# Maquina GamingService

# Writeup: GamingService (TryHackMe)

## Introducción

Este writeup documenta la explotación de la máquina **GamingService** en la plataforma TryHackMe. Se detalla el proceso desde el reconocimiento inicial hasta la exploración de vulnerabilidades encontradas.

## Reconocimiento Inicial

### Escaneo de Puertos

Se realizó un escaneo inicial con **Nmap** para identificar los puertos abiertos en la máquina objetivo.

```
┌──(zikuta㉿zikuta)-[~]
└─$ nmap -A --top-ports 100 10.10.163.91
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-13 17:01 CDT
Nmap scan report for 10.10.163.91
Host is up (0.18s latency).
Not shown: 98 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol
2.0)
| ssh-hostkey:
|   2048 34:0e:fe:06:12:67:3e:a4:eb:ab:7a:c4:81:6d:fe:a9 (RSA)
|   256 49:61:1e:f4:52:6e:7b:29:98:db:30:2d:16:ed:f4:8b (ECDSA)
|_  256 b8:60:c4:5b:b7:b2:d0:23:a0:c7:56:59:5c:63:1e:c4 (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: House of danak
Device type: general purpose
Running: Linux 4.X
OS CPE: cpe:/o:linux:linux_kernel:4.15
OS details: Linux 4.15
Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

TRACEROUTE (using port 80/tcp)
HOP RTT       ADDRESS
```

```
1   183.77 ms 10.23.0.1
2   183.88 ms 10.10.163.91

OS and Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 17.43 seconds
```

**Resultados:**

- **Puerto 22 (SSH):** Servidor SSH activo.
- **Puerto 80 (HTTP):** Servidor web activo.

## Análisis del Servidor Web

Navegando al puerto 80, se encontró un servidor web funcional. Para identificar posibles recursos ocultos, se realizó un escaneo de directorios con **dirsearch**.

# Escaneo de Directorios

El comando ejecutado fue:

```
┌──(zikuta㉿zikuta)-[~]
└─$ dirsearch -u http://10.10.163.91
/usr/lib/python3/dist-packages/dirsearch/dirsearch.py:23: DeprecationWarning:
pkg_resources is deprecated as an API. See
https://setuptools.pypa.io/en/latest/pkg_resources.html
  from pkg_resources import DistributionNotFound, VersionConflict

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

Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 25
Wordlist size: 11460

Output File: /home/zikuta/reports/http_10.10.163.91/_25-06-13_17-02-00.txt

Target: http://10.10.163.91/

[17:02:00] Starting:
[17:02:09] 403 -  277B  - /.ht_wsr.txt
[17:02:09] 403 -  277B  - /.htaccess.bak1
[17:02:09] 403 -  277B  - /.htaccess.orig
[17:02:09] 403 -  277B  - /.htaccess.sample
[17:02:09] 403 -  277B  - /.htaccess.save
```

```
[17:02:09] 403 -  277B  - /.htaccess_orig
[17:02:09] 403 -  277B  - /.htaccess_extra
[17:02:09] 403 -  277B  - /.htaccess_sc
[17:02:09] 403 -  277B  - /.htaccessBAK
[17:02:09] 403 -  277B  - /.htaccessOLD
[17:02:09] 403 -  277B  - /.htaccessOLD2
[17:02:09] 403 -  277B  - /.html
[17:02:09] 403 -  277B  - /.htm
[17:02:09] 403 -  277B  - /.htpasswds
[17:02:09] 403 -  277B  - /.htpasswd_test
[17:02:09] 403 -  277B  - /.httr-oauth
[17:02:11] 403 -  277B  - /.php
[17:02:19] 200 -    2KB - /about.php
[17:02:19] 200 -  524B  - /about.html
[17:03:18] 200 -   33B  - /robots.txt
[17:03:19] 200 -  455B  - /secret/
[17:03:19] 301 -  313B  - /secret  ->  http://10.10.163.91/secret/
[17:03:20] 403 -  277B  - /server-status/
[17:03:20] 403 -  277B  - /server-status
[17:03:31] 301 -  314B  - /uploads  ->  http://10.10.163.91/uploads/
[17:03:31] 200 -  522B  - /uploads/
Task Completed
```

**Resultados principales:**

- `/about.php` (200 OK)
- `/about.html` (200 OK)
- `/robots.txt` (200 OK)
- `/secret/` (200 OK)
- `/uploads/` (200 OK)
- Otros directorios como `/.htaccess`, `/.php`, etc., retornaron 403 Forbidden.

# Análisis de Archivos Directorios

## Archivo en `/uploads/`: `dic.lst`

Dentro del directorio `/uploads/`, se identificó un archivo llamado `dic.lst`. Este archivo contenía una lista extensa de contraseñas. Estas claves serán útiles para futuras tareas de fuerza bruta o descifrado.

## Archivo en `/secret/`: `id_rsa`

En el directorio `/secret/`, se encontró un archivo `id_rsa`. Este archivo es una clave privada SSH que puede utilizarse para acceder al servicio SSH en el puerto 22. Sin embargo, para

usarla, se requería una **passphrase** (frase de acceso), que estaba protegida por un hash.

## Cracking de la `id_rsa` con John the Ripper

Para descifrar la **passphrase**, se utilizó la herramienta **John the Ripper**. Este proceso involucró los siguientes pasos:

1. **Convertir la clave privada en un formato utilizable por John:**

   El comando utilizado fue:

```
python3 /usr/share/jhon/ssh2john.py id_rsa > rsa_hash.txt
```

Este paso extrae el hash de la clave privada SSH, permitiendo que John lo procese.

2. **Crackear el hash utilizando el archivo `dic.lst` como diccionario:**

   Con el hash extraído, se ejecutó John con el siguiente comando:

```
┌──(zikuta㉿zikuta)-[~/Desktop]
└─$ john --wordlist=contrasenas.txt rsha_hash.txt
```

Esto me dio la contrasena del passphrase de SSH la cual es `letmein`

## Uso de la Clave SSH Descifrada

Con la **passphrase** obtenida ( `letmein` ), se intentó acceder al servidor SSH utilizando la clave privada `id_rsa` .

## Procedimiento:

1. Cambiar los permisos de la clave privada.

```
chmod600 id_rsa
```

2. Conectar al servidor SSH:

```
ssh john@10.10.10.16.139.30 -i id_rsa
```

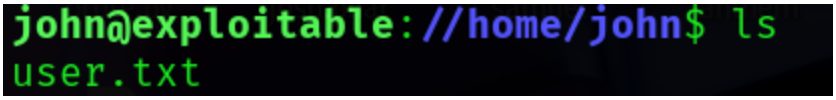
3. Nos pedira el passphrase y introduciremos `letmein`

## Explicación Detallada

- **Clave Privada SSH (`id_rsa`):** Es un archivo utilizado para la autenticación SSH sin contraseña, pero suele estar protegida por una **passphrase** para mayor seguridad.
- **Herramienta `ssh2john`:** Convierte la clave SSH en un hash que puede ser procesado por herramientas de cracking como John the Ripper.
- **Cracking con John:** Se realizó un ataque de diccionario utilizando el archivo `dic.lst`. John probó cada contraseña hasta encontrar la correcta que descifró el hash y reveló la **passphrase**.

# User Flag

ahora en el escritorio como el usuario john procedemos a leer la flag de usuario.




# Escalada de privilegios

Mientras buscaba vulnerabilidades para escalar privilegios, al momento de ver los grupos a los que pertenecia me encontre algo bastante curioso

```
john@exploitable:~$ id
uid=1000(john) gid=1000(john)
groups=1000(john),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lxd)
```

### Análisis del Grupo LXD y Preparación del Entorno

Durante la enumeración de privilegios, se identificó que el usuario `john` pertenecía al grupo `lxd` (Listado: `id`). LXD (*Linux Container Daemon*) es un sistema de gestión de contenedores privilegiados que, si no está correctamente configurado, puede permitir la escalada de privilegios mediante la creación de contenedores con acceso al sistema de archivos del host.

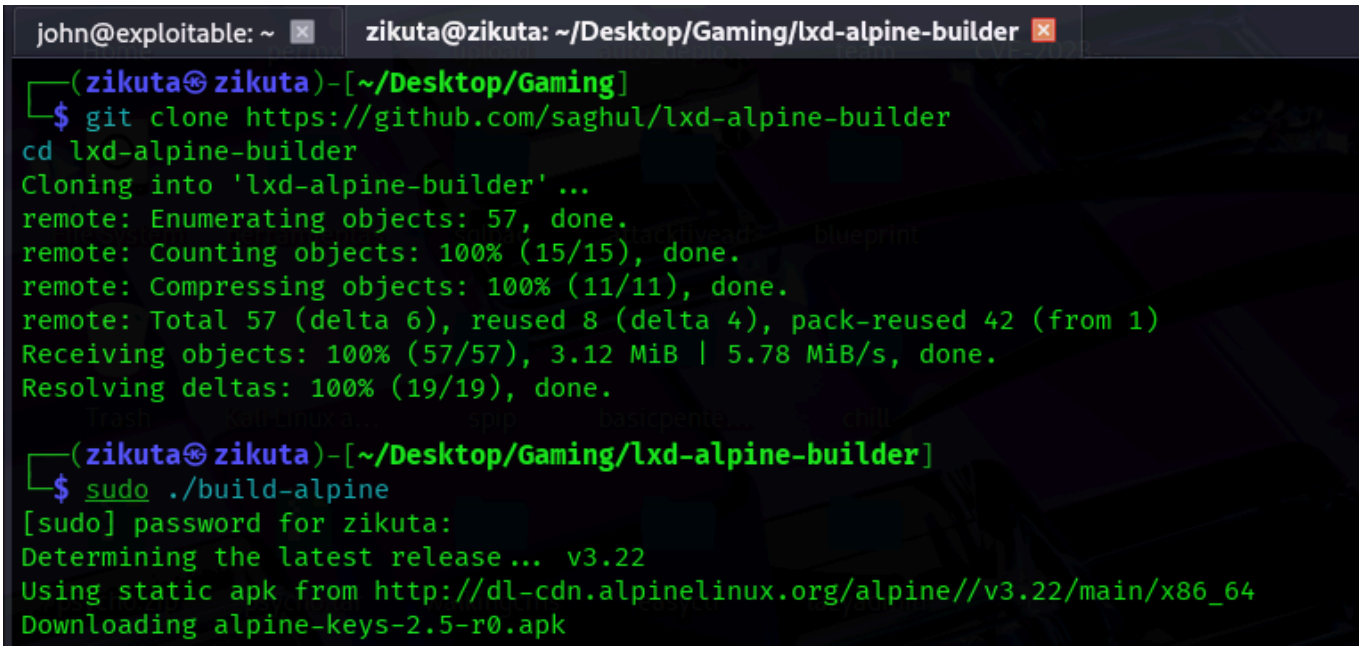
# ¿Qué es Alpine Linux?

**Alpine Linux** es una distribución de Linux ultra ligera y segura, muy usada en contenedores (Docker, LXC/LXD) por su pequeño tamaño (~5MB) y su enfoque en seguridad (usa **musl libc** en lugar de glibc).

- **Ventajas para pentesting**:
  - Es minimalista (ideal para transferir en entornos restringidos).

- Fácil de modificar para incluir herramientas de explotación.

la descargamos en nuestra maquina host




# ¿Por qué lo necesitamos para explotar LXD?

Para abusar del grupo `lxd`, necesitamos:

1. **Una imagen de contenedor maliciosa**:

- Debe permitir ejecutar comandos como **root** en el host.
- Alpine es ideal porque es pequeño y fácil de manipular.

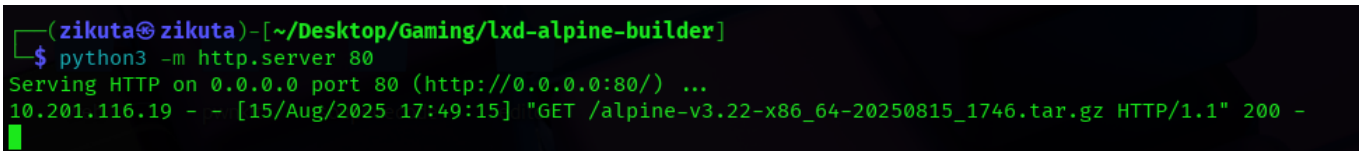
**Montar el sistema de archivos del host:**

- El builder crea una imagen que podemos configurar para montar / (directorio raíz del host) dentro del contenedor.

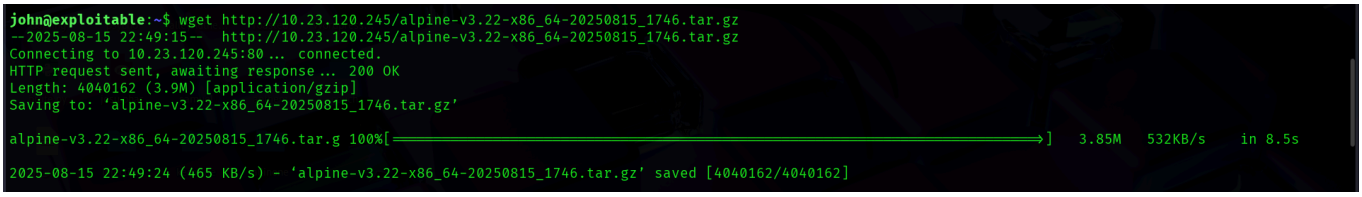
**Evitar dependencias complejas**:

- Otras imágenes (Ubuntu, CentOS) son más grandes y pueden fallar en entornos sin internet.
- Alpine pesa ~5MB (fácil de transferir incluso con `wget` o `python3 -m http.server`).

despues de instalar el software procederemos a hostearlo en el puerto 80 para luego descargarnoslo desde la maquina victima




luego en la maquina victima procederemos a descargar el archivo




El primer paso crítico es importar la imagen de Alpine Linux que generamos previamente (o descargamos) en el sistema vulnerable usando el comando `lxc image import`.

```
john@exploitable:~$ lxc image import  ./alpine-v3.22-x86_64-
20250815_1746.tar.gz --alias myimage
Image imported with fingerprint:
8cc7ac3fa7474017aebfa712aebb9c730904be6d3f15f82b04d9572dccc2b865
```

# Qué hace este comando?

`lxc image import:`

- **Es el comando de LXD para cargar una nueva imagen en su repositorio local**.
- **Las imágenes en LXD son plantillas que se usan para crear contenedores**.

`./alpine-v3.14-x86_64-20210804_1750.tar.gz:`

- **Es el archivo comprimido que contiene el sistema de archivos de Alpine Linux**.
- **Este archivo fue generado previamente con el script build-alpine y transferido al sistema objetivo.**

`--alias myimage:`

- **Asigna un nombre fácil de recordar (myimage) a la imagen importada.**
- **Esto evita tener que referirse a la imagen por su nombre largo (alpine-v3.14...).**

# ¿Por qué es necesario?

- **Sin importar la imagen, no podríamos crear un contenedor.**
- **LXD trabaja con imágenes predefinidas (como Docker), y necesitamos una que podamos controlar para explotar el sistema.**

# Crear un Contenedor Privilegiado y Montar el Sistema de Archivos del Host

Una vez importada la imagen, el siguiente paso es crear un contenedor basado en ella, pero con configuraciones especiales que permitan el acceso al sistema de archivos del host.

```
john@exploitable:~$ lxc init myimage ignite -c security.privileged=true
Creating ignite
```

## Explicación Detallada:

### 1. `lxc init myimage ignite -c security.privileged=true`

- `lxc init` : Crea un nuevo contenedor (similar a `docker create` ).
- `myimage` : Es la imagen que importamos previamente.
- `ignite` : Nombre que le damos al contenedor (podría ser cualquier otro).
- `-c security.privileged=true` :
  - Esta es la parte **más importante** del exploit.
- Por defecto, los contenedores LXD se ejecutan en un entorno aislado (sin acceso completo al host).
  - Al establecer `security.privileged=true` , el contenedor se ejecutará con **privilegios de root en el sistema host**, rompiendo el aislamiento.

Luego ejecutaremos el comando

```
john@exploitable:~$ lxc config device add ignite mydevice disk source=/
path=/mnt/root recursive=true
Device mydevice added to ignite
```

`lxc config device add ignite mydevice disk source=/ path=/mnt/root recursive=true`

Este comando configura un "dispositivo" en el contenedor que **monta el directorio raíz ( `/` ) del host** dentro del contenedor.

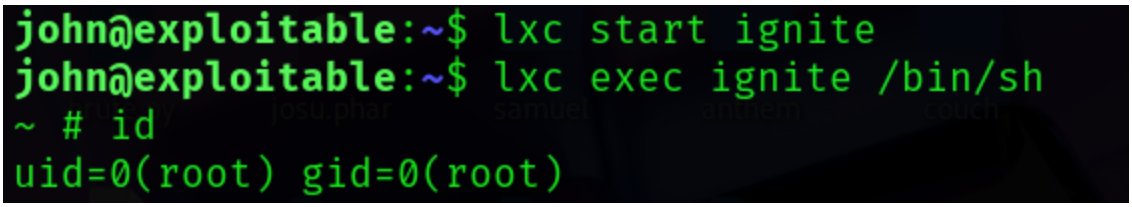
- `ignite` : Nombre del contenedor.
- `mydevice` : Nombre arbitrario para el dispositivo (podría ser `hostroot` o similar).
- `disk` : Tipo de dispositivo (en este caso, un disco virtual).
- `source=/` : Indica que el origen es el directorio raíz ( `/` ) del **host** (no del contenedor).
- `path=/mnt/root` : Monta el directorio raíz del host en `/mnt/root` dentro del contenedor.
- `recursive=true` : Asegura que todos los subdirectorios también sean accesibles.

### 3. `lxc start ignite`

- Simplemente inicia el contenedor ( `ignite` ).
- Sin esto, el contenedor estaría creado pero no en ejecución.

## Ejecutar una Shell en el Contenedor y Escalar a Root

Finalmente, ejecutamos una shell dentro del contenedor y usamos el comando `lxc exec ignite /bin/sh`




y finalmente obtuvimos ROOT!!

## ¿Por qué esto nos da acceso root?

- Al montar `/` del host en `/mnt/root` dentro del contenedor, **tenemos acceso a TODO el sistema de archivos del host**.
- Como el contenedor se ejecuta en modo privilegiado ( `security.privileged=true` ), podemos modificar archivos críticos como:
  - `/etc/shadow` (para cambiar la contraseña de root).
  - `/root/.ssh/authorized_keys` (para agregar una clave SSH y acceder como root).
  - Cualquier otro archivo del sistema.

## Conclusión

La explotación de la máquina **GamingService** en TryHackMe demostró una serie de vulnerabilidades críticas que permitieron obtener acceso no autorizado y escalar privilegios hasta convertirse en **root**. A continuación, se detallan los puntos clave que facilitaron la vulneración del sistema, cómo se llevó a cabo el ataque y las posibles medidas de mitigación.

### Puntos Clave que Permitieron el Acceso Total:

1. **Exposición de información sensible**:
   - Se encontró un archivo `id_rsa` (clave privada SSH) en el directorio `/secret/` , protegido por una passphrase.

- El archivo `dic.lst` en `/uploads/` contenía contraseñas potenciales, lo que facilitó el cracking de la passphrase.

2. **Uso de credenciales débiles**:
   - La passphrase de la clave SSH ( `letmein` ) era fácil de descifrar con un ataque de diccionario.
3. **Mala configuración de permisos**:
   - El usuario `john` pertenecía al grupo `lxd` , lo que permitió abusar de contenedores privilegiados para escalar a root.
4. **Falta de hardening en LXD**:
   - No se restringió el uso de contenedores privilegiados, lo que permitió montar el sistema de archivos del host y modificar archivos críticos.

## Técnicas Utilizadas:

| Técnica | Herramienta/Comando | Propósito |
|---|---|---|
| Escaneo de puertos | `nmap -A --top-ports 100` | Identificar servicios activos (SSH en puerto 22, HTTP en puerto 80). |
| Escaneo de directorios | `dirsearch -u <URL>` | Encontrar recursos ocultos ( `/secret/` , `/uploads/` , `robots.txt` ). |
| Fuerza bruta con diccionario | `john --wordlist=dic.lst` | Descifrar la passphrase de la clave SSH ( `id_rsa` ). |
| Explotación de grupo `lxd` | `lxc init` , `lxc config` | Crear un contenedor privilegiado para montar el sistema de archivos del host. |
| Escalada de privilegios | `lxc exec ignite /bin/sh` | Obtener una shell como root dentro del contenedor. |

## Posibles Medidas de Mitigación:

1. **Protección de archivos sensibles**:
   - Evitar almacenar claves privadas ( `id_rsa` ) o diccionarios de contraseñas en directorios accesibles públicamente.
   - Usar permisos estrictos (ej: `chmod 600` para claves SSH).
2. **Configuración segura de SSH**:
   - Implementar autenticación de dos factores (2FA).
   - Deshabilitar el acceso SSH con claves débiles o sin passphrase.
3. **Restricción de grupos de usuarios**:
   - Limitar la membresía en grupos críticos como `lxd` , `sudo` , o `docker` solo a usuarios necesarios.

4. **Hardening de LXD/LXC**:
    - Deshabilitar contenedores privilegiados ( `security.privileged=false` ).
    - Aplicar políticas de seguridad para evitar el montaje de directorios del host.
5. **Monitoreo y auditoría**:
    - Revisar logs de acceso a servicios como Apache y SSH para detectar actividades sospechosas.
    - Usar herramientas como `auditd` para rastrear cambios críticos en el sistema.

## Reflexión Final:

Este caso resalta la importancia de la configuración segura y la gestión adecuada de credenciales. Pequeños descuidos, como exponer archivos sensibles o permitir grupos innecesarios, pueden llevar a compromisos totales del sistema. La mitigación requiere un enfoque proactivo, combinando hardening, monitoreo y educación en seguridad para los administradores.