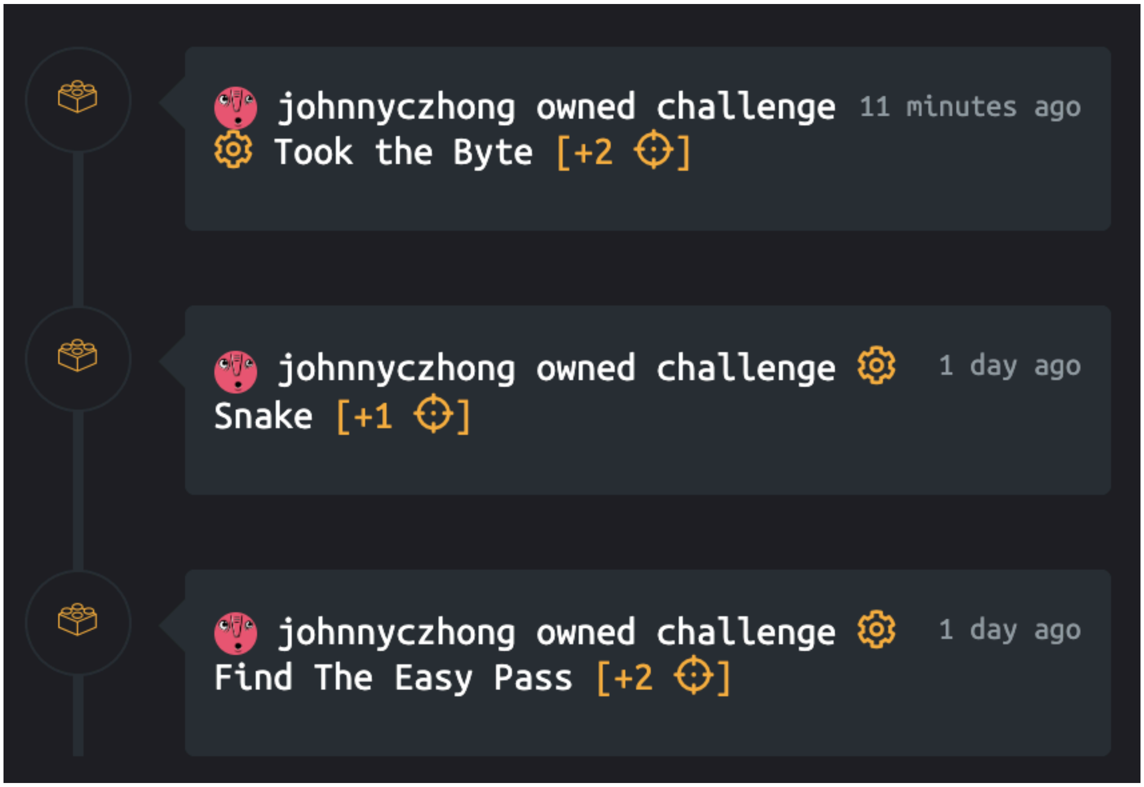
Johnny Zhong

CS373 – Defense Against the Dark Arts

Aug 12 2019

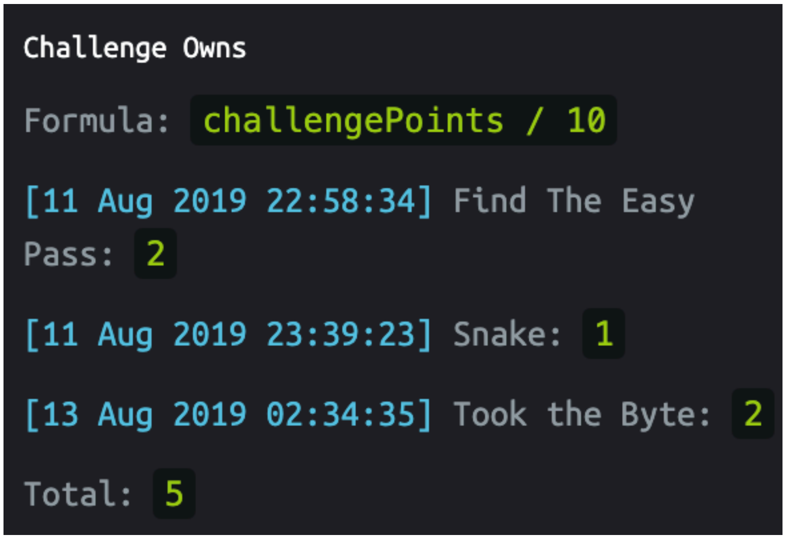
Hack the Box Assignments



johnnyczhong owned challenge 11 minutes ago
⚙ Took the Byte [+2 ⌖]

johnnyczhong owned challenge ⚙ 1 day ago
Snake [+1 ⌖]

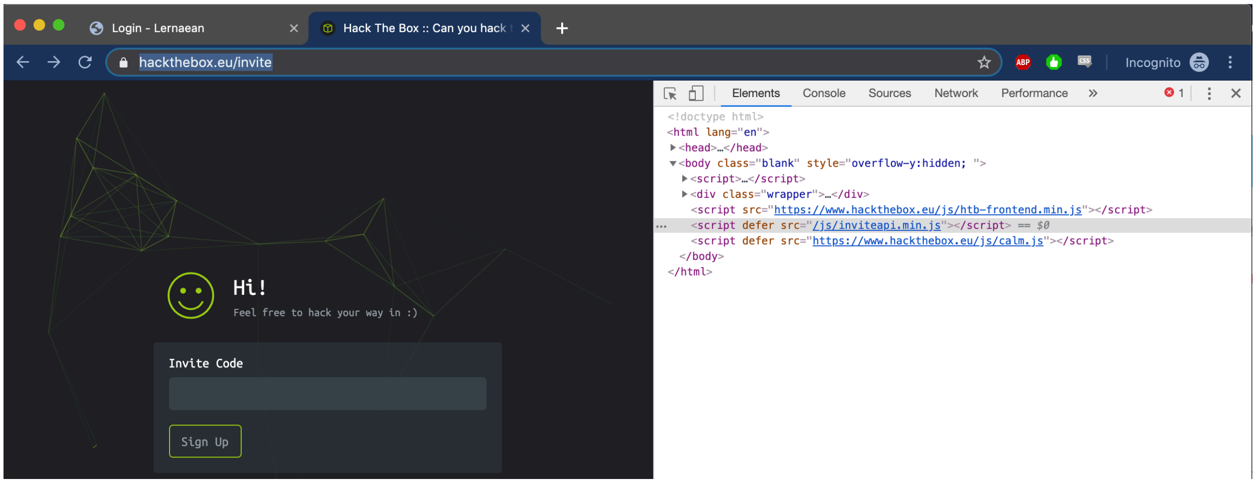
johnnyczhong owned challenge ⚙ 1 day ago
Find The Easy Pass [+2 ⌖]



```
Challenge Owns

Formula: challengePoints / 10

[11 Aug 2019 22:58:34] Find The Easy
Pass: 2

[11 Aug 2019 23:39:23] Snake: 1

[13 Aug 2019 02:34:35] Took the Byte: 2

Total: 5
```

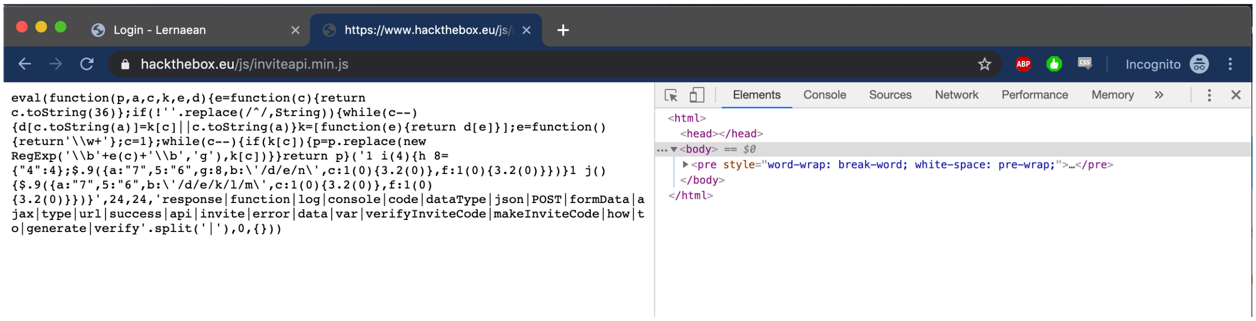
Getting the invitation code:

Using inspect, we see that there’s a script in the directory /js/inviteapi.min.js.



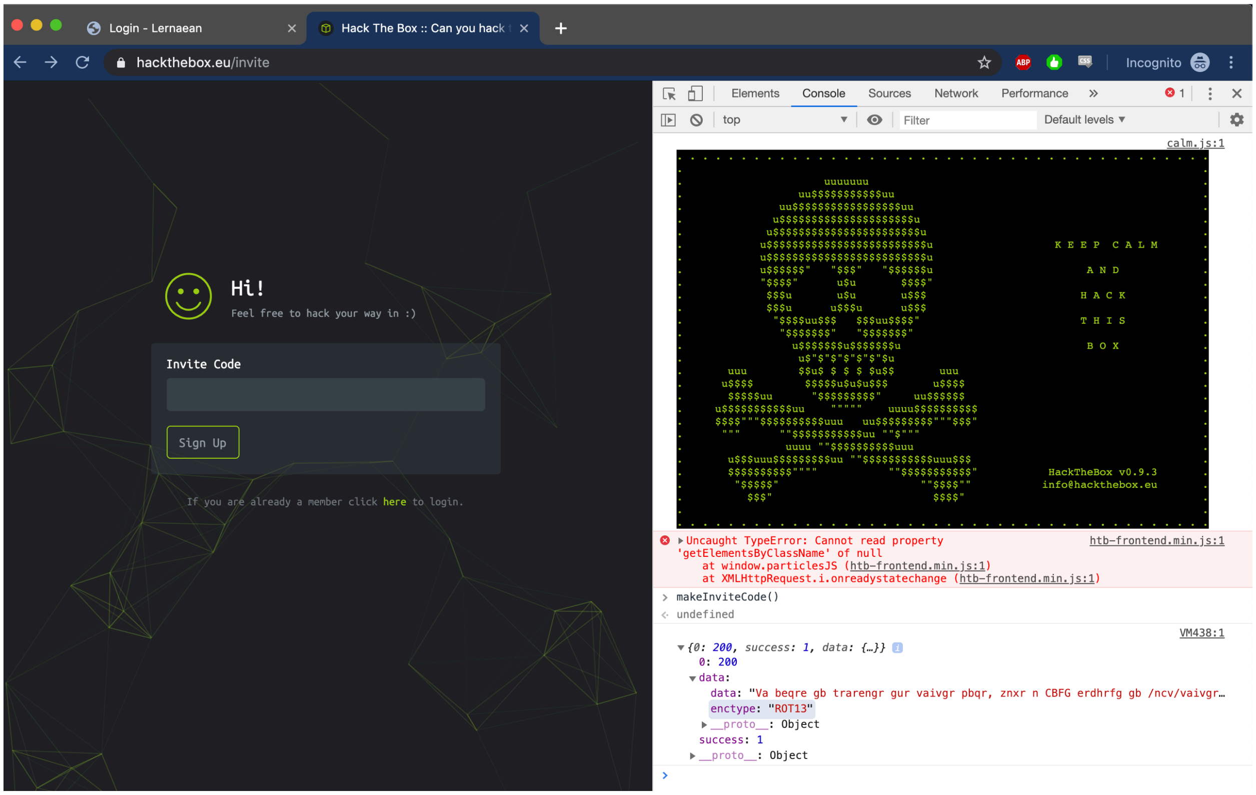
```
<!doctype html>
<html lang="en">
 ▶<head>…</head>
 ▼<body class="blank" style="overflow-y:hidden; ">
   ▶<script>…</script>
   ▶<div class="wrapper">…</div>
    <script src="https://www.hackthebox.eu/js/htb-frontend.min.js"></script>
    <script defer src="/js/inviteapi.min.js"></script> == $0
    <script defer src="https://www.hackthebox.eu/js/calm.js"></script>
  </body>
</html>
```

After we go to the site, we see that there’s a function called “makeInviteCode”. Using the console tab in chrome’s inspection window, we run that function and it generates a series of alphabetical characters.



```
eval(function(p,a,c,k,e,d){e=function(c){return
c.toString(36)};if(!''.replace(/^/,String)){while(c--)
{d[c.toString(a)]=k[c]||c.toString(a)}k=[function(e){return d[e]}];e=function()
{return'\\w+'};c=1};while(c--){if(k[c]){p=p.replace(new
RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}('1 i(4){h 8=
{"4":4};$.9({a:"7",5:"6",g:8,b:\'/d/e/n\',c:1(0){3.2(0)},f:1(0){3.2(0)}})}1 j()
{$.9({a:"7",5:"6",b:\'/d/e/k/l/m\',c:1(0){3.2(0)},f:1(0)
{3.2(0)}})}',24,24,'response|function|log|console|code|dataType|json|POST|formData|a
jax|type|url|success|api|invite|error|data|var|verifyInviteCode|makeInviteCode|how|t
o|generate|verify'.split('|'),0,{}))
```

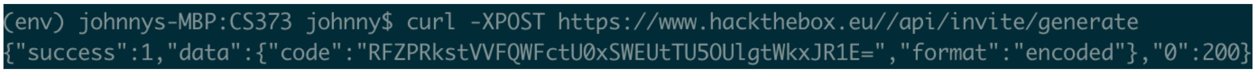
```
<html>
  <head></head>
 ▼<body> == $0
   ▶<pre style="word-wrap: break-word; white-space: pre-wrap;">…</pre>
  </body>
</html>
```



```
Uncaught TypeError: Cannot read property                    htb-frontend.min.js:1
'getElementsByClassName' of null
    at window.particlesJS (htb-frontend.min.js:1)
    at XMLHttpRequest.i.onreadystatechange (htb-frontend.min.js:1)
> makeInviteCode()
< undefined
                                                                        VM438:1
  {0: 200, success: 1, data: {…}}
    0: 200
    data:
      data: "Va beqre gb trarengr gur vaivgr pbqr, znxr n CBFG erdhrfg gb /ncv/vaivgr…
      enctype: "ROT13"
      __proto__: Object
    success: 1
    __proto__: Object
>
```

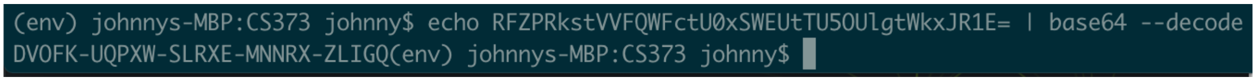
Next to the value, we see the enctype as "ROT13". This refers to the ROT cipher. Using a decoder, we get the string "INORDERTOGENERATETHEINVITECODEMAKEAPOSTREQUESTTOAPIINVITEGENERATE".

We make the post request using curl and get a string back. It looks like a base64 encoded string, based on how it ends with an "=" sign.



```
(env) johnnys-MBP:CS373 johnny$ curl -XPOST https://www.hackthebox.eu//api/invite/generate
{"success":1,"data":{"code":"RFZPRkstVVFQWFctU0xSWEUtTU50UlgtWkxJR1E=","format":"encoded"},"0":200}
```

In a terminal, we can echo the string we got and pipe it into a base64 decoder: | base 64 –decode



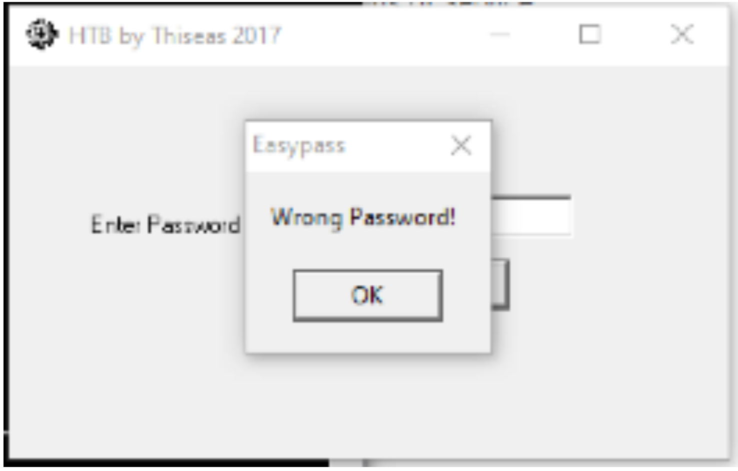
```
(env) johnnys-MBP:CS373 johnny$ echo RFZPRkstVVFQWFctU0xSWEUtTU50UlgtWkxJR1E= | base64 --decode
DVOFK-UQPXW-SLRXE-MNNRX-ZLIGQ(env) johnnys-MBP:CS373 johnny$
```

Here, we get strings of alphanumeric characters separated by dashes. I enter this into the invite code site and it functions as an invite code.

Find the Easy Password:

Checked the sha256 sum.

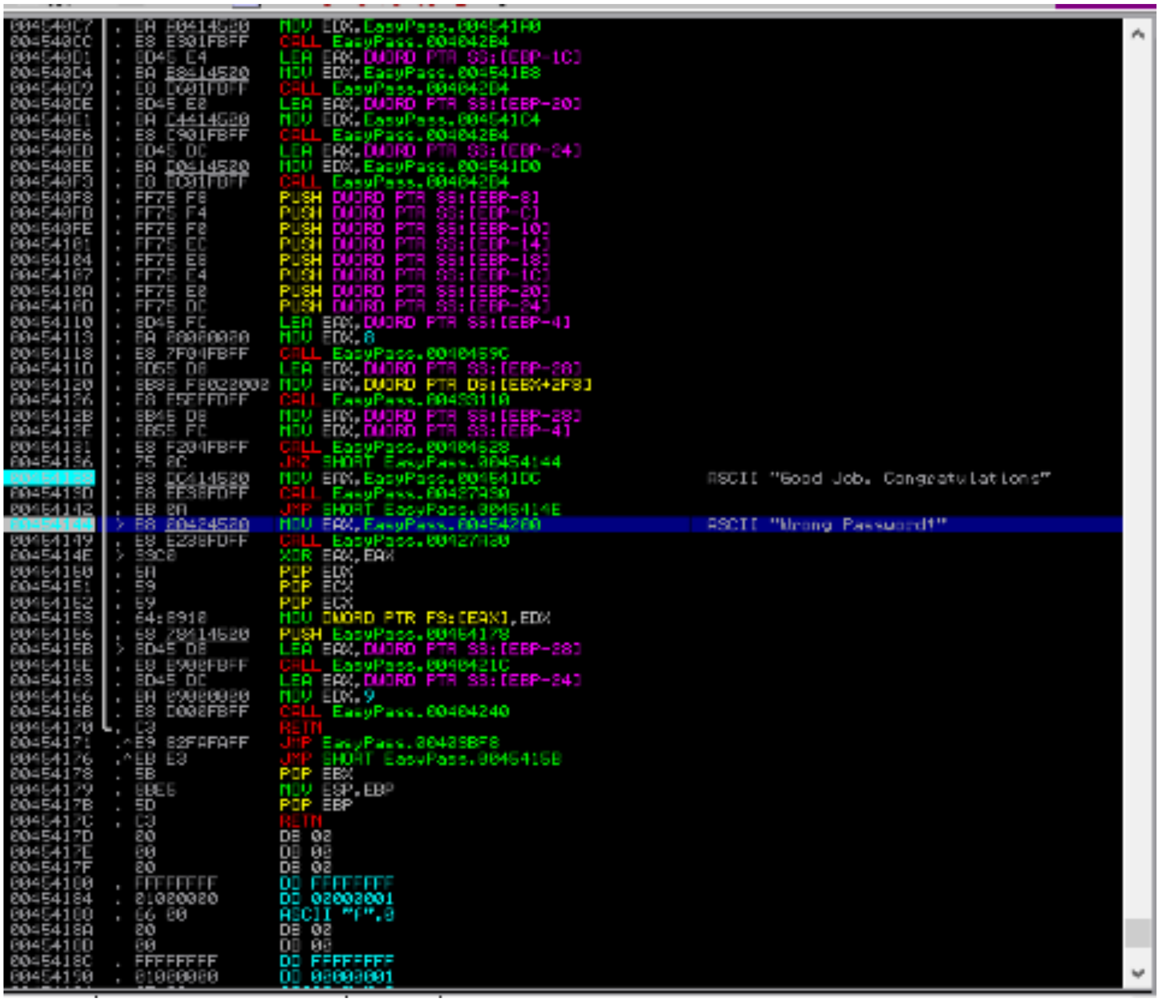
Opened up the program. Tried a dummy password on the program.



Wrong Password!

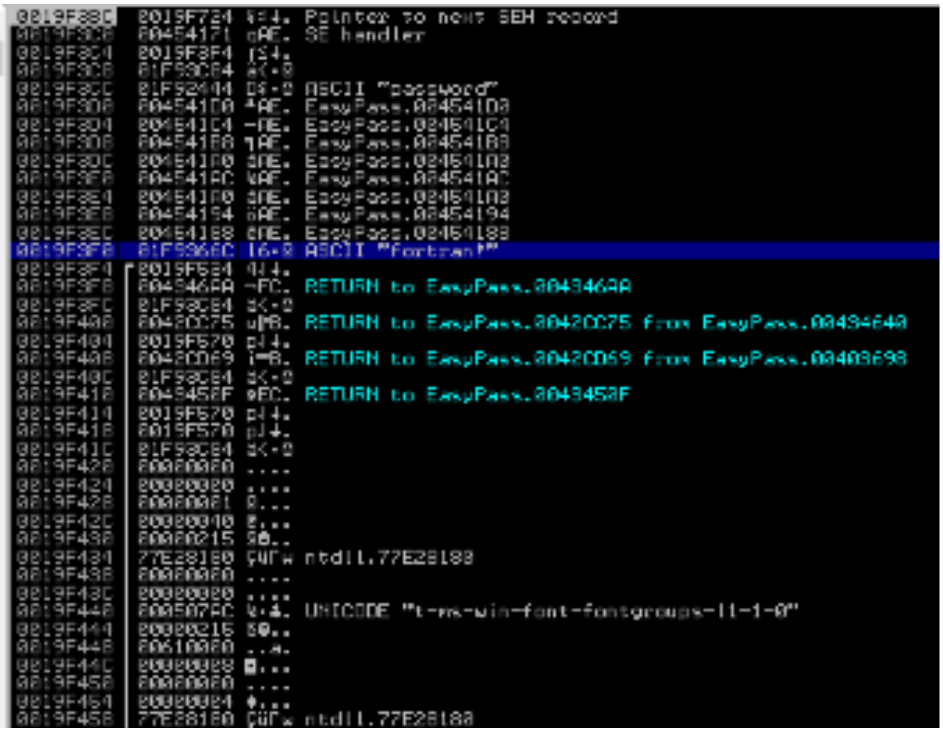
Opened the program using Immunity Debugger and searched for "all referenced text strings".

Set breakpoints on the strings indicating the success or failure of the attempt, as seen in the following screenshot.



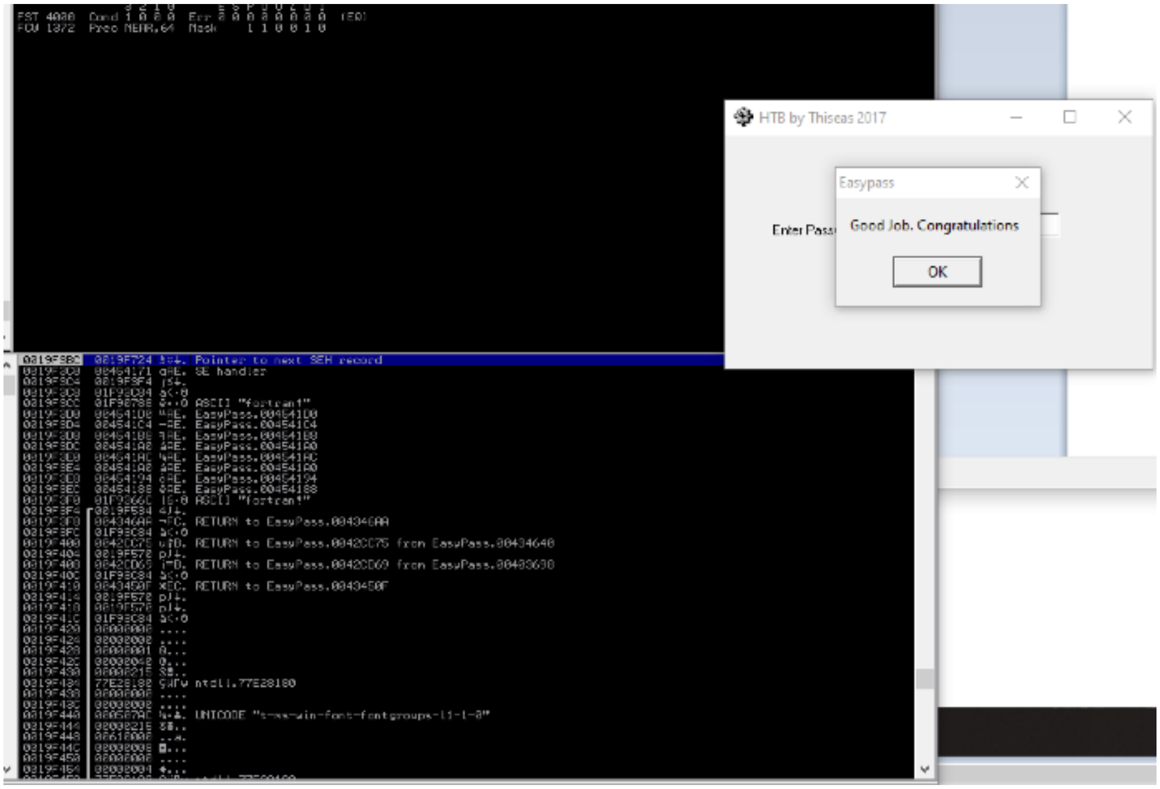
```
004540C7  .  BA A0414500    MOV EDX,EasyPass.004541A0
004540CC  .  E8 E301FBFF    CALL EasyPass.004042B4
004540D1  .  8D45 E4        LEA EAX,DWORD PTR SS:[EBP-1C]
004540D4  .  BA B8414500    MOV EDX,EasyPass.004541B8
004540D9  .  E8 D601FBFF    CALL EasyPass.004042B4
004540DE  .  8D45 E0        LEA EAX,DWORD PTR SS:[EBP-20]
004540E1  .  BA C4414500    MOV EDX,EasyPass.004541C4
004540E6  .  E8 C901FBFF    CALL EasyPass.004042B4
004540EB  .  8D45 DC        LEA EAX,DWORD PTR SS:[EBP-24]
004540EE  .  BA D0414500    MOV EDX,EasyPass.004541D0
004540F3  .  E8 BC01FBFF    CALL EasyPass.004042B4
004540F8  .  FF75 F8        PUSH DWORD PTR SS:[EBP-8]
004540FB  .  FF75 F4        PUSH DWORD PTR SS:[EBP-C]
004540FE  .  FF75 F0        PUSH DWORD PTR SS:[EBP-10]
00454101  .  FF75 EC        PUSH DWORD PTR SS:[EBP-14]
00454104  .  FF75 E8        PUSH DWORD PTR SS:[EBP-18]
00454107  .  FF75 E4        PUSH DWORD PTR SS:[EBP-1C]
0045410A  .  FF75 E0        PUSH DWORD PTR SS:[EBP-20]
0045410D  .  FF75 DC        PUSH DWORD PTR SS:[EBP-24]
00454110  .  8D45 FC        LEA EAX,DWORD PTR SS:[EBP-4]
00454113  .  BA 08000000    MOV EDX,8
00454118  .  E8 7F04FBFF    CALL EasyPass.0040459C
0045411D  .  8D55 D8        LEA EDX,DWORD PTR SS:[EBP-28]
00454120  .  8B83 F8020000  MOV EAX,DWORD PTR DS:[EBX+2F8]
00454126  .  E8 F5EFFDFF    CALL EasyPass.00433110
0045412B  .  8B45 D8        MOV EAX,DWORD PTR SS:[EBP-28]
0045412E  .  8B55 FC        MOV EDX,DWORD PTR SS:[EBP-4]
00454131  .  E8 F204FBFF    CALL EasyPass.00404628
00454136  .  75 0C          JNZ SHORT EasyPass.00454144
00454138  .  B8 DC414500    MOV EAX,EasyPass.004541DC                ASCII "Good Job. Congratulations"
0045413D  .  E8 FE38FDFF    CALL EasyPass.00427A40
00454142  .  EB 0A          JMP SHORT EasyPass.0045414E
00454144  >  B8 00424500    MOV EAX,EasyPass.00454200                ASCII "Wrong Password!"
00454149  .  E8 E238FDFF    CALL EasyPass.00427A30
0045414E  >  33C0           XOR EAX,EAX
00454150  .  5A             POP EDX
00454151  .  59             POP ECX
00454152  .  59             POP ECX
00454153  .  64:8910        MOV DWORD PTR FS:[EAX],EDX
00454156  .  68 78414500    PUSH EasyPass.00454178
0045415B  >  8D45 D8        LEA EAX,DWORD PTR SS:[EBP-28]
0045415E  .  E8 B900FBFF    CALL EasyPass.0040421C
00454163  .  8D45 DC        LEA EAX,DWORD PTR SS:[EBP-24]
00454166  .  BA 09000000    MOV EDX,9
0045416B  .  E8 D000FBFF    CALL EasyPass.00404240
00454170  .  C3             RETN
00454171  .^ E9 82FAFAFF    JMP EasyPass.00403BF8
00454176  .^ EB E3          JMP SHORT EasyPass.0045415B
00454178  .  5B             POP EBX
00454179  .  8BE5           MOV ESP,EBP
0045417B  .  5D             POP EBP
0045417C  .  C3             RETN
0045417D     00             DB 00
0045417E     00             DB 00
0045417F     00             DB 00
00454180  .  FFFFFFFF       DD FFFFFFFF
00454184  .  01000000       DD 00000001
00454188  .  66 00          ASCII "f",0
0045418A     00             DB 00
0045418B     00             DB 00
0045418C  .  FFFFFFFF       DD FFFFFFFF
00454190  .  01000000       DD 00000001
```

This allowed me to read the registers to determine the string being used to compare the incorrect value, which would be the password: fortran!



```
0019F3BC  0019F724  $=↓.  Pointer to next SEH record
0019F3C0  00454171  qAE.  SE handler
0019F3C4  0019F3F4  ⌠S↓.
0019F3C8  01F93C84  ä<•@
0019F3CC  01F92444  D$•@  ASCII "password"
0019F3D0  004541D0  ╨AE.  EasyPass.004541D0
0019F3D4  004541C4  ─AE.  EasyPass.004541C4
0019F3D8  004541B8  ╕AE.  EasyPass.004541B8
0019F3DC  004541A0  áAE.  EasyPass.004541A0
0019F3E0  004541AC  ¼AE.  EasyPass.004541AC
0019F3E4  004541A0  áAE.  EasyPass.004541A0
0019F3E8  00454194  öAE.  EasyPass.00454194
0019F3EC  00454188  êAE.  EasyPass.00454188
0019F3F0  01F9366C  l6•@  ASCII "fortran!"
0019F3F4 ┌0019F534  4J↓.
0019F3F8 │004346AA  ¬FC.  RETURN to EasyPass.004346AA
0019F3FC │01F93C84  ä<•@
0019F400 │0042CC75  u╠B.  RETURN to EasyPass.0042CC75 from EasyPass.00434640
0019F404 │0019F570  pJ↓.
0019F408 │0042CD69  i═B.  RETURN to EasyPass.0042CD69 from EasyPass.00403698
0019F40C │01F93C84  ä<•@
0019F410 │0043450F  ☼EC.  RETURN to EasyPass.0043450F
0019F414 │0019F570  pJ↓.
0019F418 │0019F570  pJ↓.
0019F41C │01F93C84  ä<•@
0019F420 │00000000  ....
0019F424 │00000000  ....
0019F428 │00000001  ☺...
0019F42C │00000040  @...
0019F430 │00000215  §☻..
0019F434 │77E28180  ÇüΓw ntdll.77E28180
0019F438 │00000000  ....
0019F43C │00000000  ....
0019F440 │0050A7AC  ¼•P.  UNICODE "t-ms-win-font-fontgroups-l1-1-0"
0019F444 │00000215  §☻..
0019F448 │00610000  ..a.
0019F44C │00000008  ◘...
0019F450 │00000000  ....
0019F454 │00000004  ♦...
0019F458 │77E28180  ÇüΓw ntdll.77E28180
```

With this knowledge, I was able to provide the right password:



```
                    3 2 1 0          E S P U O Z D I
FST 4000  Cond 1 0 0 0  Err 0 0 0 0 0 0 0 0  (EQ)
FCW 1372  Prec NEAR,64  Mask    1 1 0 0 1 0
```

Good Job. Congratulations

```
0019F3BC  0019F724  $=↓.  Pointer to next SEH record
0019F3C0  00454171  qAE.  SE handler
0019F3C4  0019F3F4  ⌠S↓.
0019F3C8  01F93C84  ä<•@
0019F3CC  01F98788  ê••@  ASCII "fortran!"
0019F3D0  004541D0  ╨AE.  EasyPass.004541D0
0019F3D4  004541C4  ─AE.  EasyPass.004541C4
0019F3D8  004541B8  ╕AE.  EasyPass.004541B8
0019F3DC  004541A0  áAE.  EasyPass.004541A0
0019F3E0  004541AC  ¼AE.  EasyPass.004541AC
0019F3E4  004541A0  áAE.  EasyPass.004541A0
0019F3E8  00454194  öAE.  EasyPass.00454194
0019F3EC  00454188  êAE.  EasyPass.00454188
0019F3F0  01F9366C  l6•@  ASCII "fortran!"
0019F3F4 ┌0019F534  4J↓.
0019F3F8 │004346AA  ¬FC.  RETURN to EasyPass.004346AA
0019F3FC │01F93C84  ä<•@
0019F400 │0042CC75  u╠B.  RETURN to EasyPass.0042CC75 from EasyPass.00434640
0019F404 │0019F570  pJ↓.
0019F408 │0042CD69  i═B.  RETURN to EasyPass.0042CD69 from EasyPass.00403698
0019F40C │01F93C84  ä<•@
0019F410 │0043450F  ☼EC.  RETURN to EasyPass.0043450F
0019F414 │0019F570  pJ↓.
0019F418 │0019F570  pJ↓.
0019F41C │01F93C84  ä<•@
0019F420 │00000000  ....
0019F424 │00000000  ....
0019F428 │00000001  ☺...
0019F42C │00000040  @...
0019F430 │00000215  §☻..
0019F434 │77E28180  ÇüΓw ntdll.77E28180
0019F438 │00000000  ....
0019F43C │00000000  ....
0019F440 │0050A7AC  ¼•P.  UNICODE "t-ms-win-font-fontgroups-l1-1-0"
0019F444 │00000215  §☻..
0019F448 │00610000  ..a.
0019F44C │00000008  ◘...
0019F450 │00000000  ....
0019F454 │00000004  ♦...
```

Snake.py

This required some python knowledge. Here, we're given a script to work through and the goal is to find the username and a key. The username is easy enough to get, as we just need to print the username that is required to pass the string matching requirement.

If I print "slither" this should be enough to pass this first part. It's "anaconda" by the way.

```
slither = aa + db + nn + ef + rr + gh + lr + ty

print 'Authentication required'

print ''

user_input = raw_input('Enter your username\n')

if user_input == slither:

    pass


else:

    print 'Wrong username try harder'

    exit()
```

This next part is a little trickier, as the "password" is misleading. We determine that the "key" in question is the hexadecimal string created by the following:

```
keys = [0x70, 0x61, 0x73, 0x73, 0x77, 0x6f, 0x72, 0x64, 0x21, 0x21]
```

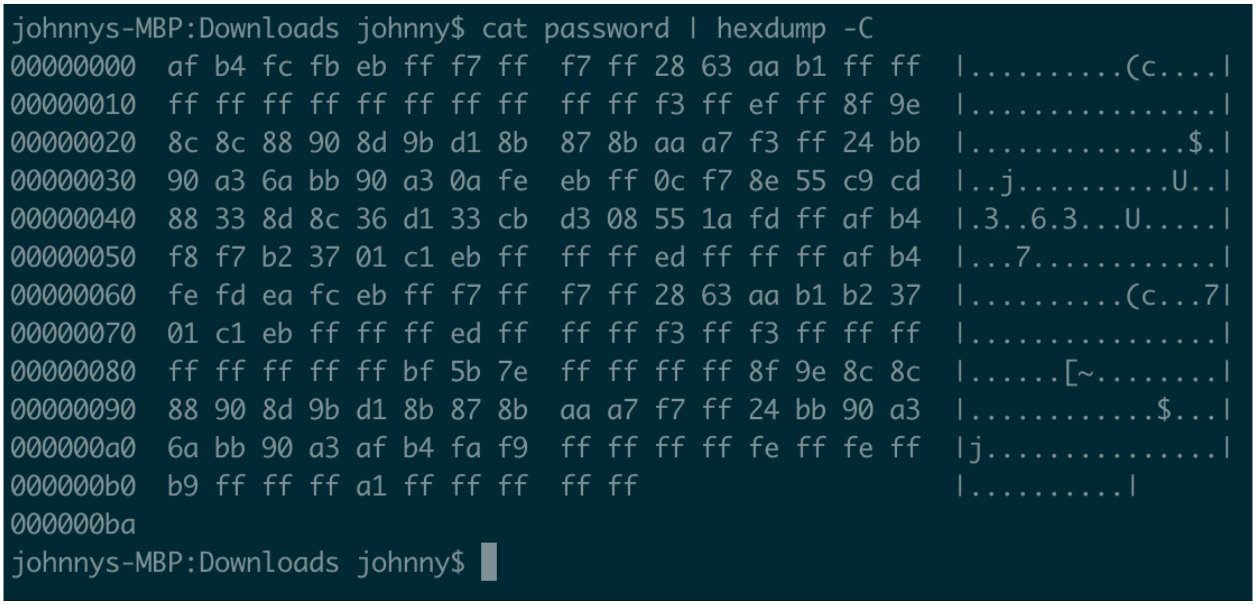
This is modified by a "lock", which is some random integer modifier. We can look at the changed values by simply using the following list comprehension after the encryption:

```
for key in keys:

    keys_encrypt = lock ^ key

    chars.append(keys_encrypt)

print ''.join([str(chr(x)) for x in chars])
```

This provides us the value "udvvrjwa$$", which used in conjunction with "anaconda" passes this challenge.

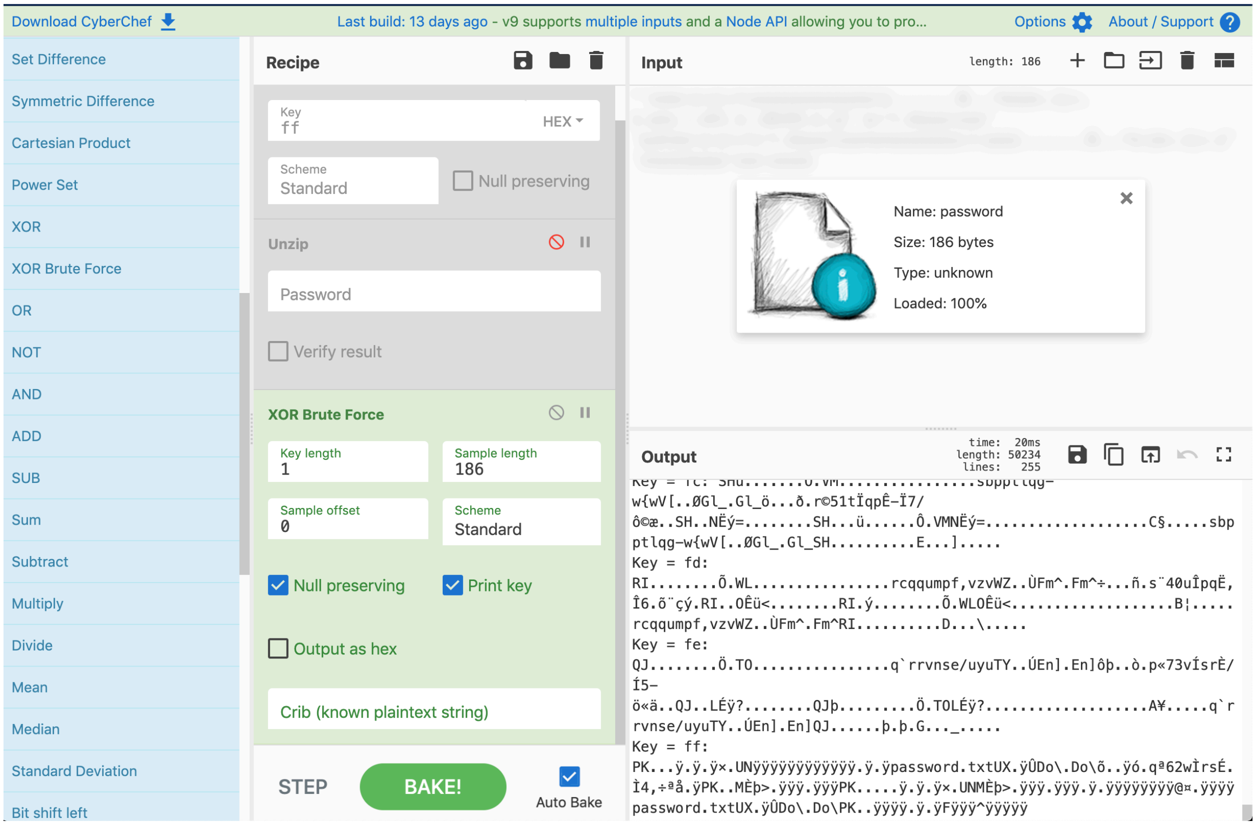
Took the Byte:

This challenge was difficult in the pure number of options it had to solve it. The hint: “Someone took my bytes! Can you recover my password for me?“ was moderately useful for solving this problem. The way it’s phrased leads the user to think that a byte is missing. After attempting to add a byte (all the byte combinations), I stepped back and looked at the file:



```
johnnys-MBP:Downloads johnny$ cat password | hexdump -C
00000000  af b4 fc fb eb ff f7 ff  f7 ff 28 63 aa b1 ff ff  |..........(c....|
00000010  ff ff ff ff ff ff ff ff  ff ff f3 ff ef ff 8f 9e  |................|
00000020  8c 8c 88 90 8d 9b d1 8b  87 8b aa a7 f3 ff 24 bb  |..............$.|
00000030  90 a3 6a bb 90 a3 0a fe  eb ff 0c f7 8e 55 c9 cd  |..j..........U..|
00000040  88 33 8d 8c 36 d1 33 cb  d3 08 55 1a fd ff af b4  |.3..6.3...U.....|
00000050  f8 f7 b2 37 01 c1 eb ff  ff ff ed ff ff ff af b4  |...7............|
00000060  fe fd ea fc eb ff f7 ff  f7 ff 28 63 aa b1 b2 37  |..........(c...7|
00000070  01 c1 eb ff ff ff ed ff  ff ff f3 ff f3 ff ff ff  |................|
00000080  ff ff ff ff ff bf 5b 7e  ff ff ff ff 8f 9e 8c 8c  |......[~........|
00000090  88 90 8d 9b d1 8b 87 8b  aa a7 f7 ff 24 bb 90 a3  |............$...|
000000a0  6a bb 90 a3 af b4 fa f9  ff ff ff ff fe ff fe ff  |j...............|
000000b0  b9 ff ff ff a1 ff ff ff  ff ff                    |..........|
000000ba
johnnys-MBP:Downloads johnny$
```

From this, it looks like we could do some modifications on the bytes present to look for a hidden message. The tool “cyberchef” offers a brute force XOR tool, which was used to look for hidden messages.



Key ff HEX
Scheme Standard
□ Null preserving

Unzip

Password

□ Verify result

XOR Brute Force

Key length 1
Sample length 186
Sample offset 0
Scheme Standard
☑ Null preserving
☑ Print key
□ Output as hex
Crib (known plaintext string)

☑ Auto Bake

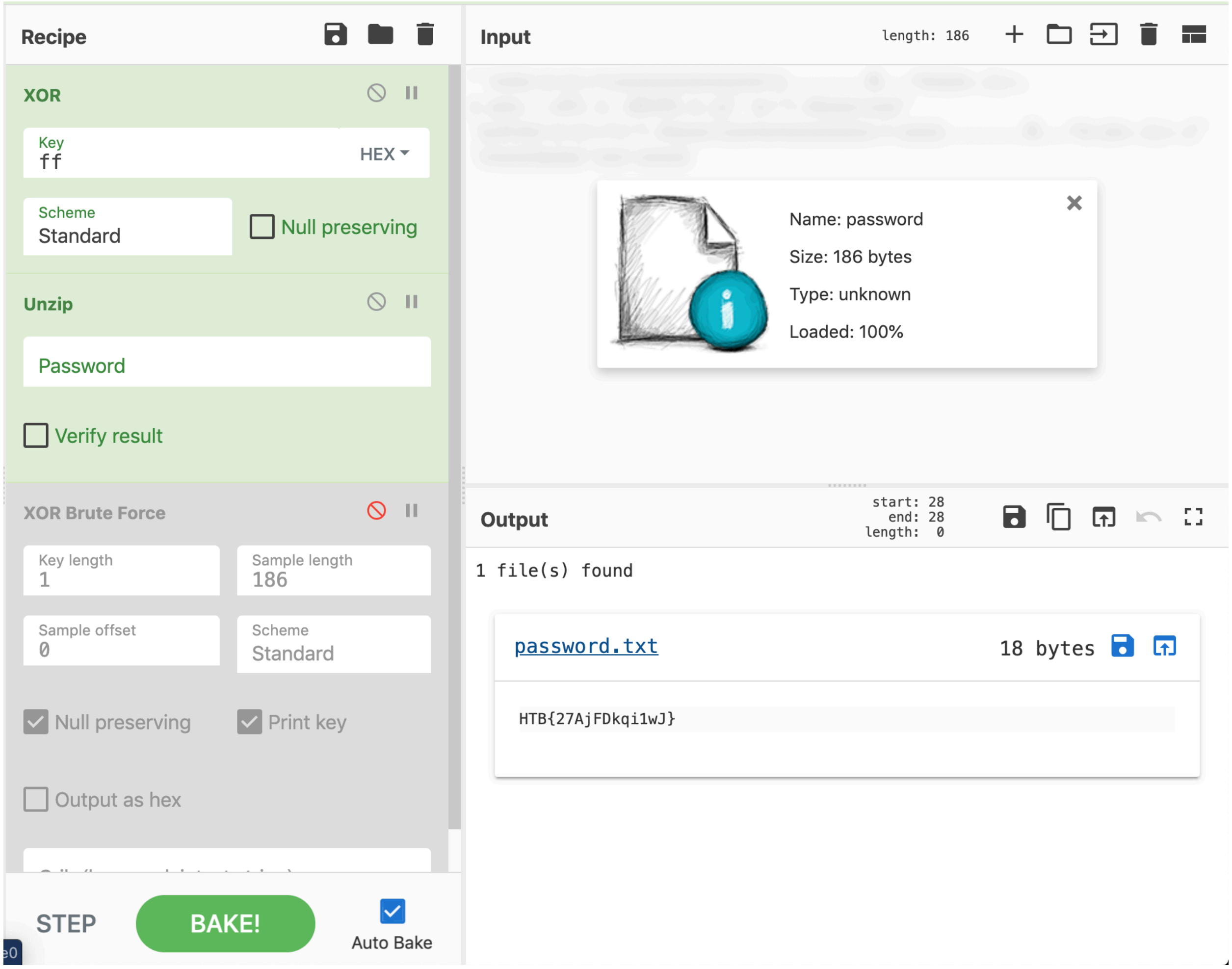
Input length: 186

Name: password
Size: 186 bytes
Type: unknown
Loaded: 100%

Output time: 20ms length: 50234 lines: 255

```
Key = fc: SHû.......Ô.VM...............sbpptlqg–
w{wV[..ØGl_.Gl_ö...ð.r©51tÏqpÊ–Ï7/
ô©æ..SH..NËý=........SH...ü......Ô.VMNËý=..................C§.....sbp
ptlqg-w{wV[..ØGl_.Gl_SH..........E...].....
Key = fd:
RI........Õ.WL................rcqqumpf,vzvWZ..ÙFm^.Fm^÷...ñ.s¨40uÎpqË,
Î6.õ¨çý.RI..OÊü<........RI.ý........Õ.WLOÊü<...................B¦.....
rcqqumpf,vzvWZ..ÙFm^.Fm^RI..........D...\.....
Key = fe:
QJ........Ö.TO................q`rrvnse/uyuTY..ÚEn].En]ôþ..ò.p«73vÍsrÈ/
Í5–
ö«ä..QJ..LÉÿ?........QJþ.........Ö.TOLÉÿ?...................A¥.....q`r
rvnse/uyuTY..ÚEn].En]QJ......þ.þ.G..._.....
Key = ff:
PK...ÿ.ÿ.ÿ×.UNÿÿÿÿÿÿÿÿÿÿÿÿ.ÿ.ÿpassword.txtUX.ÿÛDo\.Do\õ..ÿó.qª62wÌrsÉ.
Ì4,÷ªå.ÿPK..MÈþ>.ÿÿÿ.ÿÿÿPK.....ÿ.ÿ.ÿ×.UNMÈþ>.ÿÿÿ.ÿÿÿ.ÿ.ÿÿÿÿÿÿÿÿ@¤.ÿÿÿÿ
password.txtUX.ÿÛDo\.Do\PK..ÿÿÿÿ.ÿ.ÿFÿÿÿ^ÿÿÿÿÿ
```

The last entry looks interesting, as the file seems to be identified as a “PK” file and the text “password.txt” is in clear ascii. Looking online, a file with the PK header seems to be a zip file.



XOR

Key: ff (HEX)

Scheme: Standard

☐ Null preserving

Unzip

Password

☐ Verify result

XOR Brute Force

Key length: 1

Sample length: 186

Sample offset: 0

Scheme: Standard

☑ Null preserving

☑ Print key

☐ Output as hex

☑ Auto Bake

Input length: 186

Name: password

Size: 186 bytes

Type: unknown

Loaded: 100%

```
start: 28
  end: 28
length:  0
```

```
1 file(s) found

password.txt                                  18 bytes

HTB{27AjFDkqi1wJ}
```

We unzip the file and the “password.txt” file is made available to us. This gives us the key to enter.