Challenge URL: /dvwa/vulnerabilities/brute/
Objective: Brute force the administrator's password.
Bonus objective: Find the other four users' usernames and brute force each password.
Tools needed: Hydra, a remote shell from Challenge 4 (optional)

Did you remember to read this section's README?

The Guide

The Administrator's Account

Okay, let's navigate to the challenge. We are presented with a classic username/password web form.

We're obviously assuming that we don't know the administrator's password, though I know we do. That's what we used to log into DVWA, after all. However, for convenience, we won't try and brute force the administrator's username, which is "admin". I tried it for fun, but it took way too much time and I feel that's not what the spirit of the challenge is. So our objective is to find the password for "admin".

I don't know about you, Dear Reader, but I'm not manually brute forcing anything! So let's take a quick step back and examine what tools Kali has to automate brute forcing. Under the Kali menu "Applications" > "05 - Password Attacks", we see plenty - John, Medusa, Ophcrack, pyrit. I personally prefer the command-line cracker Hydra, so let's try that.

But what arguments do we need to pass to Hydra? I'm guessing we'll need some, right? Let's try running the command `hydra` in the terminal and find out!

All right! I'll save us all some time and say which ones we need to know about:

- `-l [username]` : The username we want to attack. In our case, we don't need to pass in a file. We can just specify "admin" as a string.
- `-P [wordlist_file]` : The password we want to try. We don't know what this is yet, so we'll pull in a wordlist. More on that below.
- `-s [port]` : The port our service is on. This should be port 80, as we didn't slap an SSL certificate on our web server. Yours may vary depending on how you set up your web server.
- `[service]` : The kind of service (ie: what kind of HTTP request) this form uses. More below.
- `[host]` : Our target web server. Mine is "dvwa".
- `[target_URI]` : The relative path to the vulnerable form, with some additional arguments formatted in a specific way. More below.

Let's define the missing parts:

- `-P [wordlist_file]` : In Kali, wordlists are stored at **/usr/share/wordlists/**. I think "rockyou" is a phenomenal list, so let's extract the TXT file in "rockyou.tar.gz" to our working directory.
- `[service]` : We can find the service in multiple ways:
    - Examining the client-side source code in our browser (Right-click > "Inspect Element");

    - Clicking the "View Source" button on the bottom right of the page, which pulls the server's PHP file; or

    - Analyzing the traffic in Burp Suite. It's good practice if you don't know Burp well, but it's overkill for our purposes.

    We can see multiple instances of the word "GET", which makes it clear that `[service]` should be **http-get-form**!

- `[target_URI]`: Formatting this is tricky, so I'll tell you that this part will require three parts, separated by "**:**":
    1. **The full target URL**. We know the exact URL of the form already: **/dvwa/vulnerabilities/brute/index.php**.
    2. **Any parameters**. If we view the client-side source code above, we can see two parameters listed: **username** and **password**. We also see an action, **Login**, which we have to pass in so the form knows what to do with our data.
    3. **Failed login message**. We need to pass in a way for Hydra to know if a password isn't valid. We can replicate this by giving false credentials to the form, thereby telling us that the failed login message is "**Username and/or password incorrect.**".

Combining all that together, we get our final command.

```
hydra -l admin -P rockyou.txt -s 80 dvwa http-get-form "/dvwa/vulnerabilities/brute/index.php:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.
```

If it works, our successful username/password pair will be highlighted in green. Let's run it and see what happens!

Wait, why did we get 16 valid pairs?! Take a second and see if you can find out why.

...

All set?

Here's what happened: When we ran our command, we didn't actually give a way for Hydra to authenticate to our target page. Hence, without us knowing, Hydra actually ran the attack unauthenticated, which means the attack ran againt **/dvwa/login.php**. Since the failed-login error message for that page isn't the same as our target, Hydra can't really tell what a valid credential set is. So it just throws out the first 16 passwords in rockyou.txt at random for each of Hydra's 16 process threads (see the Hydra parameter screenshot above for why it's 16).

So is there a way we can give or trick Hydra into using a valid session, so that it can "see" the actual target?

We know cookies are often used in web authentication, so let's start by examining the cookies our browser has. You can do this by opening the Firefox developer web console (Control+Shift+K) and entering the string `document.cookie`. We see two fields, **security** and **PHPSESSID**. **security** is our current security setting, "low". **PHPSESSID** is our session ID, which is actually what we need!

Let's re-run our command, passing our cookie in the way Hydra expects:

```
hydra -l admin -P rockyou.txt -s 80 dvwa http-get-form "/dvwa/vulnerabilities/brute/index.php:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie: security=low; PHPSESSID=[your_value_here]"
```

Behold!

Let's test our cracked password in the form:

We got the success message. Nice! But we're not quite done yet ...

The Other Accounts

So let's examine that weird white box (or actual image if your setup was ~~better~~different than mine) below the success message. If we inspect that element as before, we see an interesting line: `<img src="./hackable/users/admin.jpg">`. We can logically assume if we try to visit that **/users/** directory, we may find the other usernames in the format **[username].jpg**. So after some playing around with the URL in Firefox, we find the directory.

Other possible attack vectors include uploading a reverse shell in Challenge 4, SQL injection, or passing in a wordlist of all possible usernames and brute forcing that at the same time as the passwords.

Regardless of how you found them, we find out that the other four usernames are:

- "smithy"
- "gordonb"
- "pablo"
- "1337"

With this, we can easily modify our earlier Hydra command. We only need to make two changes. We capitalize the `-l` switch to `-L` to indicate we're now passing in a file. We then actually pass in a text file with all the usernames instead of the string "admin" (I called mine "users.txt").

```
hydra -L users.txt -P rockyou.txt -s 80 dvwa http-get-form "/dvwa/vulnerabilities/brute/index.php:username=^USER^&password=^PASS^&Login=Login:Username and/or password incorrect.:H=Cookie: security=low; PHPSESSID=[your_value_here]"
```

Let's see the result.

We can test each as before, and they all give us a similar success message.

Challenge complete. Great work!