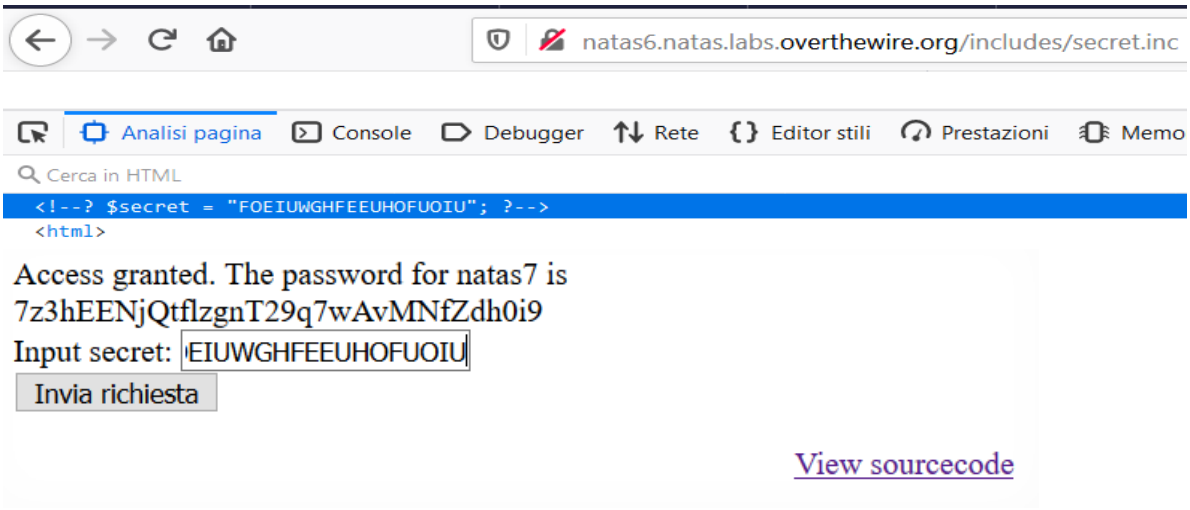

Figure 8: password for natas7 found

- Natas7: When I browsed to natas 7 there were only the home page and two buttons, both related to two different pages. In index page’s HTML code (specifically in the content block on the body, but I could also find in every of the three pages), I had a hint where the password would be. I copied that link and put as argument of the page query argument in the URL and then I get the password for next level.

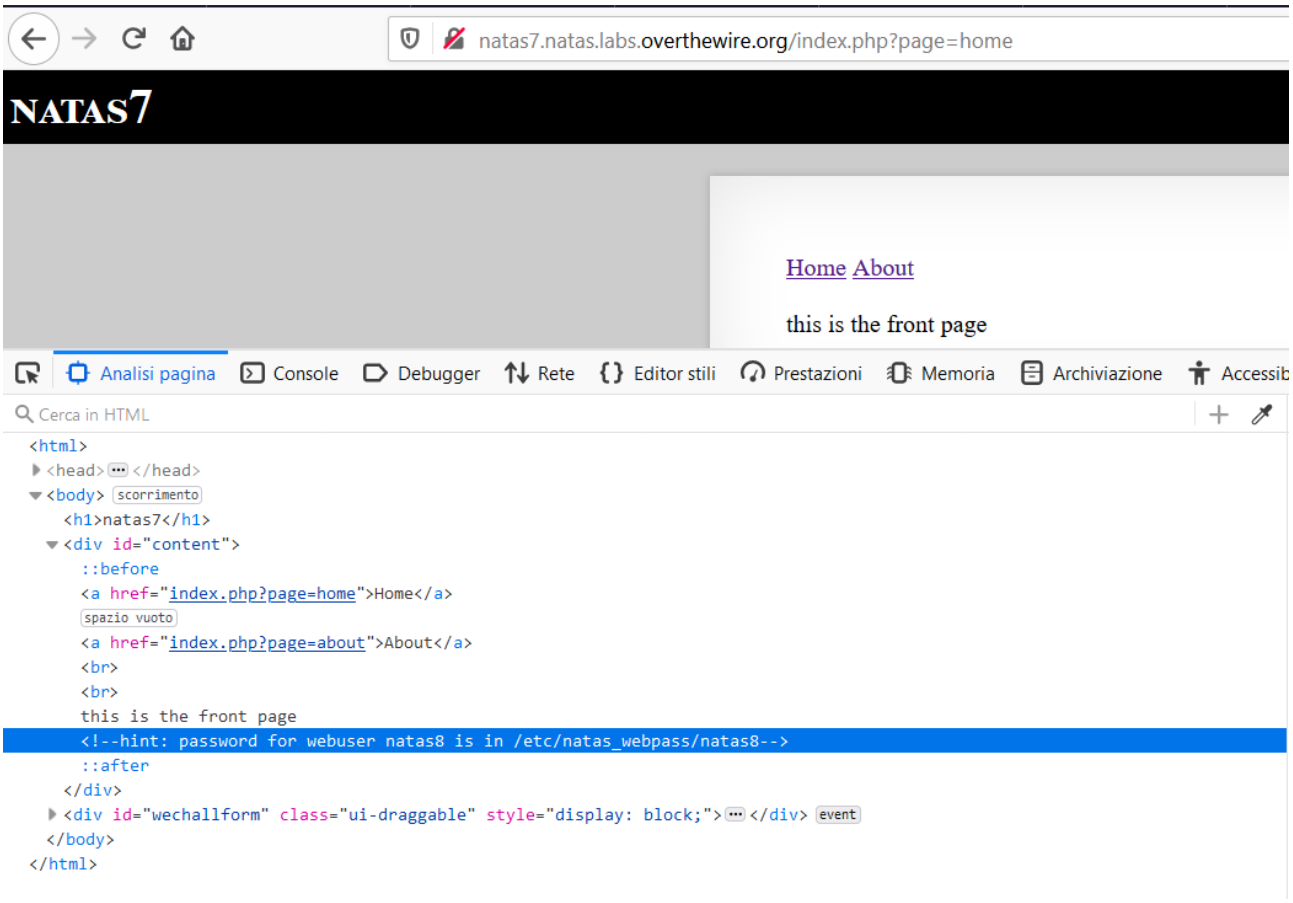

Figure 9: page inspection on natas7

Figure 10: password found for natas8

- Natas 8: When I browsed on natas 8 I saw a button with the source code. I inspected it and noticed an encoded secret and the process made to obtain it. First, I tried to make the reverse process with Python, but I do not know why I had some troubles. So, I decided to make it with online tools but maybe some of them are not well implemented. Then, I browsed to a php sandbox and put the reverse process,

finally I get the output. I put that output in the first page of natas 8 and got the password for natas 9.

```php
<?php
        //Enter your code here, enjoy!

$encodedSecret = "3d3d516343746d4d6d6c315669563362";

function encodeSecret($secret) {
    return bin2hex(strrev(base64_encode($secret)));
}

function decodeSecret($encodedSecret) {
    return base64_decode(strrev(hex2bin($encodedSecret)));
}

echo decodeSecret($encodedSecret);
```

Run on PHP version: 7.4.7
Output: Textbox

Execute code | Save or share your code

Result:
oubWYf2kBq

Figure 11: reverse php code

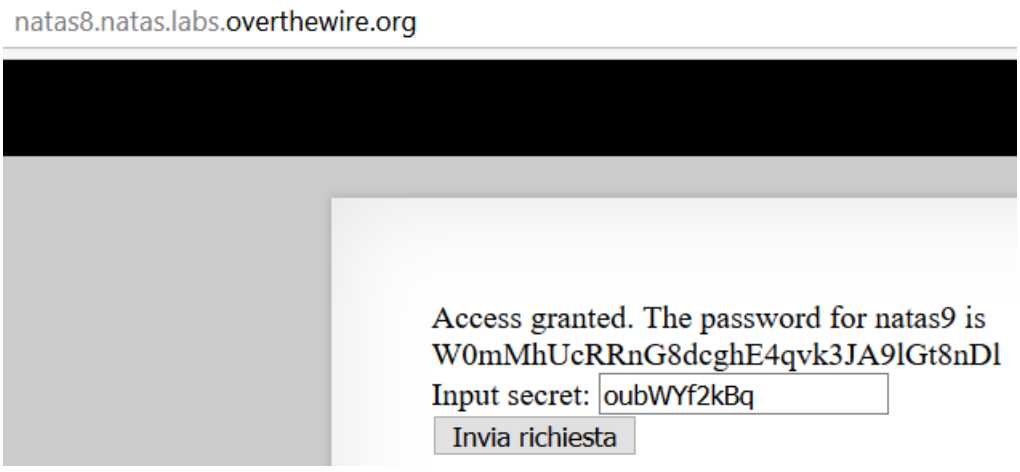

Figure 12: natas9 password found

Task 3 – HTTP Requests and Sessions

When I browsed to the website, using Burp I sent the request to the reapter in case I had to change something (as your hint was manage to authenticate and we haven't done yet SQL injection and also by looking to task title, I thought that there was something to change). In the repeater I noticed the cookies and when I overlapped on it I saw it was encrypted in base64 (as was another hint). Decoding it, the message was "authenticated=false". I had to change it to "authenticated=true", so I encoded in base64 this sentence and put the decoding on the cookie, sent to the repeater and found the flag.

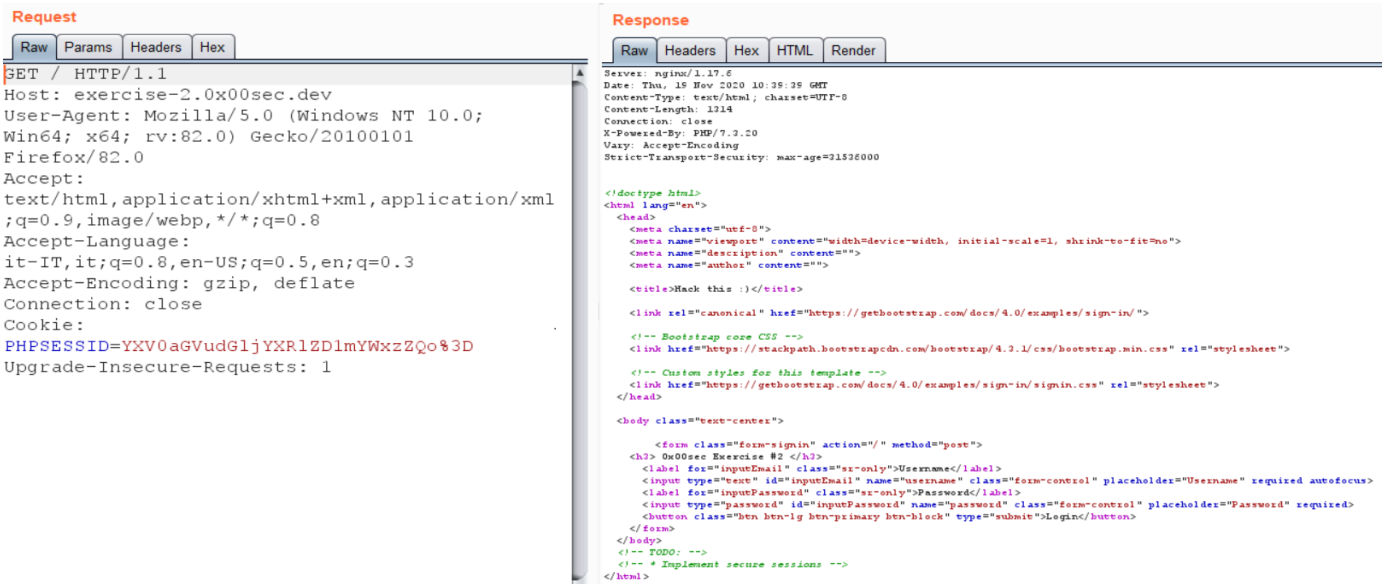

Figure 13: first request and relative response from Burp's repeater

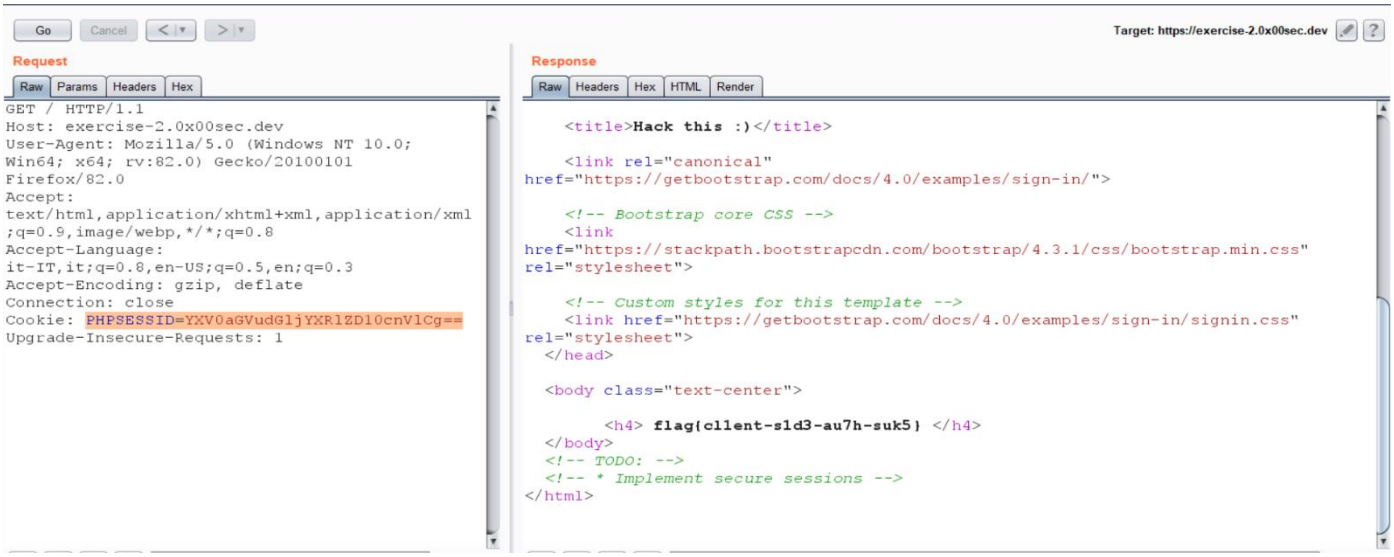

Figure 14: request and relative response from Burp's repeater after changing cookies

Why is vulnerable/insecure? In my opinion the vulnerability is that the cookie is in base64 which is not a secure cyphertext. What captured my attention is the word "PHPSESSID", so I searched in the web and found that is a keyword used by PHP to keep track of cookie sessions. If it is that, why is it so easy to reveal that value? So, in my opinion not only the base64 is a vulnerability but also the storage way even if now I have not yet the ability to know how to secure session.

Bonus Task – Requests/Responses with Python

In this exercise I had to manage requests and responses with Python instead of Burp for the last exercise in both assignment 1 and assignment 2. I proceeded in two different ways using urllib3 library:

- Method 1: I created a function that given the URL and the header to be modified just makes a request with the desired values and sends it. From the given response I extract the flag.

```
def findflag (url, headers, flagstart, flagend):
    import urllib3
    http = urllib3.PoolManager() #create http object
    req = http.request('GET', url, headers=headers) #make correct request
    resp = str(req.data) #read response
    flag = flagstart + str((resp.split(flagstart))[1].split(flagend)[0]) + flagend #check flag
    return flag
```

Figure 15: function that I made for method 1

- Method 2: I created two functions (one for each assignment) that automatically performs the all exercise. In the first assignment the function sends a basic request to the web page, reads the response and if it contains the unauthorized browser, changes the User Agent with the desired one. This is done by only giving to the function the URL and how the flag starts and ends. In the second assignment the function sends a simple request and reads the cookies, decoding them from base64 to ascii (first I check that the decoded cookie means something, I make the opposite and decrypt it). The function sends the changed request to the web page and in the response checks for the flag. Here again the only inputs are the URL and how the flag starts and ends.

```python
def autoflag1(url, flagstart, flagend):
    import urllib3
    http = urllib3.PoolManager() #first I create an http oject
    req = http.request('GET', url) #make the simple request
    if "not picobrowser!" in str(req.data): #check if incorrect browser in response
        header = {'User-Agent': 'picobrowser'} #create a dictionary to change user agent
    else:
        print("Error \n")
    req2 = http.request('GET', url, headers=header) #send request with correct user agent
    resp = str(req2.data) #read response
    flag = flagstart + str((resp.split(flagstart))[1].split(flagend)[0]) + flagend #find flag
    return flag
```

Figure 16: function that I made for method 2 assignment 1 using urllib3

```python
def autoflag2(url, flagstart, flagend):
    import urllib3
    import base64
    http = urllib3.PoolManager() #create http object
    req = http.request('GET', url) #make request
    cookie0 = str(req.info().get_all('Set-Cookie')) #read cookie
    cookie0value = ((cookie0.split("PHPSESSID="))[1].split(";")[0]).split("%")[0] + "==" #read cookie value
    #I had to change character %3D with character == otherwise decryption and encryption was incorrect
    decode_cookie0 = cookie0value.encode('ascii') #from line 22 to 24 I make cookie decryption
    decode_cookie0 = base64.b64decode(decode_cookie0)
    decode_cookie0 = decode_cookie0.decode('ascii')
    if str(decode_cookie0) == "authenticated=false\n": #check if message is "authenticated=false\n"
        encode_cookie1 = str("authenticated=true\n") #encode "authenticated=true\n", correct cookie value
        encode_cookie1 = encode_cookie1.encode('ascii') #from line 27 to 29 cookie encryption
        encode_cookie1 = base64.b64encode(encode_cookie1)
        encode_cookie1 = encode_cookie1.decode('ascii')
        cookie1 = 'PHPSESSID=' + str(encode_cookie1) #reconstruct cookie value
    else:
        print("Error \n")
    head2 = {'Cookie':cookie1} #create an header
    req2 = http.request('GET', url, headers=head2) #make request with correct cookie
    resp = str(req2.data) #read response
    flag = flagstart + str((resp.split(flagstart))[1].split(flagend)[0]) + flagend #find flag
    return flag
```

Figure 17: function that I made for method 2 assighment 2 using urllib3

After that I had the curiosity to know how much library “urllib3” changes from library “requests” (that I already knew). So, I decided to perform method 1 using requests library. I read that with requests I can work with simple requests or creating sessions, so I decided to try both and check the result. Even, I also noticed that the cookie change for assignment 2 could be done with a dictionary or creating an appropriate cookie object, I tried both.

```python
def findflag2(url, flagstart, flagend, head={}, session=0, cookiejar=0):
    import requests
    if cookiejar==1: #check if want to use cookiejar
        domain = input("Insert web page domain\n")
        path = input("Insert web page path \n")
        cookiename = input("Insert cookie name \n")
        cookievalue = input("Insert cookie value \n")
        cookies_jar = requests.cookies.RequestsCookieJar() #create cookiejar object
        cookies_jar.set(cookiename, cookievalue, domain=domain, path=path) #fill cookiejar
        header=cookies_jar #rename object
    elif cookiejar==0:
        header=head #if not uses cookiejar, the cookies are set from input as dictionary
    else:
        print("Incorrect number \n")
    if session==0:
        req = requests.get(url, headers=header) #make request
    elif session==1: #make request with using session
        sess = requests.Session()
        req = sess.get(url, headers=header)
    else:
        print("Invalid session\n")
    resp = str(req.text) #read response
    flag = flagstart + str((resp.split(flagstart))[1].split(flagend)[0]) + flagend #check flag
    return flag
```

Figure 18: find flag for both assignment 1 and assignment 2 but using requests library

```
Flag for assignment 1 exercise 3:  picoCTF{p1c0_s3cr3t_ag3nt_84f9c865}
Flag for assignment 2 exercise 3:  flag{cl1ent-s1d3-au7h-suk5}
The auto generated flag for assignment 1 exercise 3  picoCTF{p1c0_s3cr3t_ag3nt_84f9c865}
The auto generated flag for assignment 2 exercise 3  flag{cl1ent-s1d3-au7h-suk5}
Insert web page domain
exercise-2.0x00sec.dev
Insert web page path
/
Insert cookie name
Cookie
Insert cookie value
PHPSESSID=YXV0aGVudGljYXRlZD10cnVlCg==
Insert web page domain
exercise-2.0x00sec.dev
Insert web page path
/
Insert cookie name
Cookie
Insert cookie value
PHPSESSID=YXV0aGVudGljYXRlZD10cnVlCg==
Flag assignment 1 exercise 3 with requests  picoCTF{p1c0_s3cr3t_ag3nt_84f9c865}
Flag assignment 1 exercise 3 with session  picoCTF{p1c0_s3cr3t_ag3nt_84f9c865}
Flag assignment 2 exercise 3 with requests  flag{cl1ent-s1d3-au7h-suk5}
Flag assignment 2 exercise 3 with session  flag{cl1ent-s1d3-au7h-suk5}
Flag assignment 2 exercise 3 with requests and cookiejar  flag{cl1ent-s1d3-au7h-suk5}
Flag assignment 2 exercise 3 with session and cookiejar  flag{cl1ent-s1d3-au7h-suk5}
```

Figure 19: output from the code I wrote. First two are for method 1, second one are for method 2 and the last six are generated with requests library