reto-seleccion

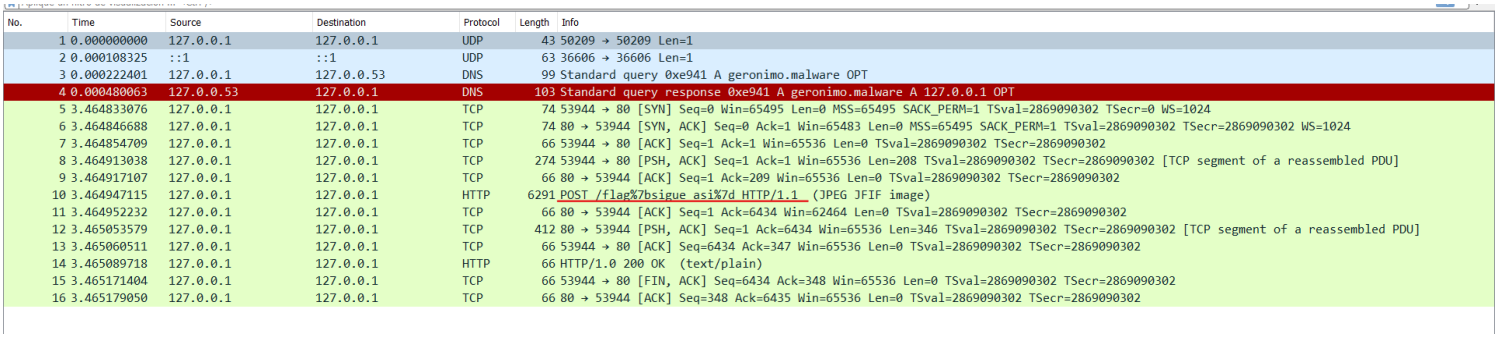
Name: Reto Selección ISDEFE
Category: Steganography and Pcap Tracing
Description: Find all the flags starting by init.txt

First-Contact

We were provided with two files to start with, ***init.txt*** and ***flag.pcapng***.
1. First, we obtain the content of the file ***init.txt***: ***cat init.txt*** → ***flag{buen_comienzo}***, showing the first flag of the challenge.
2. Secondly, we check the ***strings init.txt*** output, looking for hidden data in the file: No valuable output.
3. We recognise the pcapng as a trace file saved from the *WireShark* software, so we open it. → ***flag{sigue_así}***

WireShark

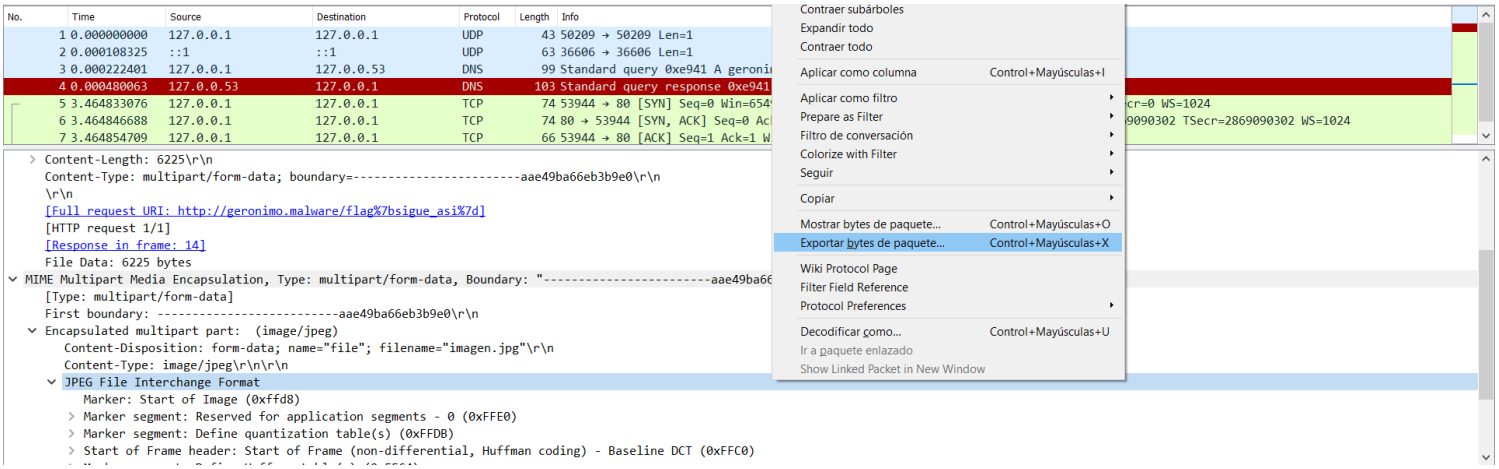
When we open the file ***flag.pcapng*** we can see the following traces:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	43	50209 → 50209 Len=1
2	0.000108325	::1	::1	UDP	63	36606 → 36606 Len=1
3	0.000222401	127.0.0.1	127.0.0.53	DNS	99	Standard query 0xe941 A geronimo.malware OPT
4	0.000480063	127.0.0.53	127.0.0.1	DNS	103	Standard query response 0xe941 A geronimo.malware A 127.0.0.1 OPT
5	3.464833076	127.0.0.1	127.0.0.1	TCP	74	53944 → 80 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=2869090302 TSecr=0 WS=1024
6	3.464846688	127.0.0.1	127.0.0.1	TCP	74	80 → 53944 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=2869090302 TSecr=2869090302 WS=1024
7	3.464854709	127.0.0.1	127.0.0.1	TCP	66	53944 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2869090302 TSecr=2869090302
8	3.464913038	127.0.0.1	127.0.0.1	TCP	274	53944 → 80 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=208 TSval=2869090302 TSecr=2869090302 [TCP segment of a reassembled PDU]
9	3.464917107	127.0.0.1	127.0.0.1	TCP	66	80 → 53944 [ACK] Seq=1 Ack=209 Win=65536 Len=0 TSval=2869090302 TSecr=2869090302
10	3.464947115	127.0.0.1	127.0.0.1	HTTP	6291	POST /flag%7bsigue_asi%7d HTTP/1.1 (JPEG JFIF image)
11	3.464952232	127.0.0.1	127.0.0.1	TCP	66	80 → 53944 [ACK] Seq=1 Ack=6434 Win=62464 Len=0 TSval=2869090302 TSecr=2869090302
12	3.465053579	127.0.0.1	127.0.0.1	TCP	412	80 → 53944 [PSH, ACK] Seq=1 Ack=6434 Win=65536 Len=346 TSval=2869090302 TSecr=2869090302 [TCP segment of a reassembled PDU]
13	3.465060511	127.0.0.1	127.0.0.1	TCP	66	53944 → 80 [ACK] Seq=6434 Ack=347 Win=65536 Len=0 TSval=2869090302 TSecr=2869090302
14	3.465089718	127.0.0.1	127.0.0.1	HTTP	66	HTTP/1.0 200 OK (text/plain)
15	3.465171404	127.0.0.1	127.0.0.1	TCP	66	53944 → 80 [FIN, ACK] Seq=6434 Ack=348 Win=65536 Len=0 TSval=2869090302 TSecr=2869090302
16	3.465179050	127.0.0.1	127.0.0.1	TCP	66	80 → 53944 [ACK] Seq=348 Ack=6435 Win=65536 Len=0 TSval=2869090302 TSecr=2869090302

First, we find traces related with the local DNS assignment of the target computer. After that, we see a 3-step connection to the web page ***http://geronimo.malware***, also notice that the trace 10 contains a special text in the post header, this trace reveals the flag ***flag{sigue_asi}***.

Also, in that same trace, notice that WireShark displays the message ***(JPEG JFIF Image)*** which means that the packet contains a *JPEG* image. Let's try downloading that image from the packet's body:



In the image above we go to the trace 10, open the *MIME Multipart Media Encaps.* section and look for the *JPEG File Interchange Format* field. Rigth-Click that title and select *Export bytes from package*, this will display a popup where we will save the image as ***flag3.jpeg***.

Finally, we open the downloaded image with any image viewer, and obtain the 4th flag:
flag{welcome_to_stego}

steghide

As we saw in the WireShark page, the 3rd flag is contained in an image and it's content is ***flag{welcome_to_stego}***. This flag suggests that the image we downloaded has a file hidden within its binary data, this process is called *Steganography*. There are several tools tha can be used to detect, insert and extract files with steganography, we are going to use ***steghide***, available in linux apt repos.

```
sudo apt install steghide
```

With the help of steghide we try to extract the supposed hidden file inside ***flag3.jpeg***. The execution of the program to extract embedded files usually go as follows:

```
steghide extract -p passPhrase -sf fileName
```

Notice: The files hidden with stego are usually password protected, in our first try we will insert an empty passphrase and if we fail we could try guessing or brute forcing the password. With the ***-sf fileName*** parameter we specify the name of the file from where to extract the hidden file.

Let's run this program against the file ***flag3.jpeg*** and try to extract any file hidden with an empty passphrase:

```
alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/reto_seleccion$ steghide extract -p "" -sf
flags/flag3.jpeg
wrote extracted data to "flag.txt".
alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/reto_seleccion$ cat flag.txt
flag{este_es_casi_el_final}









alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/reto_seleccion$
```

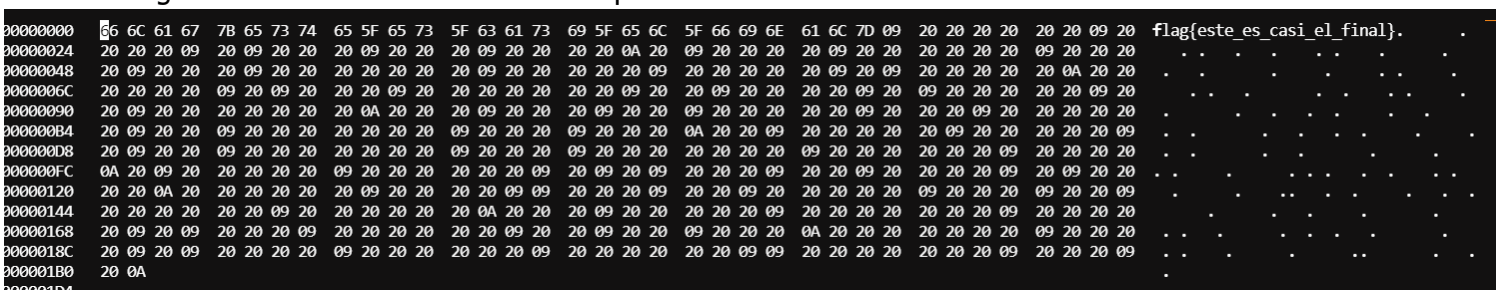
At this point we discovered the 4th flag: ***flag{este_es_casi_el_final}***, that suggests we are almost at the end of the challenge.

stegsnow

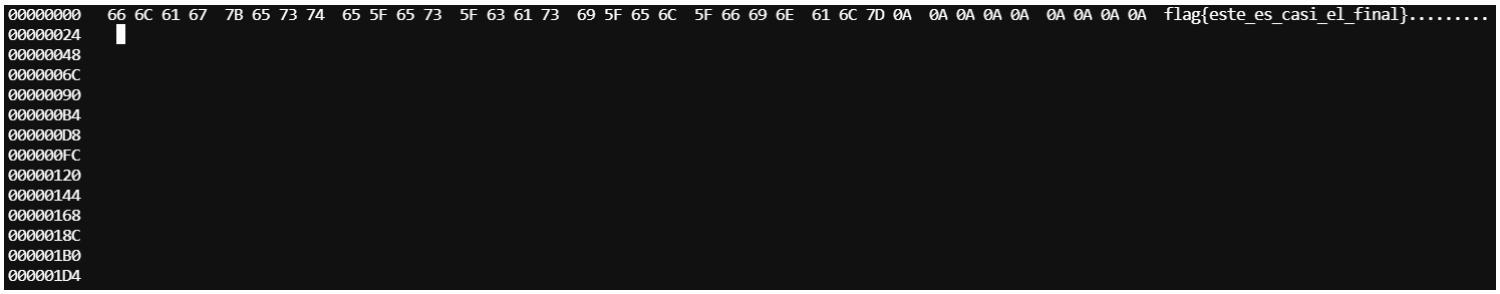
The file containing the 4th flag is a `.txt` file, this means *steghide* will not be able to extract any files from it (as it is not an image). Anyway, if we have closer look at the contents of `flag4.txt`:

```
alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/reto_seleccion$ cat flags/flag4.txt
flag{este_es_casi_el_final}



alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/reto_seleccion$
```

We can notice that the plain text is not the only information stored in this file, so we can assume there is something hidden in the blank characters found in the file. To make sure that we are right, we are going to run a `hexedit` against the file and see the ascii output:



We can see that the *hexdump* of the file is irregular, to be sure we can compare it with a new file which contains only the text and 5 '\n's:



There is an obvious difference between the two outputs, so we are sure there is something hidden in that dot grid. After researching about ways to hide files or data in .txt using spaces and tabulations we discovered ***snow*** a steganographic program capable of embedding complete files inside .txt, .docx, etc. files. We download the program ***stegsnow*** from the linux apt repository and run some tests on the test file we created:

```
alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/reto_seleccion$ stegsnow -p "test" -m "Hi"
flag_test.txt flag_hidden.txt
Message used approximately 20.51% of available space.
alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/reto_seleccion$ cat flag_hidden.txt
flag{este_es_casi_el_final}



alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/reto_seleccion$
```

```
66 6C 61 67  7B 65 73 74  65 5F 65 73  5F 63 61 73  69 5F 65 6C  5F 66 69 6E  61 6C 7D 09  09 20 20 20  20 20 09 20  flag{este_es_casi_el_final}..    .
20 20 20 20  20 09 20 09  20 20 20 20  20 0A 09 0A  0A 0A                                    . .    .....
```

Success!!! We inserted the message *Hi* in the file `flag_hidden.txt` and, as we can see, the output obtained is similar to the file extracted with `steghide`. The point is that `stegsnow`, as well as `steghide`, requires a passphrase to embed or extract files or messages. Let's give it a try with an empty passphrase:

```
alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/reto_seleccion$ stegsnow -p "" flags/
flag4.txt
Warning: an empty password is being used
\'xK
    {kri"vR#alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/reto_seleccion$
```

The passphrase ended up in failure, but not completely, the fact that the extraction give us some output demonstrates that there is some message hidden, but the passphrase was not correct.
Up to this point we can follow two paths, trying to guess the password with information found in the previous steps or running bruteforce against the file to try to find the passphrase.

1. Let's try to guess the passphrase with the information gathered up to now:
   - ***isdefe*** (name of the organisation that hosts the challenge page) - Failure
   - ***stego*** (from flag3.jpeg) - Failure
   - ***steganography*** - Failure
   - ***snow*** (the stego program) - Failure
   - ***geronimo.malware*** (from flag.pcapng) - Failure
   - ***geronimo*** - Failure
   - ***malware*** - Failure
   - ***flag{este_es_casi_el_final}*** (last known flag) - Failure

2. Let's run a simple python program to try to crack the passphrase. To do this, we could use a popular wordlist within brute forcing, ***rockyou.txt***, the callback is that this file has 1.435.000 combinations and lots of time to analyze them all. Instead we will use a custom wordlist with words from 'a' to 'zzzzzzzz', because the rest of the flags where easy to capture and we assume that, if there is a passphrase, it must be shorter than 8 characters.
   - First, create the wordlist with the script ***dic_creator.py*** obtained from github.
   - Second, run the script ***snow_decrypt.py*** against the ***flag4.txt*** file and try to extract the file.
   - We left the script running tests while continuing investigating other paths to follow.

```
 alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/reto_seleccion$ python3 scripts/
snow_decrypt.py
Trying with:         dn8, tried: 17343 times.
```

After a deeper research on the ***stegsnow*** program use, we discovered that in some unusual cases you can use the program to extract without the ***-p*** argument: ***stegsnow flags/flag4.txt***, let's try it:

```
alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/reto_seleccion$ stegsnow flags/flag4.txt
flag{este_es_el_verdadero_final}alex@DESKTOP-MQKMDU5:/mnt/c/Ciberseg/
reto_seleccion$
```

Success!! The last step is to redirect the output of the execution:

```
stegsnow flags/flag4.txt > flags/flag5.txt
```

and we solved the challenge to the final flag.

dic_creator.py

```
import os
import sys
import time
import string
import argparse
import itertools


def createWordList(chrs, min_length, max_length, output):
    """
    :param `chrs` is characters to iterate.
    :param `min_length` is minimum length of characters.
    :param `max_length` is maximum length of characters.
    :param `output` is output of wordlist file.
    """
    if min_length > max_length:
        print ("[!] Please `min_length` must smaller or same as with
`max_length`")
        sys.exit()

    if os.path.exists(os.path.dirname(output)) == False:
        os.makedirs(os.path.dirname(output))

    print ('[+] Creating wordlist at `%s`...' % output)
    print ('[i] Starting time: %s' % time.strftime('%H:%M:%S'))

    output = open(output, 'w')

    for n in range(min_length, max_length + 1):
        for xs in itertools.product(chrs, repeat=n):
            chars = ''.join(xs)
            output.write("%s\n" % chars)
            sys.stdout.write('\r[+] saving character `%s`' % chars)
            sys.stdout.flush()
    output.close()

    print ('\n[i] End time: %s' % time.strftime('%H:%M:%S'))


if __name__ == '__main__':
    parser = argparse.ArgumentParser(
        formatter_class=argparse.RawTextHelpFormatter,
        description='Python Wordlist Generator')
    parser.add_argument(
        '-chr', '--chars',
        default=None, help='characters to iterate')
    parser.add_argument(
        '-min', '--min_length', type=int,
        default=1, help='minimum length of characters')
```

```
parser.add_argument(
    '-max', '--max_length', type=int,
    default=2, help='maximum length of characters')
parser.add_argument(
    '-out', '--output',
    default='output/wordlist.txt', help='output of wordlist file.')

args = parser.parse_args()
if args.chars is None:
    args.chars = string.printable.replace(' \t\n\r\x0b\x0c', '')
createWordList(args.chars, args.min_length, args.max_length, args.output)
```

snow_decrypt.py

```
import os

dict_file = open('../wordlists/snow_brute.txt', 'r') # Open wordlist file
res_file = open('result.txt', 'r') # Output file from the snow execution

count = 0
for line in dict_file:
    print("Trying with: %10s, tried: %5d times."%
(line.replace('\n',''),count),end='\r')

    # Run stegsnow against 'line' passphrase
    os.system('stegsnow -Q -p "%s" flags/flag4.txt result.txt' % line)

    # Read the output file and check for 'flag' string in its content
    text = res_file.read()
    count += 1
    if 'flag' in res_file.read():
        print(line)
        break
else:
    print("The file flag4.txt couldn't be extracted with any passphrase of
snow_brute")

# Close the opened streams
res_file.close()
dict_file.close()
```