Joshua Lane
CYSE450 Section 23190
11/08/2022

Laboratory Exercise – Metasploitable CTF Part 5 (Exercise J5)

1. Overview

In this lesson, students will learn how to analyze a web application to discover a hidden flag. They will use Burp Suite to capture requests, write a bash script to capture a file before it is deleted, write a deobfuscation script in Ruby, and base64 decode in the terminal.

2. Resources required

This exercise requires the Kali Linux with Metasploitable (2020.09) running in the Cyber Range.

3. Initial Setup

For this exercise, you will log in to your Cyber Range account, select the Kali Linux with Metasploitable (2020.09) environment, click “start” to start your environment, and then “join” to get to your Linux desktop.

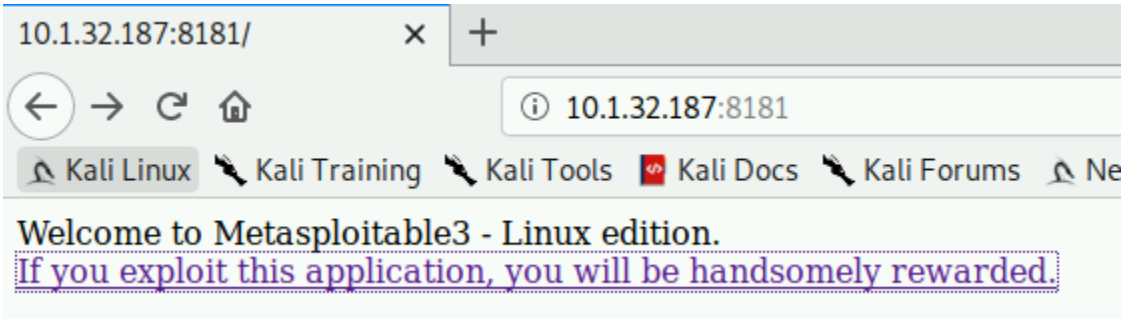
4. Tasks

Task 1: Find the 6_of_Clubs Flag

From our previous scans we saw that port 8181 was open and running a WEBrick 1.3.1 Ruby 2.3.7 version.

```
msf5 > workspace metasploitable
[*] Workspace: metasploitable
msf5 > services
Services
========

host         port  proto  name          state     info
----         ----  -----  ----          -----     ----
10.1.32.187  21    tcp    ftp           open      ProFTPD 1.3.5
10.1.32.187  22    tcp    ssh           open      OpenSSH 6.6.1p1 Ubuntu 2ubuntu2 Ubuntu Linux; protocol 2.0
10.1.32.187  80    tcp    http          open      Apache httpd 2.4.7
10.1.32.187  137   tcp    netbios-ns    filtered
10.1.32.187  139   tcp    netbios-ssn   filtered
10.1.32.187  445   tcp    netbios-ssn   open      Samba smbd 3.X - 4.X workgroup: WORKGROUP
10.1.32.187  631   tcp    ipp           open      CUPS 1.7
10.1.32.187  3000  tcp    ppp           closed
10.1.32.187  3306  tcp    mysql         open      MySQL unauthorized
10.1.32.187  3500  tcp    http          open      WEBrick httpd 1.3.1 Ruby 2.3.7 (2018-03-28)
10.1.32.187  6607  tcp                  filtered
10.1.32.187  6667  tcp    irc           filtered
10.1.32.187  6697  tcp    irc           open      UnrealIRCd
10.1.32.187  8181  tcp    http          open      WEBrick httpd 1.3.1 Ruby 2.3.7 (2018-03-28)
10.1.32.187  8484  tcp    unknown       filtered
10.1.32.187  8585  tcp                  filtered
10.1.32.187  8989  tcp    sunwebadmins  filtered
10.1.32.187  9200  tcp    wap-wsp       filtered

msf5 >
```

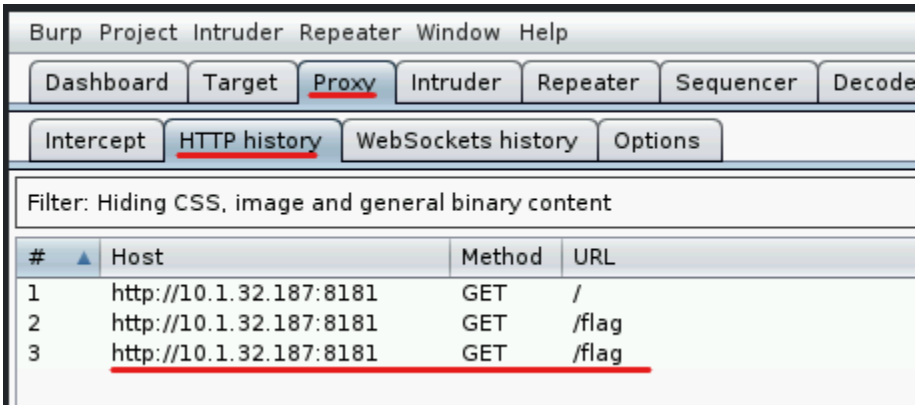
- Start Burp Suite and turn the intercept off.
- (If not already added) In a browser, install the FoxyProxy add-on and set the proxy up for Burp Suite as we have in previous lessons. (See Module 6, Lesson 3 Web App Manual Recon with Burp Task 1.)
- In the browser, navigate to <metasploitable ip>:8181 and press enter.
- Click the link on the web page.



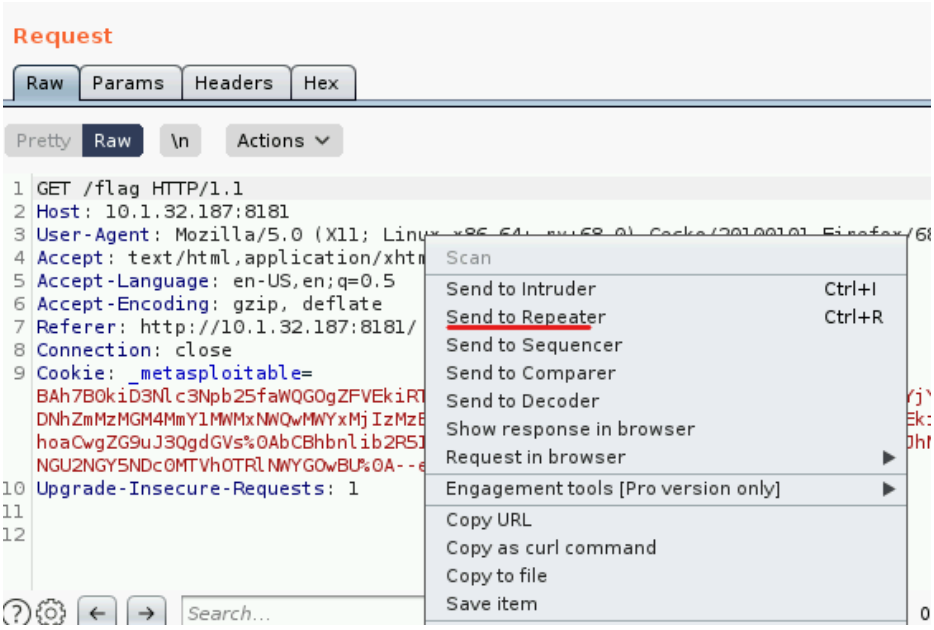
- In Burp Suite, click the **Proxy** tab and the **HTTP history** tab.

Here you see the webpage we visited in the browser.

- Click on the HTTP History URL that contains the /flag.



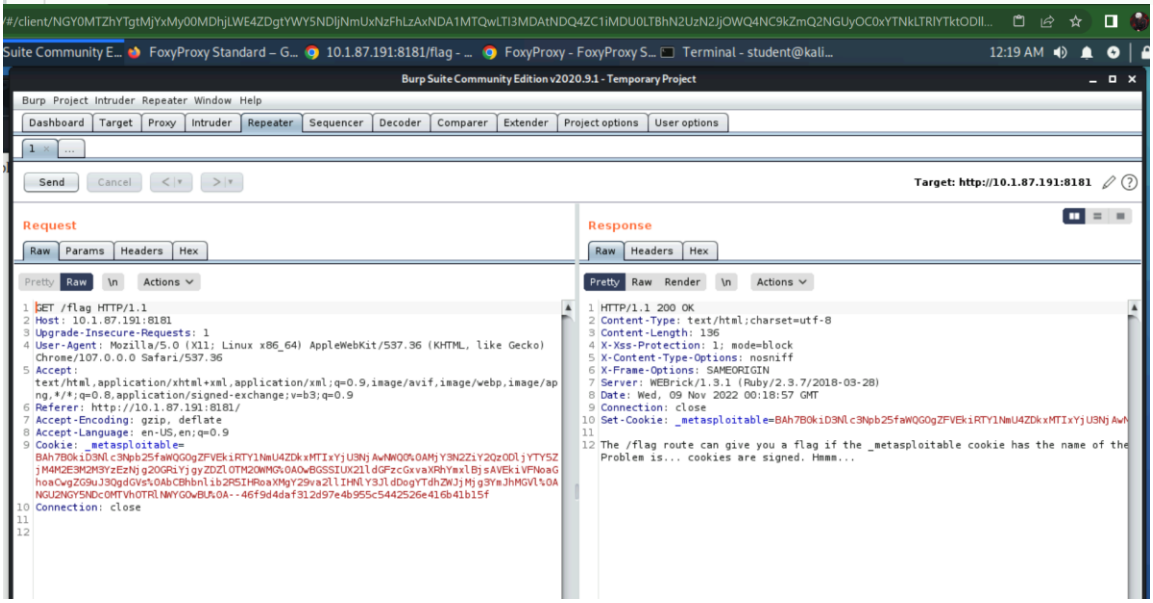
- Right click on the Raw request and then on the **Send to Repeater**.



- Click on the **Repeater** tab and click the **Send** button to make the request.

Looking at this response, we can see the cookie is being passed as a base64 encoded string using an old Ruby technique. You may not recognize this as a newcomer; however, the **=BAh** is the first hint for Ruby and Base64. You may have also noticed from previous lessons that Ruby was running on the web application server (WEBrick).

```
HTTP/1.1 200 OK
Content-Type: text/html;charset=utf-8
Content-Length: 136
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Server: WEBrick/1.3.1 (Ruby/2.3.7/2018-03-28)
Date: Thu, 15 Jul 2021 01:50:51 GMT
Connection: close
Set-Cookie: _metasploitable=BAh7B0kiD3Nlc3Npb25faWQGOgZFVEkiRTdlMTA1ZDYxNjA5NDZhOGJkZWI0%0ANTBhND

The /flag route can give you a flag if the _metasploitable cookie has the name of the flag.<br>
Problem is... cookies are signed. Hmmm...
```

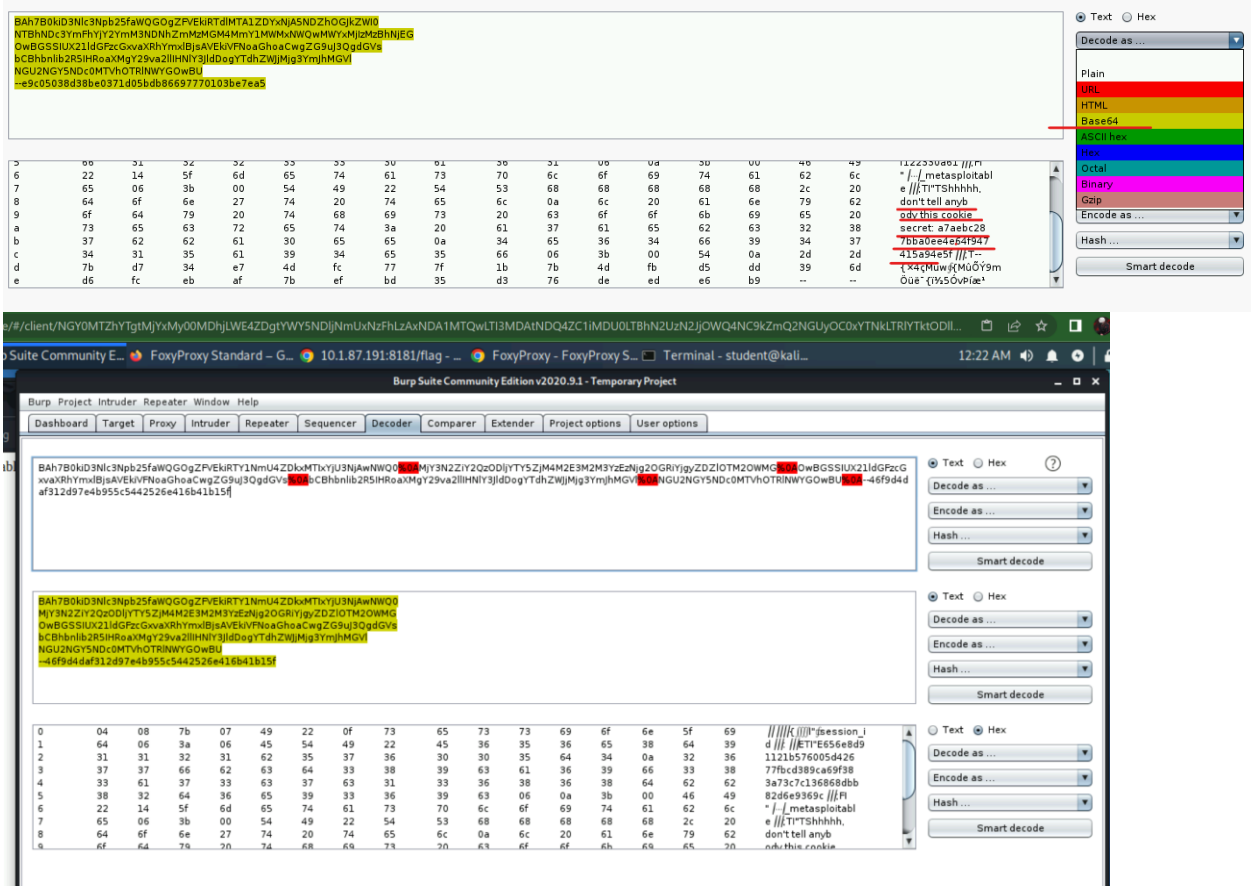


Here is my screenshot for the above steps.

- In the Request, select all the red text starting with and after Bah, right click on it, and then click **Send to Decoder**.
- Click the **Decoder** tab and click **Smart decode**.

- Now click the **Decode as** drop down and choose **Base64**.

This appears to be the key to decode the cookie.



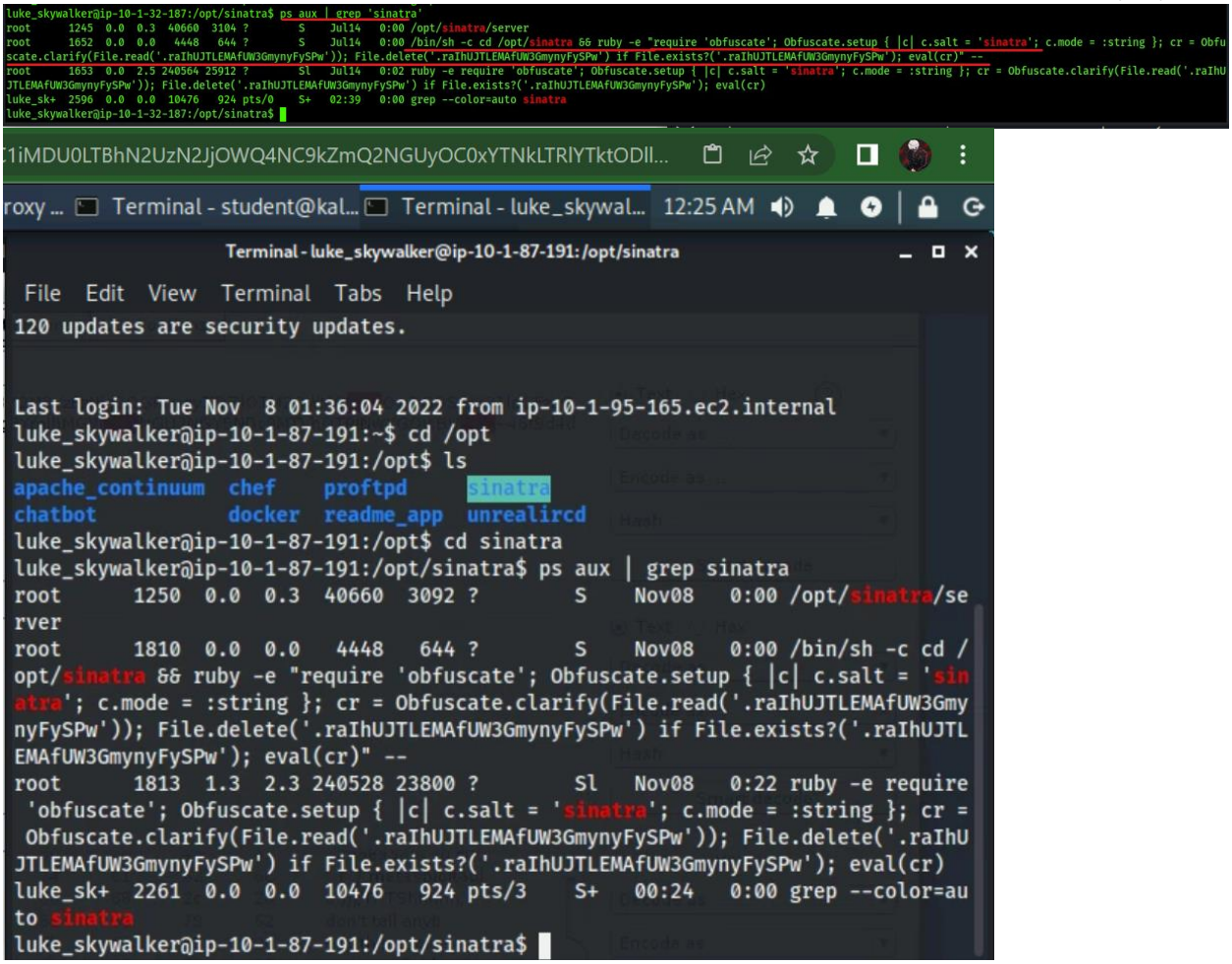
Here is my screenshot for decoding the base64.

This particular method of grabbing the flag can be tricky. We would have to use the extracted key, add 6_of_clubs to the _metasploitable cookie, sign the cookie, and then send the request. There is another way to grab the flag because we have access to the application on the metasploitable box itself. Let's see what else we can find to solve this flag. Logon as luke_skywalker to see what we can find about the Ruby web application.

- In a root terminal, log on as luke_skywalker using the **ssh** command as completed in previous lessons.
- Type **`cd /opt`** and press enter.

Notice the sinatra directory. Sinatra is an open-source Ruby web application library. To see all running processes that are being executed by the sinatra library, we can use the **ps aux** command. You can visit the following site for a better understanding of the **ps aux** command if you are not familiar with it:
https://www.computernetworkingnotes.com/linux-tutorials/ps-aux-command-and-ps-command-explained.html&sa=D&source=editors&ust=1626829763004000&usg=AOvVaw1ITpucA7Y8cXehZNahZ7Vq

- In the SSH terminal, type **`cd sinatra`** and press enter.
- In the SSH terminal, type **`ps aux | grep sinatra`** and press enter.



Here is my screenshot after logging into luke_skywalker and using the above commands.

Wow, this is a great hint. The output that is important reads:

```
"require 'obfuscate'; Obfuscate.setup { |c| c.salt = 'sinatra'; c.mode =
:string }; cr = Obfuscate.clarify(File.read('.raIhUJTLEMAfUW3GmynyFySPw'));
File.delete('.raIhUJTLEMAfUW3GmynyFySPw') if
File.exists?('.raIhUJTLEMAfUW3GmynyFySPw'); eval(cr)" --
```

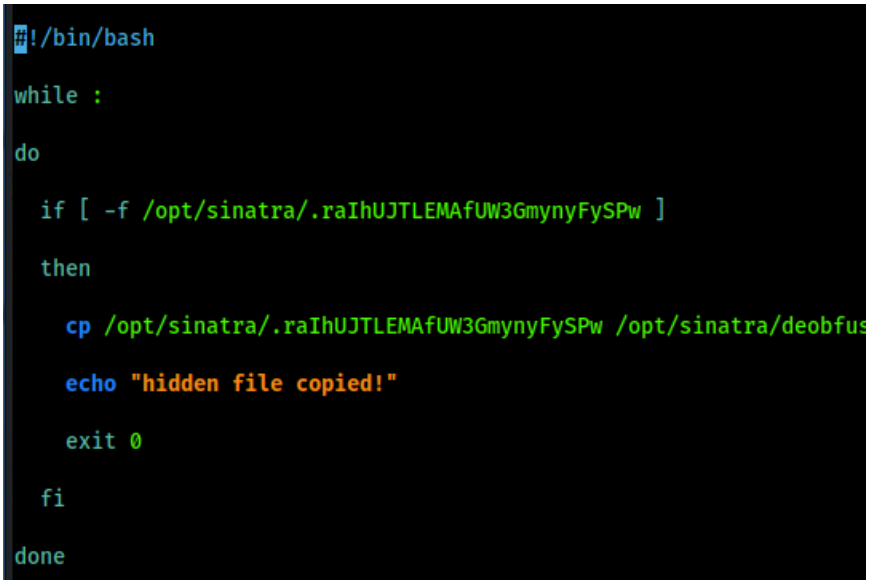
This is the Ruby code for obfuscating the hidden file `.raIhUJTLEMAfUW3GmynyFySPw`. I am betting this is our flag. The code then deletes the hidden file and this is why we can't see it. We need to grab the file before it is deleted. Once we have the file, we can deobfuscate it with this code. For more information, the Ruby Obfuscator github repository can be found here.

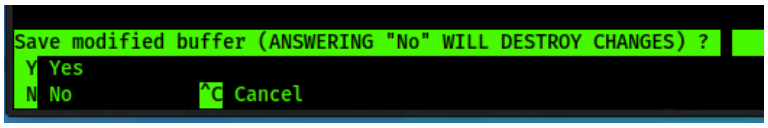
We first need to obtain the file. Since we know sinatra is the service creating the file, we can create a quick bash script to copy the file when it is seen.

- On the metasploitable ssh terminal, type **`mkdir deobfuscate`** and press enter.
- Type **`cd deobfuscate`** and press enter.
- Type **`nano flagfile.txt`** and press enter.
- Copy and paste the following script:

```
#!/bin/bash
while :
do
  if [ -f /opt/sinatra/.raIhUJTLEMAfUW3GmynyFySPw ]
  then
      cp /opt/sinatra/.raIhUJTLEMAfUW3GmynyFySPw
/opt/sinatra/deobfuscate
      echo "Hidden file copied!"
      exit 0
  fi
done
```

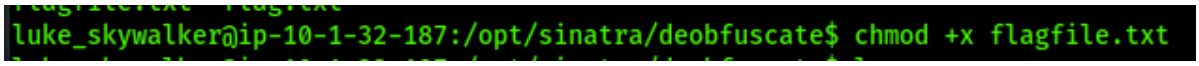
- Press `CTRL+X` and when prompted to save type **`Y`** and press enter.

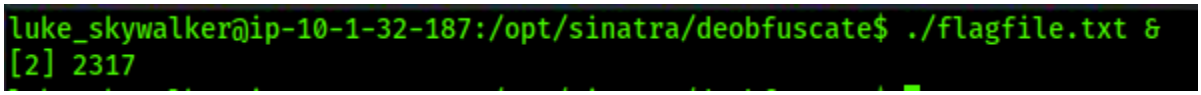




We need to make the script executable and have it run in the background.

- Type **`chmod +x flagfile.txt`** and press enter.
- Type **`./flagfile.txt &`** and press enter.





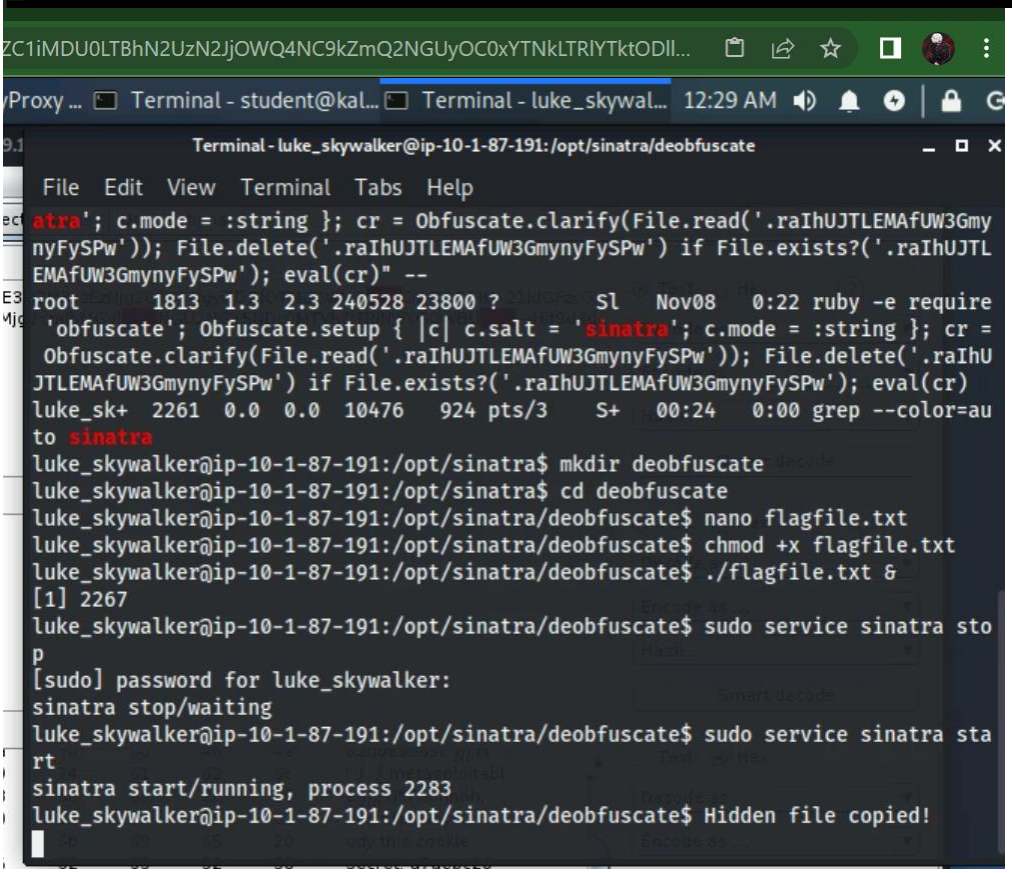
Before the script will fire, we need to stop and start the sinatra service.

- Type **`sudo service sinatra stop`** and press enter.
- You may need to type the password for luke_skywalker and press enter.
- Type **`sudo service sinatra start`** and press enter.

You will see a prompt "***hidden file copied!***"

- Press **`CTRL+C`** to exit the prompt from the script.

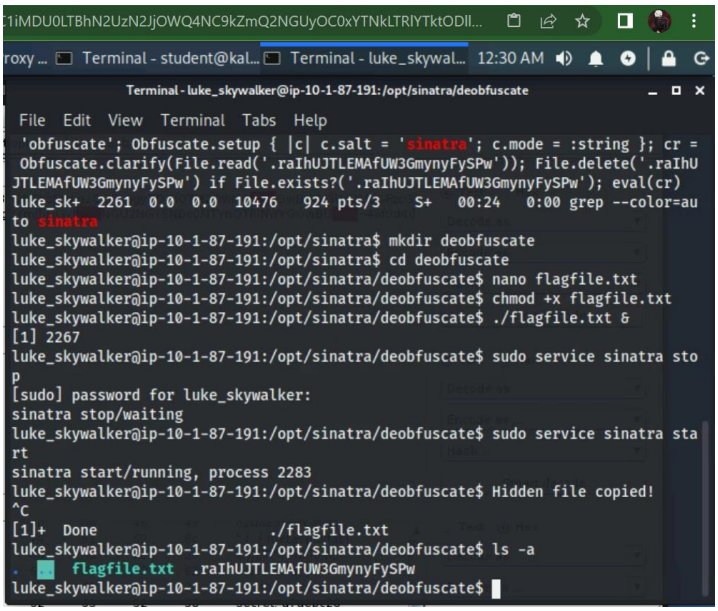
```
luke_skywalker@ip-10-1-32-187:/opt/sinatra/deobfuscate$ sudo service sinatra stop
stop: Unknown instance:
luke_skywalker@ip-10-1-32-187:/opt/sinatra/deobfuscate$ sudo service sinatra start
sinatra start/running, process 2385
luke_skywalker@ip-10-1-32-187:/opt/sinatra/deobfuscate$ hidden file copied!
```



Here is my hidden file copied.

- Type **`ls -a`** and press enter.

```
luke_skywalker@ip-10-1-32-187:/opt/sinatra/deobfuscate$ ls -a
.  ..  flagfile.txt  flag.txt  .raIhUJTLEMAfUW3GmynyFySPw
luke_skywalker@ip-10-1-32-187:/opt/sinatra/deobfuscate$
```



Here you can see my ls -a command.

Now we need to create a script to deobfuscate the code. This script was given to us when we found the sinatra process running using the ***ps aux | grep sinatra*** command. In our case, however, we are not going to add the delete file section of the script and we need to direct the "Obfuscate.clarify" to the location of our file.

- Open a text editor on the Kali VM.
- Copy and paste the following code and save it as **deobfuscate.rb** to your **flags** directory.

CODE:

```
require 'obfuscate'
Obfuscate.setup do |c|
c.salt = 'sinatra'
c.mode = :string
end
cr =
Obfuscate.clarify(File.read('/opt/sinatra/deobfuscate/.raIhUJTLEMAfUW3GmynyFy
SPw'))
print cr
```

- In a new root terminal, change directory to **flags** and copy the deobfuscate.rb to the metasploitable system by typing:

`scp deobfuscate.rb luke_skywalker@<metasploitable ip>:/opt/sinatra` and pressing enter.

```
ZarRonRoot@/Flags$:scp deobfuscate.rb luke_skywalker@10.1.32.187:/opt/sinatra
luke_skywalker@10.1.32.187's password:
deobfuscate.rb
ZarRonRoot@/Flags$:
```

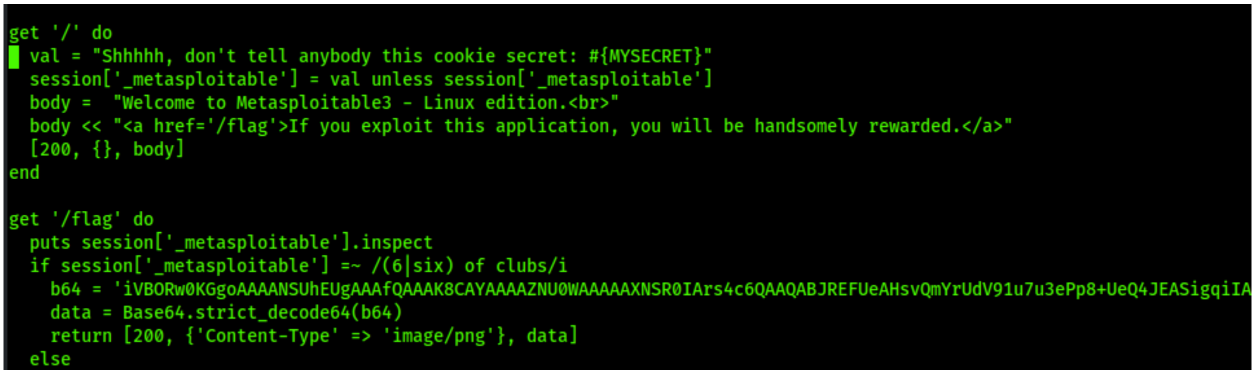
- In the metasploitable ssh terminal, be sure you are in the sinatra directory; if not, type **`cd /opt/sinatra`** and press enter.
- Type **`ruby deobfuscate.rb > /opt/sinatra/deobfuscate/flag.txt`** and press enter.

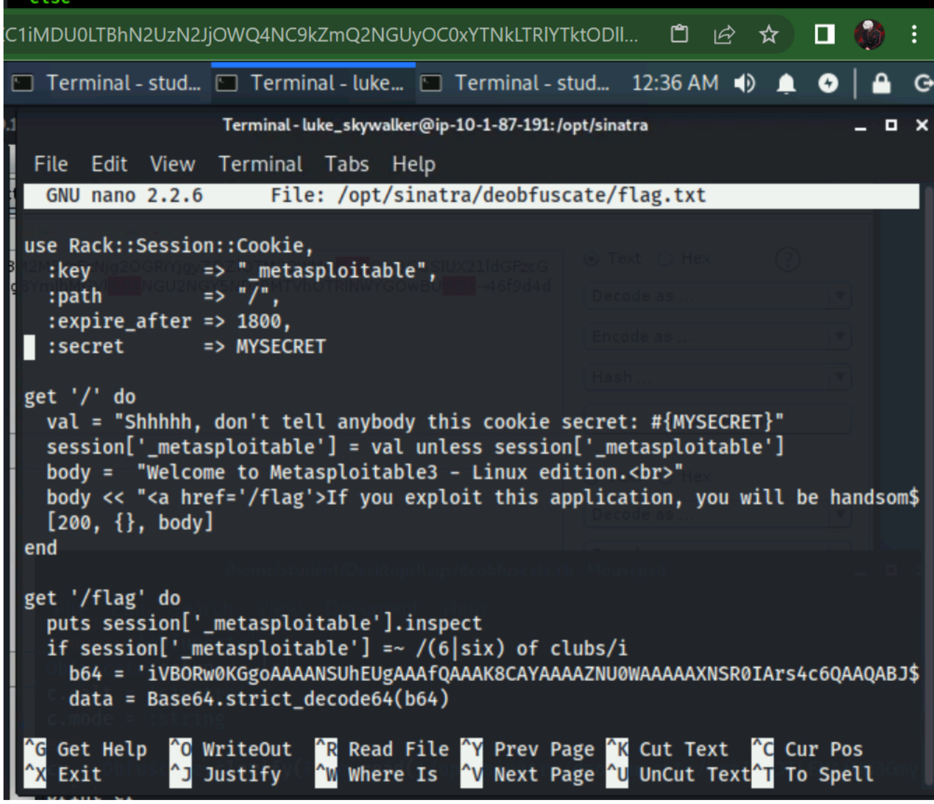
```
luke_skywalker@ip-10-1-32-187:/opt/sinatra$ ruby deobfuscate.rb > /opt/sinatra/deobfuscate/flag.txt
```

- Type **`nano /opt/sinatra/deobfuscate/flag.txt`** and press enter.

You can mouse scroll to analyze the whole file.

```
#!/usr/bin/env ruby

require 'sinatra'
require 'erubis'
require 'active_support'
require 'webrick'
require 'base64'

MYSECRET = 'a7aebc287bba0ee4e64f947415a94e5f'

set :environment, :development
set :bind, '0.0.0.0'
set :port, 8181

# These settings are specific for Sinatra 2.0.0rc2
set :logging, false
set :quiet, true
dev_null = WEBrick::Log::new("/dev/null", 7)
set :server_settings, {:Logger => dev_null, :AccessLog => dev_null}

use Rack::Session::Cookie,
  :key          => "_metasploitable",
  :path         => "/",
  :expire_after => 1800,
  :secret       => MYSECRET
```



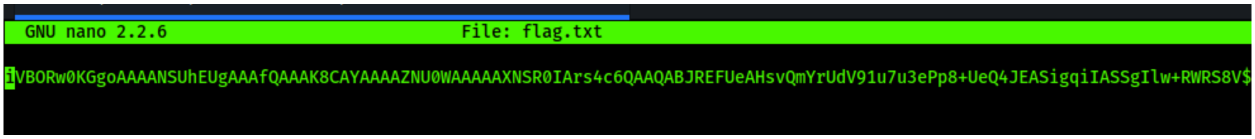


Here is showing the message in the file.

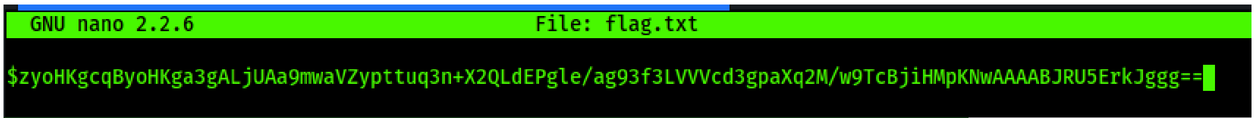
Now we can see the same cookie secret we discovered by using the base64 decode when we captured the request earlier using Burp Suite. We also see the base64 blob that contains the flag. We have to extract only the base64 code so we can decode the png. This can be tricky in the SSH terminal.

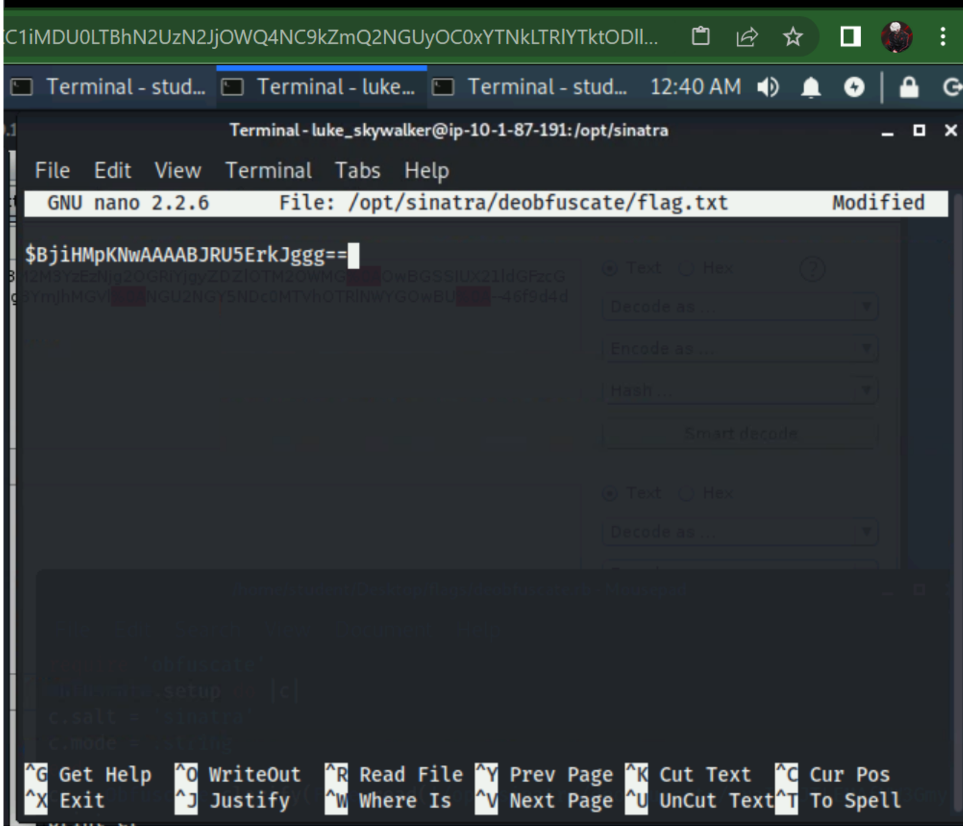
- Delete all the lines except for the line that contains b64. There are several ways of doing this; however, what worked for me was using the arrow keys and CTRL+K to delete each line.
- Delete everything before the i on the b64 line using the arrow keys and the delete button.
- Delete everything after == at the end of the line. To get to the end of the line press CTRL+E.

Beginning of the line:



End of the line:



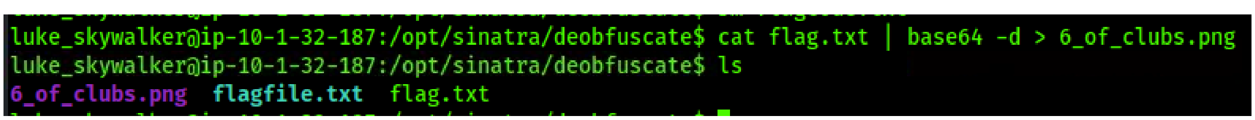


Here is my screenshot of the end of the line.

- Press CTRL+X.
- Type **Y** and press enter.
- Type **`cd deobfuscate`** and press enter.

Now we can decode the base64 png.

- Type **`cat flag.txt | base64 -d > 6_of_clubs.png`** and press enter.

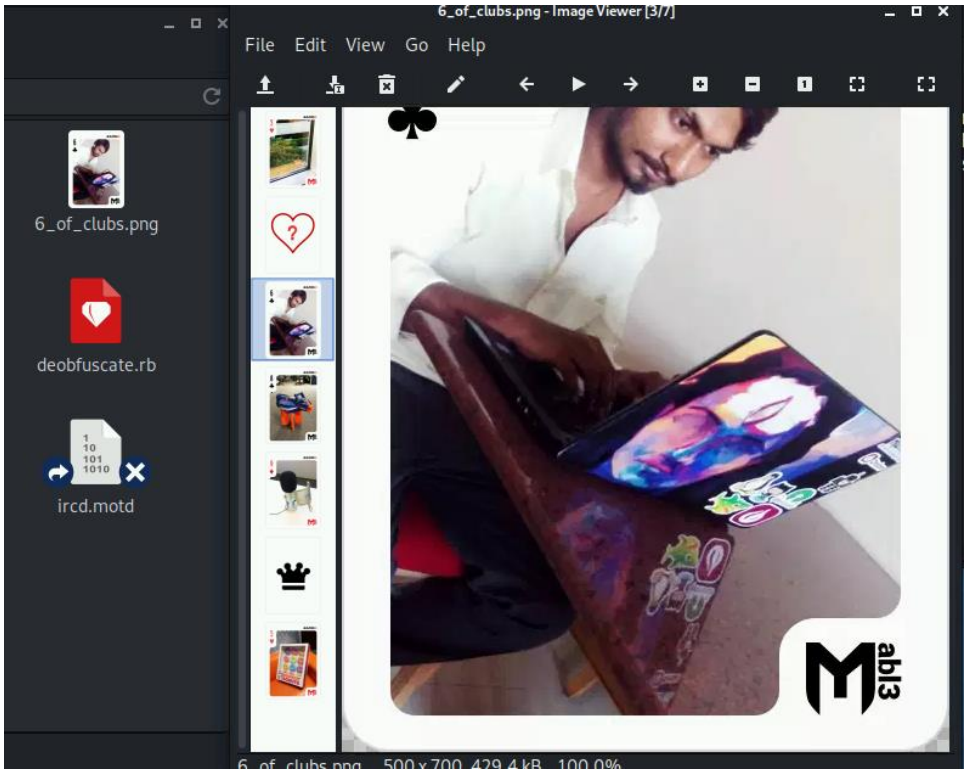


To see the flag, we will have to transfer it to the Kali VM.

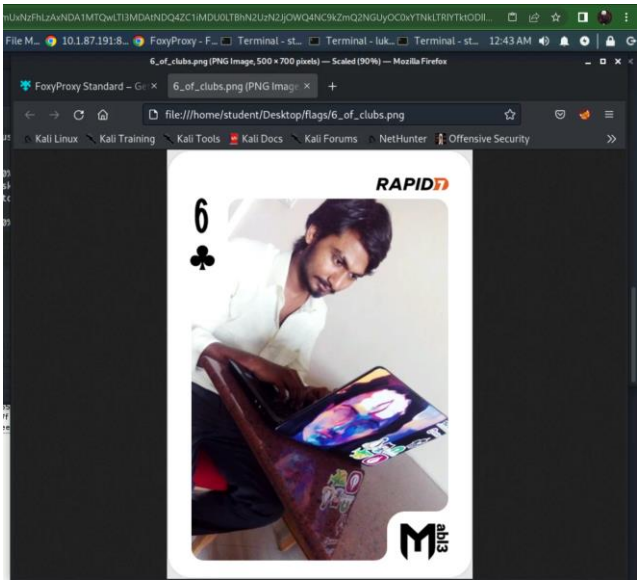
- In the Kali VM terminal, type
`scp luke_skywalker@<metasploitable ip>:/opt/sinatra/deobfuscate/6_of_clubs.png /home/student/Desktop/flags/` and press enter.

```
ZarRonRoot@/flags$:scp luke_skywalker@10.1.32.187:/opt/sinatra/deobfuscate/6_of_clubs.png  /home/student/Desktop/flags/
luke_skywalker@10.1.32.187's password:
6_of_clubs.png                                100%  419KB  79.6MB/s   00:00
ZarRonRoot@/flags$:
```

To open the flag, navigate to the **flags** folder and double click the **6_of_clubs.png** file. Congratulations, you have successfully obtained the flag!



MY SCREENSHOT OF THE FLAG IS ON NEXT PAGE



Here you can see the screenshot of the 6 of clubs flag.

In this lesson, you learned how to analyze a web application to discover a hidden flag. You used Burp Suite to capture requests and decode a cookie, analyzed a running web application using the ps aux command, wrote a bash script to capture a file before it was deleted, wrote a deobfuscation script in Ruby, and base64 decoded an image in the terminal.