


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

## i18 Challenge - Part 1

=====

🕒 5 minute read ✍ Published: 10 Jul, 2019

> I was given the link to this CTF that ran back in 2015, and I thought I would  
> take the opportunity to start writing about my thought process using these  
> challenges. If you want to try for yourself, the link is at the beginning of  
> the post.

The i18 has 13 challenges that you can find at [this address](#) . This CTF is in Norwegian, but has a very small amount of text so google translate is more than enough, but I will translate the text and clues to english here. This is the first part of my writeup where I will do the first 4 challenges. It is very possible that I will make mistakes or take longer path to the flag, so if you have any recommandation or correction, feel free to contact me by email or twitter. And same if you have questions or things you would like to know more about! Alright let's jump in!

### ## 1: Ren rutine

**Task:** First task and this is simple. The answer we are looking for is sha256(lett som en plett).

This one seems fairly easy: just generates the sha256 hash for the sentence 'lett som en plett'. Just type this command:

```
echo -n 'lett som en plett' | sha256sum
```

It will give: b3df0e709c60b6c3790941a49d989dcd1911b1e25eba18afdb08f78820a1b8cf

And it works! First flag!

## ## 2: This Must Be The Place

**Task:** We have put this on our head completely.

No more info than that. But it says the “head” so I would imagine it has something to do with the header, so let’s look at the page header! A quick way to do that is to right click, inspect element, go to to the network tab and click on the first GET request which is for this specific page. In the header, we do find a strange parameter called X-Answer with the value:

b8e63fba4504b3c29e582cc3d1f6cc56918998d50e55b1af3382080506f2f969

And indeed it works! We could also have just pinged the page to get the header back:

```
curl -I https://challenge.i18.no/level/fbmserjd3dwtvrspnk8ektmxsvdenwvp
```

## ## 3: I Skyggene

Task: nsm:\$1\$asH66rpo\$gkIQMSAA.QS/ah5VZvrYi0:16403:0:99999:7:::

Clue one: Passwords that follow predictable patterns make them easier to crack.

Clue two: An article explaining why Summerferie2014 is a bad password.

The clues are completely essential for me here because I have very little experience in crypto challenges. I can easily guess that it will follow the pattern of Summerferie2014 though. So one word and then one number (probably a year). As it is in Norwegian, we can't use already premade wordlists I have in English.

To find a list of words wasn't easy but I found an old scrabble website that had one in their rules page by googling ordboka: <https://www.ordspill.no/rules/> which redirects us in the end to this website with a downloadable list:

<http://www2.scrabbleforbundet.no/wp-content/uploads/2016/10/nsf2016.zip>

I wonder if they know their innocent scrabble list can be used to crack passwords. Anyway this probably took me almost an hour but we got it! Oh yes, the actual challenge now.

We save the hash in a text file and after running a normal command with john (john hash.txt -wordlist=nsf2016.txt), it doesn't manage to crack the password. So we probably need to help it by adding a rule as we know the format of the possible password (common word + number/year).

According to this article: <https://www.gracefulsecurity.com/custom-rules-for-john-the-ripper/> - we should be able to create that by adding to the john's Config file (etc/john/john.conf) this rule:

```
[List.Rules:wordDate]  
cAz"201[0-9]"
```

Basically it will try every word on the list with a first capital letter, then 201 and add 0 to 9 at the end, giving for example Stanley2012. The "c" at the beginning tells john to remove the rule of the initial capital letter if it doesn't find anything. So after the right command:

```
john hash.txt --wordlist=nsf2016.txt --rules=wordDate
```

It tells me that it cracked it with the result being "Demokrati2015". For some reason, the CTF did not accept this answer and I couldn't understand why. But then I remembered that the previous answers were hashed in sha256 so I did the same with Demokrati2015 and it worked.


```
echo -n Demokrati2015 | sha256sum  
a4e3e09a00b0706add935ebdf627dc57bfb6e9d53b0be2c60754ad5865ce3c4f
```

## ## 4: Broen

A file. What is hiding in this.  
Clue: `sha256(lc(nameofaplace))`

Ok so we get a file to download. Awesome! I love to do forensic challenges. The file is called data.bin. Let's check what it could be with a simple command:

```
file data.bin "data.bin: pcap capture file, microsecond ts (little-endian) - version 2.4 (Ethernet, capture length 65535)"
```

Great! A pcap file. I'm deep into a network security ebook so I recognise the extension directly. Basically for those that don't know, pcap means [Packet Capture Data](#) . The file was generated through packet sniffing by the free software Wireshark. So let's open the software and have a look at this capture. It's a short packet capture and we see directly that there was a successful HTTP request and another request after that for an image:


No.	Time	Source	Destination	Protocol	Length	Info
7	0.388308	192.168.195.128	192.168.195.129	HTTP	400	GET / HTTP/1.1
8	0.388416	192.168.195.129	192.168.195.128	TCP	54	1337 → 49166 [ACK] Seq=1 Ack=347 Win=30336 Len=0
9	0.390333	192.168.195.129	192.168.195.128	TCP	71	1337 → 49166 [PSH, ACK] Seq=1 Ack=347 Win=30336 Len=0
10	0.390511	192.168.195.129	192.168.195.128	HTTP	407	HTTP/1.0 200 OK (text/html)
11	0.390735	192.168.195.128	192.168.195.129	TCP	60	49166 → 1337 [ACK] Seq=347 Ack=372 Win=65280 Len=0
12	0.391556	192.168.195.128	192.168.195.129	TCP	60	49166 → 1337 [FIN, ACK] Seq=347 Ack=372 Win=65280 Len=0
13	0.391578	192.168.195.129	192.168.195.128	TCP	54	1337 → 49166 [ACK] Seq=372 Ack=348 Win=30336 Len=0
14	0.493801	Vmware_2d:63:d6	Broadcast	ARP	60	Who has 192.168.195.1? Tell 192.168.195.128
15	0.811132	192.168.195.128	192.168.195.129	HTTP	340	GET /favicon.ico HTTP/1.1
16	0.811196	192.168.195.129	192.168.195.128	TCP	54	1337 → 49167 [ACK] Seq=1 Ack=287 Win=30336 Len=0
17	0.811577	192.168.195.129	192.168.195.128	TCP	83	1337 → 49167 [PSH, ACK] Seq=1 Ack=287 Win=30336 Len=0
18	0.811697	192.168.195.129	192.168.195.128	HTTP	369	HTTP/1.0 404 File not found (text/html)
19	0.812074	192.168.195.128	192.168.195.129	TCP	60	49167 → 1337 [ACK] Seq=287 Ack=346 Win=65280 Len=0
20	0.812814	192.168.195.128	192.168.195.129	TCP	60	49167 → 1337 [FIN, ACK] Seq=287 Ack=346 Win=65280 Len=0
21	0.812837	192.168.195.129	192.168.195.128	TCP	54	1337 → 49167 [ACK] Seq=346 Ack=288 Win=30336 Len=0
22	1.500472	Vmware_2d:63:d6	Broadcast	ARP	60	Who has 192.168.195.1? Tell 192.168.195.128
23	1.628616	192.168.195.128	192.168.195.129	TCP	66	49168 → 1337 [SYN] Seq=0 Win=8192 Len=0 MSS=1460
24	1.628661	192.168.195.129	192.168.195.128	TCP	66	1337 → 49168 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0
25	1.628916	192.168.195.128	192.168.195.129	TCP	60	49168 → 1337 [ACK] Seq=1 Ack=1 Win=65536 Len=0
26	1.629238	192.168.195.128	192.168.195.129	HTTP	448	GET /image.png HTTP/1.1
27	1.629265	192.168.195.129	192.168.195.128	TCP	54	1337 → 49168 [ACK] Seq=1 Ack=395 Win=30336 Len=0
28	1.629563	192.168.195.129	192.168.195.128	TCP	71	1337 → 49168 [PSH, ACK] Seq=1 Ack=395 Win=30336 Len=0


If we continue down through this image request, we see that it was also successful:

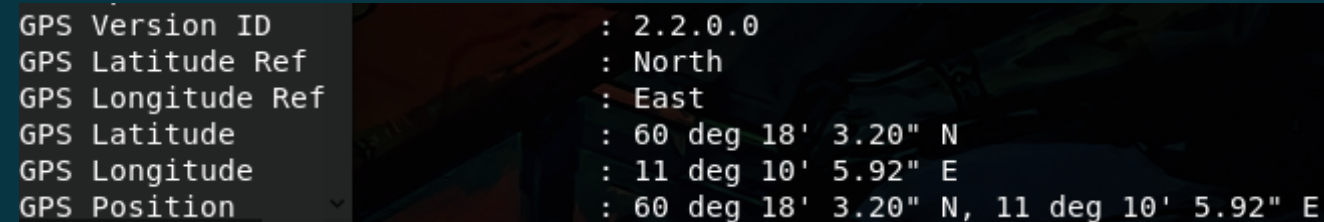
64	1.636727	192.168.195.128	192.168.195.129	TCP	60 49168 → 1337 [ACK] Seq=3
← 65	1.636736	192.168.195.129	192.168.195.128	HTTP	4652 HTTP/1.0 200 OK (PNG)
66	1.637223	192.168.195.128	192.168.195.129	TCP	60 49168 → 1337 [ACK] Seq=3

Which means there is an image in this packet capture. To get the files transferred through HTTP, go to file -> Export Objects -> HTTP. Let's have a look at this picture:





Interesting... It looks like a google map picture of a bridge (Broen, the title of the challenge means bridge in Norwegian). As the clue is about finding the name of a place, I can guess already that if we find where this bridge is, we should be good to go. Let's start by checking [the website](#)  that is written as Copyright. But we don't find anything there, just an interactive map and I'm not going to visit all the places in Norway looking for this bridge. Let's analyze the image then.

I did not find anything interesting by checking the usual commands like identify, file, exiv2. So after some googling I found [an article by Nullbyte](#)  about extracting hidden information in images through exiftool. I quickly installed it and after running it, we get an interesting information:



```
GPS Version ID      : 2.2.0.0
GPS Latitude Ref    : North
GPS Longitude Ref   : East
GPS Latitude        : 60 deg 18' 3.20" N
GPS Longitude       : 11 deg 10' 5.92" E
GPS Position        : 60 deg 18' 3.20" N, 11 deg 10' 5.92" E
```

A GPS position! Alright now we're getting somewhere. I go straight to [a GPS coordinate website](#) , enter the coordinates and BOOM! We know that the bridge is in [Eidsvoll](#) . Fun fact: the Norwegian constitution was drafted and signed in Eidsvoll on the 17th of May 1814 by the constitutional assembly. And this is now the Norwegian national day.

**Address**

Carsten Ankers veg, 2074 Eidsvoll verk, ↑

Get GPS Coordinates

---

**DD (decimal degrees)\***

Latitude 60.300889

Longitude 11.168306

Get Address

Lat,Long 60.300889,11.168306


---

**DMS (degrees, minutes, seconds)\***

Latitude ☒ N ☐ S 60 ° 18 ' 3.2 "

Longitude ☒ E ☐ W 11 ° 10 ' 5.901 "

Get Address



Let's finish the challenge now:

```
echo -n eidsvoll | sha256sum  
e6b46a50991986097e4a309076cc0760be4c7a643f62e42b07adaf14a7fe30d4
```

Success! See you in part 2!



-----  
Published by theyknow 10 Jul, 2019 in [post](#) and tagged [crypto](#), [ctf](#), [forensic](#), [i18](#),  
[john](#), [web](#) and [wireshark](#) using 1016 words.