2016

MICT1 – Exercise Week 6

MICT1 – Group 1

Zuyd University

22-Mar-16

MICT1 – Exercise Week 6

Reverse Engineering Data – Exercise week 6

Group	Group 1	
Students	Delano Cörvers	(1306669)
	Davy Heutmekers	(1309730)
	Rik Kierkels	(1354442)
Module	MICT1 – Reverse Engineering Data	
Assignment	Exercise – Week 6	
School year	2015 – 2016	

Table of content

1 What we found

We found the following hidden message embedded in a PNG image: “It’s true. All of it.”

Figure 1: Recovered PNG Image file.

2 Where we found it

The PNG image we found was hidden in a deeper file structure. The image was compressed in a .RAR file, which was compressed into a .ZIP file itself using a lossless compression method. At first we thought the PNG file would contain a text data field to hold the message with some sort of encoding. However, we couldn't discover any text field in the file. After reconstructing the image we saw that the text was embedded in the image (added afterwards).

3 How we found it

In this chapter we will describe our process of solving this week's exercise.

3.1 Inspecting the original file

While inspecting the provided file for this week's exercise we found traces of it being a ZIP file. This was very notable by looking at the trailer of the file. We weren't sure if it actually was a zip file, because the trailer could also have been from an older file (slack). Since our trusted tool ZipFix was experiencing downtime, we had to reconstruct the ZIP header ourselves to know for sure. We were able to manually reconstruct the header of the alleged PKZIP file, because the header contains mostly the same information as the trailer. The header was actually not completely damaged. We had to modify the first two bytes and the cyclic redundancy check (CRC). By using the PKZIP specification on Wikipedia and cross-referencing it with the trailer, we were able to retrieve a working zip file. The CRC was still in the trailer, so we were able to find this value and replace it in the header. The working header of the file became the following:

```
Address  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000000 50 4b 03 04 0a 03 00 00 00 00 dc bb 74 48 22 45 PK........Ü»tH"E
00000010 e0 e2 f8 be 0d 00 f8 be 0d 00 02 00 00 00 78 32 àâø¾..ø¾......x2
00000020 ad 61 72 21 1a 07 00 cf 90 73 00 00 0d 00 00 00 -ar!...Ï.s......
```

Figure 2: Part of the working header for PKZIP.

3.2 The recovered zip file

The recovered zip file was using a lossless compression method. This means that the contents of the zip were a one-on-one copy of the original data when decompressed. In figure 2 we can see that the filename of the contents in PKZIP is likely to be x2.-ar. From a little bit of research we could quickly determine that the extension –ar was actually the .RAR extension. The filename is x2. The decompressed ZIP file appears to only contain that .RAR file. In order to actually decompress its contents, we had to change the header of the file to rar instead of –ar. We did this by changing the first byte of the header (file) to the hexadecimal value 52 which is the character 'r'. The following image contains the header of the rar file:

```
Address  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000000 52 61 72 21 1a 07 00 cf 90 73 00 00 0d 00 00 00 Rar!...Ï.s......
00000010 00 00 00 00 f5 80 74 80 90 23 00 ba be 0d 00 b8 ....õ€t€.#.º¾..¸
00000020 b4 0d 00 03 5d 11 cd 92 b4 ba 74 48 1d 33 01 00 ´...].Í'´ºtH.3..
```

Figure 3: Part of the header of the RAR file.

3.3 The recovered RAR file

In the recovered RAR file we found a file without a header. This file is named x. By looking at the structure of the data we found keywords such as IHDR, PTLE, IDAT. The trickery in this file was that all bytes were encoded using Middle-Endian (ME). This means that the critical chunks weren't named IHDR and PTLE, but HIRD and TPEL.

```
Address  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000000 af 76 b8 b1 0a 0d 0a 1a 00 00 0d 00 48 49 52 44 ¯v¸±........HIRD
00000010 00 00 0f 04 00 00 73 02 06 08 00 00 60 00 3a 70 ......s.....`.:p
00000020 00 5b 00 00 62 06 47 4b 00 44 00 ff 00 ff a0 ff .[..b.GK.D.ÿ.ÿ.ÿ
00000030 a7 bd 00 93 00 00 70 09 59 48 00 73 0b 00 00 13 §½.“..p.YH.s....
```

Figure 4: File x.

By using a Unix based terminal we used a few standardized commands to swap the file from middle endian to a big endian. To do this we used the command: `dd conv=swab <original_file >new_file`.

Figure 5: Folder with the original and modified file.

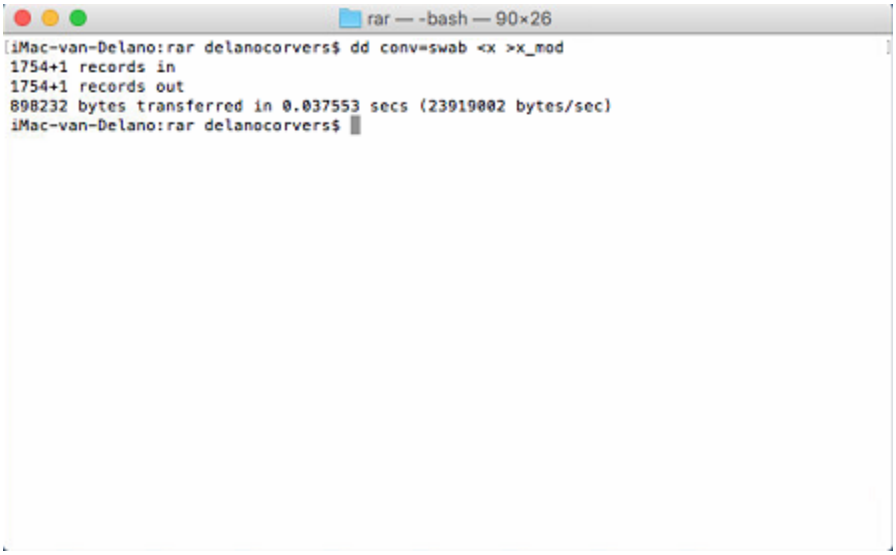

Figure 6: Terminal commands.

After doing this procedure we have the original file converted to big endian. In order to actually view the PNG we need to repair the broken header of the file. By investigating the header we were able to figure out that the bits themselves were also swapped. Hex Editor Neo has built-in functionality to flip bits. This resulted in the following change:

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	76	af	b1	b8	0d	0a	1a	0a	00	00	00	0d	49	48	44	52	v¯±¸....IHDR
00000010	00	00	04	0f	00	00	02	73	08	06	00	00	00	60	70	3a	.....s.....`p:
00000020	5b	00	00	00	06	62	4b	47	44	00	ff	00	ff	00	ff	a0	[....bKGD.ÿ.ÿ.ÿ.
00000030	bd	a7	93	00	00	00	09	70	48	59	73	00	00	0b	13	00	½§"....pHYs.....

Figure 7: Original PNG file with broken header.

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	89	50	4e	47	0d	0a	1a	0a	00	00	00	0d	49	48	44	52	‰PNG....IHDR
00000010	00	00	04	0f	00	00	02	73	08	06	00	00	00	60	70	3a	.....s.....`p:
00000020	5b	00	00	00	06	62	4b	47	44	00	ff	00	ff	00	ff	a0	[....bKGD.ÿ.ÿ.ÿ.
00000030	bd	a7	93	00	00	00	09	70	48	59	73	00	00	0b	13	00	½§"....pHYs.....

Figure 8: Modified PNG file with fixed header.