CS 373 Summer 2019

Final Write Up

Kirsten Wollam

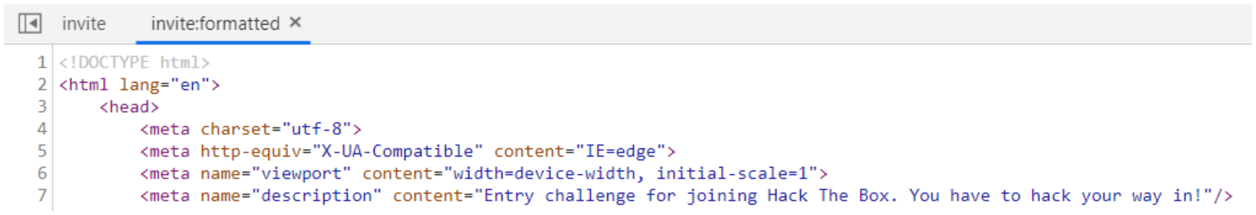
HACK THE BOX

Generating an Invite Code

To join Hack the Box in invitation code is required, but there is not information provided on how to obtain one, except a comment that says "Hi! Feel free to hack your way in :)."

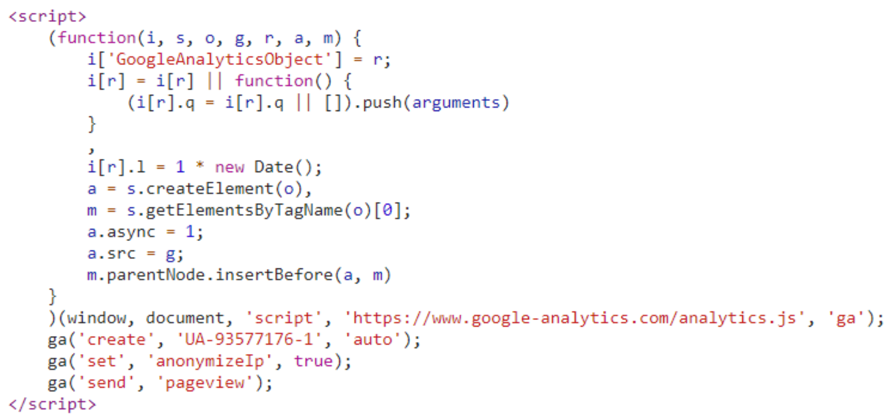
My first instinct was to start by inspecting the website in chrome using the built in tools. This provided this sweet but mostly unhelpful image.

Next I decided to take a look at the source file to see if there is anything interesting. The First thing i found in the source was confirmation of what we already knew, that we must hack our way in.



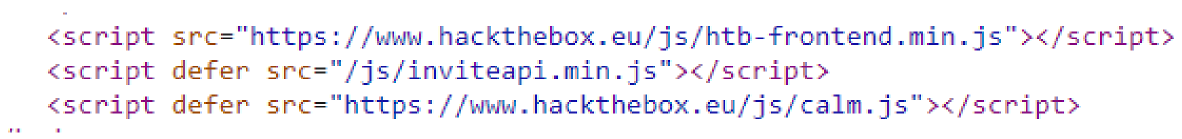
```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="description" content="Entry challenge for joining Hack The Box. You have to hack your way in!"/>
```

The next thing that caught my eye was this function:



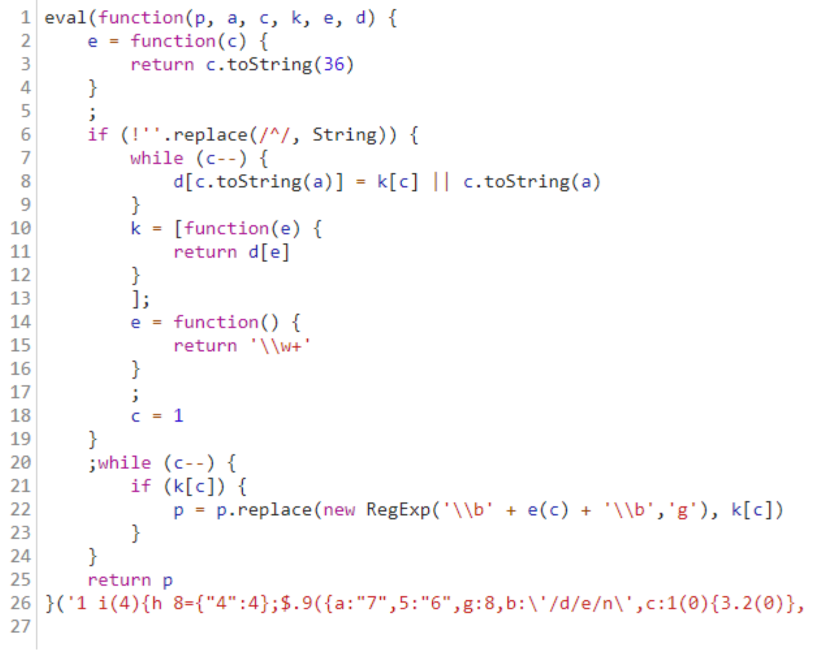
```
<script>
    (function(i, s, o, g, r, a, m) {
        i['GoogleAnalyticsObject'] = r;
        i[r] = i[r] || function() {
            (i[r].q = i[r].q || []).push(arguments)
        }
        ,
        i[r].l = 1 * new Date();
        a = s.createElement(o),
        m = s.getElementsByTagName(o)[0];
        a.async = 1;
        a.src = g;
        m.parentNode.insertBefore(a, m)
    }
    )(window, document, 'script', 'https://www.google-analytics.com/analytics.js', 'ga');
    ga('create', 'UA-93577176-1', 'auto');
    ga('set', 'anonymizeIp', true);
    ga('send', 'pageview');
</script>
```

It looked like it might be part of how an invite code is validated or something like that, but I could not make much of it, so I continued on in the source code and at the end found a few more linked source files.



```
<script src="https://www.hackthebox.eu/js/htb-frontend.min.js"></script>
<script defer src="/js/inviteapi.min.js"></script>
<script defer src="https://www.hackthebox.eu/js/calm.js"></script>
```

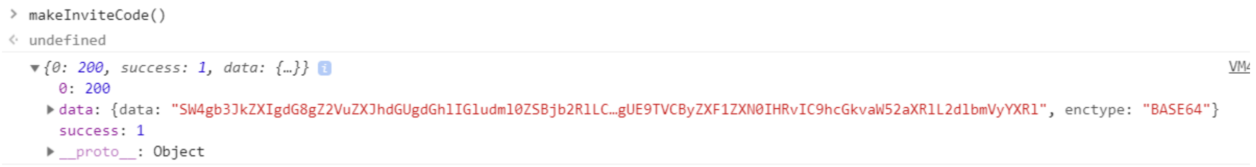
Of these inviteapi.min.js seemed like it might be moving in the right direction. This source code also has an interesting function, but I was not able to make much of it.



```
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)},
```

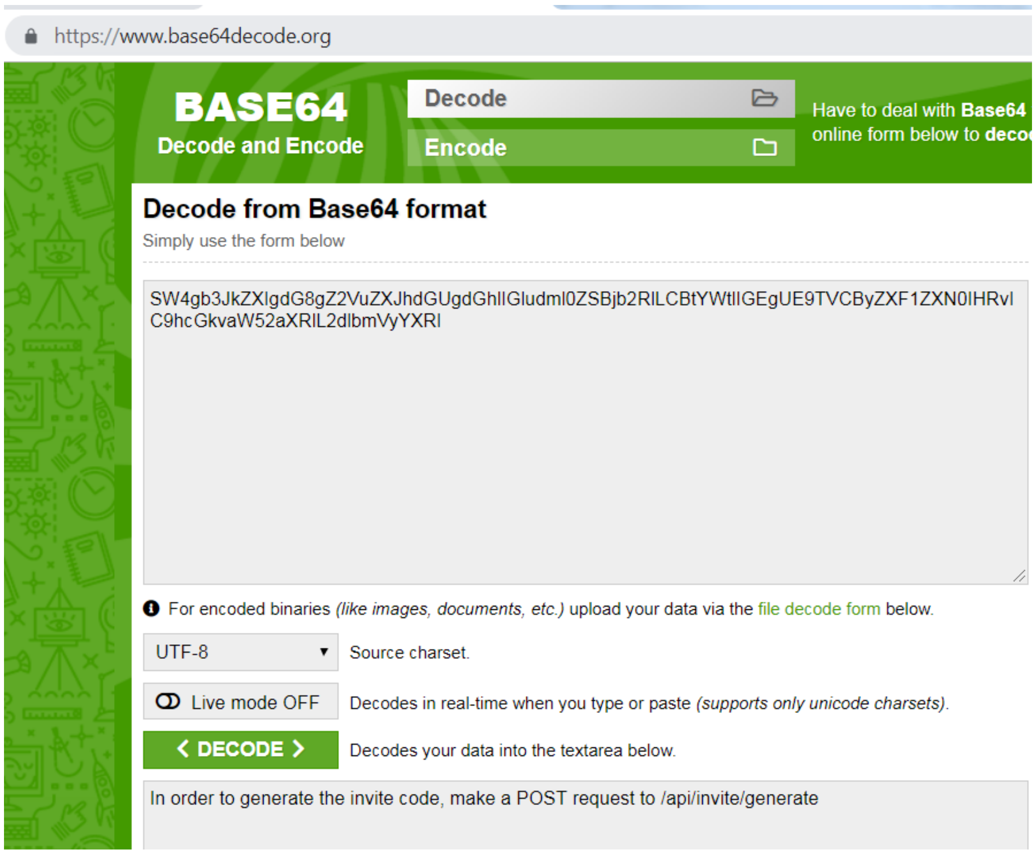
While I couldn't make out much from this function there is an interesting string at the end that seems to offer a hint towards direction:
'response|function|log|console|code|dataType|json|POST|formData|ajax|type|url|success|api|invite|error|data|var|verifyInviteCode|makeInviteCode|how|to|generate|verify'.split('|')

Particularly the "MakeInviteCode" portion of this string. I am hoping that it is a function or object I can locate. Luckily using the console to find makeInviteCode() yields something very interesting.



```
> makeInviteCode()
< undefined
  ▼{0: 200, success: 1, data: {…}}                                                                                     VM4
     0: 200
   ▶data: {data: "SW4gb3JkZXIgdG8gZ2VuZXJhdGUgdGhlIGludml0ZSBjb2RlLC…gUE9TVCByZXF1ZXN0IHRvIC9hcGkvaW52aXRlL2dlbmVyYXRl", enctype: "BASE64"}
     success: 1
   ▶__proto__: Object
```

Since this says enctype: "BASE64" I took this string and ran it though a decryptor.


https://www.base64decode.org

Decode from Base64 format

Simply use the form below

```
SW4gb3JkZXIgdG8gZ2VuZXJhdGUgdGhlIGludml0ZSBjb2RlLCBtYWtlIGEgUE9TVCByZXF1ZXN0IHRvI
C9hcGkvaW52aXRlL2dlbmVyYXRl
```

For encoded binaries *(like images, documents, etc.)* upload your data via the file decode form below.

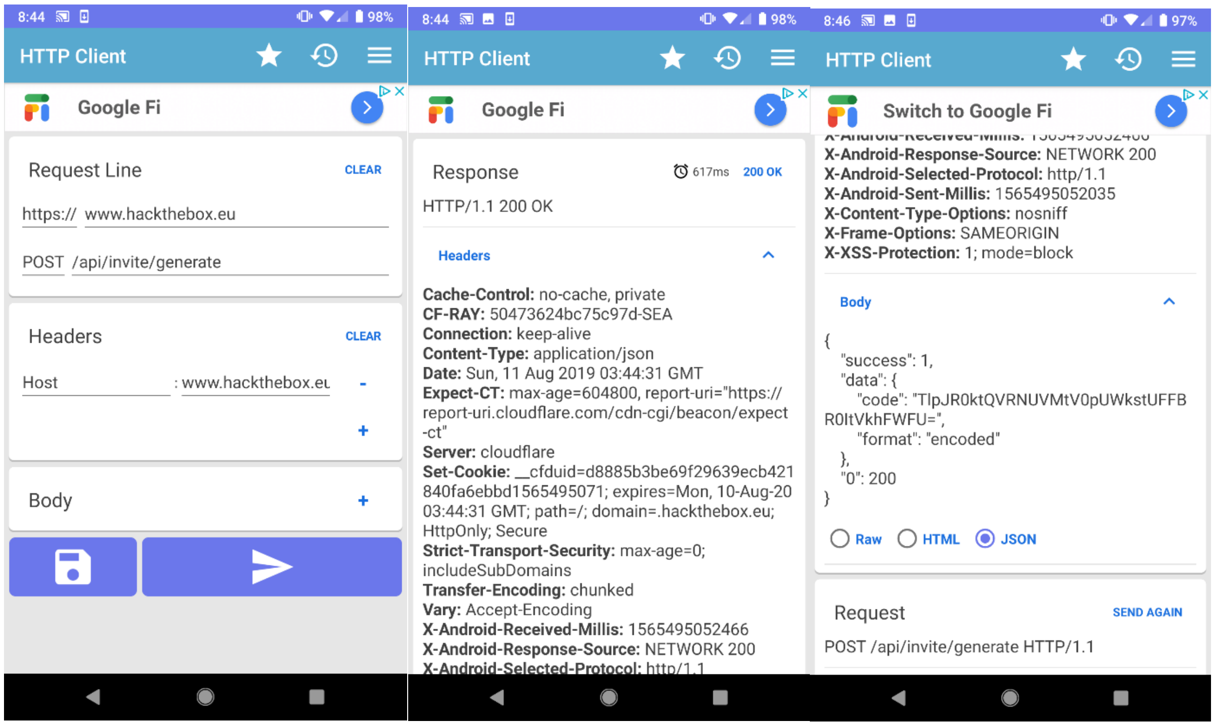
UTF-8 — Source charset.

Live mode OFF — Decodes in real-time when you type or paste *(supports only unicode charsets)*.

DECODE — Decodes your data into the textarea below.

```
In order to generate the invite code, make a POST request to /api/invite/generate
```

The next step seems super obvious. I have an app on my phone that allows me to easily send a post request, so I decided to use that.



Request Line

https:// www.hackthebox.eu

POST /api/invite/generate

Headers

Host : www.hackthebox.eu

Body

```
Response                617ms   200 OK
HTTP/1.1 200 OK

Headers

Cache-Control: no-cache, private
CF-RAY: 50473624bc75c97d-SEA
Connection: keep-alive
Content-Type: application/json
Date: Sun, 11 Aug 2019 03:44:31 GMT
Expect-CT: max-age=604800, report-uri="https://
report-uri.cloudflare.com/cdn-cgi/beacon/expect
-ct"
Server: cloudflare
Set-Cookie: __cfduid=d8885b3be69f29639ecb421
840fa6ebbd1565495071; expires=Mon, 10-Aug-20
03:44:31 GMT; path=/; domain=.hackthebox.eu;
HttpOnly; Secure
Strict-Transport-Security: max-age=0;
includeSubDomains
Transfer-Encoding: chunked
Vary: Accept-Encoding
X-Android-Received-Millis: 1565495052466
X-Android-Response-Source: NETWORK 200
X-Android-Selected-Protocol: http/1.1
```

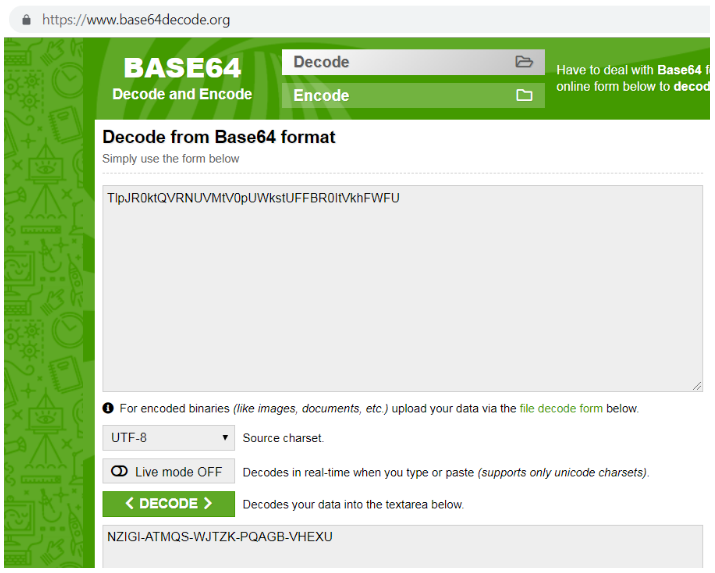
```
X-Android-Response-Source: NETWORK 200
X-Android-Selected-Protocol: http/1.1
X-Android-Sent-Millis: 1565495052035
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block

Body

{
   "success": 1,
   "data": {
      "code": "TlpJR0ktQVRNUVMtV0pUWkstUFFB
R0ItVkhFWFU=",
      "format": "encoded"
   },
   "0": 200
}
```

( ) Raw ( ) HTML [✓ JSON]

Request — SEND AGAIN

POST /api/invite/generate HTTP/1.1

This response confirms we are headed in the right direction, but does not seem to yield a working invite code. the format is listed as encoded, so I took it and ran it though the same base64 decryptor to see if that would work. When I tried this code I got an error, stating it was not valid for my IP, but then I remembered that i sent the post request from my phone. When used on my phone it worked!!



https://www.base64decode.org

Decode from Base64 format

Simply use the form below

```
TlpJR0ktQVRNUVMtV0pUWkstUFFBR0ItVkhFWFU
```

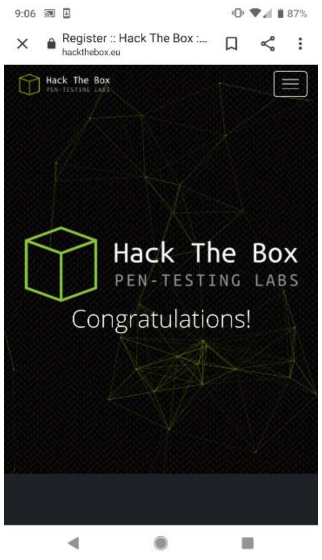
For encoded binaries *(like images, documents, etc.)* upload your data via the file decode form below.

UTF-8 Source charset.

Live mode OFF Decodes in real-time when you type or paste *(supports only unicode charsets)*.

DECODE Decodes your data into the textarea below.

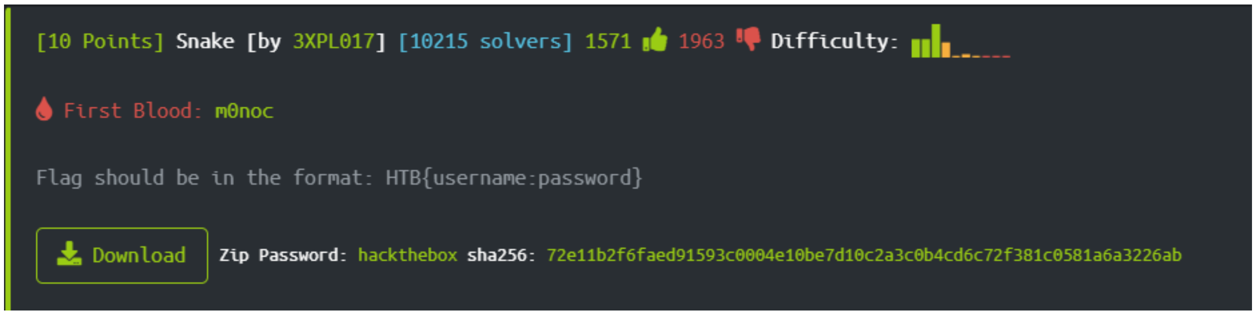
```
NZIGI-ATMQS-WJTZK-PQAGB-VHEXU
```



Challenge 1: Snake

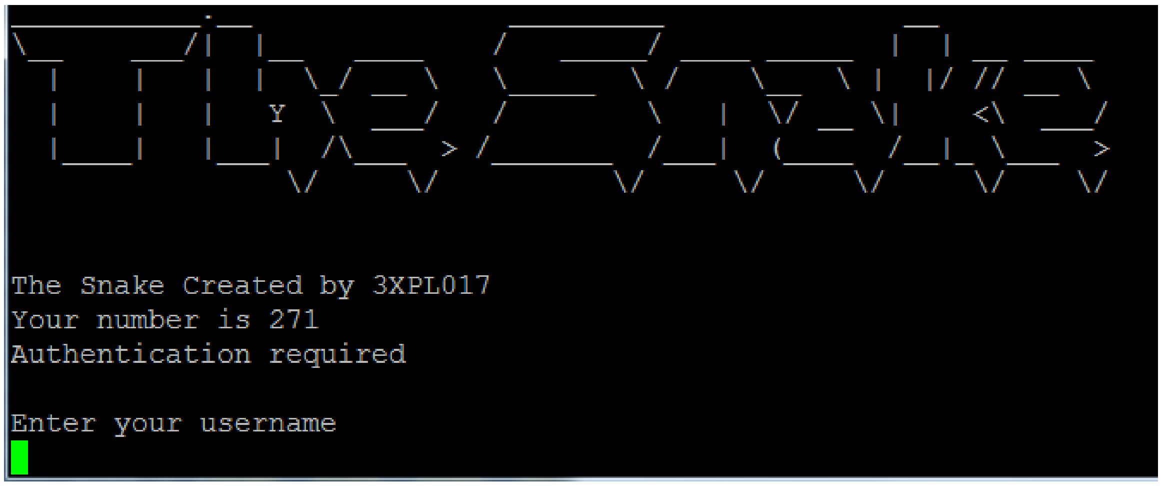
Reversing, 10 points

To start I chose what looks to be a fairly easy reversing challenge called Snake.



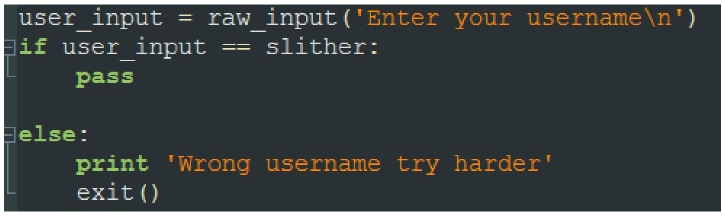
```
[10 Points] Snake [by 3XPL017] [10215 solvers] 1571 1963 Difficulty:

First Blood: m0noc

Flag should be in the format: HTB{username:password}

Download   Zip Password: hackthebox sha256: 72e11b2f6faed91593c0004e10be7d10c2a3c0b4cd6c72f381c0581a6a3226ab
```

After extracting this file, it looks to be a regular python file. Opening it in notepad++ shows a fairly straightforward python file, so I ran the file to see where we are starting. For the flag we will need to know the username and password. unsurprisingly the program prompts us to enter the username, after a sweet logo.



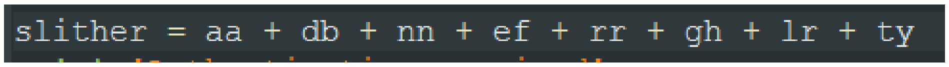
```
The Snake Created by 3XPL017
Your number is 271
Authentication required

Enter your username
```

Time to head back to the python script to see how it is working and find the user name. The code is structured fairly straightforwardly and this statement:



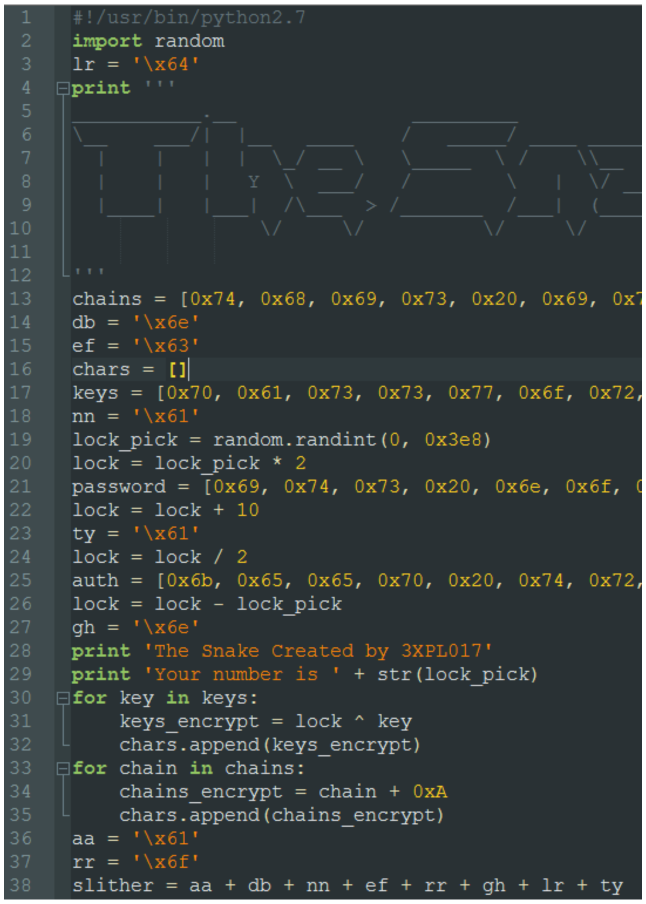
```
user_input = raw_input('Enter your username\n')
if user_input == slither:
    pass

else:
    print 'Wrong username try harder'
    exit()
```

Slither is unfortunately not a string, but it is defined previously in the file:



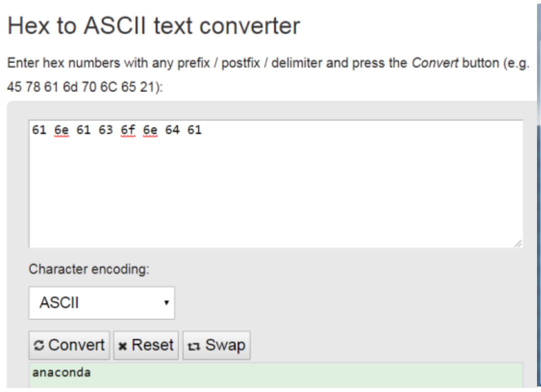
```
slither = aa + db + nn + ef + rr + gh + lr + ty
```

Well, it couldn't be too easy, slither is made up of 8 more variables. Looking through the file these are scattered in the code.



```
#!/usr/bin/python2.7
import random
lr = '\x64'
print '''
___________.__               _________
\__    ___/|  |__   ____    /   _____/ ____ _____
  |    |   |  |  \_/ __ \   \_____  \ /    \\__
  |    |   |   Y  \  ___/   /        \   |  \/ _
  |____|   |___|  /\___  > /_______  /___|  (___
                \/     \/          \/     \/


'''
chains = [0x74, 0x68, 0x69, 0x73, 0x20, 0x69, 0x7
db = '\x6e'
ef = '\x63'
chars = []
keys = [0x70, 0x61, 0x73, 0x73, 0x77, 0x6f, 0x72,
nn = '\x61'
lock_pick = random.randint(0, 0x3e8)
lock = lock_pick * 2
password = [0x69, 0x74, 0x73, 0x20, 0x6e, 0x6f, 0
lock = lock + 10
ty = '\x61'
lock = lock / 2
auth = [0x6b, 0x65, 0x65, 0x70, 0x20, 0x74, 0x72,
lock = lock - lock_pick
gh = '\x6e'
print 'The Snake Created by 3XPL017'
print 'Your number is ' + str(lock_pick)
for key in keys:
    keys_encrypt = lock ^ key
    chars.append(keys_encrypt)
for chain in chains:
    chains_encrypt = chain + 0xA
    chars.append(chains_encrypt)
aa = '\x61'
rr = '\x6f'
slither = aa + db + nn + ef + rr + gh + lr + ty
```

All of the variables that make up slither are in the form of '\x**' which is an excape clause indicating hex values. from that we can get that slither is a string created by the hex values: 61 6e 61 63 6f 6e 64 61. I threw these in a simple hex to ascii converter to see what it came out as and got the username 'anaconda'.



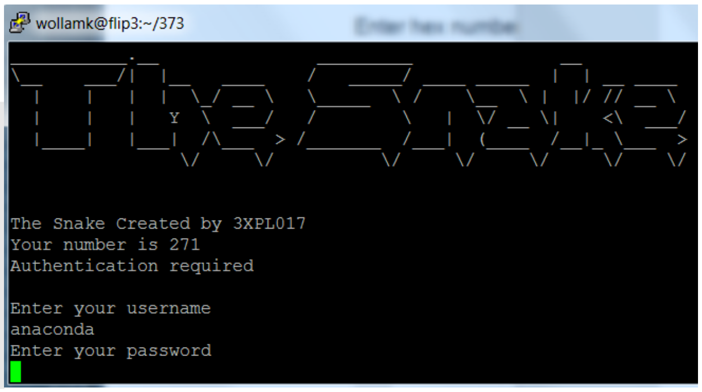
Hex to ASCII text converter

Enter hex numbers with any prefix / postfix / delimiter and press the *Convert* button (e.g. 45 78 61 6d 70 6C 65 21):

```
61 6e 61 63 6f 6e 64 61
```

Character encoding: ASCII

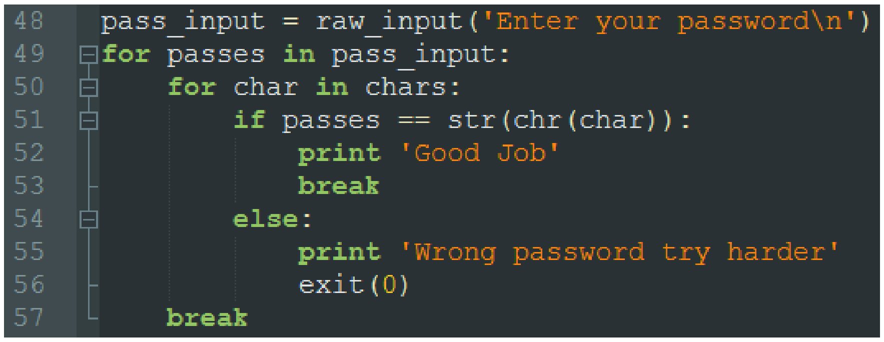
```
anaconda
```



wollamk@flip3:~/373

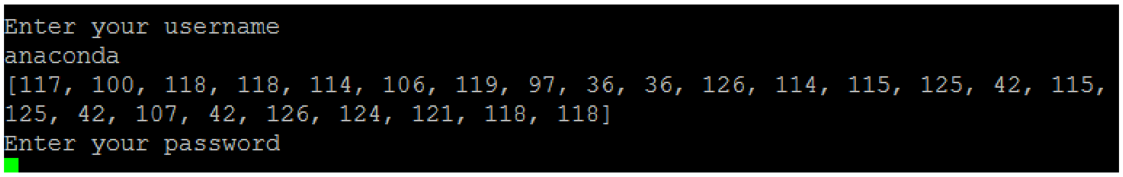
```
___________.__               _________              __
\__    ___/|  |__   ____    /   _____/ ____ _____  |  | __ ____
  |    |   |  |  \_/ __ \   \_____  \ /    \\__  \ |  |/ // __ \
  |    |   |   Y  \  ___/   /        \   |  \/ __ \|    <\  ___/
  |____|   |___|  /\___  > /_______  /___|  (____  /__|_ \\___  >
                \/     \/          \/     \/     \/     \/    \/


The Snake Created by 3XPL017
Your number is 271
Authentication required

Enter your username
anaconda
Enter your password
```

Phase 1 complete, our username seems to have worked. Now to figure out where the password is hiding.



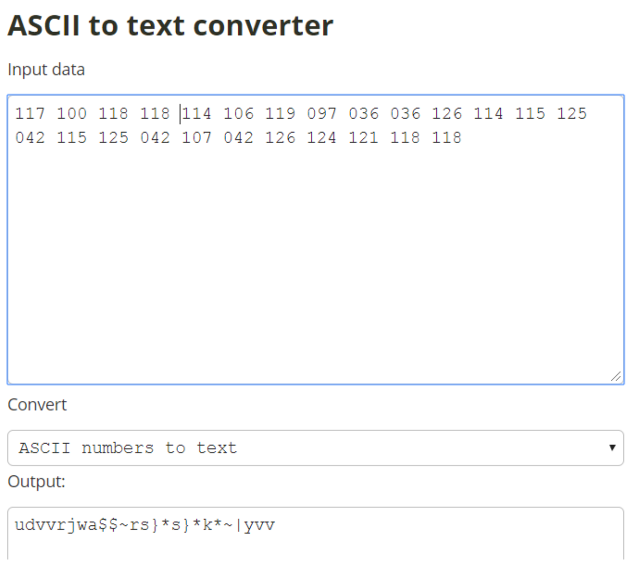
```
pass_input = raw_input('Enter your password\n')
for passes in pass_input:
    for char in chars:
        if passes == str(chr(char)):
            print 'Good Job'
            break
        else:
            print 'Wrong password try harder'
            exit(0)
    break
```

From this we see that the user input is stored as pass_input, and then iterated across by characters. Those are compared to str(chr(char)). Looking at the code, char is a fairly complicated variable. To avoid having to figure out the whole string I decided to simply add a print chars statement. This yielded a string of values.



```
Enter your username
anaconda
[117, 100, 118, 118, 114, 106, 119, 97, 36, 36, 126, 114, 115, 125, 42, 115,
125, 42, 107, 42, 126, 124, 121, 118, 118]
Enter your password
```

I then took these and converted them from ascii codes to text



ASCII to text converter

Input data

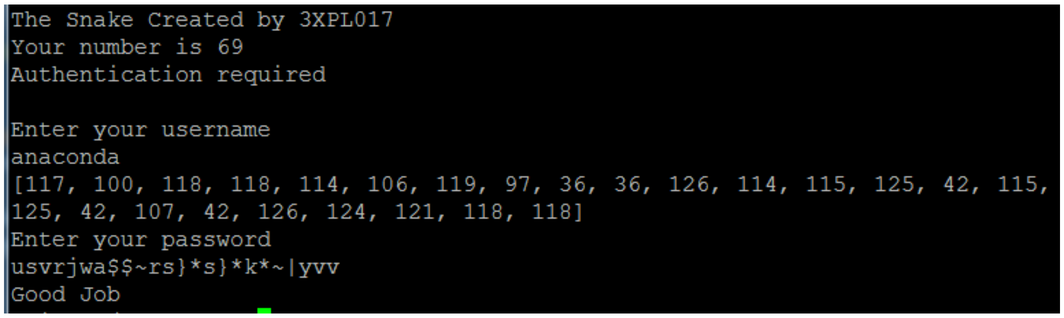
```
117 100 118 118 114 106 119 097 036 036 126 114 115 125
042 115 125 042 107 042 126 124 121 118 118
```

Convert

ASCII numbers to text

Output:

```
udvvrjwa$$~rs}*s}*k*~|yvv
```



```
The Snake Created by 3XPL017
Your number is 69
Authentication required

Enter your username
anaconda
[117, 100, 118, 118, 114, 106, 119, 97, 36, 36, 126, 114, 115, 125, 42, 115,
125, 42, 107, 42, 126, 124, 121, 118, 118]
Enter your password
usvrjwa$$~rs}*s}*k*~|yvv
Good Job
```

This password worked and we got the Good Job! Unfortunately even though the challenge states that the flag is in the format HTB{username:password} using HTB{anaconda: udvvrjwa$$~rs}*s}*k*~|yvv} does not work. To try to figure out what I was missing I took all the arrays of hex and ran them through a hex to text converter.

chains = [0x74, 0x68, 0x69, 0x73, 0x20, 0x69, 0x73, 0x20, 0x61, 0x20, 0x74, 0x72, 0x6f, 0x6c, 0x6c]

“this is a troll”

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

“password!!”

password = [0x69, 0x74, 0x73, 0x20, 0x6e, 0x6f, 0x74, 0x20, 0x74, 0x68, 0x61, 0x74, 0x20, 0x65, 0x61, 0x73, 0x79]

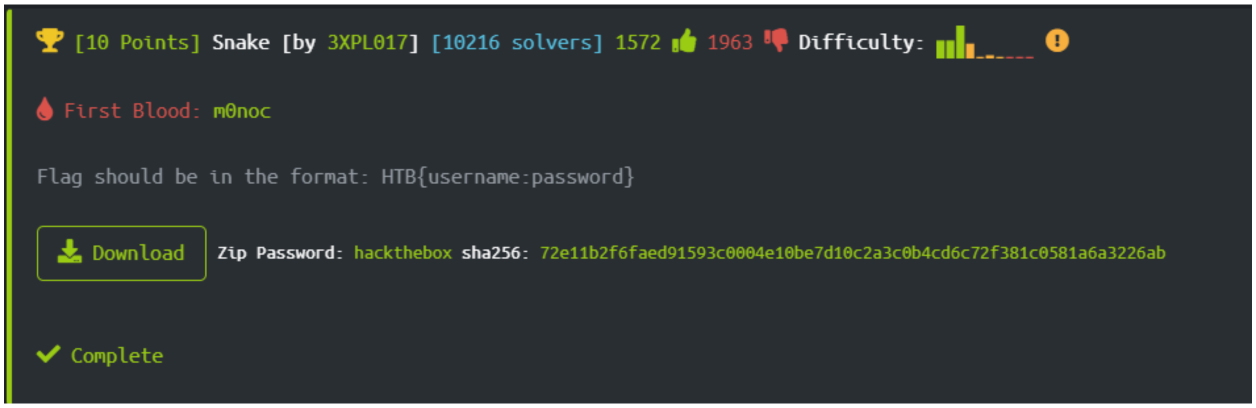
“its not that easy”

auth = [0x6b, 0x65, 0x65, 0x70, 0x20, 0x74, 0x72, 0x79, 0x69, 0x6e, 0x67]

“keep trying”

Since chars was made up of and encrypted version of keys then chains, and keys says password literally while chains says that it is a troll, I decided to try jut the first part of the password that came from keys “udvvrjwa$$” and it worked!

I also realized that I had made a couple typos in the password when I entered it, and the code only cares that it starts with ‘u’ the very first character then breaks. While I am glad that I finally got this one, I think it was actually not a very well created challenge. Thumbs down. Next time I will make sure to pick one with more thumbs ups.


[10 Points] Snake [by 3XPL017] [10216 solvers] 1572 1963 Difficulty:

First Blood: m0noc

Flag should be in the format: HTB{username:password}

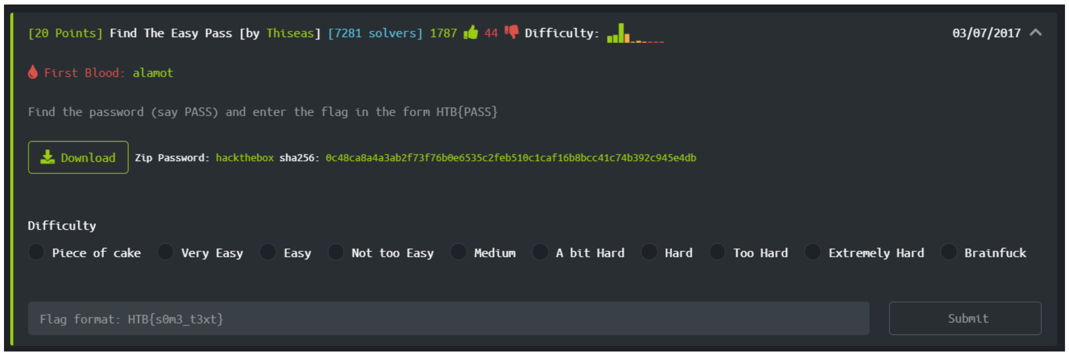
Download Zip Password: hackthebox sha256: 72e11b2f6faed91593c0004e10be7d10c2a3c0b4cd6c72f381c0581a6a3226ab

Complete

Challenge 2: Find the Easy Pass

Reversing, 20 Points

For my second challenge I went with one a little more challenging for 20 points but with a way better ratio of thumbs ups. The task is simple, find the password.



[20 Points] Find The Easy Pass [by Thiseas] [7281 solvers] 1787 44 Difficulty: 03/07/2017

First Blood: alamot

Find the password (say PASS) and enter the flag in the form HTB{PASS}

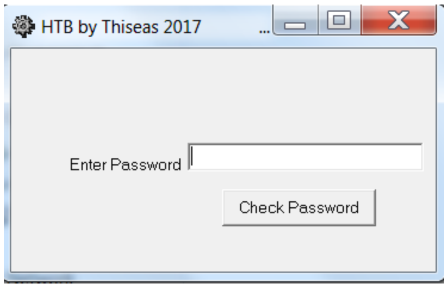
Download Zip Password: hackthebox sha256: 0c48ca8a4a3ab2f73f76b0e6535c2feb510c1caf16b8bcc41c74b392c945e4db

Difficulty

Piece of cake · Very Easy · Easy · Not too Easy · Medium · A bit Hard · Hard · Too Hard · Extremely Hard · Brainfuck

Flag format: HTB{s0m3_t3xt} · Submit

Launching EasyPass.exe gives us a simple window prompting for a password.

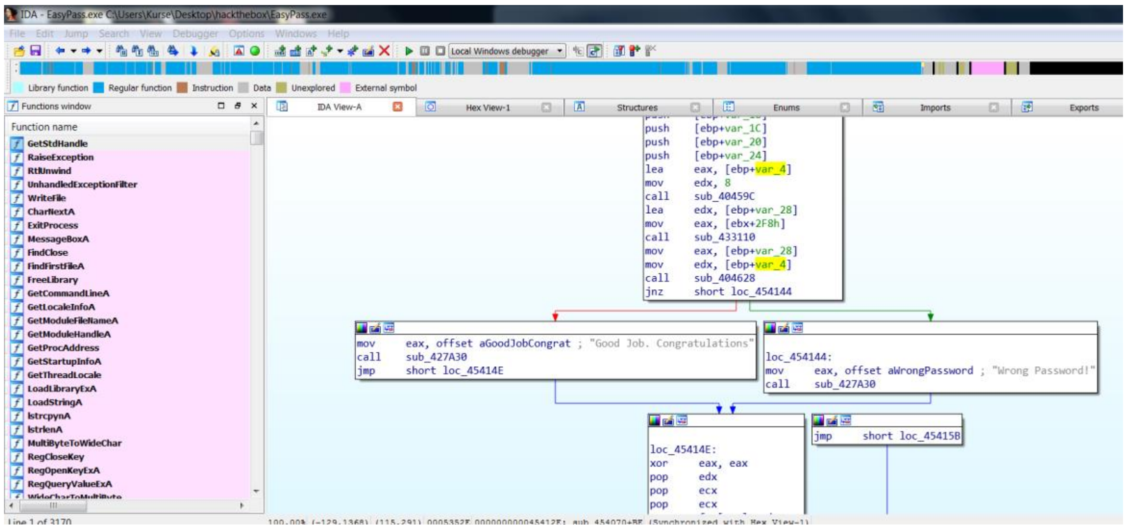


HTB by Thiseas 2017

Enter Password

Check Password

To achieve this I decided a good first step was to run the exe file through a decompiler. I downloaded the freeware version of IDA to explore. since I needed to find a password I searched for the term "password" and was able to locate the phrase "Wrong Password!"



```
push    [ebp+var_1C]
push    [ebp+var_20]
push    [ebp+var_24]
lea     eax, [ebp+var_4]
mov     edx, 8
call    sub_40459C
lea     edx, [ebp+var_28]
mov     eax, [ebx+2F8h]
call    sub_433110
mov     eax, [ebp+var_28]
mov     edx, [ebp+var_4]
call    sub_404628
jnz     short loc_454144
```

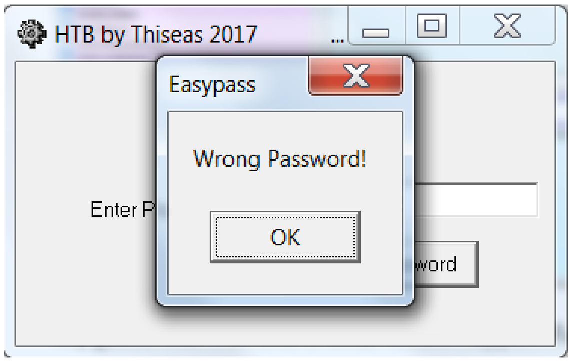
```
mov     eax, offset aGoodJobCongrat ; "Good Job. Congratulations"
call    sub_427A30
jmp     short loc_45414E
```

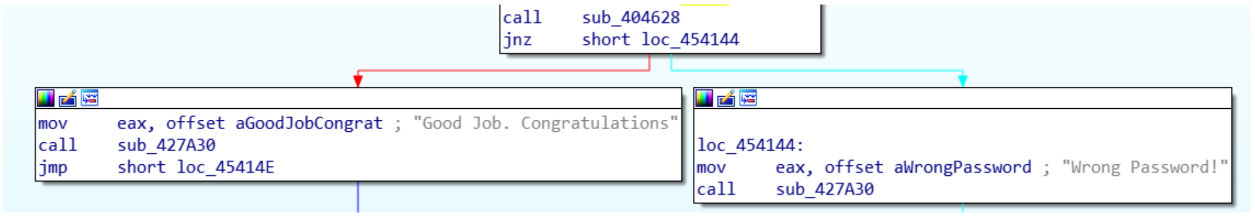
```
loc_454144:
mov     eax, offset aWrongPassword ; "Wrong Password!"
call    sub_427A30
```

```
loc_45414E:
xor     eax, eax
pop     edx
pop     ecx
pop     ecx
```

```
jmp     short loc_45415B
```

My guess was that this is what would happen if a wrong password is entered so I tried it out to confirm. Turns out that infected is not the password, but it does give us the "Wrong Password!" prompt.

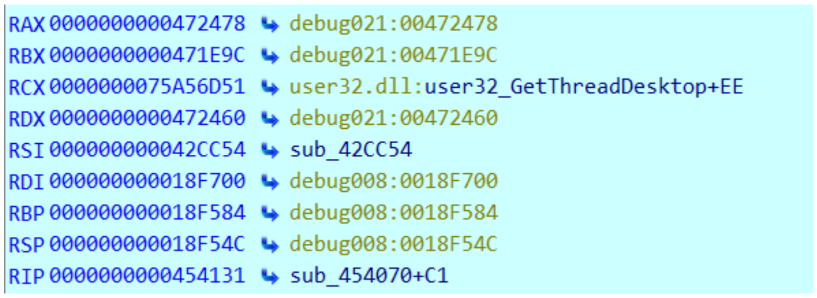




```
call    sub_404628
jnz     short loc_454144

mov     eax, offset aGoodJobCongrat ; "Good Job. Congratulations"
call    sub_427A30
jmp     short loc_45414E

loc_454144:
mov     eax, offset aWrongPassword ; "Wrong Password!"
call    sub_427A30
```

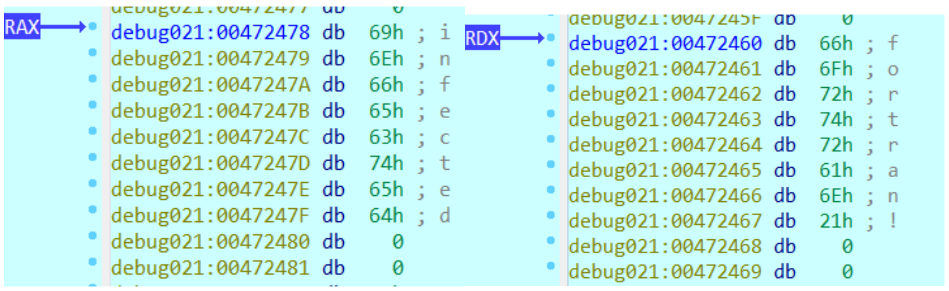
inspecting the IDA graph view we see that the "Wrong Password!" follows from a jnz jump call. If we do not jump then we would instead get a "Good Job. Congratulations." Directly before the check to jump is a call to a subprocess 404628. It seems likely that this process is likely the one that determines if the password is correct and sets the flag for the jump.

I set a breakpoint at that process call location to see if it yielded any information. The process has a cmp eax, edx call, so I decided to look into the registers



```
RAX 0000000000472478 ↳ debug021:00472478
RBX 0000000000471E9C ↳ debug021:00471E9C
RCX 0000000075A56D51 ↳ user32.dll:user32_GetThreadDesktop+EE
RDX 0000000000472460 ↳ debug021:00472460
RSI 000000000042CC54 ↳ sub_42CC54
RDI 000000000018F700 ↳ debug008:0018F700
RBP 000000000018F584 ↳ debug008:0018F584
RSP 000000000018F54C ↳ debug008:0018F54C
RIP 0000000000454131 ↳ sub_454070+C1
```

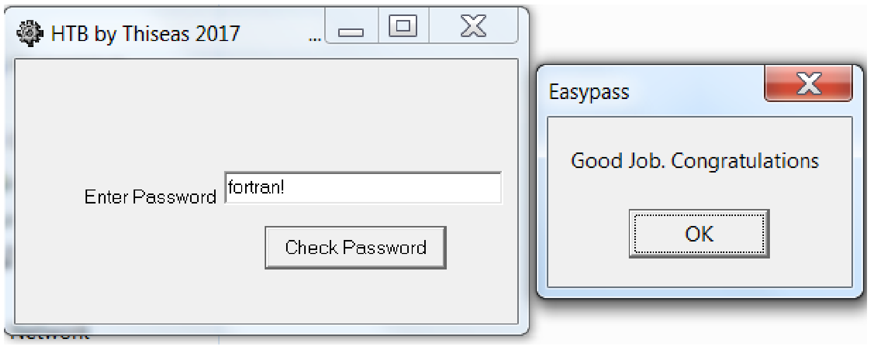
Sure enough RAX contains the password I entered, "infected". RDX also stores a clear string, "fortran!"



```
RAX → debug021:00472478 db  69h ; i
      debug021:00472479 db  6Eh ; n
      debug021:0047247A db  66h ; f
      debug021:0047247B db  65h ; e
      debug021:0047247C db  63h ; c
      debug021:0047247D db  74h ; t
      debug021:0047247E db  65h ; e
      debug021:0047247F db  64h ; d
      debug021:00472480 db    0
      debug021:00472481 db    0
```

```
RDX → debug021:00472460 db  66h ; f
      debug021:00472461 db  6Fh ; o
      debug021:00472462 db  72h ; r
      debug021:00472463 db  74h ; t
      debug021:00472464 db  72h ; r
      debug021:00472465 db  61h ; a
      debug021:00472466 db  6Eh ; n
      debug021:00472467 db  21h ; !
      debug021:00472468 db    0
      debug021:00472469 db    0
```

Entering “fortran!” yields the outcome we were looking for, “Good Job. Congratulations”


HTB by Thiseas 2017

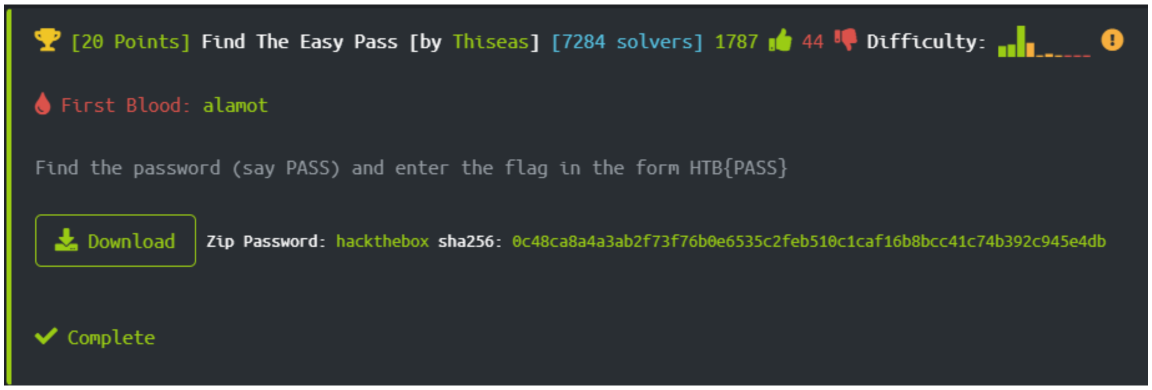
Enter Password fortran!

Check Password

Easypass

Good Job. Congratulations

OK


[20 Points] Find The Easy Pass [by Thiseas] [7284 solvers] 1787 44 Difficulty:

First Blood: alamot

Find the password (say PASS) and enter the flag in the form HTB{PASS}

Download Zip Password: hackthebox sha256: 0c48ca8a4a3ab2f73f76b0e6535c2feb510c1caf16b8bcc41c74b392c945e4db

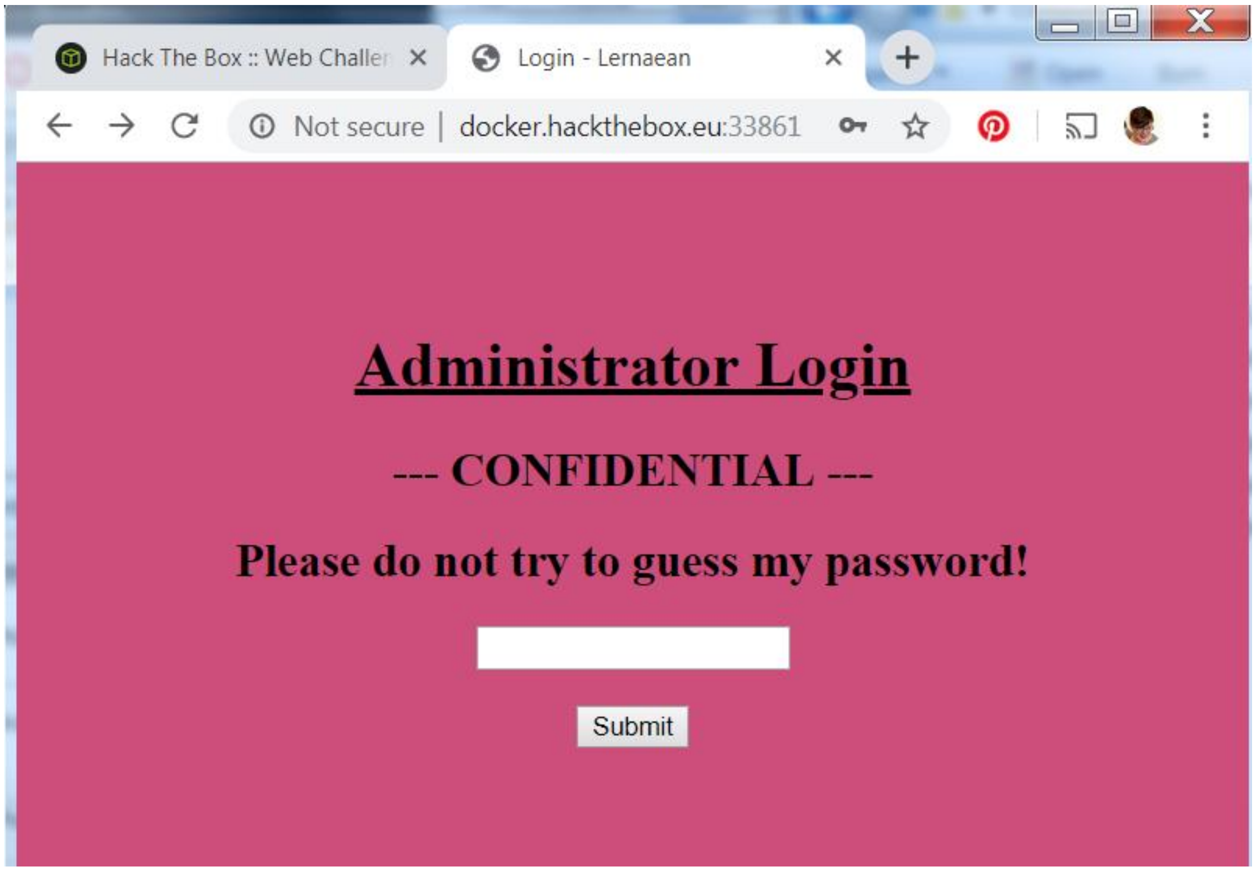
Complete

I felt this challenge was much better and gave it an enthusiastic thumbs up.

Challenge 3: Lernaean

Web, 20 Points

For my last challenge I decided to try a web challenge. I chose this one as it had a lot of up votes but was still on the easy side for this noob. The challenge has the following prompt "Your target is not very good with computers. Try and guess their password to see if they may be hiding anything!" After getting behind the openvpn I was able to access the challenge and was presented with a screen for administrator log in.

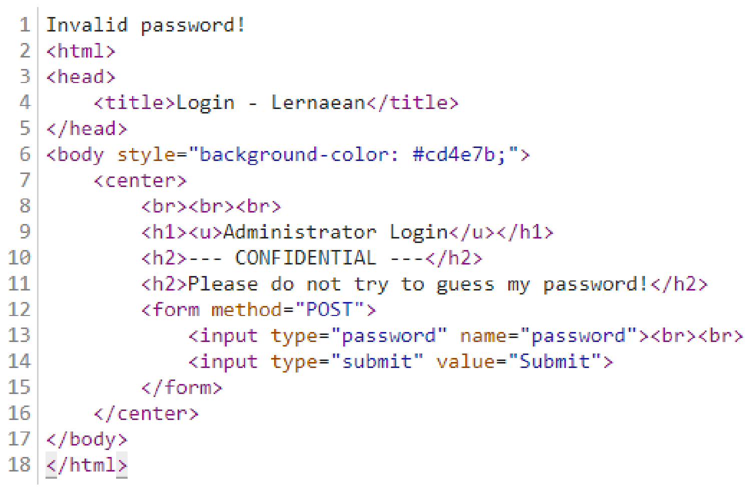

Hack The Box :: Web Challe
Login - Lernaean
Not secure | docker.hackthebox.eu:33861
Submit

Administrator Login

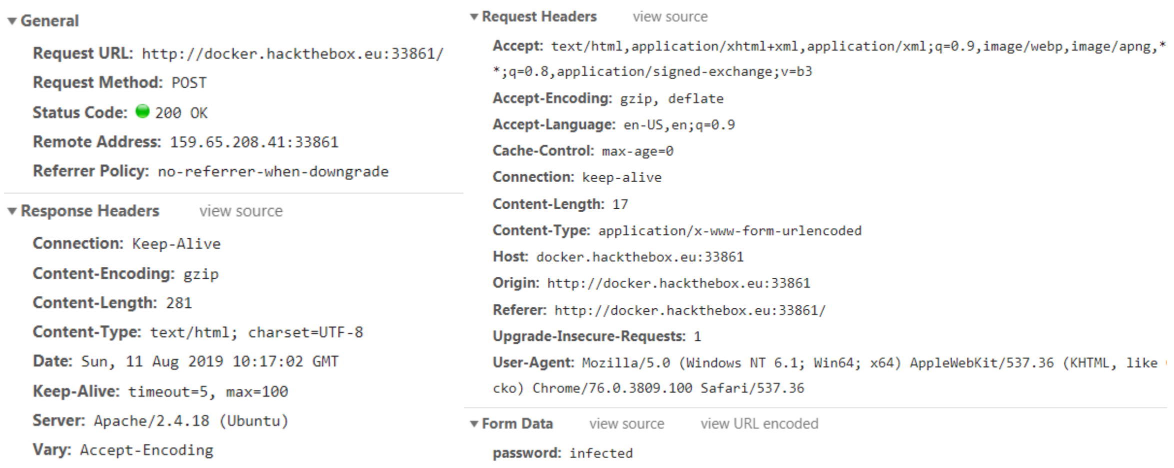
--- CONFIDENTIAL ---

Please do not try to guess my password!

When presented a challenge that states "Please do not try to guess my password!" the obvious first step is to try to guess the password. I tried "infected" which resulted in a message on the screen saying "Invalid password!" The source code for the site is quite simple, and seems to be sending the password via post but does not give us much additional information.



```
Invalid password!
<html>
<head>
    <title>Login - Lernaean</title>
</head>
<body style="background-color: #cd4e7b;">
    <center>
        <br><br><br>
        <h1><u>Administrator Login</u></h1>
        <h2>--- CONFIDENTIAL ---</h2>
        <h2>Please do not try to guess my password!</h2>
        <form method="POST">
            <input type="password" name="password"><br><br>
            <input type="submit" value="Submit">
        </form>
    </center>
</body>
</html>
```



```
▼ General
    Request URL: http://docker.hackthebox.eu:33861/
    Request Method: POST
    Status Code: ● 200 OK
    Remote Address: 159.65.208.41:33861
    Referrer Policy: no-referrer-when-downgrade

▼ Response Headers    view source
    Connection: Keep-Alive
    Content-Encoding: gzip
    Content-Length: 281
    Content-Type: text/html; charset=UTF-8
    Date: Sun, 11 Aug 2019 10:17:02 GMT
    Keep-Alive: timeout=5, max=100
    Server: Apache/2.4.18 (Ubuntu)
    Vary: Accept-Encoding

▼ Request Headers    view source
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*
    *;q=0.8,application/signed-exchange;v=b3
    Accept-Encoding: gzip, deflate
    Accept-Language: en-US,en;q=0.9
    Cache-Control: max-age=0
    Connection: keep-alive
    Content-Length: 17
    Content-Type: application/x-www-form-urlencoded
    Host: docker.hackthebox.eu:33861
    Origin: http://docker.hackthebox.eu:33861
    Referer: http://docker.hackthebox.eu:33861/
    Upgrade-Insecure-Requests: 1
    User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like
    cko) Chrome/76.0.3809.100 Safari/537.36

▼ Form Data    view source    view URL encoded
    password: infected
```

Without much more to go on I wondered if there were other clues in the challenge information. I decided to google the name of the challenge, as they usually seem to provide information but this one didn't seem to mean anything to me. Turns out Lernaean is a Hydra from Greek mythology.

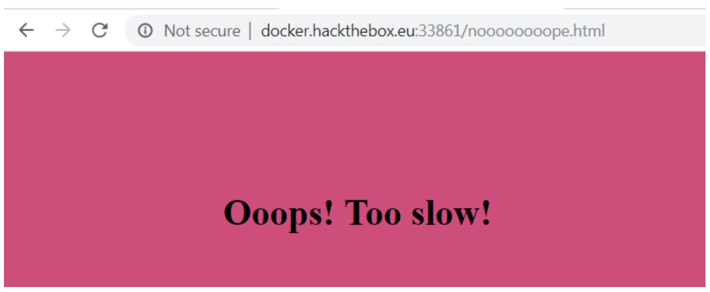
Lernaean Hydra

The Lernaean Hydra or Hydra of Lerna, more often known simply as the Hydra, is a serpentine water monster in Greek and Roman mythology. Its lair was the lake of Lerna in the Argolid, which was also the site of the myth of the Danaïdes. Wikipedia

Parents: Typhon and Echidna

Mythology: Greek mythology

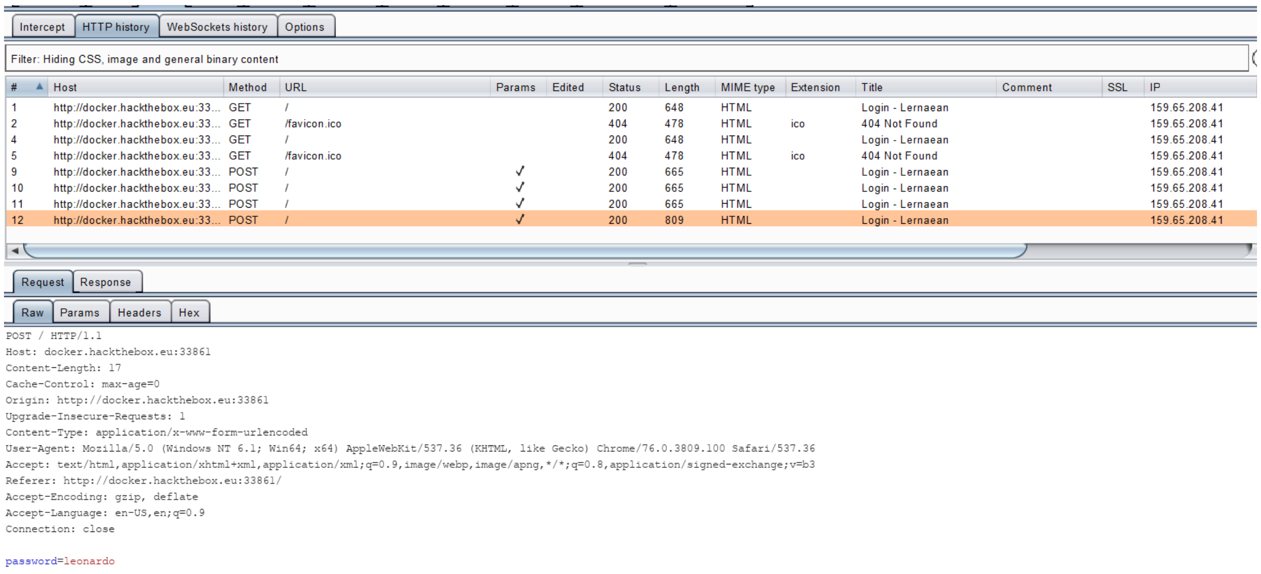
I tried hydra as the password but this did not work so I did some more google sleuthing and discovered that Hydra is actually a very popular password cracker. This lead me to believe that I would need to brute force this password. Using Hydra I was able to get a password of 'leonardo'. This does seem to work, but unfortunately just sends us to this page:



Not secure | docker.hackthebox.eu:33861/noooooooope.html

Ooops! Too slow!

This is quite disheartening. I closed out and loaded a new instance, but even putting the password in right away is not fast enough. I decided to look at the post request again. Everything seemed in order so I decided to install some more tools to try to figure out what to do and where to go. The forums had a lot to say about BurpSuite, so I found that and installed it. I spent a significant amount of time learning how to set it up and use it and then captured packets from the Lernaen instance.

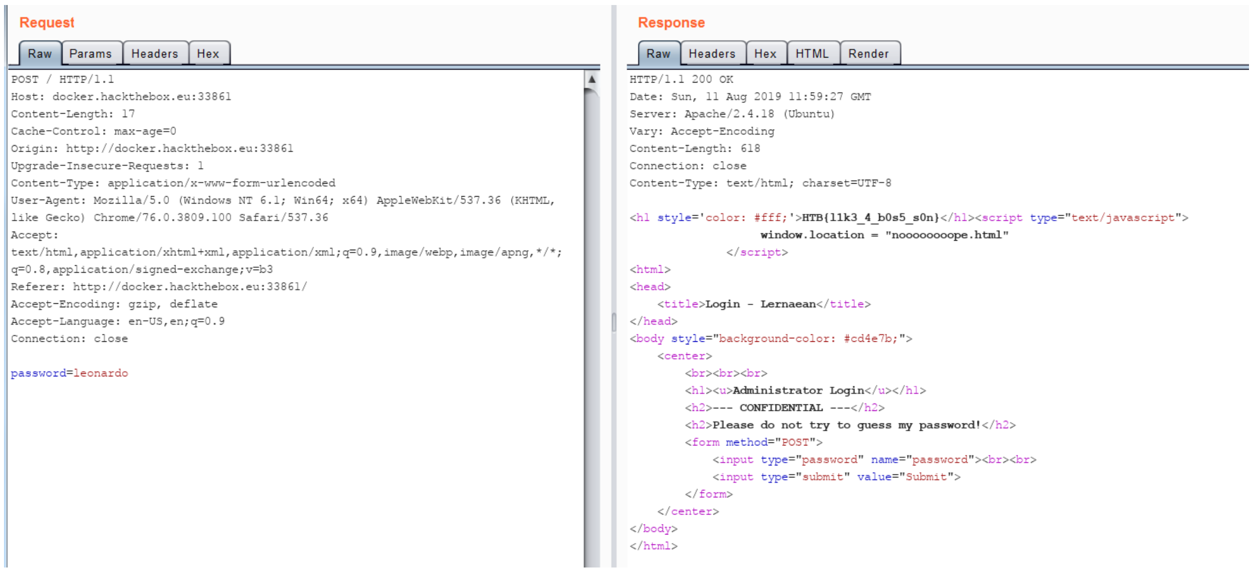


Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	SSL	IP
1	http://docker.hackthebox.eu:33...	GET	/			200	648	HTML		Login - Lernaean			159.65.208.41
2	http://docker.hackthebox.eu:33...	GET	/favicon.ico			404	478	HTML	ico	404 Not Found			159.65.208.41
4	http://docker.hackthebox.eu:33...	GET	/			200	648	HTML		Login - Lernaean			159.65.208.41
5	http://docker.hackthebox.eu:33...	GET	/favicon.ico			404	478	HTML	ico	404 Not Found			159.65.208.41
9	http://docker.hackthebox.eu:33...	POST	/	✓		200	665	HTML		Login - Lernaean			159.65.208.41
10	http://docker.hackthebox.eu:33...	POST	/	✓		200	665	HTML		Login - Lernaean			159.65.208.41
11	http://docker.hackthebox.eu:33...	POST	/	✓		200	665	HTML		Login - Lernaean			159.65.208.41
12	http://docker.hackthebox.eu:33...	POST	/	✓		200	809	HTML		Login - Lernaean			159.65.208.41

```
POST / HTTP/1.1
Host: docker.hackthebox.eu:33861
Content-Length: 17
Cache-Control: max-age=0
Origin: http://docker.hackthebox.eu:33861
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.100 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Referer: http://docker.hackthebox.eu:33861/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

password=leonardo
```

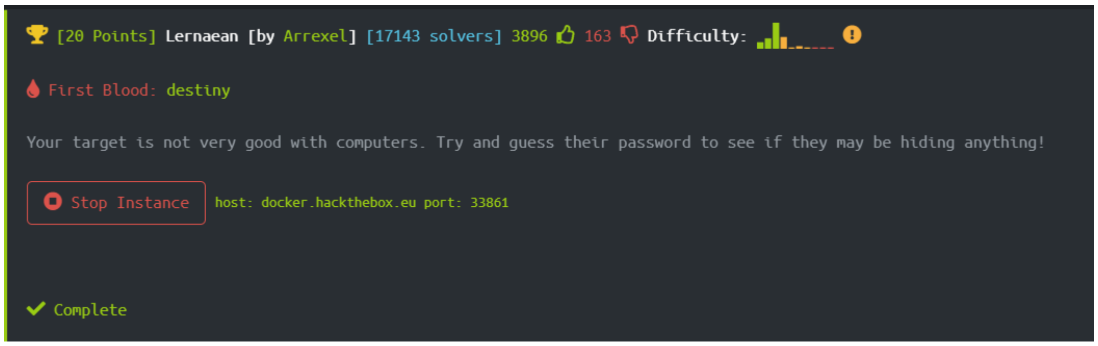
From here I poked around, and finally found gold by sending the post request to the repeater. This let me see the raw response with hilariously had the flag hidden in the html.



```
POST / HTTP/1.1
Host: docker.hackthebox.eu:33861
Content-Length: 17
Cache-Control: max-age=0
Origin: http://docker.hackthebox.eu:33861
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/76.0.3809.100 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;
q=0.8,application/signed-exchange;v=b3
Referer: http://docker.hackthebox.eu:33861/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

password=leonardo
```

```
HTTP/1.1 200 OK
Date: Sun, 11 Aug 2019 11:59:27 GMT
Server: Apache/2.4.18 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 618
Connection: close
Content-Type: text/html; charset=UTF-8

<h1 style='color: #fff;'>HTB{l1k3_4_b0s5_s0n}</h1><script type="text/javascript">
                   window.location = "noooooooope.html"
               </script>
<html>
<head>
    <title>Login - Lernaean</title>
</head>
<body style="background-color: #cd4e7b;">
    <center>
        <br><br><br>
        <h1><u>Administrator Login</u></h1>
        <h2>--- CONFIDENTIAL ---</h2>
        <h2>Please do not try to guess my password!</h2>
        <form method="POST">
            <input type="password" name="password"><br><br>
            <input type="submit" value="Submit">
        </form>
    </center>
</body>
</html>
```

I am still not sure exactly why this information doesn't show up in the build in chrome tools, but I sure am glad to have found it finally. This one took me a lot longer requiring me to learn several tools, but it was interesting and I feel like I learned a lot of good techniques.



[20 Points] Lernaean [by Arrexel] [17143 solvers] 3896 163 Difficulty:

First Blood: destiny

Your target is not very good with computers. Try and guess their password to see if they may be hiding anything!

host: docker.hackthebox.eu port: 33861

Complete