



Digital Forensics Report

Daniel Nunes – 81016

Diogo Freitas - 81586

1 Objectives of the investigation

For several years now, John Mole has been working for DroneX, a major manufacturer of drone technology. In the past few months, motivated by suspicious changes in John's behavior, DroneX started taking measures to investigate him. Since he had privileged access to the design plans of their new revolutionary drones, the fear was that he might be illicitly stealing those plans in order to sell them to competitors. To dissipate such fears, after obtaining legal counseling and authorization, DroneX assembled an auditing team to look for potential evidence of industrial espionage.

2 Artifacts for analysis

The following image is a table with the files and respective md5 found on John Mole's pen-drive before the investigation started:

File	Value
munich.txt	c6596b360ac97889c4f2d68ba6787f92
compress.py	72eab63334dcd0f73418e32999b71f05
cathedral.png	55fd5b1d42072955e15769b55a390400
oktoberfest.png	deb345aea6cdb82ca4636c0811c292df
street.png	f1ea1beaa6a838d16b4d457c6fe68fd0
wursten.png	13c85b20b6b1e481a32700f26818333e
snow.bmp	a6e56c4d34d9a541b622b74c954c3fc9
online_banking.zip	b3baa737b818db4f52a681f0cf8d440c

- munich.txt – UTF-8 Unicode text file corresponding to a Wikipedia page about the history of the city of Munich;
- compress.py – appears to be a compiled python bytecode file;
- cathedral.png – PNG image file of the Cathedral of *Frauenkirche* in Munich;
- oktoberfest.png – PNG image file of a group of young people supposedly at the festival *Oktoberfest*, a beer festival in Munich;
- street.png – PNG image file of a street where you can see the *Hofbrauhaus*, a known brewery in Munich;
- wursten.png – PNG image file of a table with sausages and beer;
- snow.bmp – BMP image file of a snowy landscape;
- online_banking.zip – ZIP file containing two other files encrypted with a password.

3 Evidence to look for

Our first approach was to try and find out if any of the image files had hidden information when converted to hexadecimal format.

Following that, we decided to compare the txt file (munich.txt) with the original Wikipedia page that offers information about Munich and tried to find any differences.

The next step was to investigate the zip file (online_banking.zip) which contained a password encrypting its contents. Inside we could see two files, a .docx file and a .bmp image file with the name drone-A. We figure this drone-A.bmp might be an image containing drone plans of the company DroneX.

Lastly, we examined the python file, finding out through an appropriate tool that it was a steganography program. We hypothesize that it is likely that some information was hidden within the images we found in John Mole's pen-drive using this steganography program. We hope to find hidden messages and incriminating evidence like bank transactions and stolen information that might link the suspect, John Mole, to any rivaling companies in the .docx file as its name is online_banking.

For that matter we will need to discover the password to the zip file mentioned above to access its information and also reverse engineer the program compress.py to find out what might be hidden in the image files we obtained.

4 Examination details

4.1 Hexadecimal Analysis of the Image Files

At 4PM on the 20th of October we started by getting proper clearance and authorization needed to make our investigation legal.

Following that, we started analysis of the image files. We used a software called HxD to analyze all the image files, looking for any content that might have been introduced into the photos. After converting the image file to hexadecimal format, we searched for any piece of text that might look relevant to our investigation. When we analyzed the file snow.bmp, at the end of the file, we found the following message *"Ich sende Ihnen f nf Dateien: (1) Drohne A Pl ne, (2) Drohne B Pl ne, (3) technische Spezifikationen, (4) Passw rter von DroneX Dateiservern"* which translates to in English "I'm sending you five files: (1) drone A plans, (2) drone B plans, (3) technical specifications, (4) passwords from DroneX file servers"

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
000E260	69	43	8E	69	43	8E	69	43	8E	69	43	8E	69	43	8E	69	ic�ic�ic�ic�ic�i
000E270	43	8E	69	43	8E	69	43	8E	69	43	8E	69	43	8D	68	42	c�ic�ic�ic�ic�.hb
000E280	8D	68	42	8D	68	42	8D	68	42	8D	68	42	8D	68	42	8D	.hb.hb.hb.hb.hb.
000E290	68	42	8D	68	42	8D	68	42	8D	68	42	8D	68	42	8E	69	hb.hb.hb.hb.hb.hb
000E2A0	43	8E	69	43	8E	69	43	8E	69	43	8E	69	43	8E	69	43	c�ic�ic�ic�ic�ic�
000E2B0	8E	69	43	8E	69	43	8D	67	44	8D	67	44	8D	67	44	8D	�ic�ic�.gd.gd.gd.
000E2C0	67	44	8D	67	44	8D	67	44	8D	67	44	8D	67	44	8D	67	gd.gd.gd.gd.gd.g
000E2D0	44	8D	67	44	8E	68	45	8E	68	45	8E	68	45	8D	67	44	D.gD����������
000E2E0	9C	66	43	8C	66	43	8E	66	43	8E	66	43	8E	66	43	8E	��������������
000E2F0	66	43	8E	66	43	8E	66	43	8E	66	43	8E	66	43	8E	66	��������������
000E300	43	8E	66	43	8E	66	43	8F	67	44	8F	67	44	80	68	45	c�ic�ic�.gd.gd.hE
000E310	90	68	45	91	69	46	90	69	43	90	69	43	90	69	43	90	.hE'�f.�c.�c.�c.
000E320	69	43	90	69	43	90	69	43	90	69	43	90	69	43	8F	68	�c.�c.�c.�c.�c.h
000E330	42	8F	68	42	8F	68	42	8F	68	42	8F	68	42	8F	68	42	B.hb.hb.hb.hb.hb
000E340	8F	68	42	8F	68	42	8F	68	41	8F	68	41	8F	68	41	8F	.hb.hb.ha.ha.ha.
000E350	68	41	8F	68	41	8E	67	40	8E	67	40	8E	67	40	8D	66	h.ha.ha.g��������
000E360	3F	8D	66	3F	8D	66	3F	8D	66	3F	8D	66	3F	8D	66	3F	?������������
000E370	8D	66	3F	8D	66	3F	8C	65	3F	8C	65	3F	8D	66	40	8D	.������������
000E380	66	40	8D	66	40	8D	66	40	8C	65	3F	8C	65	3F	8D	66	��������������
000E390	40	8D	66	40	8D	66	40	8D	66	40	8D	66	40	8D	66	40	��������������
000E3A0	8D	66	40	8D	66	40	8D	66	40	8D	66	40	8D	66	40	8D	.������������
000E3B0	66	40	8D	66	40	8D	66	40	8D	66	40	8D	66	40	8C	65	��������������
000E3C0	3F	8C	65	3F	8C	65	3F	8D	66	40	8D	66	40	8D	66	40	?������������
000E3D0	8D	66	40	8D	66	40	90	67	40	90	67	40	90	67	40	90	.������������
000E3E0	67	40	90	67	40	90	67	40	90	67	40	90	67	40	91	68	��������������
000E3F0	41	91	68	41	90	67	40	90	67	40	90	67	40	8F	66	3F	A'ha.g����������
000E400	8F	66	3F	8F	66	3F	8F	65	40	8F	65	40	90	66	41	90	.������������
000E410	66	41	90	66	41	90	66	41	90	66	41	90	66	41	91	67	��������������
000E420	42	91	67	42	91	67	42	91	67	42	91	67	42	91	67	42	B'gB'gB'gB'gB'gB
000E430	92	68	43	92	68	43	89	63	68	20	73	65	6E	64	65	20	'hc'�h����������
000E440	89	68	6E	65	6E	20	66	C3	BC	6E	66	20	44	61	74	65	��������������
000E450	89	65	6E	3A	20	28	31	29	20	44	72	6F	68	6E	65	20	��������������
000E460	41	20	90	6C	C3	A4	6E	65	2C	20	28	32	29	20	44	72	A Pl�ne, (2) D�
000E470	8F	68	6E	65	20	42	20	50	6C	C3	A4	6E	65	2C	20	28	�hne B Pl�ne, (3
000E480	33	29	20	74	65	63	68	6E	69	73	63	68	65	20	53	70) technische Sp
000E490	65	7A	69	66	69	6B	61	74	69	6F	6E	65	6E	2C	20	28	ezifikationen, (4
000E4A0	34	29	20	50	61	73	73	77	C3	B6	72	74	65	72	20	76	4) Passw�rter v
000E4B0	8F	6E	20	44	72	6F	6E	65	58	20	44	61	74	65	69	73	on DroneX Dateis
000E4C0	65	72	76	65	72	6E	64										ervern.

With this hidden message we discovered exactly what we should be looking for inside the pen-drive.

4.2 Analysis of the file munich.txt

After this discovery, at 4:30PM we analyzed the file munich.txt. Our first idea was to find any difference between this file and the text in the original Wikipedia page about Munich. After copying the mentioned Wikipedia text to a txt file, we ran the diff command to find any discrepancies between the two files. Nothing was found proving the texts were identical.

```
root@kali:~/Desktop/csf-lab1-artifacts/diff# diff original.txt copia.txt
root@kali:~/Desktop/csf-lab1-artifacts/diff#
```

4.3 Analysis of the file online_banking.zip

At 5PM we analyzed the zip file, online_banking.zip, and, through the tool fcrackzip, we tried a brute force method to find the password encrypting the files in the zip. We used the file munich.txt as a dictionary for the brute force method, creating a new file munich_new.txt we used the cat command to sort alphabetically and separate each word of the file in a new line.

```
root@kali:~/Downloads/csf-lab1-artifacts# cat munich.txt | tr -s '\n' '\n' | sort -u > munich_new.txt
root@kali:~/Downloads/csf-lab1-artifacts#
```

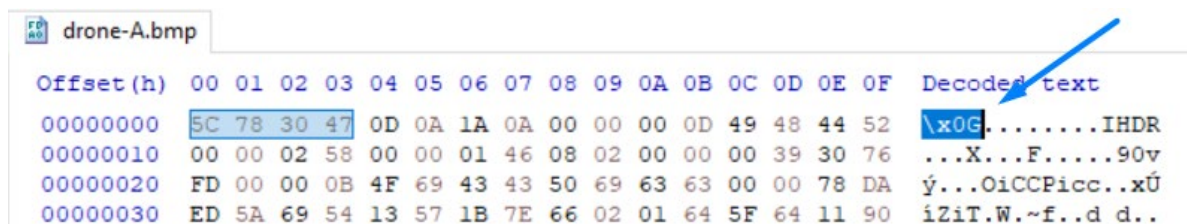
Executing the fcrackzip tool we found the password: "Stadelheim", the name of a prison in Munich.

```
root@kali:~/Downloads/csf-lab1-artifacts# fcrackzip -b -D -u -v -p munich_new.txt online_banking.zip
found file 'online_banking.docx', (size cp/uc 12936/ 22313, flags 9, chk 59ae)
found file 'drone-A.bmp', (size cp/uc 87952/ 92401, flags 9, chk 71a4)

PASSWORD FOUND!!!!: pw == Stadelheim
root@kali:~/Downloads/csf-lab1-artifacts#
```

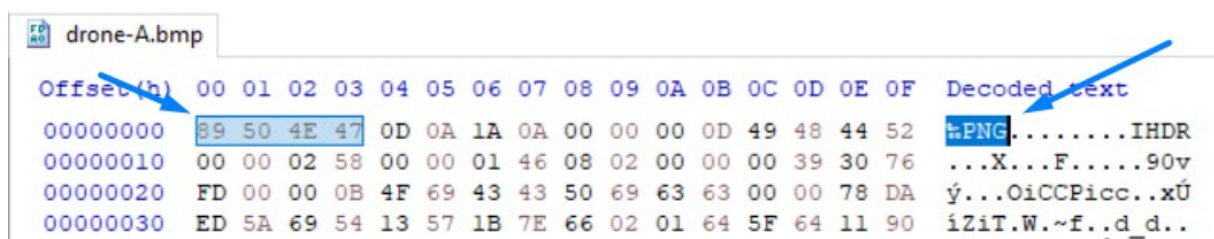
4.4 Analysis of the files contained in the zip file

After discovering the password, we used it to gain access to the files inside online_banking.zip at 5:30PM and we found the files online_banking.docx and drone-A.bmp. This last file had his signature corrupted. Once again, we used the software HxD to analyze the file and we noticed the first bytes didn't correspond to a .bmp file. Looking carefully at the hexadecimal format of the image we found the fields IHDR and IDAT which we can typically find in a .png file.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	5C	78	30	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	\x0G.....IHDR
00000010	00	00	02	58	00	00	01	46	08	02	00	00	00	39	30	76	...X...F.....90v
00000020	FD	00	00	0B	4F	69	43	43	50	69	63	63	00	00	78	DA	ý...OiCCPicc..xÜ
00000030	ED	5A	69	54	13	57	1B	7E	66	02	01	64	5F	64	11	90	iZiT.W.~f..d_d..

We then changed the signature of the file to a .png file signature (89 50 4e 47 0D 0a 1a 0A) and verified, as we opened the now uncorrupted file, that it was a drone plan. We called the uncorrupted file drone-A.png.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	\tPNG.....IHDR
00000010	00	00	02	58	00	00	01	46	08	02	00	00	00	39	30	76	...X...F.....90v
00000020	FD	00	00	0B	4F	69	43	43	50	69	63	63	00	00	78	DA	ý...OiCCPicc..xÜ
00000030	ED	5A	69	54	13	57	1B	7E	66	02	01	64	5F	64	11	90	iZiT.W.~f..d_d..

drone-A.png MD5: d99f500968d444b5e0a1c9fd1dd69274

The file online_banking.docx contained only a password which we hope to be able to use later. Nothing else was found on this file that might feel relevant to the investigation.

4.5 Analysis of the file compress.py

Finally, at 6PM on the 22nd of October, through the tool uncompyle2 the program compress.py was decrypted producing the file uncomp_compress.py. After analyzing its contents, we reached the conclusion that it was a steganography tool to hide information inside images using the technique LSB.

```
C:\Users\user\Desktop>uncompyle6 compress.pyc
# uncompyle6 version 3.2.3
# Python bytecode 2.7 (62211)
# Decompiled from: Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (Intel)]
# Embedded file name: csfsteg/csfsteghide.py
# Compiled at: 2018-10-13 11:57:39
import sys, struct, numpy, PIL as pillow
from PIL import Image

def decompose(data):
    v = []
    fSize = len(data)
    bytes = [ ord(b) for b in struct.pack('i', fSize) ]
    bytes += [ ord(b) for b in data ]
    for b in bytes:
        for i in range(7, -1, -1):
            v.append(b >> i & 1)

    return v

def set_bit(n, i, x):
    mask = 1 << i
    n &= ~mask
    if x:
        n |= mask
    return n

def embed(imgFile, payload, password):
    img = Image.open(imgFile)
    width, height = img.size
    conv = img.convert('RGBA').getdata()
    print '[*] Input image size: %dx%d pixels.' % (width, height)
    max_size = width * height * 3.0 / 8 / 1024
    print '[*] Usable payload size: %.2f KB.' % max_size
    f = open(payload, 'rb')
    data = f.read()
    f.close()
    print '[+] Payload size: %.3f KB ' % (len(data) / 1024.0)
    v = decompose(data)
    while len(v) % 6:
        v.append(0)
```

uncomp_compress.py MD5: [df79d4920b843912aacd99e515eb49d0](#)

4.6 Reverse Engineering the python code

After analyzing the code, we understood the information was being converted into an array of bits which contained the size of the information and the information itself through the method *decompose* and then being masked into the last two bits of the color codes (RGB) of the pixels after passing a certain number of pixels determined by a password (we hypothesize that the password in the *online_banking.docx* might be the password that John Mole might have used in his steganography program) that could be part of the input of the program (*displacement*) through the method *set_bit*.

```
def decompose(data):
    v = []
    fSize = len(data)
    bytes = [ ord(b) for b in struct.pack('i', fSize) ]
    bytes += [ ord(b) for b in data ]
    for b in bytes:
        for i in range(7, -1, -1):
            v.append(b >> i & 1)
    return v

def set_bit(n, i, x):
    mask = 1 << i
    n &= ~mask
    if x:
        n |= mask
    return n
```

```
def embed(imgFile, payload, password):
    img = Image.open(imgFile)
    width, height = img.size
    conv = img.convert('RGBA').getdata()
    print('[*] Input image size: %dx%d pixels.' % (width, height))
    max_size = width * height * 3.0 / 8 / 1024
    print('[*] Usable payload size: %.2f KB.' % max_size)
    f = open(payload, 'rb')
    data = f.read()
    f.close()
    print('[+] Payload size: %.3f KB ' % (len(data) / 1024.0))
    v = decompose(data)
    while len(v) % 6:
        v.append(0)

    payload_size = len(v) / 8 / 1024.0
    print('[+] Embedded payload size: %.3f KB ' % payload_size)
    if payload_size > max_size - 4:
        print('[-] Cannot embed. File too large')
        sys.exit()

    steg_img = Image.new('RGBA', (width, height))
    data_img = steg_img.getdata()
    idx = 0
    displacement = 0
    for h in range(height):
        for w in range(width):
            if displacement < password:
                displacement = displacement + 1
                continue
            r, g, b, a = conv.getpixel((w, h))
            if idx < len(v):
                r = set_bit(r, 0, v[idx])
                r = set_bit(r, 1, v[idx + 1])
                g = set_bit(g, 0, v[idx + 2])
                g = set_bit(g, 1, v[idx + 3])
                b = set_bit(b, 0, v[idx + 4])
                b = set_bit(b, 1, v[idx + 5])
                idx = idx + 6
            data_img.putpixel((w, h), (r, g, b, a))

    steg_img.save(imgFile + '-stego.png', 'PNG')
    print('[+] %s embedded successfully!' % payload)
```

Understanding this, we decided to draw a truth table with the variables that would define how the bits were masked into the image.

n	mask	n & ~mask
00	01	00
00	10	00
01	01	01
01	10	00
10	01	00
10	10	10
11	01	01
11	10	10

n	x	o
00	00	00
00	01	10
00	10	01
00	11	11
01	00	00
01	01	10
01	10	01
01	11	11
10	00	00
10	01	10
10	10	01
10	11	11
11	00	00
11	01	10
11	10	01
11	11	11

n	mask	n mask
00	01	01
00	10	10
01	01	01
01	10	11
10	01	11
10	10	10
11	01	11
11	10	11

decompress.py MD5: DANIEL PREENCHE AQUI

inoffensive images. The fact that server passwords were included in the transmitted information might mean the idea was to continue extracting confidential information from the company DroneX.