



**Universidad**  
Zaragoza



Escuela Universitaria  
Politécnica - Teruel  
**Universidad**Zaragoza

# Práctica 2

Ingeniería inversa de un programa

**Radu Constantin Robu**  
**Alberto Pérez Blasco**

Sistemas Legados

Escuela Universitaria Politécnica de Teruel  
Departamento de Informática e Ingeniería de Sistemas

5 de diciembre de 2021

## Resumen ejecutivo

Práctica dedicada a la ejecución crackeada de un programa del cual se ha perdido la documentación y la llave de acceso que se conecta a un puerto paralelo (LPT1).

Se ha estudiado el código máquina y el código desensamblado del programa con el fin de realizar los cambios necesarios para saltar las posibles barreras del programa.

Para saltar la primera barrera se ha ejecutado el comando `sudo chmod 7777 legado1` para cambiar los permisos de ejecución del ejecutable.

Para saltar la segunda barrera se ha ejecutado el comando `sudo chown root legado1` para cambiar el usuario propietario del ejecutable porque la llave de acceso necesita permisos de super-usuario.

Para saltar la tercera barrera se han tenido que modificar los saltos de las distintas opciones del menú de opciones del programa ejecutable.

Por último se ha documentado en detalle todos estos aspectos en el informe escrito.

## Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Desensamblado del código</b>	<b>2</b>
<b>3. Características del programa original</b>	<b>3</b>
<b>4. Ejecución 32-bit libc6-i386</b>	<b>3</b>
<b>5. Salto de la primera barrera</b>	<b>4</b>
<b>6. Salto de la segunda barrera</b>	<b>5</b>
<b>7. Script para saltar las barreras</b>	<b>5</b>
<b>8. Problema de ejecución: Debian vs Ubuntu</b>	<b>6</b>
<b>9. Salto de la tercera barrera</b>	<b>7</b>
<b>Referencias</b>	<b>11</b>

## 1. Introducción

La práctica 2 de la asignatura de Sistemas Legados tiene como objetivo principal conseguir la ejecución de un programa del cual se ha perdido la documentación original y la llave de protección necesaria para la ejecución del programa.

Las tareas principales que se han intentado realizar son saltar las posibles barreras que impidan la ejecución del programa y saltar la ejecución mediante la llave de acceso del programa original.

El entorno de trabajo han sido dos máquinas con sistemas operativos Ubuntu y Linux Debian respectivamente. En principio, el trabajo realizado consta de la edición directa del fichero binario del programa mediante el editor hexadecimal `hexedit`. La edición se ha realizado directamente en la terminal de los sistemas empleados para la realización de la práctica.

## 2. Desensamblado del código

Para el desensamblado del código, se utilizó la herramienta `objdump` con distintos flags, para así poder ver las distintas características del ejecutable.

Como primera versión, se utilizó el siguiente comando: `objdump -d legado1 >legado1.asm`. Con la opción `-d`, únicamente se obtuvo el código desensamblado, por lo que se optó por realizar una segunda versión, en la que se incluían más datos sobre el ejecutable.

Para la segunda versión, se utilizó el comando `objdump -f -s -d --source legado1 >legado1v2.asm` y la funcionalidad de cada una de las flags es:

- **-f**                    opción para obtener el contenido de las cabeceras
- **-s**                    opción para mostrar todos los contenidos del resto de las secciones
- **-d**                    opción para obtener el código ensamblador de las secciones ejecutables
- **- -source**            opción para mostrar tanto el código fuente como el desensamblado

```

1  legado1:      file format elf32-i386
2
3
4
5  Disassembly of section .init:
6
7  00000584 <.init>:
8  584: 53                push    %ebx
9  585: 83 ec 08          sub     $0x8,%esp
10 588: e8 83 01 00 00    call   710 <_gmon_start__@plt+0x48>
11 58d: 81 c3 13 2a 00 00 add     $0x2a13,%ebx
12 593: 8b 83 54 00 00 00 mov     0x54(%ebx),%eax
13 599: 85 c9             test    %eax,%eax
14 59b: 74 05             je      5a2 <strsr@plt+0x1e>
15 59d: e8 26 01 00 00    call   6c8 <_gmon_start__@plt>
16 5a2: 83 c4 08          add     $0x8,%esp
17 5a5: 5b                pop     %ebx
18 5a6: c3                ret

```

Figura 1: Cabecera del código en la primera versión.

```

1  legado1:      file format elf32-i386
2
3  architecture: i386, flags 0x00000150:
4  HAS_SYMS, DYNAMIC, D_PAGED
5  start address 0x000006d0
6
7  Contents of section .interp:
8  0154 2f6c962 2f6c642d 6c696e75 782e736f /lib/ld-linux.so
9  0164 2e3200                                .2.
10 Contents of section .note.ABI-tag:
11 0168 04000000 10000000 01000000 474e5500 .....GNU.
12 0178 00000000 03000000 02000000 00000000 .....
13 Contents of section .note.gnu.build-id:
14 0188 04000000 14000000 03000000 474e5500 .....GNU.
15 0198 2d4492df a6c74203 6cc6b1fe 9f3ea83d -D....B.l....>.=
16 01a8 7ea80b19
17 Contents of section .gnu.hash:
18 01ac 02000000 15000000 01000000 05000000 .....

```

Figura 2: Cabecera del código en la segunda versión

### 3. Características del programa original

Una vez desensamblado el ejecutable, se han podido observar algunas de las características del ejecutable.

```
legado1:      file format elf32-i386
architecture: i386, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x000006d0
```

Figura 3: Cabecera del código desensamblado.

Según la cabecera del código ensamblador, el formato del ejecutable es elf32-i386, compilado para arquitectura i386.

El código 'elf32-i386' indica que el fichero es un *Executable and Linkable Format* de 32 bits para arquitectura i386.

Ahora se conoce el tipo de ejecutable y la arquitectura para la cual está compilado.

### 4. Ejecución 32-bit libc6-i386

Como se ha mencionado en el apartado 1, las máquinas empleadas para la realización de la práctica tenían sistemas operativos Ubuntu y Linux Debian.

Al intentar ejecutar el programa en cada una de las máquinas, se obtuvieron resultados diferentes. En la máquina con sistema operativo Ubuntu, el programa se consiguió ejecutar, llegando a observar en la pantalla la primera barrera del programa. Sin embargo, en el caso de la máquina Debian, la aplicación no se consiguió ejecutar, obteniendo el siguiente mensaje de la figura 4.

```
rick@rick-ZenBook-UX431FLC-UX431FL:~/Desktop/P1_Sistemas_Legados/Practica2$ ./legado1
bash: ./legado1: No such file or directory
rick@rick-ZenBook-UX431FLC-UX431FL:~/Desktop/P1_Sistemas_Legados/Practica2$
```

Figura 4: Ejecución sin la librería instalada.

Además, en el apartado 3 se ha descubierto que el fichero ejecutable es un fichero de 32 bits.

Según la comunidad de programadores ([2], [3]), para realizar la ejecución de un programa de 32bits en un sistema operativo de 64bits, se necesita una librería que contiene las directrices necesarias para llevar a cabo la ejecución.

En el caso de esta práctica, se ha hallado que el programa que se desea crackear es un programa ensamblado para la arquitectura i386, se ha considerado que solamente la librería específica para esta arquitectura será necesaria para la ejecución. Esta librería fue necesaria instalarla en la máquina Debian para conseguir la ejecución del programa.

```
rick@rick-ZenBook-UX431FLC-UX431FL:~/Desktop/P1_Sistemas_Legados/Practica2$ sudo apt install libc6-i386
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  libc6-i386
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 2.723 kB of archives.
After this operation, 14,7 MB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 libc6-i386 amd64 2.31-0ubuntu9.2 [2.723 kB]
Fetched 2.723 kB in 1s (4.952 kB/s)
Selecting previously unselected package libc6-i386.
(Reading database ... 363015 files and directories currently installed.)
Preparing to unpack .../libc6-i386_2.31-0ubuntu9.2_amd64.deb ...
Unpacking libc6-i386 (2.31-0ubuntu9.2) ...
Setting up libc6-i386 (2.31-0ubuntu9.2) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
rick@rick-ZenBook-UX431FLC-UX431FL:~/Desktop/P1_Sistemas_Legados/Practica2$
```

Figura 5: Instalación de la librería `libc6-i386` en Linux Debian.

Por último, se ha comprobado que la ejecución es posible ahora que la librería necesaria ha sido instalada.

```
rick@rick-ZenBook-UX431FLC-UX431FL:~/Desktop/P1_Sistemas_Legados/Practica2$ ./legado1
Stocks
Error acceso llave de protección
rick@rick-ZenBook-UX431FLC-UX431FL:~/Desktop/P1_Sistemas_Legados/Practica2$
```

Figura 6: Ejecución del programa con todas las librerías necesarias instaladas.

## 5. Salto de la primera barrera

Una vez instalada la librería correspondiente (apartado 4), se ha procedido a la ejecución del programa. Sin embargo, durante la ejecución se ha obtenido un error de permisos de ejecución.

```
alberto@alberto-X99 ~/D/G/a/a/S/Practica2 (main)> ./legado1
Stocks
Error de permisos de ejecución
```

Figura 7: Error primera barrera.

La solución propuesta es asignar los permisos al fichero con el comando `sudo chmod 777 legado1`.

Al volver a ejecutar el ejecutable, el error de la Figura (figura 7) seguía ocurriendo. Por tanto, la primera solución no es suficiente. La segunda solución propuesta a la primera barrera es activar el sticky bit, además de cambiar los permisos de ejecución, empleando, para ello, el comando `sudo chmod 7777 legado1`.

Por último, se ha ejecutado el comando `ls -l` sobre cada uno de los ficheros, para asegurarse que el programa y todos los ficheros necesarios para que su ejecución tuviese activo el sticky bit (opción `-t`).

```
-rwxrwxrwx 1 alberto alberto 11936 nov 26 12:52 legado1* -rwxrwxrwt 1 alberto alberto 11936 nov 26 12:43 legado1*
```

Figura 8: Código sin la opción del sticky bit activa y todos permisos (777). Figura 9: Código con la opción del sticky bit activa y resto de permisos (7777).

Una vez cambiados los permisos y el sticky bit del fichero, el error durante la ejecución cambió al de la figura 10, por lo que se pudo confirmar que la primera barrera se había pasado.

```
alberto@alberto-X99 ~/D/G/a/a/S/Practica2 (main)> ./legadol  
Stocks  
Error acceso llave de protección
```

Figura 10: Segunda barrera a pasar.

## 6. Salto de la segunda barrera

La segunda barrera que se ha tenido que saltar para conseguir el funcionamiento de la aplicación ha sido la barrera del error de acceso de la llave de protección (figura 10). Esta barrera ocurre debido a que el usuario que está intentando ejecutar el programa original no tiene los permisos para realizar determinadas llamadas al sistema, como podría ser el acceso a la llave de protección.

En este caso, el problema fue que se utilizaba la llamada al sistema `ioperm` (llamada que establece los permisos en los puertos de entrada/salida), por lo que su empleo, requería permisos de super-usuario.

La solución propuesta para saltar la barrera de error de acceso a la llave de protección es cambiar el usuario propietario del fichero ejecutable del programa a un usuario que pueda realizar llamadas al sistema o un super-usuario.

En este caso, se ha elegido cambiar el usuario propietario original del fichero ejecutable al super-usuario `root` mediante el comando `sudo chown root legadol`. Este comando cambiará el usuario propietario del fichero y permitirá saltar la segunda barrera del programa, pudiendo realizar el ejecutable de esta forma la llamada al sistema `ioperm`.

```
alberto@alberto-X99 ~/D/G/a/a/S/Practica2 (main)> ./legadol  
Stocks  
Error acceso llave de protección
```

Figura 11: Mensaje de error al ejecutar sin cambiar el usuario a super-usuario.

```
alberto@alberto-AORUS-15-SA ~/D/G/a/S/Practica2 (main)> ./legadol  
Stocks  
Llave protección no encontrada
```

Figura 12: Mensaje de error al ejecutar después de cambiar el usuario a super-usuario.

Con esto se comprueba que se ha conseguido saltar la segunda barrera del programa y que la tercera barrera es la que impide ahora ejecutar el programa.

## 7. Script para saltar las barreras

Para facilitar la ejecución y salto de las dos primeras barreras, se ha decidido implementar en `bash` un script corto que ejecute los dos comandos necesarios para cambiar el usuario propietario y los permisos de ejecución del programa.

La necesidad de implementar un script ha surgido porque se halló que los comandos deben ser ejecutados en un orden particular para conseguir saltar las dos

barreras.

Primero se debe cambiar el usuario propietario a root y, posteriormente, se deben cambiar los permisos de ejecución.

```
#!/usr/bin/bash
```

```
sudo chown root legado1
```

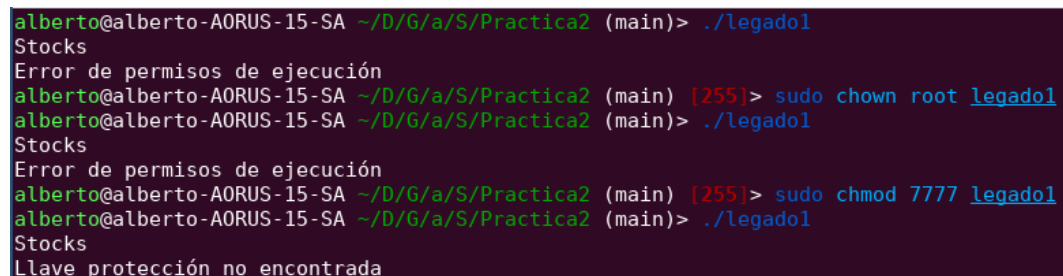
```
sudo chmod 7777 legado1
```

Código fuente 1: Script implementado para el salto automático de las barreras 1 y 2.

Este script se debe ejecutar una única vez antes de la ejecución inicial del programa, ya que una vez ejecutado el script el programa ya tendrá todas las modificaciones necesarias implementadas para su funcionamiento.

## 8. Problema de ejecución: Debian vs Ubuntu

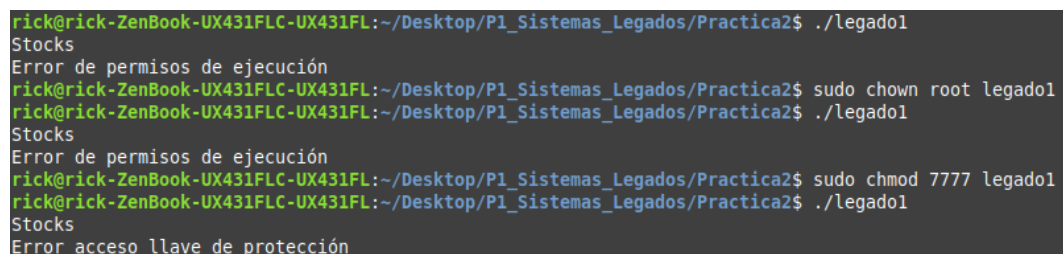
Durante la realización de la práctica, se ha observado un comportamiento peculiar del programa dependiendo si se intentaba ejecutar el programa en Ubuntu o en Debian. Al intentar saltar las barreras 1 y 2 en los dos sistemas operativos, el comportamiento del programa cambiaba. En el primer sistema operativo (Ubuntu), empleando las dos soluciones propuestas para el salto de las barreras 1 y 2, se conseguía llegar a ejecutar el programa para mostrar de forma correcta el mensaje de la barrera 3.



```
alberto@alberto-AORUS-15-SA ~/D/G/a/S/Practica2 (main)> ./legado1
Stocks
Error de permisos de ejecución
alberto@alberto-AORUS-15-SA ~/D/G/a/S/Practica2 (main) [255]> sudo chown root legado1
alberto@alberto-AORUS-15-SA ~/D/G/a/S/Practica2 (main)> ./legado1
Stocks
Error de permisos de ejecución
alberto@alberto-AORUS-15-SA ~/D/G/a/S/Practica2 (main) [255]> sudo chmod 7777 legado1
alberto@alberto-AORUS-15-SA ~/D/G/a/S/Practica2 (main)> ./legado1
Stocks
Llave protección no encontrada
```

Figura 13: Ejecución correcta en sistema operativo Ubuntu.

Sin embargo, en el sistema operativo Debian, las soluciones propuestas no permiten ejecutar el programa debido a que no se consigue saltar las barreras 1 y 2.



```
rick@rick-ZenBook-UX431FLC-UX431FL:~/Desktop/P1_Sistemas_Legados/Practica2$ ./legado1
Stocks
Error de permisos de ejecución
rick@rick-ZenBook-UX431FLC-UX431FL:~/Desktop/P1_Sistemas_Legados/Practica2$ sudo chown root legado1
rick@rick-ZenBook-UX431FLC-UX431FL:~/Desktop/P1_Sistemas_Legados/Practica2$ ./legado1
Stocks
Error de permisos de ejecución
rick@rick-ZenBook-UX431FLC-UX431FL:~/Desktop/P1_Sistemas_Legados/Practica2$ sudo chmod 7777 legado1
rick@rick-ZenBook-UX431FLC-UX431FL:~/Desktop/P1_Sistemas_Legados/Practica2$ ./legado1
Stocks
Error acceso llave de protección
```

Figura 14: Ejecución estancada en sistema operativo Debian.

En Ubuntu (figura 13) después de ejecutar los comandos de las dos soluciones propuestas, se consigue llegar a la tercera barrera durante la ejecución del programa.

Sin embargo, en Debian (figura 14) no se consigue llegar a la tercera barrera, siendo imposible saltar la segunda.

Una posible solución sería modificar el código máquina de la zona del código donde se está produciendo la comprobación del acceso a la llave de protección para saltar la barrera directamente del código máquina.

Sin embargo, los intentos realizados para conseguir este objetivo han producido numerosos errores de fallo de segmentación, por lo que, últimamente, se han abandonado los intentos de ejecución del programa en la máquina Debian.

## 9. Salto de la tercera barrera

Para el salto de la tercera barrera, se comenzó estudiando el código para saber donde se obtenía la llave de protección, debido a que el error que se mostraba era el siguiente (figura 15).

```
alberto@alberto-AORUS-15-SA ~/D/G/a/S/Practica2 (main)> ./legado2
Stocks
Llave protección no encontrada
```

Figura 15: Tercera barrera

Para entender el código, se utilizó la herramienta IDA, debido a que es una herramienta que ayuda a tener una mejor visualización del código en ensamblador, al separarlo en las distintas funciones, unirlos con flechas para ver como funciona el flujo de ejecución o incluso un visor hexadecimal ya incorporado (figura 16).

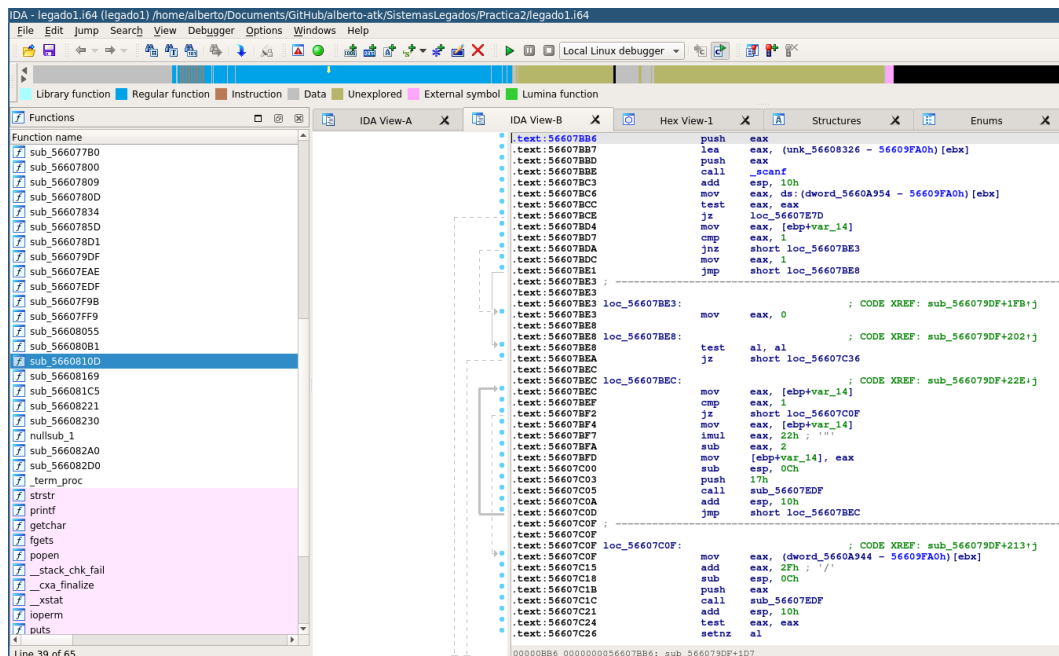


Figura 16: Código desensamblado en IDA

Para identificar la zona del código máquina en la que ocurre la lectura de la llave de protección se estudió el código en busca de la llamada a la función de salida del programa después de la escritura en pantalla del mensaje de error correspondiente. Según el código desensamblado, la función de salida del programa se llama



*exit* y tiene una dirección de memoria 0x670. Esto se ha observado en la sección de declaraciones del código desensamblado, donde la función *exit* aparecía declarada como: 00000670 <exit@plt>:.

El siguiente paso fue el estudio del código para identificar la zona en la que se llamaba a la función *exit*. Esto se realizó cambiando la instrucción del código máquina de la función *print* por una operación *nop*, ejecutando seguidamente el código para observar si el mensaje de error ya no se escribía en pantalla.

Durante el proceso de conseguir saltar la barrera 2 de la aplicación, se encontró en el código desensamblado, la función *ioperm*. Esta función se encargaba de la lectura de los permisos del ejecutable. Por tanto, la zona de búsqueda de la tercera barrera se determinó como la zona comprendida entre la llamada a la función *ioperm* y el final del código.

Cuando este mensaje de error ya no aparecía, se halló la zona de memoria de la entrada de la llave de acceso de forma manual mediante prueba y error.

Una vez se identificó la instrucción que generaba el mensaje de error, se detectó el salto que mostraba ese mensaje, por lo que se procedió a cambiar la condición. Con este cambio, se producía un bucle infinito en el que siempre se mostraba la primera opción, es decir, independientemente del número que se introdujese (formando parte del menú o no), se mostraba la primera opción y no se podía salir de la ejecución, a menos que se mandase una señal de finalización (ctrl + c).

```
aba: 74 1e je ada <__gmon_start__@plt+0x412>
```

Figura 17: Instrucción que salta si existe la llave

Posteriormente, con el uso de IDA y Visual Studio Code, se identificó que la única vez que se accedía a la llave de protección, era en una única función (con el uso de la instrucción *in*). Esta función, se llamaba una sola vez por otra función, por lo que hay una función intermediaria entre el flujo de ejecución principal del programa y el acceso a la llave de protección. La función en la que se accedía a la llave de protección es la siguiente:

```
80d: 55 push %ebp
80e: 89 e5 mov %esp,%ebp
810: 83 ec 14 sub $0x14,%esp
813: e8 09 0a 00 00 call 1221 <__gmon_start__@plt+0xb59>
818: 05 88 27 00 00 add $0x2788,%eax
81d: 8b 45 08 mov 0x8(%ebp),%eax
820: 66 89 45 ec mov %ax,-0x14(%ebp)
824: 0f b7 45 ec movzwl -0x14(%ebp),%eax
828: 89 c2 mov %eax,%edx
82a: ec in (%dx),%al
82b: 88 45 ff mov %al,-0x1(%ebp)
82e: 0f b6 45 ff movzbl -0x1(%ebp),%eax
832: c9 leave
833: c3 ret
```

Figura 18: Función de acceso a la llave de ejecución

Como primera solución, se trató de cambiar la llave de acceso devolviendo un uno al pensar que se trataba de una función booleana (comprobaba si la llave de acceso estaba, y en caso de acierto devolvía 1 (figura 26).

```

80d: 55          push    %ebp
80e: 89 e5       mov     %esp,%ebp
810: 83 ec 14    sub     $0x14,%esp
813: e8 09 0a 00 00 call    1221 <__gmon_start__@plt+0xb59>
818: 05 88 27 00 00 add     $0x2788,%eax
81d: 8b 45 08     mov     0x8(%ebp),%eax
820: 66 89 45 ec  mov     %ax,-0x14(%ebp)
824: 0f b7 45 ec  movzwl  -0x14(%ebp),%eax
828: 89 c2       mov     %eax,%edx
82a: ec         in     (%dx),%al
82b: 88 45 ff     mov     %al,-0x1(%ebp)
82e: c6 45 ff 01  movb    $0x1,-0x1(%ebp)
832: c9         leave
833: c3         ret

```

Figura 19: Cambios en la función in

Para comprobar que realmente la función devolvía uno, se cambió la condición previa al salto comentado anteriormente (condición en la que se evaluaba la llave de acceso), cambiando la comparación (test) entre dos registros iguales (%al) por una comparación entre el registro y un inmediato. Con esto,

```

ab8: a8 00      test    $0x1,%al
aba: 74 1e      je     ada <__gmon_start__@plt+0x412>

```

Figura 20: Comprobación de existencia de la llave de acceso

```

ab8: 84 c0      test    %al,%al
aba: 74 1e      je     ada <__gmon_start__@plt+0x412>

```

Figura 21: Comprobación original

Al suceder esto, se observó que seguía sin funcionar, debido a que volvía a producirse el error de la tercera barrera (figura 15), por lo que se decidió buscar la instrucción que producía el bucle infinito, es decir, constantemente mostraba la primera opción y hacía volver al inicio. Para ello, se cambió la condición de test para comprobar con un 0 (no realizándose el salto). Después de analizar el código en detalle, se llegó a la conclusión que la línea que llamaba a la función que producía este error era la siguiente:

```

bce: 0f 84 a9 02 00 00 je     e7d <__gmon_start__@plt+0x7b5>

```

Figura 22: Instrucción original que genera el bucle infinito

```

bce: 0f 85 a9 02 00 00 jne    e7d <__gmon_start__@plt+0x7b5>

```

Figura 23: Cambio de la condición de la instrucción para no generar bucle infinito

Cuando se consiguió esto, la funcionalidad del programa era mínima, ya que algunas funcionaban correctamente y otras daban error en la llave de acceso. Para solucionarlo, se procedió a modificar cada una de las opciones del menú por separado de la siguiente forma:

- Con la herramienta find de Visual Studio Code, se identificaba la comparación de cada uno de los números con el registro %eax, para así ver el fragmento de código sobre el que se realizaban las pruebas de los saltos.
- Una vez encontrado, se modificaban las instrucciones de salto invirtiendo los condicionales
- Después de esto, se seguía un patrón en todas las opciones, debido a que se llamaba a la función 0xedf, la cual llamaba a la 0x80d (comprobación de la llave de acceso), y en las líneas posteriores, se comprobaba si estaba la llave de acceso, por lo que con invertir el salto posterior a la comprobación, se solucionaban la mayoría de las opciones.

Estas fotos a continuación, están asociadas al proceso de cambio para el correcto funcionamiento de una opción:

```
ce0: 8b 45 ec      mov     -0x14(%ebp),%eax
ce3: 83 f8 03      cmp     $0x3,%eax
ce6: 75 3c         jne     d24 <__gmon_start__@plt+0x65c>
```

Figura 24: Comprobación de la opción del menú

En esta primera figura, se produce una comparación con el inmediato tres para saber la opción, en caso de que sean iguales se continuará, y en el caso contrario, se realizará un salto a una instrucción similar a la 0xce0, pero situada en la 0xd24 para realizar el mismo proceso pero con la opción 4.

```
d05: e8 d5 01 00 00 1 call    edf <__gmon_start__@plt+0x817>
d0a: 83 c4 10      add     $0x10,%esp
d0d: 85 c0         test    %eax,%eax
d0f: 0f 95 c0      setne   %al
d12: 84 c0         2 test    %al,%al
d14: 0f 84 44 01 00 00 3 je      e5e <__gmon_start__@plt+0x796>
d1a: e8 92 03 00 00 4 call    10b1 <__gmon_start__@plt+0x9e9>
d1f: e9 b7 fd ff ff 4 jmp     adb <__gmon_start__@plt+0x413>
```

Figura 25: Ejecución de una opción

En la segunda imagen, en el número 1, se ve la llamada a la función de la dirección 0xedf para obtener la llave, y posteriormente, se comprueba en el punto 2, por lo que la instrucción a cambiar, en este caso sería la 0xd14, invirtiendo el salto al introducir un cambio de je por jne (cambio del 84 por un 85). Finalmente, en el punto 4, se retornará al menú de inicio para esperar a que se introduzca otra opción.

```
f4b: 50           push    %eax
f4c: e8 bc f8 ff ff call    80d <__gmon_start__@plt+0x145>
f51: 83 c4 10      add     $0x10,%esp
```

Figura 26: Llamada a la función que comprueba de la llave de acceso desde 0xedf

## Referencias

- [1] *80386 Programmer's reference manual 1986*. Intel Corporation, 1988.
- [2] Francesco, *No such file or directory? but the file exists!* Mayo de 2012. dirección: <https://askubuntu.com/questions/133389/no-such-file-or-directory-but-the-file-exists>.
- [3] T. Stuefe, *How to run 32-bit app in ubuntu 64-bit?* Abr. de 2014. dirección: <https://askubuntu.com/questions/454253/how-to-run-32-bit-app-in-ubuntu-64-bit>.