INSTITUTO SUPERIOR TÉCNICO

Departamento de Engenharia Informática

Forensics Cyber Security

MEIC / METI 2018-2019 – 1st Semester

Digital Forensics Report

Ana Beatriz Nogueira	João Silveira	Martim Zanatti
82433	80789	82517

1 Objectives of the investigation

The investigation consists of the auditing process of John Mole, a DroneX employee suspected of industrial espionage. John Mole is suspected of having stolen the design plans for DroneX's new revolutionary drone technology in order to sell them to competitors. The following sections present in detail the analysis of files found on a pen drive the suspect was carrying along with him after returning from a trip to Germany.

2 Artifacts for analysis

The artifacts handed over appear to include four PNG files (cathedral.png, oktoberfest.png, street.png and wursten.png), one BMP file (snow.bmp), a text file (munich.txt), a ZIP compressed file (online_banking.zip) and a python file (compress.py). We first verified the integrity of the artifacts by checking the MD5 fingerprint provided when the files were handed in.

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

3 Evidence to look for

Before starting, we booted Kali (*forensics mode*) on a virtual machine and carried on with our investigation from there. We consider this an important step as we didn't know what kind of files we would be dealing with and wanted to make sure we wouldn't be running any malicious software on our own machines, and possibly compromise them and the artifacts.

We began our investigation by exploring each file with commands *file*, *hexdump* and *strings*, since it is a straightforward way to verify if the files are what they claim to be and/or check if they have any concealed information.

By examining the **snow.bmp** image, we found the first secret - someone had planted some text at the end of the file, leveraging the BMP file format. We first found the message through *strings* but it presented strangely formatted results, so we decided to run the *tail* command and we managed to correctly extract the first secret. The text found was written in german (we promptly translated it with the help of Google), a message stating that 5 (it

should be 4 as the 5 was apparently a typo) items of information would be sent: plans for drones A and B, technical specifications and data server passwords. We saved this data on a new **secret-message.txt** file.

"Ich sende Ihnen fünf Dateien: (1) Drohne A Pläne, (2) Drohne B Pläne, (3) technische Spezifikationen, (4) Passwörter von DroneX Dateiservern"	I'm sending you five files: (1) drone A plans, (2) drone B plans, (3) technical specifications, (4) passwords from DroneX file server

Text found in snow.bmp, original german version (left), english translation (right)

We then focused our attention on the **compress.py** file. By running *file* on compress.py we found out that it was a python2.7 byte compiled file, so it wasn't directly fully readable, but executing the *strings* command let us gather some information nonetheless. We were able to perceive that it was a Least Significant Bit Steganography tool. We temporarily put it on hold and examined the remaining files.

Exploring the **online_banking.zip** file with the archive manager, we were able to see the directory listing for the files inside (online_banking.docx and drone-A.bmp), but the archive itself was password protected. We could not extract any more information from it, but speculated that the password would be hidden somewhere in the other artifacts.

For the remaining files - **munich.txt** and all the **PNG images** - at this stage we were not able to get any information. If there was anything hidden in these files, we would need more specific tools to find it.

Taking a closer look at **online_banking.zip**, our first attempt was to try and crack the password. We searched for a tool that could perform this task. An Internet search led us to the ***fcrackzip*** utility. First we used the tool with the brute force algorithm, but upon experimenting we found it to be quite a lengthy process. The **munich.txt** file appeared to be harmless and to contain no hidden data - the file itself appeared to be nothing more than the Wikipedia article for the city of Munich. It then occurred to us that it might be a dictionary for cracking the zip file's password. *fcrackzip*'s dictionary mode uses a sorted file with a password per line and outputs the correct password, if present in the file. Thus we created a script (**parse_doc.py**) that parses the **munich.txt** file and creates a text file (**dictionary.txt**) with a word per line, alphabetically sorted. This script simply removes the spaces and puts a word per line. We thought of removing all punctuation and include only alphabetical words, but since the password could indeed include punctuation and we found it using the simple script, there was no need to dig further.

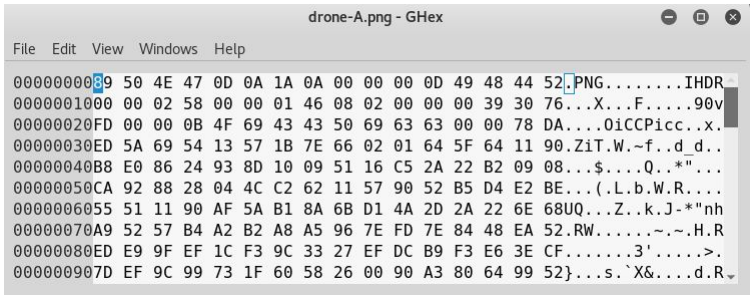
Then using *fcrackzip* with our dictionary, we were able to recover the password - "Stadelheim".

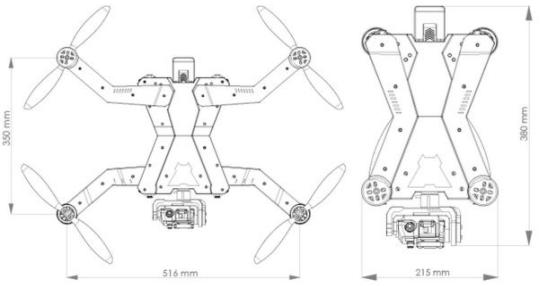
```
root@kali:~/Desktop/csf_lab1# python parse_doc.py
root@kali:~/Desktop/csf_lab1# fcrackzip -u -D -p dictionary.txt online_banking.zip


PASSWORD FOUND!!!!: pw == Stadelheim
```

The found password allowed us to unzip the compressed file and get access to the **drone-A.bmp** and **online_banking.docx** files.

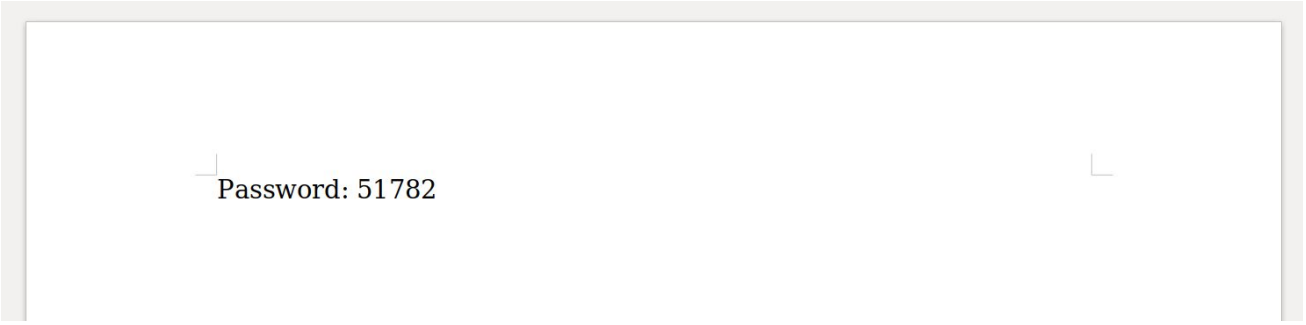
```
root@kali:~/Desktop/csf_lab1# unzip online_banking.zip
Archive:  online_banking.zip
[online_banking.zip] online_banking.docx password:
  inflating: online_banking.docx
  inflating: drone-A.bmp
```

We tried to open these files. We were unable to open drone-A.bmp with regular image software, and after executing ***file drone-A.bmp***, it did confirm that it wasn't a BMP image as advertised by the extension, but in fact just data, which meant that this file did not have any known magic number, or had the wrong one. Once again, with the help of *hexdump*, we found some interesting information at the end of the file . We found the strings *"%tEXtdate:create"* and *"%tEXtdate:modify"* which after a quick search appeared to be linked to PNG images. So we looked at the beginning of the file and looked at where a magic number could be. It seemed to be a corrupted version of a PNG's magic number. So we tried editing the file using the ***ghex*** utility, changing the beginning of the file to include the right number for PNG files. This finally revealed a PNG image with the plans for drone-A (**drone-A.png**).





Moving on to **online_banking.docx**, it appeared to be just an innocuous Microsoft Word document. The only information it contained was the following:



We suspected that there might have been more information inside the .docx file, and used *unzip* to investigate if we could find something, but we found no concealed information. As of this moment, we were still unsure of what the utility of the password was.

We moved on and took a look at the PNG images. We tried to use some LSB steganography tools to try and extract possible hidden information from the images. We tried both ***steghide*** and **LSBSteg** but neither was able to extract anything.

We then looked into the byte compiled file. We figured it could be of use to have the original source code for the program in order to understand how or why it might have been used. Our objective was to decompile it, so we looked for tools that might do the trick. We stumbled upon ***uncompyle6***, a tool that receives a **.pyc** file and reverts the compilation process, returning the original code.

We only had to change the extension of the **compress.py** file to **.pyc** and we got the code (**compress_code.py**). We thoroughly analysed it and understood how the LSB algorithm was implemented in this particular program. We could see that the code worked for the PNG format, present several times in our set of artifacts. This, and the lack of evidence found for the PNG files mentioned in the first analysis, led us to believe that the compress.py program might have been used to conceal information in those files.

Since the other tools didn't seem to work for this specific encoding, we used our newly acquired knowledge of the algorithm to build our own program to extract the information.

We had previously understood that the steganography program encoded information in the two least significant bits of each pixel's R, G and B channels (Red, Green and Blue, respectively). We also found out the order in which it encoded information: in the least significant bit first, and then on the second least significant bit. Furthermore, the algorithm also adds the size of the payload to the beginning of the hidden data.

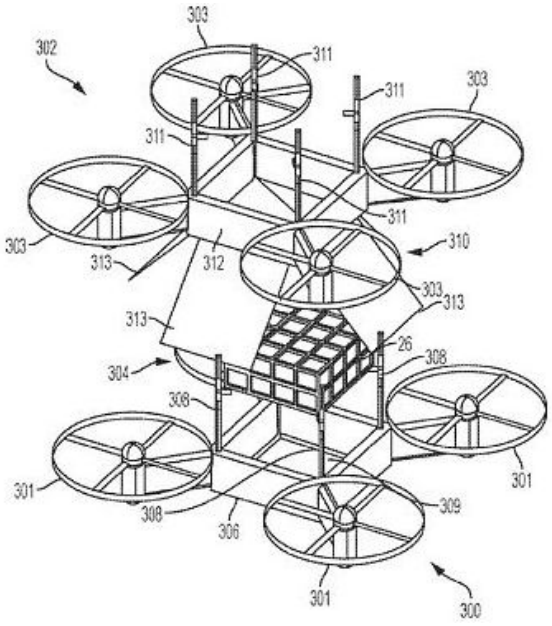
Examining the password method, for the stego tool, we understood that the modulo 13 operation would limit the password to 0-12. This way, we could manually try each password in an incremental manner.

So we wrote a program (**uncompress_code.py**) that printed the extracted information to the screen (when running it, we redirected the output to a file).

We ran this program over all the images present in the artifact set.

For **cathedral.png**, our program found nothing hidden for all 13 possible passwords. We then moved on to **oktoberfest.png**, with password 0. We received some output which we sent to **oktoberfest-out** and then interpreted with the *file* command, learning that it was a PNG image. All we had to do was change the file extension and then we could open the file to reveal the plans for drone B (**drone-B.png**).

```
root@kali:~/Desktop/csf_lab1# python2 uncompress_code.py oktoberfest.png 0 > oktoberfest-out
root@kali:~/Desktop/csf_lab1# file oktoberfest-out
oktoberfest-out: PNG image data, 343 x 376, 8-bit/color RGBA, non-interlaced
root@kali:~/Desktop/csf_lab1# mv oktoberfest-out drone-B.png
```



Running the program once again, now for the **street.png** image, using password 0 we extracted a **street-out** file, and using *file* again we found out it was a new PNG image depicting the Neuschwanstein Castle in Bavaria. It seemed like a strange result, but since we now had a new image (**castle.png**), we could run the program again over it. This way we found an ASCII text file containing the drones' technical specifications. We renamed it to **drone-specs.txt**.

One more PNG file remaining, we executed uncompress_code.py over **wursten.png**. Examining the output (**wursten-out**) with the *file* command again, we found another ASCII text file, containing the passwords mentioned in **secret-message.txt**. We saved this content on a new file **access-passwords.txt**.

```
OPERATING SPECIFICATIONS

DRONE A

Operational Altitude: 5,000 - 12,000ft WGS, no lift loss
(FAA limited at this time)
Sensor agnostic system
Full autonomous flying, terrain following, target tracking, real time GPS waypoints
Dual Mode - Wireless and Laptop
Automated return home
Max / min altitude setting
Payload up to 15lbs
Can fly in 30mph crosswinds
Take off and land in small areas
Flight Management Integration
Auto Takeoff, flight, and Landing
Waypoints - 10cm accuracy terrain following
Remote camera settings
GPS integration
Real time downlink data
Flight Time
Gas Powered: 60 - 90 minutes w / 10 - 15lb payload
Electric Powered: 25 - 45 minutes w / 10 - 15lb payload

DRONE B

Sensor agnostic system
Full autonomous flying, target tracking, real time GPS waypoints
Dual Mode - Wireless and Laptop
Automated return home
Max / min altitude setting
Payload up to 1lb
Take off and land in small areas
Flight Management Integration
Auto Takeoff, flight, and Landing
Waypoints - 5m accuracy
Remote camera settings
GPS integration
Real time downlink data
Flight Time
Electric Powered: 20 minutes w / 1lb payload
```

ACCESS PASSWORDS FOR DRONEX SERVERS

SERVER 1: Sr!_01llxt

SERVER 2: p_GEtKl4dA

4 Analysis results

We present below the list of files found during the investigation, a short description of their contents and their respective MD5 fingerprints. The files in bold are the five secrets.

File name	File contents	MD5
secret-message.txt	Message appended to the end of the snow.bmp artifact.	c8351db6ce4db8af451dfa1759646117
parse_doc.py	Python script used to format the munich.txt file into an fcrackzip dictionary.	7b41a0f17d5bc5589eb7b08c999f3c5b
dictionary.txt	Output of the execution of parse_doc.py over munich.txt. Dictionary file for fcrackzip.	448b8d3b3a63ed4319011918af93553c
drone-A.bmp	File found after unzipping the online_banking.zip package. Has a BMP extension but it is not a BMP file.	05029f0ae6af62ca3350f5b094584b22
online_banking.docx	Microsoft Word document containing an unknown	b70702822417bd39a7997a0f8c73941f

	password. File found after unzipping the online_banking.zip package.	
drone-A.png	PNG file with the plans of drone-A. File obtained by changing the magic number and extension on drone-A.bmp.	`d99f500968d444b5e0a1c9fd1dd69274`
compress.pyc	File obtained by changing the extension of compress.py so we could execute it as input to uncompyle6.	`72eab63334dcd0f73418e32999b71f05`
compress_code.py	Output of uncompyle6. Python file containing the original source code for the compress.py program and the LSB steganography tool.	`a5dda1478b809f134469fd162b10ab89`
uncompress_code.py	Python program developed by us to extract information off PNG image artifacts.	`8e67c2d2049a621f3c82ded3b3deb0c5`
oktoberfest-out	Output of the execution of uncompress_code.py over the oktoberfest.png image.	`5217527b444f5dc5f44fd9c8c8663aad`
drone-B.png	PNG file with the plans of drone-B. oktoberfest-out file with the correct extension.	`5217527b444f5dc5f44fd9c8c8663aad`
street-out	Output of the execution of uncompress_code.py over the street.png image.	`7e6d51a4997704a832d3866405c02792`
castle.png	PNG file, photograph. street-out file with the correct extension.	`7e6d51a4997704a832d3866405c02792`
drone-specs.txt	ASCII text file containing the drone specifications. Output of the execution of uncompress_code.py over the castle.png image.	`dfb4cc39c2a873eca69403ee1e532e25`
wursten-out	Output of the execution of uncompress_code.py over the wurst.png image.	`c5c7d9afc5fd46d7d4e44404b82bb660`
access-passwords.txt	ASCII text file containing the supposed access passwords for the DroneX servers. wursten-out file with the correct extension.	`c5c7d9afc5fd46d7d4e44404b82bb660`

After the five secrets were discovered, we thought about what the utility of the password found on the online_banking.docx file would be. It became clear that this password had no role in discovering these secrets, but was probably meant for something else.

There is no proof that John is really guilty, because these secrets could have either been introduced in his pen drive without his knowledge or he could simply not know they contained any secrets in the first place. This password could be the access to his private account in the company, or something similar. So this could be a way to incriminate John, because he is expected to be the only one with knowledge of his password. Again, we have no proof of this.

On the other hand, if we consider that John is guilty, we can think that this password could be a coordinate or a code to defining the next meeting. And the PNG image, cathedral.png, could correspond to the meeting place.

Without knowing the addresses of the access servers, we cannot confirm whether the passwords are legitimate. The same happens for the drone specifications and plans. Only with further internal information and investigation could we verify the legitimacy of the artifacts found.

5 Conclusions

The evidence obtained and analysed, by itself, cannot prove John Mole's guilt or lack thereof. The possession of sensitive information about DroneX's new drone technology in a covert way might be suspicious, but we cannot infer the suspect's guilt without further proof.

As we mentioned before, some entity might be trying to frame him. Also, his suspicious behaviour could just be a coincidence, and his intentions might have never been to give this information to the competition.