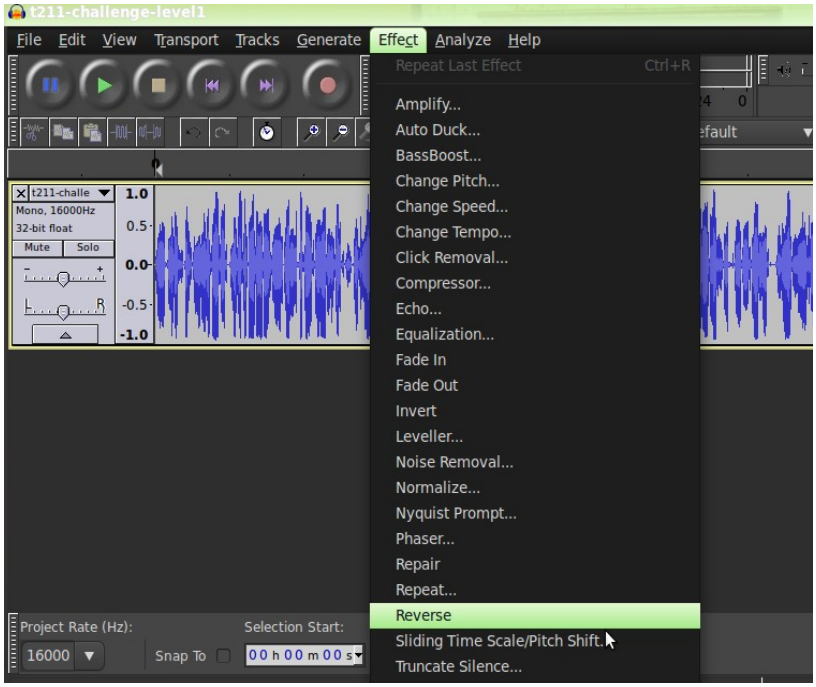
A war-story of

Solving the T2 '11 challenge

as told by Timo Teräs

Level 1 – The audiogram

The competition started off with a mp3 file. After the first listening, it was obvious that the file was reversed English speech. This called to fire up `audacity`[1] the sound editor of preference.



After reversing the audio clip, the voice needed just a minor reset of sampling rate (to 32kHz) to get some understandable English.

The military alphabet used to spell letters is fairly well known, but the list is also available in Wikipedia[2]. The only trouble was recognizing the part 'b8a5' as the '*alfa*' was slightly obscure.

So the URL was:
http://t2.fi/ext/challenge?level=09d642f60cbed162b8a589bde2d18674

But as my habits require, I was forced to also take a look at the file using `file`[3] and `hd`[4], my default utilities to recognize file and to check the binary contents.

As ID3 tag was present, I decided to take a look at it also with `id3v2`[5]:

```
$ id3v2 -l t211-challenge-lreevel1.mp3
id3v2 tag info for t211-challenge-level1.mp3:
TIT2 (Title/songname/content description): Revolution 9
TALB (Album/Movie/Show title): The Beatles
TRCK (Track number/Position in set): 5
TCON (Content type): Rock & Roll (78)
TPE2 (Band/orchestra/accompaniment): Beatles
TDRC ():  frame
```

So... we seem to have a Beatles theme this year ;)

1 apt-get install audacity
2 http://en.wikipedia.org/wiki/NATO_phonetic_alphabet
3 apt-get install file
4 apt-get install bsdmainutils
5 apt-get install id3v2

Level 2 – The pictogram

The file given for level 2 was a .tif image. And yes, `file` says it was indeed a TIFF image. So the first thing is to see it an image editor – `gimp`[6] in my case. However, something was definitely fishy as gimp complained with various errors like:

```
Line length mismatch at line 0 of strip
0 (got 529, expected 23)
Line length mismatch at line 2 of strip
0 (got 24, expected 23)
```

However, the image opened up, but looked just a random collection of black dots. My first guess is that there's something funky going on with the image width (TIFF RLE encoding is not allowed to overlap line boundaries).

Next I decided to take a look with `hd` for the hex dump. And since it look liked meta-data being present, I next chose to dump that using `exiv2`[7] (output modified for brevity):

```
$ exiv2 pr -Pnct 2vxv93-t211-challenge-
level2.tif
ImageWidth          1  23
ImageLength         1  69
BitsPerSample       1  1
ImageDescription 38 A Hard Day's Night
(1964) Album Cover
Make                7  Google
Model               6  R7cdl
StripOffsets        1  530
RowsPerStrip        1  122
StripByteCounts     1  156
Software           10  IrfanView
Artist              8  Beatles
HostComputer        6  llTU9
UserComment        18  Reggae Set
XPTitle            76  A Hard Day's Night
(1964) Album Cover
```

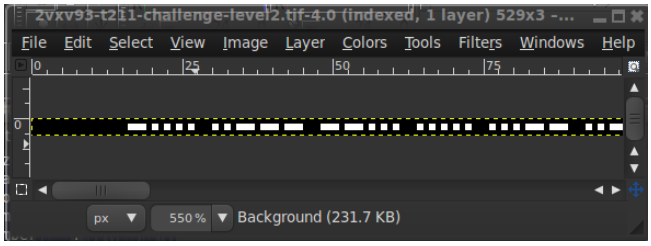
Now the *Make, Model* and *HostComputer* fields look unusual in addition to the *ImageDescription* hinting again at the Beatles. Now, googling for R7cdl and llTU9 did not result in much of interest. I quickly remembered the Google URL shortening service: http://goo.gl/. And yes, those are valid goo.gl URLs.

First, http://goo.gl/R7cdl lead to a hint about morse code[8]. The other URL http://goo.gl/llTU9 lead to factorization of 23 by 69 (the height and width of the image)[9]. This indeed hinted that the image dimensions were actually something else.

The obvious step is to patch the binary file using `hexedit`[10]. But where in the file are the dimensions? After a quick man search, I found `dcparse`[11] which gave the information:

```
tag=0x100 256, type=3, count=1,
offset=000012, data= 0017
tag=0x101 257, type=3, count=1,
offset=00001e, data= 0045
```

The bytes are at offsets 0x12 and 0x1e. There are were only few combinations to test (ways how the image size can be factorized) and trial-and-error revealed that the height 3 and width 529 looked interesting... morse code.



I decided to google for a morse decoder, and found one to decode dots and dashes to plain text[12]. The image was fairly quickly converted by hand to be:

```
-.... ..---  --... ..... ...-- ..---
----- ----. -.-. ....- -----
-.. .---- ..-. ----- .---- ..... ---..
--... ---.. ----- ..... .---- -... .....
.....  -... ---.. .- ..-. -.. ..-.
```

Which was translated by the web service for me as:

```
62753209c40d1f015878051b55b8afdf
```

The pass code for next level!

---

6 apt-get install gimp
7 apt-get install exiv2

8 http://www.wolframalpha.com/input/?i=morse+easter+egg
9 http://www.wolframalpha.com/input/?i=factorize+23*69
10 apt-get install hexedit
11 apt-get install dcraw
12 http://webnet77.com/cgi-bin/helpers/morse.pl

Level 3 – The cryptogram

And now received a network capture. This calls for a `wireshark`[13] analysis. It was obvious that there was only a single TCP-session, that looked simple dialogue. I just saved this as C-format arrays for further analysis.

And cool, we have a Beatles hint again.

After the dialogue goes to 'mode 2' it seems that the replies are quite close to the expected plain text, but getting added an increasing offset. So I drew a small table with text editor on the differences:

O	O	+0
K	L	+1
P	Q	+1
A	C	+2
U	X	+3
L	Q	+5
R	Z	+8
I	V	+13
N	c	+21
G	i	+34
O	0x86	+55

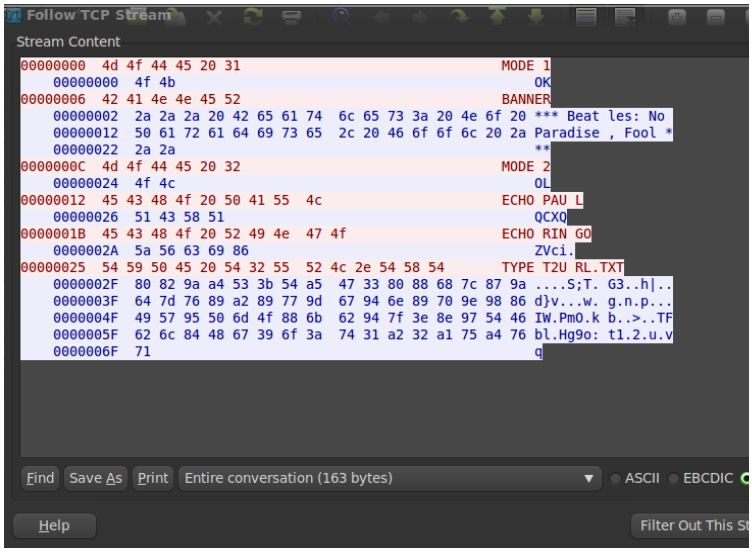
Now this is a sequence all self-respecting hackers will recognize right away: the Fibonacci sequence.

So the obvious first try is to generate the Fibonacci sequence from 55 onwards, and try to decrease those values from the final message. However, that turned out to be garbage.

It was also notable that the reply to the 'type' command was exactly the length of t2.fi URLs with the hash, so expected plain text would likely start with http://t2.fi/ext/...

So assuming that, and that the generator for the cipher stream is Fibonacci style (sum of two previous elements in sequence), we could guess the starting sequence elements. The first item would be 0x80-'h'=24 and the second one would be 0x82-'t'=14.

And this revealed only the beginning of the



URL: 'http' and after that we got unexpected characters; however, it was become soon evident that the further characters were coded in similar manner. That is, starting decoding from the fourth character, and using known plain text to guess beginning offsets revealed another group of expected characters.

After quite a bit of trial-and-errors it became clear that the offset is modified whenever it goes above 'A'. So the C-code for deciphering was ready:

```
for (i = 0; i < len; i++) {
  if (x1 >= 'A')
   x1 -= 'A';
  printf("%c", msg[i]-x1);
  x3 = x2;
  x2 = x1;
  x1 = x2 + x3;
}
printf("\n");
```

And now to just to decrypt the message using this, we get:

http://t2.fi/ext/challenge?level=1c9e0af6cc2fc9895e3313750b2a5e85

13 apt-get install wireshark

Level 4 – The logic'gram

The final level was a html page with java applet that presented a monstrous looking beast to be solved. It looks like a Sudoku puzzle... but it's just bigger!

Looking at the picture more and more it indeed looked like a 16x16 sudoku with some given digits within 0-9 and a-f. And yes, google says that there's a variant of hex sudoku that is exactly this.

So I just searched for the first hex sudoku solver and found one by Thomas Grønneløv[14]. I came with a quick and dirty Linux shell magic to convert the html page to a preset situation this program would accept:

```
$ cat t211-challenge-level4.html | grep
name= | sed -e 's/<[^>]*>//g' -e 's/[
\t]*//g' -e 's/0/16/g' -e 's/^$/0/g' -e
's/a/10/g' -e 's/b/11/g' -e 's/c/12/g'
-e 's/d/13/g' -e 's/e/14/g' -e
's/f/15/g' | tr "\n" ,
```

The output was almost perfect, just removing the first zero and fixing the end, did the trick.

Now to just have the solver crunch the problem... and wait... it was Sauna time :)

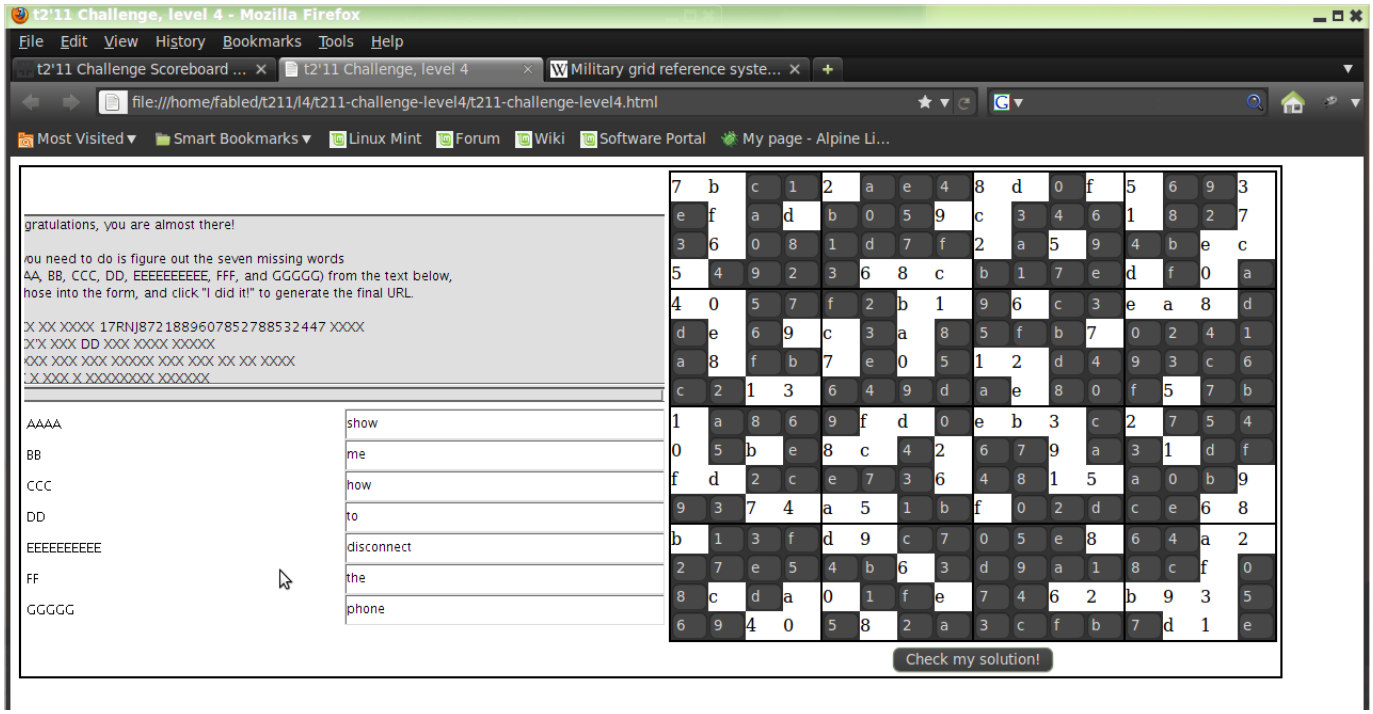
After some rounds of Sauna, the program did report a correct solution for the 16x16 Sudoku.

Now I was not feeling like coding after the Sauna, so I just typed the correct solution to the puzzle..... and this resulted in the final piece of the competition: to recognize missing words.

The immediate guess was that it's a Beatles (so many references to these guys!) lyric (the words patterns had repetitions as-in a poem or lyrics). But which one – they have hundreds of songs. And alas, at this point someone had solved the level 4... so it was back to Sauna.

Later... the pattern X.X.X.X. gave out the song in question quite fast. The words of "Back in the U.S.S.R." fit the given template perfectly. And it was easy to recognize the missing words. After filling in the missing words, the Java applet gave back the final URL of:

http://t2.fi/ext/challenge?level=e450f76afe9b3844228bdf4846d6f301



14 http://www.greenleaf.dk/projects/sudoku

At the end of the rainbow

If one is to solve a puzzle, one should do it well.

So what were the mysterious multi-character strings in the song lyrics? As they substituted geographic locations, the first thought is that they are coordinates intended as a hint. And yes, indeed they are Military Grid Reference System coordinates[15]. Out of curiosity I used an online service[16] to translate these to more known GPS coordinates, and then Google Maps to see where they point:

```
17RNJ8721889607852788532447
25°47'26.4"N 080°07'48.2"W
N.OceanRoad/Lincoln Road, Miami Beach

36UUU6416964766060099795269
48°22'46.0"N 031°09'56.1"E
Kirovohrads'ka oblast, Ukraine

37UDB1324500360079764979033
55°45'20.8"N 037°37'03.5"E
Proyezd Voskresenskiye Vorota
1/5, gorod Moskva, Russia

17SLR2015601780459479826465
32°09'26.8"N 082°54'25.6"W
New Bethel Church Rd, Helena,
GA 31037, Georgia
```

Nothing too exciting here.

However, as a final check, I wanted to figure out if the pass-codes (MD5 signatures) for each level would have some rational meaning or not. Thus I ran them using an online MD5-rainbow dictionary[17]. It did recognize the last two of the MD5s to be “Help!” and “Revolver”. Where have I seen these before?

Yes, these are the Beatles album names which I saw while hunting the song lyrics for the last level. So quickly to get a list of all the Beatles album names, and check which two would match the other levels.

The matches were “Please Please Me” and “A Hard Day's Night”.

So the level codes were:

```
Level 1
09d642f60cbed162b8a589bde2d18674
Please Please Me

Level 2
62753209c40d1f015878051b55b8afdf
A Hard Day's Night

Level 3
1c9e0af6cc2fc9895e3313750b2a5e85
Help!

Level 4
e450f76afe9b3844228bdf4846d6f301
Revolver
```

15 http://en.wikipedia.org/wiki/Military_grid_reference_system

16 http://geographiclib.sourceforge.net/cgi-bin/GeoConvert

17 http://www.md5rainbow.com