

Práctica 2

SISTEMAS LEGADOS



[Fecha]

[Nombre de la compañía]

[Dirección de la compañía]



# Resumen ejecutivo

El problema al que nos enfrentamos es un programa, el cual hemos perdido la llave de acceso, por lo que no podemos acceder a él. Para poder acceder al programa, debemos eliminar esos accesos a las llaves.

A continuación, se explica como hemos crackeado un programa desensamblándolo y viendo su estructura en lenguaje ensamblador, para poder identificar las líneas de código que debemos modificar para saltar el acceso a la llave. Para conseguir el objetivo hemos realizado una serie de pruebas como se ve a continuación.

Contenido

[Resumen ejecutivo 1](#_Toc89862619)

[Introducción 1](#_Toc89862620)

[Objetivos 1](#_Toc89862621)

[Desemsamblado 1](#_Toc89862622)

[Magic number 2](#_Toc89862623)

[Primer error 2](#_Toc89862624)

[Prueba 1 2](#_Toc89862625)

[Prueba 2 3](#_Toc89862626)

[Prueba 3 3](#_Toc89862627)



# Introducción

Para crakear un programa debimos saber en qué sistema operativo y con que lenguaje de programación había sido compilado. Una vez desensamblado tuvimos que estudiar el código y mediante un editor hexadecimal, ir probando cambios en el código y comprobando si se podía acceder al programa. Estas pruebas se muestran a continuación.

# Objetivos

* Aprender como desensamblar un programa del cual sólo tenemos el ejecutable.
* Conocer el magic number de un ejecutable.
* Trabajar con un editor hexadecimal para modificar código máquina.

# Desemsamblado

El primer paso es identificar en que lenguaje de programación y en que SO se ha compilado para poder desensamblarlo.

Para conocer el sistema operativo bastó con abrir el ejecutable desde un editor hexadecimal y fijarnos en el texto decodificado que aparece en la parte derecha:

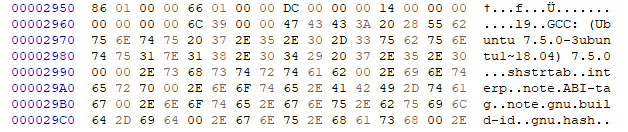


Ilustración : Ejecutable abierto desde un editor hexadecimal

Como se puede observar, el ejecutable se compiló en Ubuntu 18.04

FALTA DECIR COMO SUPIMOS EN QUE LENGUAJE DE PROGRAMACIÓN SE COMPILO Y EN QUE ARQUITECTURA DE SISTEMA OPERATIVO (32 BITS)

Para desensamblarlo usamos el siguiente comando:

Objdump -M i386 -drC legado5

# Magic number

# Primer error

Para poder ejecutar el programa debemos cambiar los permisos de ejecución. Para ello ejecutamos el comando: sudo chmod 777 nombreArchivo



Ilustración : Comando para dar todos los permisos al ejecutable

Después de dar los permisos, debemos ejecutar el siguiente comando para cambiar el usuario:



Ilustración : Comando para cambiar el usuario en un sistema UNIX

Una vez ejecutado estos dos comandos, para ejecutar el programa debemos hacer lo siguiente:



Ilustración : Ejecución programa

Una vez sabemos como ejecutar el programa, es hora de realizar pruebas para conseguir crakear el programa.

# Prueba 1

Buscando en Google acerca del puerto LPT1 vimos que la dirección de inicio de dicho puerto es la 0x378 como se puede comprobar en la siguiente tabla:

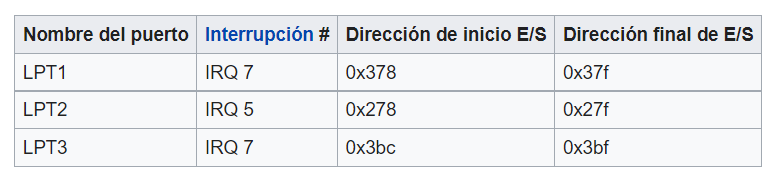


Ilustración 5: Direcciones base de los puertos paralelos

Buscamos dentro del desensamblado este número y encontramos 9 referencias. Por lo tanto, hemos decidido cambiar esas instrucciones por instrucciones nop.



Ilustración 6: Instrucciones nop

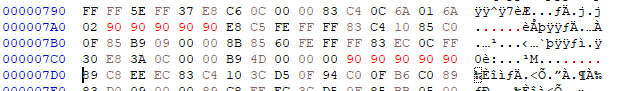


Ilustración 7: Código máquina

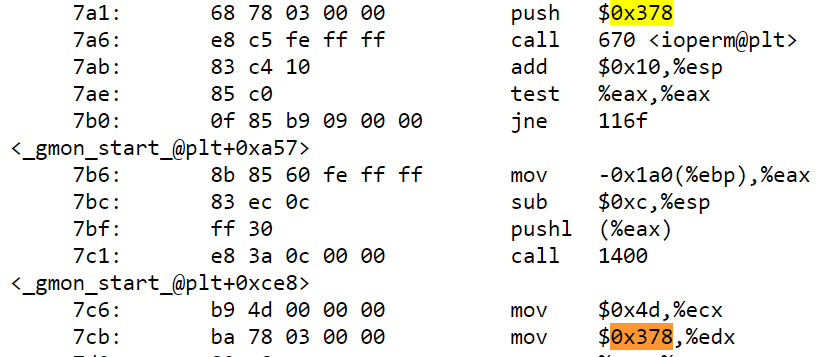


Ilustración 8: Código desensamblado

Con esta prueba nos salía el siguiente error:

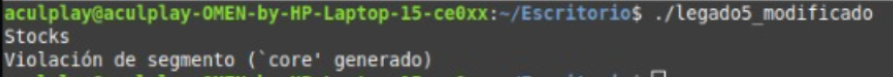


Ilustración : Error programa

Esto quiere decir que alguna variable o puntero accede a una posición que no tiene permitida.

Por consiguiente, esta prueba ha resultado fallida.

# Prueba 2

Comprobamos que la dirección “0x378” se usa una vez para apilar en la pila y 8 para usar en variables. En la prueba anterior quitábamos todas las operaciones con nops, incluyendo la instrucción que almacena en la pila. Es por eso que se descolocaba todo el programa ya que se apilaba una variable menos.

En esta segunda prueba hemos probado apilar un valor aleatorio (un 0) en la pila y quitar las otras 8 instrucciones con nops. Al ejecutarlo nos sale lo siguiente:

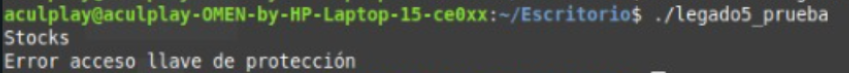


Ilustración : Error programa

# Prueba 3

En esta prueba vamos a comprobar las instrucciones siguientes a la aparición de la constante 0x378.

Estudiando las instrucciones vimos dos instrucciones que nos llamaron la atención:



Ilustración :Instrucciones entrada-salida

* **Instrucción IN (Entrada desde el puerto)**: Transfiere información desde un puerto de entrada a un registro.

Transfiere un byte, palabra o palabra doble desde una entrada puerto a AL, AX o EAX. Si un programa especifica AL con la instrucción IN, el procesador transfiere 8 bits del puerto seleccionado a AL.

* **Instrucción OUT (Salida a puerto):** Escribe información en un puerto de salida desde un registro.

Transfiere un byte, palabra o palabra doble a una salida puerto de AL, AX o EAX. El programa puede especificar el número de puerto.

Vimos que estas instrucciones siempre usan los mismos registros (%dx y %al), por lo que decidimos estudiar cada registro:

Los registros generales para entrada/salida son: EAX (32-bits), AX (16-bit) y AL (8-bit).

El registro DX especifica la dirección de un puerto en el espacio de direcciones de E / S (hasta 64K).

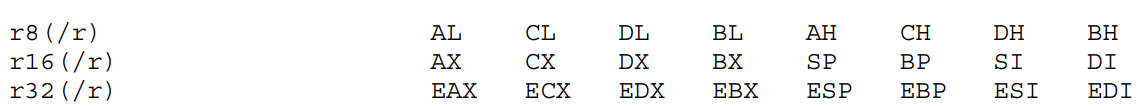


Ilustración 12: Todos los registros del sistema y sus tamaños

Investigando más vimos que los registros generales se pueden dividir en registros más pequeños como se ve en la siguiente ilustración:

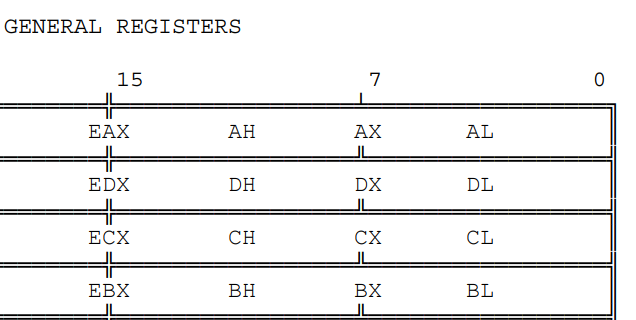


Ilustración : División de los registros generales

Por lo cual vimos que el registro EAX se puede dividir en AH, AX y AL.

# Bibliografía

1º Puerto paralelo from: <https://es.wikipedia.org/wiki/Puerto_paralelo>

2º Instrucciones ensamblador from: <http://atc2.aut.uah.es/~avicente/asignaturas/ec/pdf/ec_t4.pdf>

3º Intel 80386 programer reference manual from <https://css.csail.mit.edu/6.858/2013/readings/i386.pdf>