## Arquitecturas de las Computadoras

## TP 1 - Análisis de Binarios

#### Ejercicio 1

El programa **strings** de Linux, busca en los archivos strings, y hace una lista de éstos.

1. Corra el programa ***fortune***
2. Ejecute **strings** utilizando como argumento el programa ***fortune.*** Identifique todas las *fortunas* que dicho programa estaría indicando.

Break into jail and claim police brutality.

Never be led astray onto the path of virtue.

You will forget that you ever knew me.

Your society will be sought by people of taste and refinement.

You will be honored for contributing your time and skill to a worthy cause.

Expect the worst, it's the least you can do.

You may not get this fortune

1. ¿Cómo hace dicho programa para encontrar los strings? Implemente su propia versión.

El programa **strings** muestra como strings aquellas secuencias que contengan al menos 4 caracteres de longitud (o del número especificado en los parámetros de ejecución) y que vayan seguidas por algún carácter no imprimible. De forma predeterminada, sólo muestra las cadenas de las secciones inicializadas y cargadas de ficheros objeto; para otros tipos ficheros, muestra las cadenas de todo el fichero entero.

Una implementación posible sería, leer caracteres de a uno y si se detectan 4 o más caracteres válidos en ASCII, imprimirlos, siendo impresa su representación ASCII si es una letra, número, símbolo y un punto si es un ASCII no imprimible.

#### 

#### Ejercicio 2

Abra con el **Bless Hex Editor** y el programa ***fortune*** búsque los Strings que aparecen en el programa y cambielos.

1. Corrija los *horrores* de ortografía.

Los errores de ortografía se pueden corregir.

1. Haga más corto el mensaje de bienvenida.

No es posible borrar caracteres.

##### Ejercicio 3

La herramienta **objdump** de Linux permite hacer un análisis de archivos objeto revelando información importante de cómo está compuesto dicho archivo.

1. Haga un ***disassembly*** del código objeto y deduzca cómo es que se elige la fortuna a mostrarle al usuario. Si agrega el argumento “-M intel” podrá ver dicho código en formato Intel.
2. Investigue cuales son las secciones que tiene dicho archivo. ¿En qué sección del código se encuentran los Strings?

Con el flag -s, se puede ver el desensamblado de una manera más cómoda para buscar los Strings. Éstos se encuentran en la sección “.rodata”.

1. Identifique todas las etiquetas del archivo. ¿Cuáles reconoce?

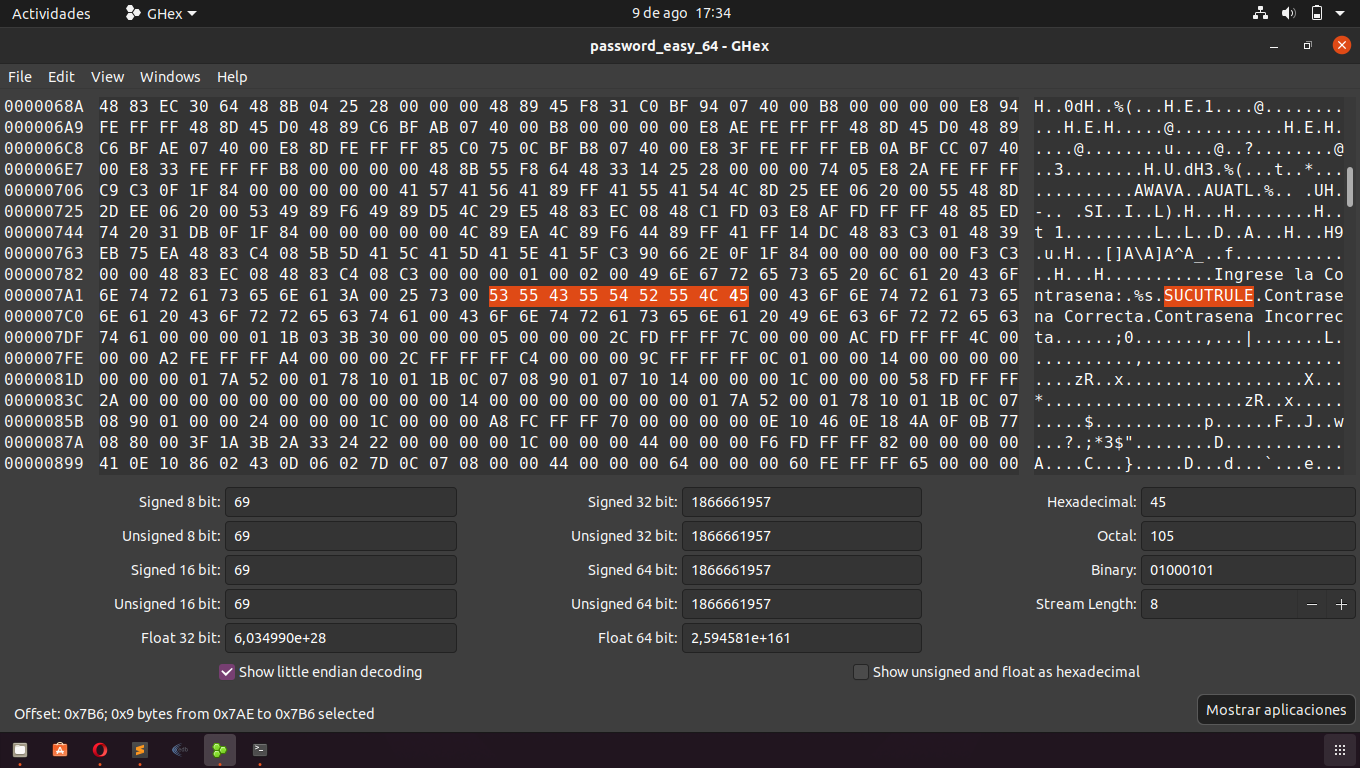
##### 

#### Ejercicio 4

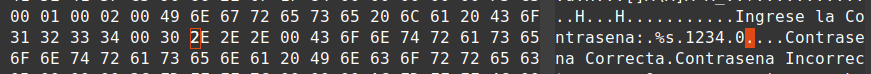
Con los conocimientos adquiridos en los puntos anteriores,

1. Obtenga la contraseña del programa ***password\_easy***.

La contraseña del programa es “SUCUTRULE”.



1. Cámbiela por “1234”.



Para finalizar la contraseña, se debe poner el ascii 00.

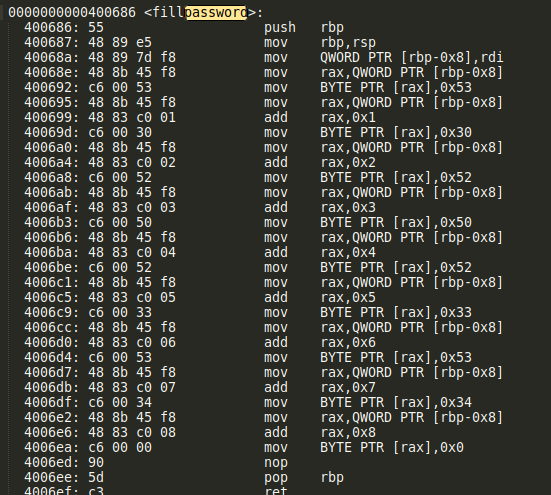
##### 

#### Ejercicio 5

Con los conocimientos adquiridos en los puntos anteriores,

1. Obtenga la contraseña del programa **password\_ofuscated1**

La contraseña la obtenemos haciendo el *disassemble* del programa.

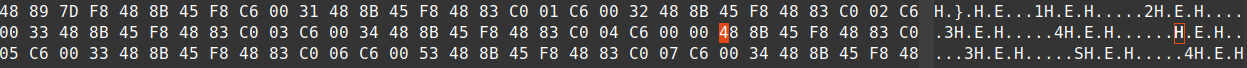


La contraseña en hexa es: *53 30 52 50 52 33 53 34*. Y el último carácter que es 0, indica el fin de la contraseña.

Pasando a letras es: S0RPR3S4.

1. Cámbiela por “1234”

Para cambiarla abrimos el programa en GHEX.



##### 

#### Ejercicio 6

Considere el siguiente programa:

|  |
| --- |
| **#include <stdio.h>**  **#define DIMENSION 1024**  **int** matriz[DIMENSION][DIMENSION];  **int** **main**(**void**) {   printf("Hello World!\n");  **return** 0; } |

Compile y analice el tamaño del binario final. Piense, ¿Cuánto espacio ocupa una matriz de 1024x1024 de *ints*? ¿Dónde está toda esa información en el binario? ¿Qué tamaño esperaba?

Una matriz de 1024x1024 bits ocupa…

Al compilar el programa, como no está inicializada la matriz, gcc no reserva espacio en disco. Recién se reserva el espacio en memoria cuando se va a ejecutar el programa.

¿Qué ocurriría si dicha matriz estuviera inicializada con 0’s? ¿Y con otro valor? Pruebe cómo varía el tamaño de dicho binario con las distintas alternativas.

|  |
| --- |
| **int** matriz[DIMENSION][DIMENSION] = {0}; |

Si dicha matriz está inicializada con 0’s, ocurre lo mismo, el programa reserva ese espacio en memoria recién cuando se va a ejecutar.

Si está inicializada con otro valor, gcc decide reservar directamente el espacio que ocupa la matriz en el disco, es por eso que el tamaño del ejecutable incrementa considerablemente.

##### 

##### Ejercicio 7

Utilice el programa **Evan’s Debugger** con el programa **fortune** de los puntos anteriores. Corra cualquier programa e identifique los siguientes elementos de cada programa:

1. Zona de Código
2. Zona de Datos
3. Stack
4. Conteste las siguientes preguntas:  
    ¿En qué lugar físico de la PC está la información que está visualizando?

La información visualizada se encuentra en memoria.  
 ¿Algunas secciones están solapadas? ¿Por qué en cada pantalla la información visualizada es distinta?

No están solapadas ni deberían estarlo, son conceptualmente diferentes: los registros no están en memoria, sino en el procesador. En cada pantalla la información es distinta porque son zonas de memoria distintas.   
 ¿Qué diferencia hay con los ejemplos anteriores? ¿En qué parte de la PC está la información que está analizando?

1. Confirme su suposición de cómo se está calculando la fortuna del usuario
2. Abra el Programa **password\_ofuscated1** y observe paso a paso cómo cambia la sección de datos a medida que se va generando la contraseña.
3. Modifique y guarde dicho programa de tal forma que sin importar la contraseña, siempre de cómo correcta.

##### 

##### Ejercicio 8

Utilice el **debugger** para deducir cómo son las contraseñas que son válidas para el programa **password\_hard.**