

NetworkSec — The Tangled Web

Assignment Summary

The assignment consists of several challenges, each of which is a website that hides a secret URL. In order to win a particular challenge and pass to the next, you need to find the secret URL by exploiting vulnerabilities and misconfigurations of the Web site in question. Every challenge have been beaten successfully. The table below summarises the secret URLs that correspond to each challenge. The remaining sections of the report detail the steps I went through in order to beat each challenge. Please note that many of the failed attempts have been omitted for the sake of brevity.

Level n.	URL
1	removed
2	removed
3	removed
4	removed
5	removed
6	removed
7	removed
8	removed
9	removed
10	removed
11	removed
12	removed
13	removed
14	removed
Victory	won

Level 1

The (username, password) pair is validated client-side. Thus every needed information is accessible by any client that visits the vulnerable website. In particular, credentials are validated using a JS script.

By inspecting the HTML source code of the website's home page, I found the mentioned script (*verify.js*) and, once displayed its content through a GET request, I found the valid credentials (*SCRIPT, KIDDIE*) and consequently the secret URL.

secret: http://—

Level 2

The password required by the form was stored into an image. In order to find the password, I inspected the HTML source code of the website's home page and I found an image not visible just by watching at the rendered page. That hidden image (*scrt.png*) contained the correct password: *NINJAkitten*, which allowed me to beat the challenge and to obtain the secret URL.

secret: http://—

Level 3

The form of this website didn't even had a real password. Using BurpSuite for analysing the client-server packets exchange, I found a hidden form value (named *admin*) which allows to login if it is set to *True*. Briefly, the form didn't rely on valid credentials, rather, on a single "hidden" HTML value.


In order to bypass such a poor authentication mechanism, it's been enough to send a POST request with that hidden parameter set to *True*. I did so by changing the HTML value directly, even though it may have been performed using BurpSuite as well or any other similar tool.





secret: http://—

Level 4

The vulnerable website does answer only to requests coming from Russian (.ru) websites and to those requests that have 'Kevin Browser Rocks' as HTTP User-Agent header. This weak authentication technique can be bypassed just by modifying the mentioned header and the HTTP Referer header. In order to beat the challenge I used a Mozilla Firefox extension, named "Modify Header Value v0.1.5", for changing the User-Agent and the Referer headers. As shown by the screenshots below, I've been able to bypass the authentication and consequently to obtain the secret URL.

Enter a desired value for (URL), (Header Name) and (Header Value), then press Enter or click on the (+) button to add it to the Table.

URL	Header Name	Header Value	Add
url (i.e. https://www.google.com/ or *)	Referer	random.ru	

#	URL	Domain	Sub	Header Name	Add	Modify	Remove	Header Value	Disable	Delete
1		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Referer	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	random.ru		
2		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	User-Agent	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Kevin Browser Rocks		

secret: http://—

Level 5

Inspecting the source code of the rendered HomePage, I found a commented line which says "never show secret.txt to the users!" and indeed, as shown below, it is not possible to access the file using a standard GET request sent by a Web browser.

In order to access the secret, as the message says, it's necessary to change the HTTP Referer header to '[domain]/supersecurepage'. For doing so I used the same MozillaFirefox add-on of the previous challenge. I changed the Referer header of the GET request and then just by requesting the cgi-bin/secret.txt file, I've been able to obtain the secret and beat the challenge.

secret: http://—

Level 6

I found the valid credentials thanks to a weak password and due to a vulnerability which discloses what usernames are registered on the server: if the username is not registered then the website warns the client about that. Once I successfully logged in using '*admin*' as both the username and password value, I realised the Web server was suffering from another vulnerability: it trusts client-supplied input. Thus, by changing the '*uid*' GET parameter it's been possible to logon as an another user without necessarily being aware of the password or the username itself. Using BurpSuite I've been able to test several values for the mentioned parameter, which led me to find several usernames (such as Angelo, Davide, Andrei, Remco, Kevin, and Tsutomu) and the secret URL.

secret: http://—

Level 7

This website allows every guest user to login using their own e-mail address and, once a user is successfully authenticated, a cookie value (named *u*) is set by the server. After some tests I found the cookie consisted of the ROT13 (shift 13) encrypted value associated to the used e-mail address. Since the website gives the secret only to teachers, and it doesn't really care about the associate password, I logged on using one teacher's address (name@[domain_name]) in order to beat the challenge and access the secret URL.

secret: http://—

Level 8

As for the challenge n. 6, the same mentioned vulnerability led me to find the default administration credentials: (admin, admin). However, once authenticated I wasn't able to read any secret. I then went inspecting the cookies set by the server and as the previous challenge, I've found the username ROT13 (still with shift=13) encrypted value that was being set as a cookie. The second security flaw that allowed me to beat the challenge is the server trusting client-supplied inputs: particularly, cookie values. Replacing the cookie value set by the server with the corresponding ROT13 encrypted value (shift=13) of the '*messyadmin*' user (found in the homepage of the server) I beaten the challenge and obtained the secret URL. In order to change the cookie value set by the server, I used the BurpSuite software: I sent the GET request to the repeater and then I changed the required value.

secret: http://—

Level 9

Initially, I tried to guess the right password without any success. Only later, I inspected the source code of the Web server's homepage and I found a comment that says "Never show sessions.txt to the users!", this led me to visit the URL of that file. The file contained a username (admin) and a Cookie value (u=456b7016a916a4b178dd72b947c152b7). Using BurpSuite I replaced the cookie value sent by the browser with the one found on the leaked file and then I was able to obtain the secret URL. Naturally, the vulnerability exploited here is due to the disclosure of a valid admin cookie.

secret: http://—

Level 10

Even this web server validates credentials using JavaScript, which is executed by the browser Web. Inspecting the HTML homepage source code I found several usernames with their MD5 password hashes. I've cracked the admin hash (6eea9b7ef19179a06954edd0f6c05ceb) which corresponds to the password '*qwertyuiop*' and then I used the password together with the admin username for beating the challenge and obtaining the secret URL. One more time the

authentication process is performed by the client's browser, which in turns allow any malicious user to (potentially) crack the hash and bypass the authentication procedure.

secret: http://—

Level 11

Even though using an integer as the only authentication factor is not a good practice, what is worst is the fact that the website explicitly require a integer which belongs to a particular range and the authentication is still performed client side. The range in question is visible by inspecting the source code of the website's homepage. By using the following javascript piece of code, executed into any browser's console (I used the last available version of Safari for this challenge), I've been able to try every possible value, thus guessing the correct one and obtaining the secret URL.

```
var i;
for(i=0; i<999999; i++){ var hash = hex_md5('localsalt' + i); if(hash ==
'c1d0b2864ecdafb8650201ce3537b41f'){ console.log(i); console.log("done!!!"); break;}}
```

The correct integer value is 444400.

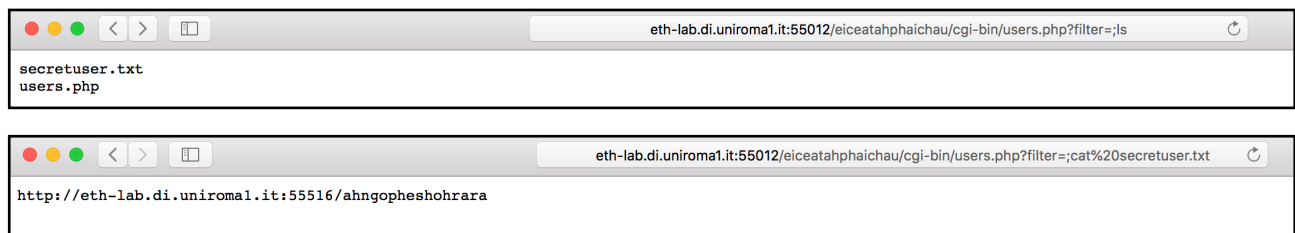
secret: http://—

Level 12

The website's home page shows a list of usernames which seemed to be retrieved directly from a Linux operating system(eg `cat /etc/passwd |cut -d ":" -f 1`). However, I had to spend some time in order to figure out how exactly those values were retrieved. Once realised that the server was using a CGI PHP (`cgi-bin/users.php`) script, specifically using the '*filter*' GET parameter, I've been able to execute arbitrary command-line utilities as if I had a shell on that server. Just by adding a '`;`[utility]'' as *filter* CGI parameter, I've been able to exploit the command execution vulnerability that affects the involved webserver.

Initially, I listed the content of the directory where the CGI script were located. However, the identified `secretuser.txt` file weren't accessible just by a GET

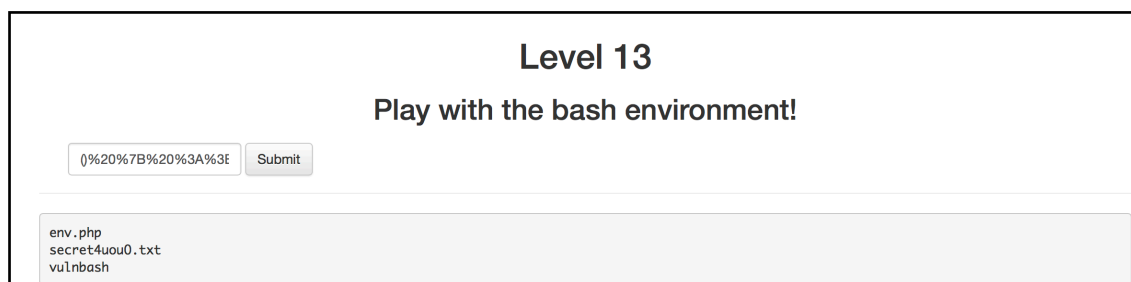
to its URL. I've been able to display its content using the `cat` command-line utility in the same way I've executed the `ls` command. Details are summarised by the following screenshots.



secret: http://—

Level 13

The vulnerability exploited for beating this challenge is the “Bash Bug”, also known as ShellShock vulnerability. It is a serious security vulnerability which, if exposed to the Internet, may lead to a complete remote control of the affected system and if exploited locally, it may lead to several other issues such as privilege escalation. Regarding the challenge's website, I've been able to execute arbitrary commands and consequently to obtain the secret URL. Details are depicted by the next two screenshots.



For

listing the directory content of the CGI script:

`()%20%7B%20%3A%3B%7D%20%3B%20ls`

For displaying the secret (note: the dot had to be encoded too):

`()%20%7B%20%3A%3B%7D%20%3B%20cat%20secret4uouO%2etxt`

secret: http://—

Level 14

The website of the last challenge is affected by a SQL injection vulnerability. I started by enumerating the web server and I found a SQL injection vulnerability affecting the create user procedure. In particular, the username parameter is not properly validated/sanitised, thus allowing any malicious user to complete the query that checks for existing usernames as he wishes.

Using “random' union select password,2,3,4,5,6 from users;#” as username value I obtained the passwords of both the users kevin and tsutomu. Reading the blogs already stored into the server’s database seemed that tsutomu blog was hiding something important. Once I logged in as tsutomu user using the password ‘kevinmitnickisanarrogantbastard’ found by exploiting the SQLi vulnerability I’ve been able to get the secret URL and consequently to beat the last challenge, as detailed below.

We found users already registered with this name

Username: asd
First: 3
Last: 4

Username: asdasd
First: 3
Last: 4

Username: iamthegreatesthackerofall!
First: 3
Last: 4

Username: kevinmitnickisanarrogantbastard
First: 3
Last: 4

An error occurred:
A user with the specified name already exists.

Final secret: http://—