i18 Challenge - Part 2

4 minute read Published: 17 Jul, 2019

> Here is the second part of the i18 CTF with 4 more challenges. This time I
> learned about Reverse Engineering, DNS lookup, more RE, and finally some
> steganography that ended up not working.

5: Lett fluidmekanikk

An executable file.
Clue one: The program also accepts the password as an argument.
Clue two: The password exists (perhaps) in a dictionary.

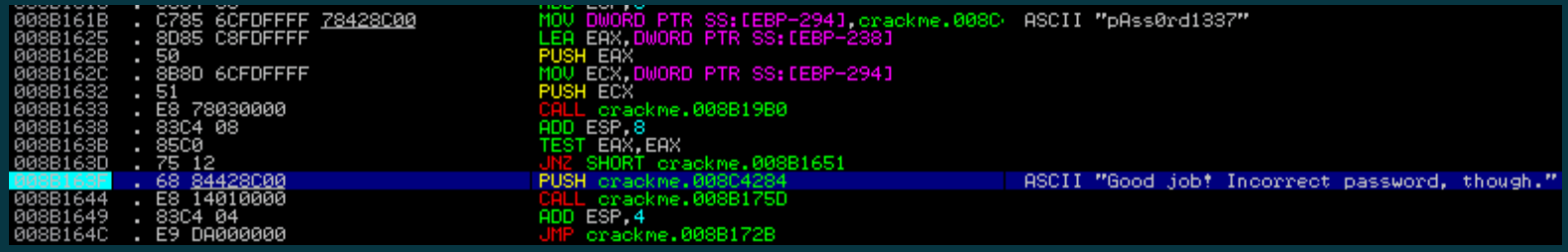
We get an executable file asking for a password. Enter the wrong one and it quits. I'm a complete beginner in Reverse Engineering but I have enjoyed a few tutorial videos so let's jump right in with Immunity.

> When you open the file, right click > View > module 'crackme'. Then right click > Search for > All reference text strings. It will show you the lines of text that the software shows us, for example "ASCII "Please enter your password: "". Double click on this one.

Now the goal is to find a way to bypass the login system. If we check the other strings we can see a password!

```
008B161B . C785 6CFDFFFF 78428C00 MOV DWORD PTR SS:[EBP-294],crackme.008C4>;
ASCII "pAss0rd1337"
```

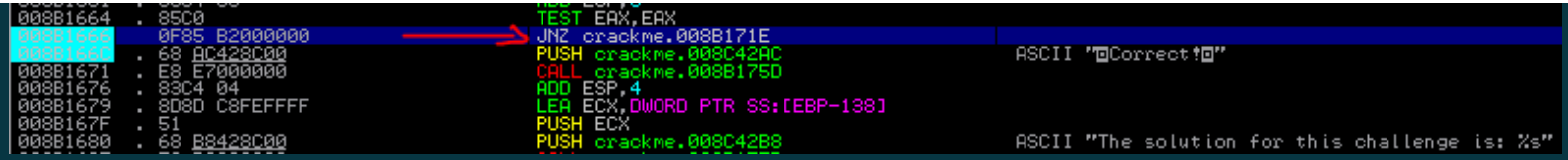
If we try it though, the program just trolls us and shows “Good job! Incorrect password, though.” before exiting.



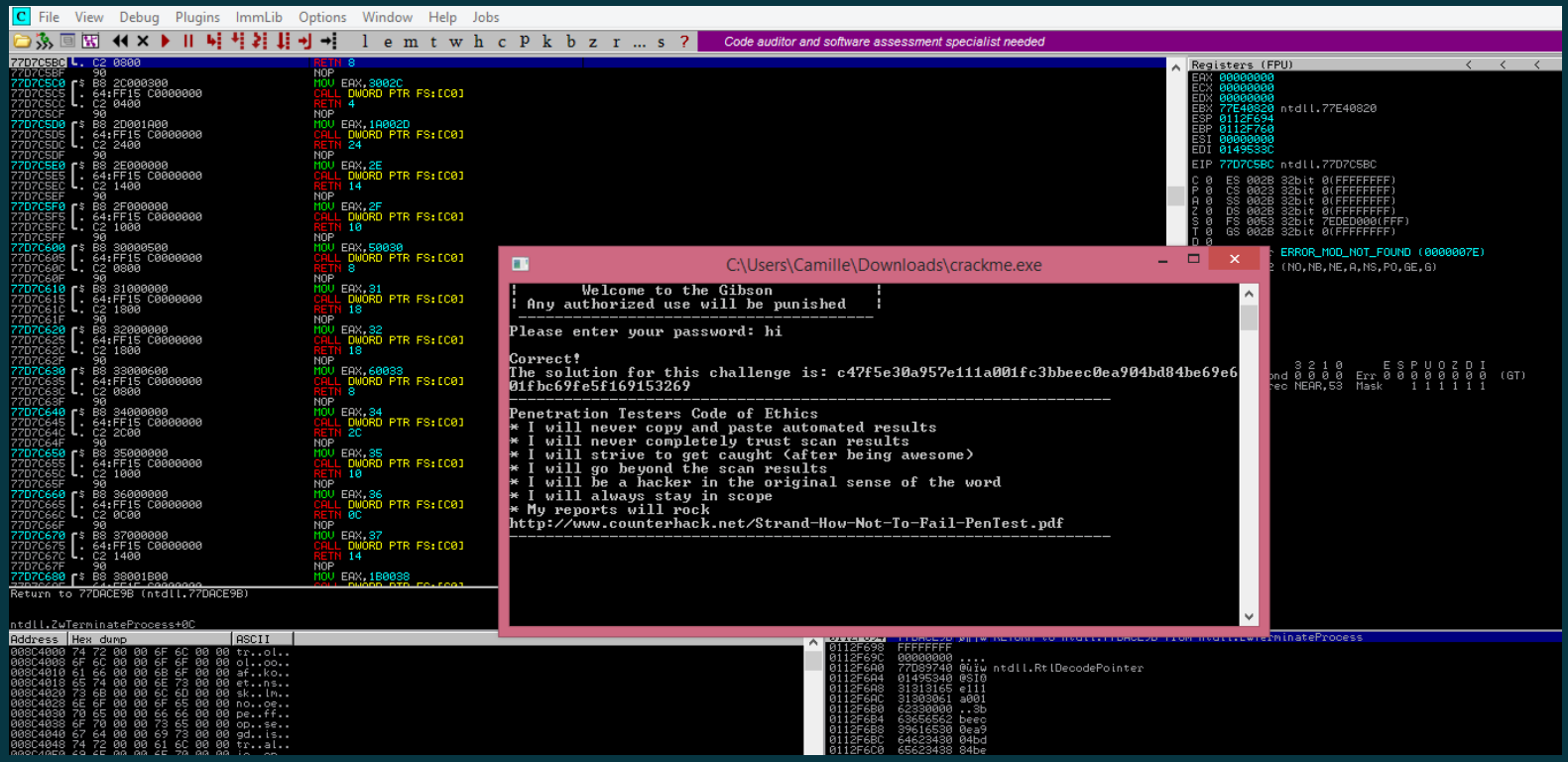
The important part to look for now are the JNZ:

```
JNZ - The jump WILL take place if the Z Flag is NOT zero (1)
CMP - If the two values are equal, the Z Flag is set (1) otherwise it is not
set (0)
```

We see several JNZ, and if we check the one right after the fake password trolling, it goes to 008B172B. So this is the exit of the program. What about the correct answer though?



In this screenshot, we can see that right before the Correct line, we have a jump happening that goes to 008B171E which is the Incorrect option and then the program exit. So by simply replacing the assembly of this jump with JZ, we bypass the verification system as now it will avoid the jump if the password is NOT the right one and give us the Correct answer!



6: For the record…

This is just to look up.
Clue one: For example, in a distributed, hierarchical directory service.
Clue two: challenge.i18.no. CLASS1 TYPE16

The look up part gave it up kind of for me. I started directly looking for a DNS recon using nslookup:

```
nslookup challenge.i18.no
Server: 10.11.0.1
Address: 10.11.0.1#53

Non-authoritative answer:
Name: challenge.i18.no
Address: 185.56.186.11
```

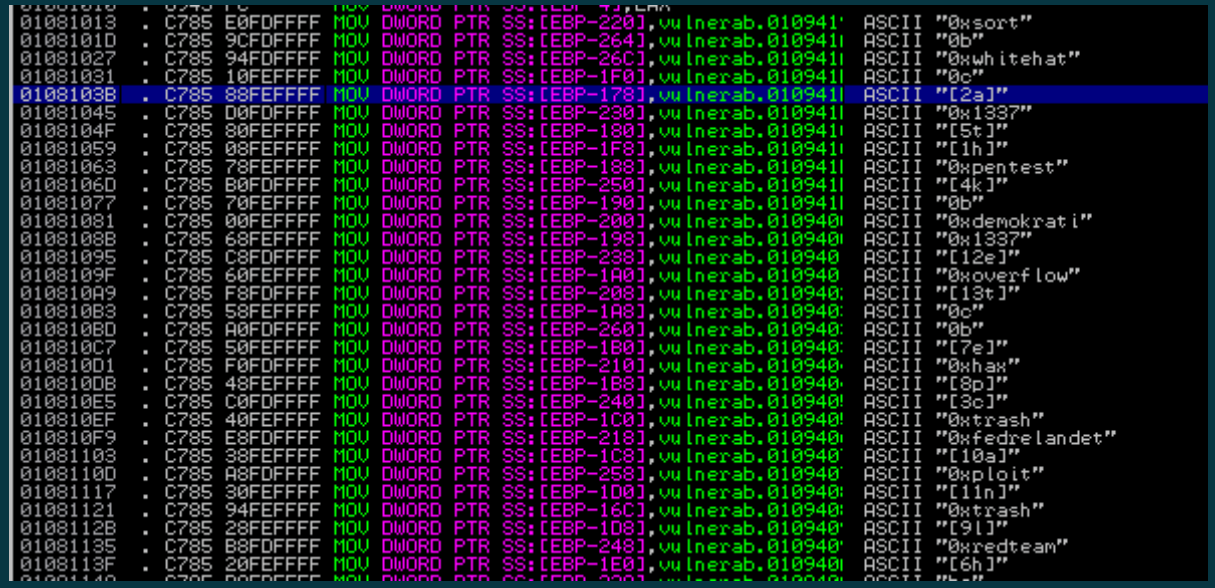
Now I checked all the possible record types: A, MX, and TXT.

In the TXT records we find the answer!

7: Not quite a 0day

Another executable file. Apparently vulnerable, by the file name to judge. Hmm.
Clue one: The task can be solved in several ways.
Clue two: Some features of C can make your program vulnerable if you use them incorrectly.
Clue three: Look for square brackets and align in ascending order.

I spent a long time on this one. I thought it would be another Reverse Engineering job so started to look at the program in Immunity, but for almost 2hrs I couldn't find anything. I checked the first two clues which confused me even more so after another half an hour of googling about C vulnerabilities, I checked the last clue. It said to look for square brackets, so I did just that in Immunity.



Found 13 lines of them and after ordering them, the letters contained in each bracket make up the line “hacktheplanet”.

```
echo -n hacktheplanet | sha256sum
48f6018bc6898a5c9e61d549b174131c07ed70542ba1c326289b9cc35af22f84
```

Probably the weirdest challenge so far, very puzzle / CTF-like.

8: På riktig bølgelengde

Clue: Things are not always as they appear.

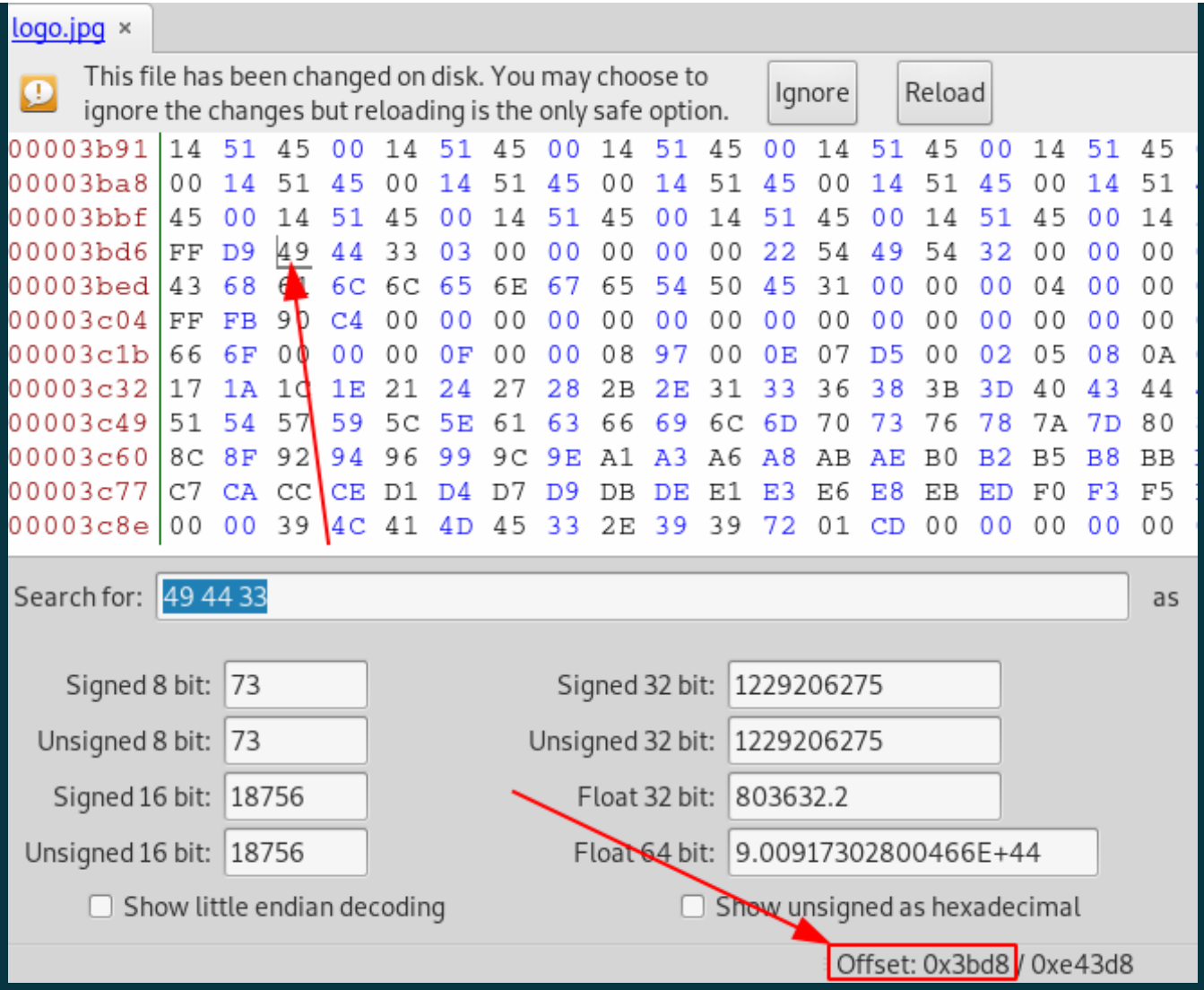
We just got this image. After doing a little bit of info recon on the file itself, not much comes up except the size of it for such a small image. So that and the clue tells me that something is up.

After a little bit of research, I find this useful article about hidden files in images. One important information is that in the HEX, a jpg has the terminating byte of ffd9. If we search for it with:

```
cat logo.jpg | grep 'ffd9'
```

```
root@kali:~/Downloads# cat logo.jpg |grep '4944'
00003bd0: 0014 5145 007f ffd9 4944 3303 0000 0000  ..QE....ID3.....
00020e50: 30a6 4944 846b 2c4c 4671 c68c 6774 0346  0.ID.k,LFq..gt.F
000240b0: 4724 ca24 73cd 848d c0c5 dcb5 cc5a 4944  G$.$s........ZID
0003e680: e640 5570 4944 158a 08d0 4444 4759 1d5d  .@UpID....DDGY.]
00049440: c65f 6ac5 79fe 6ceb 324d 09b2 0382 181b  ._j.y.l.2M......
000580b0: 252f 6a7d 2089 44e0 4944 1ab5 9e0c a768  %/j} .D.ID.....h
0006baa0: 4944 817f 35e5 0699 0b81 c042 4c04 5c94  ID..5......BL.\.
0006dd90: 60b4 4944 7421 20a9 1c84 93d2 b2b3 1809  `.IDt! .........
000737f0: cfc0 4944 a89f 6690 ae62 21c5 1244 8960  ..ID..f..b!..D.`
000a1b20: eba0 e030 22c0 97dd 11d3 4944 0040 11c0  ...0".....ID.@..
000b62e0: 4944 3122 8e42 0e73 b8f0 3fd5 4859 f87b  ID1".B.s..?.HY.{
000e02c0: 4642 6ddd 554d 4944 6c89 ce28 b1f3 b6cc  FBm.UMIDl..(....
```

We see that there is still a lot going on after the ffd9 where it should have ended. We have “4944 3303” as the following bytes, and with a quick google search we find out that it is the initial bytes of an MP3 file. Still thanks to the previous article, we know that we can use the dd command to extract it. But first we need to find the offset in an hex editor. I used Bless Hex Editor which has a nice and simple GUI. I found the offset “0x3BD8” where the MP3 file starts:

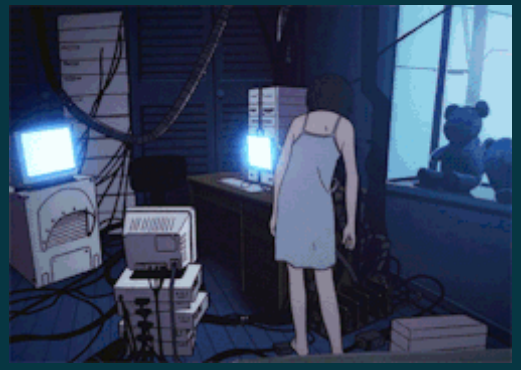


Still according to the article linked above: “dd only takes decimal values, so we convert the hexadecimal location from hex to decimal”. Which gives us the decimal 15320. Now we can run the dd command:

```
dd if=logo.jpg bs=1 skip=15320 of=logo.mp3
```

The file is created but after that I have been lost for too long. I can’t find a way to make this MP3 work so I gave up for now, and I’ll probably come back to this challenge later. If anyone spots an obvious mistake I made, please contact me!

See you for part 3!



Published by theyknow 17 Jul, 2019 in post and tagged ctf, forensic, i18, nslookup, re, reverse and web using 842 words.

Related Content

- i18 Challenge - Part 1 – 5 minutes
- Pentesting tools – 8 minutes