

[Home](#) [Posts](#) [Tools](#) [Twitter](#) [GitHub](#) [@](#)

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.

```
008B161B . C785 6CFDFFFF 78428C00 MOV DWORD PTR SS:[EBP-294],crackme.008C4>; ASCII "pAss0rd1337"
008B1625 . 8D85 C8FDFFFF LEA EAX,DWORD PTR SS:[EBP-288]
008B162B . 50 PUSH EAX
008B162C . 8B8D 6CFDFFFF MOV ECX,DWORD PTR SS:[EBP-294]
008B1632 . 51 PUSH ECX
008B1633 . E8 78030000 CALL crackme.008B19B0
008B1638 . 83C4 08 ADD ESP,8
008B163B . 85C0 TEST EAX,EAX
008B163D . 75 12 JNZ SHORT crackme.008B1651
008B163E . 68 84428C00 PUSH crackme.008C4284 ASCII "Good job! Incorrect password, though."
008B1644 . E8 14010000 CALL crackme.008B175D
008B1649 . 83C4 04 ADD ESP,4
008B164C . E9 DA000000 JMP crackme.008B172B
```

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?

```

008B1664 . 85C0          TEST EAX,EAX
008B1665 . 0F85 B2000000 JNZ crackme.008B171E
008B1666 . 68 0C428C00   PUSH crackme.008C428C ASCII "Correct!"
008B1671 . E8 E7000000   CALL crackme.008B175D
008B1676 . 83C4 04       ADD ESP,4
008B1679 . 8D8D C8FEFFFF LEA ECX,DWORD PTR SS:[EBP-138]
008B167F . 51           PUSH ECX
008B1680 . 68 B8428C00   PUSH crackme.008C4288 ASCII "The solution for this challenge is: %s"

```

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!

The screenshot shows a debugger window with assembly code on the left and a crackme application window in the center. The assembly code is for a function named `7707C5B0`. The application window displays the following text:

```

Welcome to the Gibson
Any authorized use will be punished

Please enter your password: hi

Correct!
The solution for this challenge is: c47f5e30a957e11a001fc3bbeec0ea904bd84be69e6
01fbc69fe5f169153269

Penetration Testers Code of Ethics
* I will never copy and paste automated results
* I will never completely trust scan results
* I will strive to get caught (after being awesome)
* I will go beyond the scan results
* I will be a hacker in the original sense of the word
* I will always stay in scope
* My reports will rock
http://www.counterhack.net/Strand-How-Not-To-Fail-PenTest.pdf

```

The debugger window also shows the registers (FPU) on the right, with `EAX` containing `00000000` and `ECX` containing `00000000`. The application window title is `C:\Users\Camille\Downloads\crackme.exe`.

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!

```
root@kali:~# nslookup -type=txt challenge.i18.no
Server:      10.11.0.1
Address:     10.11.0.1#53

Non-authoritative answer:
challenge.i18.no  text = "answer=f627295e1ede26fcf83fb247cdbca58376a2d07da0c46e16eb6ca85a0665ba43"

Authoritative answers can be found from:
i18.no  nameserver = ns1.hyp.net.
i18.no  nameserver = ns3.hyp.net.
i18.no  nameserver = ns2.hyp.net.
```

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.

```
01081010 . C785 E0FDFFFF MOV DWORD PTR SS:[EBP-220],vulnerab.010941 ASCII "0xsort"
01081013 . C785 9CFDFFFF MOV DWORD PTR SS:[EBP-264],vulnerab.010941 ASCII "0b"
01081027 . C785 94FDFFFF MOV DWORD PTR SS:[EBP-26C],vulnerab.010941 ASCII "0xwhitehat"
01081031 . C785 10FEFFFF MOV DWORD PTR SS:[EBP-1F0],vulnerab.010941 ASCII "0c"
0108103B . C785 88FEFFFF MOV DWORD PTR SS:[EBP-178],vulnerab.010941 ASCII "[2a]"
01081045 . C785 D0FDFFFF MOV DWORD PTR SS:[EBP-230],vulnerab.010941 ASCII "0x1337"
0108104F . C785 80FEFFFF MOV DWORD PTR SS:[EBP-180],vulnerab.010941 ASCII "[5t]"
01081059 . C785 08FEFFFF MOV DWORD PTR SS:[EBP-1F8],vulnerab.010941 ASCII "[1h]"
01081063 . C785 78FEFFFF MOV DWORD PTR SS:[EBP-188],vulnerab.010941 ASCII "0xpentest"
0108106D . C785 B0FDFFFF MOV DWORD PTR SS:[EBP-250],vulnerab.010941 ASCII "[4k]"
01081077 . C785 70FEFFFF MOV DWORD PTR SS:[EBP-190],vulnerab.010941 ASCII "0b"
01081081 . C785 00FEFFFF MOV DWORD PTR SS:[EBP-200],vulnerab.010940 ASCII "0xdemokrati"
0108108B . C785 68FEFFFF MOV DWORD PTR SS:[EBP-198],vulnerab.010940 ASCII "0x1337"
01081095 . C785 C8FDFFFF MOV DWORD PTR SS:[EBP-238],vulnerab.010940 ASCII "[12e]"
0108109F . C785 60FEFFFF MOV DWORD PTR SS:[EBP-1A0],vulnerab.010940 ASCII "0xoverflow"
010810A9 . C785 F8FDFFFF MOV DWORD PTR SS:[EBP-208],vulnerab.010940 ASCII "[13t]"
010810B3 . C785 58FEFFFF MOV DWORD PTR SS:[EBP-1A8],vulnerab.010940 ASCII "0c"
010810BD . C785 A0FDFFFF MOV DWORD PTR SS:[EBP-260],vulnerab.010940 ASCII "0b"
010810C7 . C785 50FEFFFF MOV DWORD PTR SS:[EBP-1B0],vulnerab.010940 ASCII "[7e]"
010810D1 . C785 F0FDFFFF MOV DWORD PTR SS:[EBP-210],vulnerab.010940 ASCII "0xhak"
010810DB . C785 48FEFFFF MOV DWORD PTR SS:[EBP-1B8],vulnerab.010940 ASCII "[8p]"
010810E5 . C785 C0FDFFFF MOV DWORD PTR SS:[EBP-240],vulnerab.010940 ASCII "[3c]"
010810EF . C785 40FEFFFF MOV DWORD PTR SS:[EBP-1C0],vulnerab.010940 ASCII "0xtrash"
010810F9 . C785 E8FDFFFF MOV DWORD PTR SS:[EBP-218],vulnerab.010940 ASCII "0xfedrelandet"
01081103 . C785 38FEFFFF MOV DWORD PTR SS:[EBP-1C8],vulnerab.010940 ASCII "[10a]"
0108110D . C785 A8FDFFFF MOV DWORD PTR SS:[EBP-258],vulnerab.010940 ASCII "0xploit"
01081117 . C785 30FEFFFF MOV DWORD PTR SS:[EBP-1D0],vulnerab.010940 ASCII "[11n]"
01081121 . C785 94FEFFFF MOV DWORD PTR SS:[EBP-16C],vulnerab.010940 ASCII "0xtrash"
0108112B . C785 28FEFFFF MOV DWORD PTR SS:[EBP-1D8],vulnerab.010940 ASCII "[9l]"
01081135 . C785 B8FDFFFF MOV DWORD PTR SS:[EBP-248],vulnerab.010940 ASCII "0xredteam"
0108113F . C785 20FEFFFF MOV DWORD PTR SS:[EBP-1E0],vulnerab.010940 ASCII "[6h]"
01081140 . C785 80FEFFFF MOV DWORD PTR SS:[EBP-200],vulnerab.010940 ASCII "c"
```

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:

logo.jpg x

This file has been changed on disk. You may choose to ignore the changes but reloading is the only safe option. Ignore Reload

00003b91	14	51	45	00	14	51	45	00	14	51	45	00	14	51	45	00	14	51	45
00003ba8	00	14	51	45	00	14	51	45	00	14	51	45	00	14	51	45	00	14	51
00003bbf	45	00	14	51	45	00	14	51	45	00	14	51	45	00	14	51	45	00	14
00003bd6	FF	D9	49	44	33	03	00	00	00	00	22	54	49	54	32	00	00	00	00
00003bed	43	68	6C	6C	65	6E	67	65	54	50	45	31	00	00	00	04	00	00	00
00003c04	FF	FB	90	C4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00003c1b	66	6F	00	00	00	0F	00	00	08	97	00	0E	07	D5	00	02	05	08	0A
00003c32	17	1A	1C	1E	21	24	27	28	2B	2E	31	33	36	38	3B	3D	40	43	44
00003c49	51	54	57	59	5C	5E	61	63	66	69	6C	6D	70	73	76	78	7A	7D	80
00003c60	8C	8F	92	94	96	99	9C	9E	A1	A3	A6	A8	AB	AE	B0	B2	B5	B8	BB
00003c77	C7	CA	CC	CE	D1	D4	D7	D9	DB	DE	E1	E3	E6	E8	EB	ED	F0	F3	F5
00003c8e	00	00	39	4C	41	4D	45	33	2E	39	39	72	01	CD	00	00	00	00	00

Search for: as

Signed 8 bit: Signed 32 bit:

Unsigned 8 bit: Unsigned 32 bit:

Signed 16 bit: Float 32 bit:

Unsigned 16 bit: Float 64 bit:

☐ Show little endian decoding ☐ Show unsigned as hexadecimal

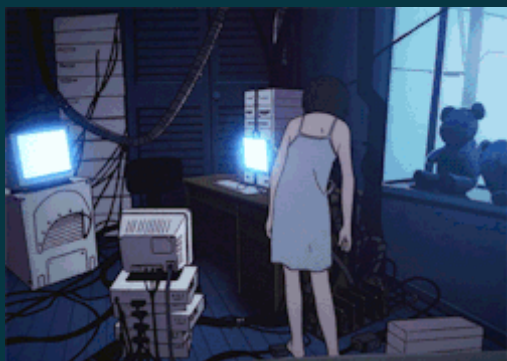
Offset: /

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

