NULL Life 20 de Octubre de 2011

Soluciones a RIC Capture The Flag - HackXColombia

Reto 1

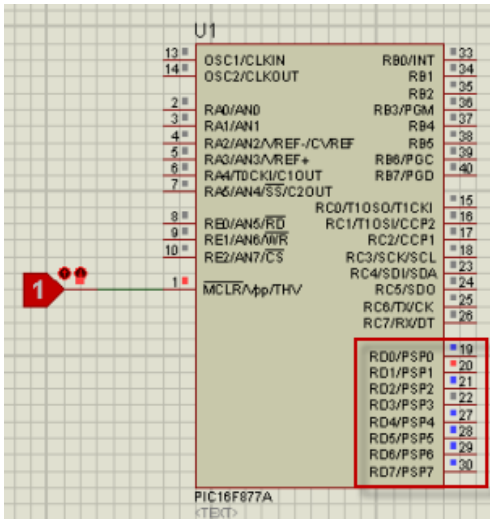
Descripción

Siempre copio de Internet mis tareas, pero esta ves no la encontré completa. Ayudame a completarla y dime que nos muestra ("un componente te mostrara la respuesta", eso me dijo mi amigo Zarek que siempre hace sus tareas).

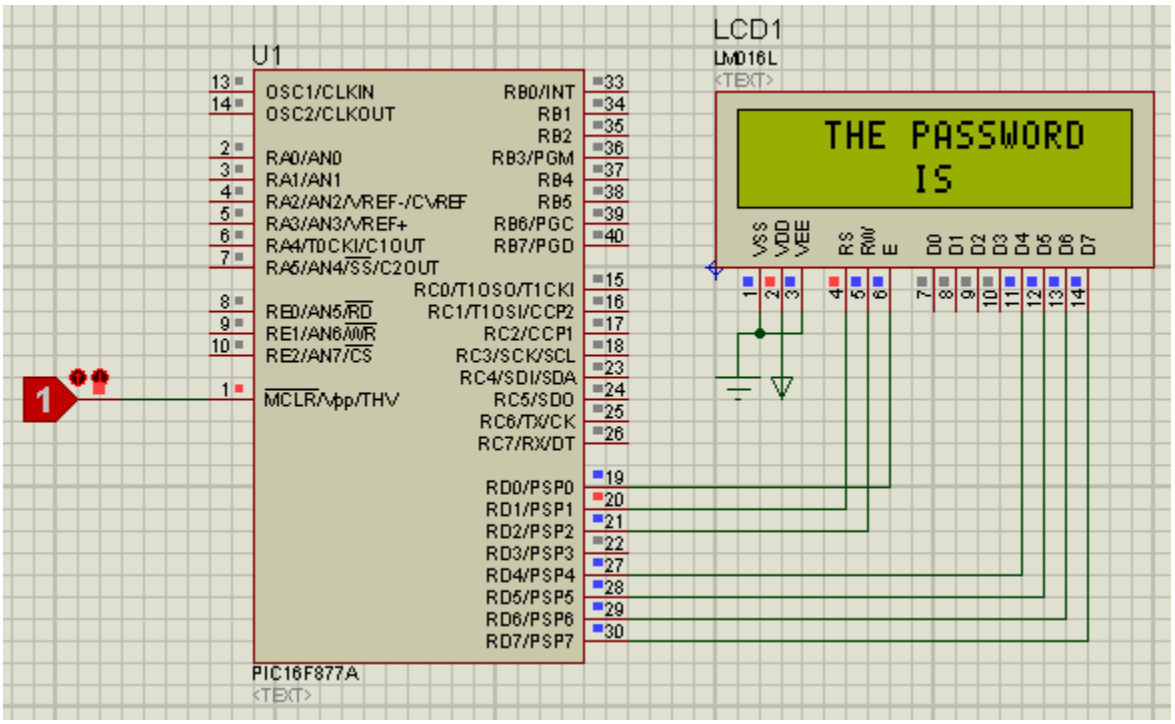
Solución

Al abrir el archivo DSN con Proteus nos muestra un PIC y un texto que dice “Algo nos mostrara la respuesta!!!”. Buscando la referencia del PIC (16F877A) en imágenes en google vemos que el componente faltante es un display.

Primero cargamos el programa que se encuentra en el archivo .hex, simulamos el circuito y vemos las salidas que producen señal.

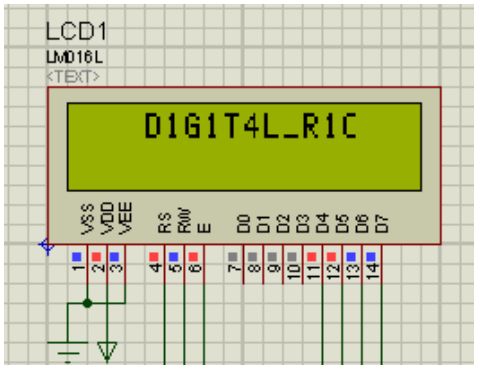


Estas salidas deben conectarse en algún lugar, agregando un display de 16 x 2 y probando combinaciones con las salidas 19, 20 y 21 obtenemos algo que parece ser casi el final del reto.



Ahora siguiendo la pista de Zarek que dio en el IRC: "**despues de que vean algo en el primer reto lo que deben hacer es imaginar que es una imagen!! .gif**"

Luego de observar un rato el display, se puede notar como hay un pequeño momento donde la imagen desaparece y vuelve a aparecer. Usando la herramienta de depuración de Proteus, podemos ir simulando paso a paso, luego de unos intentos obtenemos la clave.



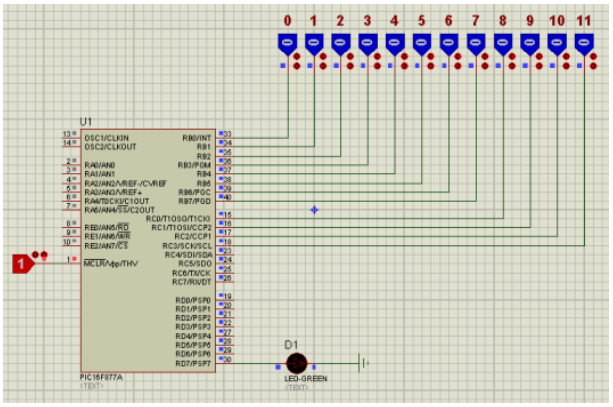
Respuesta
D1G1T4L_R1C

Reto 2

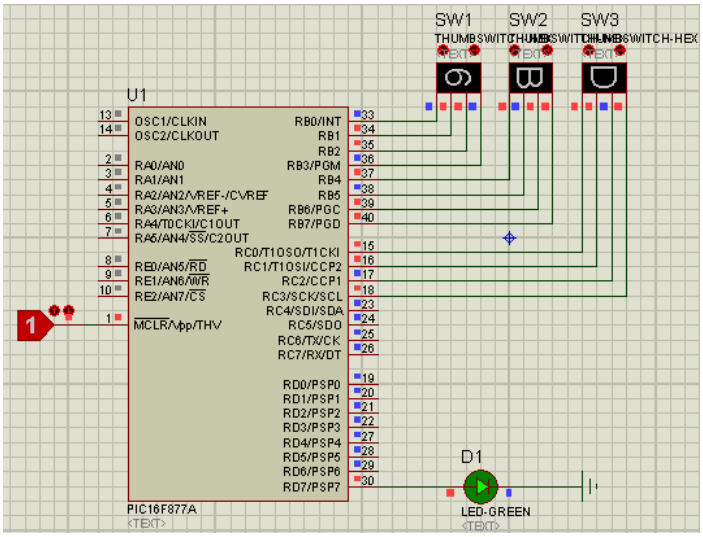
Descripción
Me haces un favor me ayudas a encender el LED, solo dime que combinación lo hace...

Solución

De nuevo nos encontramos con un reto sobre electrónica, cargamos el archivo y el programa correspondiente y observamos que lo que nos pide el autor es hacer un ataque de fuerza bruta a un binario de 12 bytes, lo que significa realizar 4096 combinaciones.



Simple, reemplazamos la entrada binaria por entradas hexadecimales y realizamos las combinaciones de una forma cómoda y rápida, para encontrar que la respuesta en hexadecimal es 6BD o su equivalente en binario 011010111101.



Respuesta

011010111101

Reto 4

Descripción

Pacman anda como raro, no lo notas? , es hora de que le ayudes a encontrar lo que anda buscando :_:

Solución

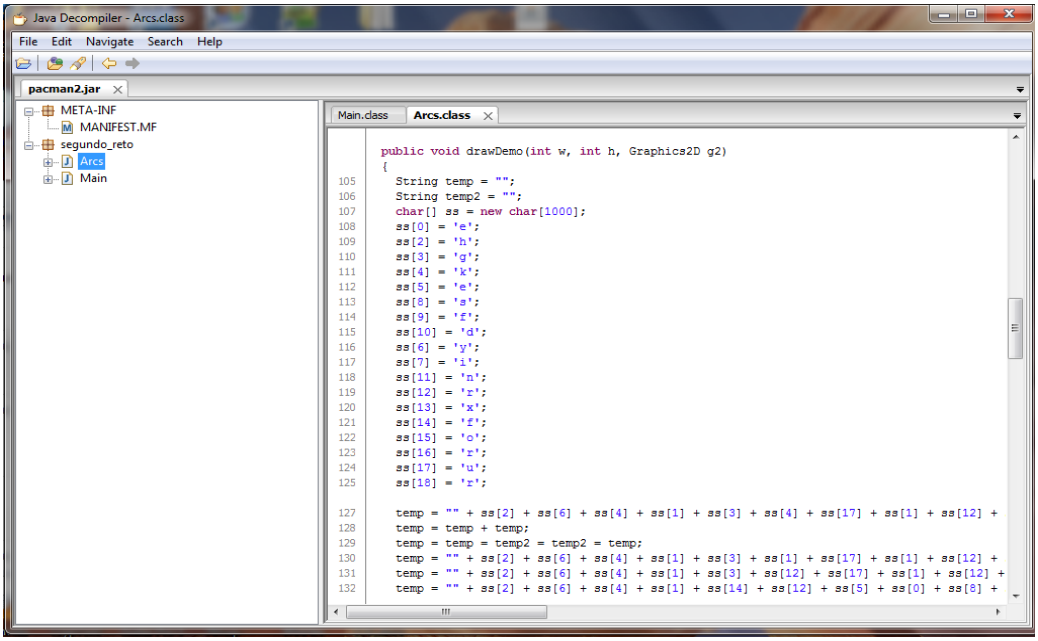
El reto nos brinda un archivo llamado pacman.jar el cual con la herramienta trid comprobamos que se trata de un “Java Archive”

```
TrID/32 - File Identifier v2.02 - (C) 2003-06 By M.Pontello
Definitions found:  4184
Analyzing...

Collecting data from file: pacman.jar
 78.3% (.JAR) Java Archive (14500/1/2)
 21.6% (.ZIP) ZIP compressed archive (4000/1)
  0.0% (.CEL) Autodesk FLIC Image File (extensions: flc, fli, cel) (7/3)
```

Ejecutamos el mismo para poder apreciar su funcionamiento

Decompilamos el archivo jar utilizando JD-GUI y obtenemos un código fuente entendible:

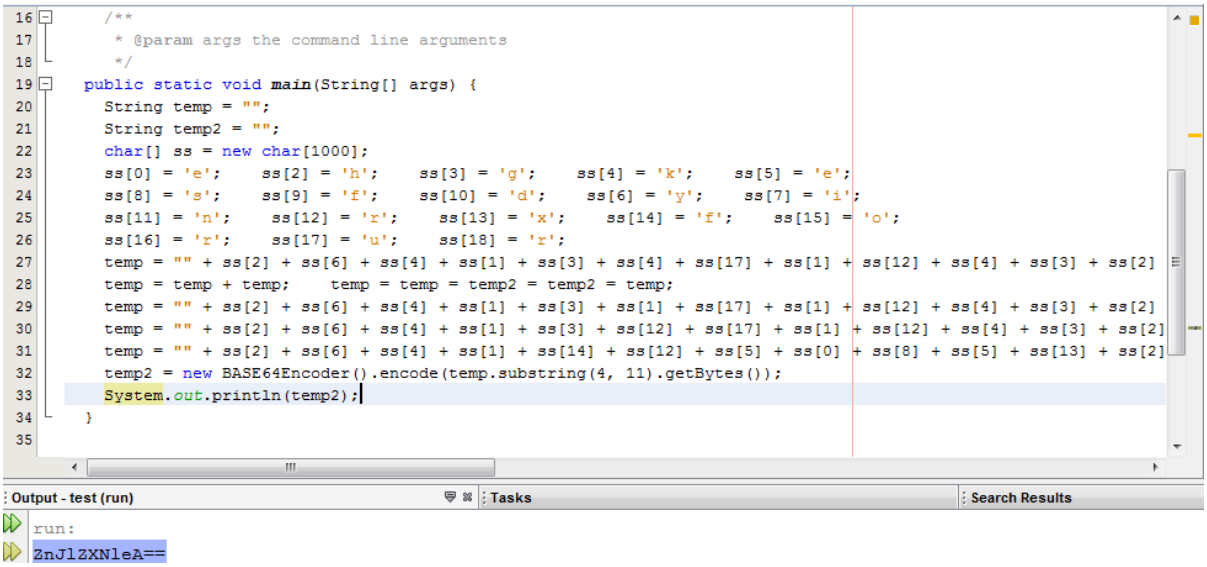


Dentro del código fuente vemos como interesante la función drawDemo la cual se encarga de dibujar las cadenas que aparecen al ejecutarlo, las lineas interesantes de esta función son las siguientes:

```
for (int i = Global.conteo_global; i < Global.conteo_global * 2; i++) {
   g2.setColor(colors[(i % 5)]);
   g2.drawString(" the key is :", (int)(i * Math.random()), (int)(i * Math.random()));
}

temp2 = new BASE64Encoder().encode(temp.substring(4, 11).getBytes());
g2.drawString(temp2, 999, 999);
```

Podemos apreciar que la variable temp2 tendra la flag del reto por lo tanto emulamos el codigo que generara el contenido de esta variable:



```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
      String temp = "";
      String temp2 = "";
      char[] ss = new char[1000];
      ss[0] = 'e';    ss[2] = 'h';    ss[3] = 'g';    ss[4] = 'k';    ss[5] = 'e';
      ss[8] = 's';    ss[9] = 'f';    ss[10] = 'd';    ss[6] = 'y';    ss[7] = 'i';
      ss[11] = 'n';    ss[12] = 'r';    ss[13] = 'x';    ss[14] = 'f';    ss[15] = 'o';
      ss[16] = 'r';    ss[17] = 'u';    ss[18] = 'r';
      temp = "" + ss[2] + ss[6] + ss[4] + ss[1] + ss[3] + ss[4] + ss[17] + ss[1] + ss[12] + ss[4] + ss[3] + ss[2]
      temp = temp + temp;    temp = temp = temp2 = temp2 = temp;
      temp = "" + ss[2] + ss[6] + ss[4] + ss[1] + ss[3] + ss[1] + ss[17] + ss[1] + ss[12] + ss[4] + ss[3] + ss[2]
      temp = "" + ss[2] + ss[6] + ss[4] + ss[1] + ss[3] + ss[12] + ss[17] + ss[1] + ss[12] + ss[4] + ss[3] + ss[2]
      temp = "" + ss[2] + ss[6] + ss[4] + ss[1] + ss[14] + ss[12] + ss[5] + ss[0] + ss[8] + ss[5] + ss[13] + ss[2]
      temp2 = new BASE64Encoder().encode(temp.substring(4, 11).getBytes());
      System.out.println(temp2);
    }
```

Output - test (run) | Tasks | Search Results

```
run:
ZnJlZXNleA==
```

Decodificando el base64 obtenemos nuestra flag.

Respuesta
freesex

Reto 5

Descripción

Inversame al hacerlo el token sera el password para el usuario Phicar

Solución

Primero analizamos el archivo, podemos apreciar que se trata de un ejecutable para los S.O. GNU/Linux de 64bits:

```
[guest@sinfocol ~]$ file RETO_5
RETO_5: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.15, not stripped
```

Ejecutamos el programa sin argumentos y nos muestra la forma de utilizarlo:

```
[guest@sinfocol ~]$ ./RETO_5
usage: rev [username] [password]
```

Utilizamos GDB, el depurador por excelencia para GNU/Linux, ejeuctamos

```
(gdb) r Phicar AAA
```

Desensamblamos la función principal y podemos apreciar algunas comparaciones interesantes:

```
0x0000000000400672 <+110>:   mov    %rax,%rdi
0x0000000000400675 <+113>:   callq  0x4007b5 <getTal>
0x000000000040067a <+118>:   mov    %rax,-0x28(%rbp)
0x000000000040067e <+122>:   movl   $0x0,-0x14(%rbp)
0x0000000000400685 <+129>:   movl   $0x0,-0x18(%rbp)
0x000000000040068c <+136>:   jmp    0x4006d2 <main+206>
      .
      .
      .
```

```
   0x00000000004006ce <+202>:   addl   $0x1,-0x18(%rbp)
   0x00000000004006d2 <+206>:   mov    -0x18(%rbp),%eax
   0x00000000004006d5 <+209>:   movslq %eax,%rbx
   0x00000000004006d8 <+212>:   mov    -0x2f20(%rbp),%rax
   0x00000000004006df <+219>:   add    $0x10,%rax
   0x00000000004006e3 <+223>:   mov    (%rax),%rax
   0x00000000004006e6 <+226>:   mov    %rax,%rdi//rax = 2arg
   0x00000000004006e9 <+229>:   callq  0x4004f8 <strlen@plt>
=> 0x00000000004006ee <+234>:   cmp    %rax,%rbx
   0x00000000004006f1 <+237>:   jb     0x40068e <main+138>
```

Aca apreciamos que %rax tendrá el valor del tamaño de la cadena del segundo argumento pasado al programa y %rbx es un contador por lo que vemos un bucle que itera a través del segundo argumento ejecutando el siguiente código:

```
       CODE A:
   0x000000000040068e <+138>:   mov    -0x2f20(%rbp),%rax
   0x0000000000400695 <+145>:   add    $0x10,%rax
   0x0000000000400699 <+149>:   mov    (%rax),%rdx
   0x000000000040069c <+152>:   mov    -0x18(%rbp),%eax
   0x000000000040069f <+155>:   cltq
   0x00000000004006a1 <+157>:   add    %rax,%rdx
   0x00000000004006a4 <+160>:   mov    -0x2f20(%rbp),%rax
   0x00000000004006ab <+167>:   add    $0x10,%rax
   0x00000000004006af <+171>:   mov    (%rax),%rcx
   0x00000000004006b2 <+174>:   mov    -0x18(%rbp),%eax
   0x00000000004006b5 <+177>:   cltq
   0x00000000004006b7 <+179>:   lea    (%rcx,%rax,1),%rax
   0x00000000004006bb <+183>:   movzbl (%rax),%ecx
   0x00000000004006be <+186>:   mov    -0x18(%rbp),%eax
   0x00000000004006c1 <+189>:   cltq
   0x00000000004006c3 <+191>:   mov    -0x2f10(%rbp,%rax,4),%eax
   0x00000000004006ca <+198>:   xor    %ecx,%eax
   0x00000000004006cc <+200>:   mov    %al,(%rdx)
   0x00000000004006ce <+202>:   addl   $0x1,-0x18(%rbp)
```

Acá apreciamos que toma un valor calculado anteriormente en la función **getTal** y le aplica una operación xor junto con el valor del cáracter actual del segundo argumento ingresado.

```
  0x000000000040072f <+299>:   addl   $0x1,-0x18(%rbp)
   0x0000000000400733 <+303>:   mov    -0x28(%rbp),%rax
   0x0000000000400737 <+307>:   mov    %rax,%rdi
=> 0x000000000040073a <+310>:   callq  0x4004f8 <strlen@plt>
   0x000000000040073f <+315>:   mov    %eax,%ebx
   0x0000000000400741 <+317>:   mov    -0x2f20(%rbp),%rax
   0x0000000000400748 <+324>:   add    $0x10,%rax
   0x000000000040074c <+328>:   mov    (%rax),%rax
   0x000000000040074f <+331>:   mov    %rax,%rdi
   0x0000000000400752 <+334>:   callq  0x4004f8 <strlen@plt>
   0x0000000000400757 <+339>:   mov    %ebx,%esi
   0x0000000000400759 <+341>:   mov    %eax,%edi
   0x000000000040075b <+343>:   callq  0x400a3e <min>
   0x0000000000400760 <+348>:   cmp    -0x18(%rbp),%eax
   0x0000000000400763 <+351>:   jg     0x4006fc <main+248>
```

Acá podemos apreciar que compara el tamaño del segundo argumento ingresado con otro

valor calculado anteriormente que para este usuario especifico ('Phicar') es 8, si estos valores son diferentes muestra un mensaje de error caso contrario continua con el siguiente código:

```
   0x00000000004006fc <+248>:   mov    -0x2f20(%rbp),%rax
   0x0000000000400703 <+255>:   add    $0x10,%rax
   0x0000000000400707 <+259>:   mov    (%rax),%rdx
   0x000000000040070a <+262>:   mov    -0x18(%rbp),%eax
   0x000000000040070d <+265>:   cltq
   0x000000000040070f <+267>:   lea    (%rdx,%rax,1),%rax
   0x0000000000400713 <+271>:   movzbl (%rax),%edx
   0x0000000000400716 <+274>:   mov    -0x18(%rbp),%eax
   0x0000000000400719 <+277>:   cltq
   0x000000000040071b <+279>:   add    -0x28(%rbp),%rax
   0x000000000040071f <+283>:   movzbl (%rax),%eax
   0x0000000000400722 <+286>:   cmp    %al,%dl
   0x0000000000400724 <+288>:   je     0x40072f <main+299>
```

Acá vemos que compara 8 valores con los valores generados en CODE A: si todos son correctos termina satisfactoriamente.

Es correcto el user y pwd, ejecutamos nuevamente el programa en GDB como sigue:

```
(gdb) r Phicar 12345678
```

Ponemos un breakpoint en *main+198 para poder obtener los valores contra los que aplica una operacion xor a nuestra pwd, que son: A1 41 26 42 BF BC 43 AD.
Luego ponemos otro breakpoint en *main+286 para ver los valores correctos luego de aplicar 0xA1412642BFBC43AD xor PASSWORD y obtenemos : F4 06 4E 32 E6 8E 05 D4 por lo que entonces sabemos que:

0xA1412642BFBC43AD ^ "123456789" = 0xF4064E32E68E05D4

Por lo tanto PASSWORD = 0x5547687059324671 = "UGhpY2Fy"

```
[guest@sinfocol ~]$ ./RETO_5 Phicar UGhpY2Fy
Joder! que bien :)
```

Respuesta
UGhpY2Fy

Reto 6

Descripción
Yo se que no es real, pero si le pones una cámara la fantasía si transforma en realidad

Solución
En este reto nos proporcionan una imagen que contiene un código de barras QR Code

Al enviarlo a un decodificador online el texto que nos muestra es:
**“aHR0cHM6Ly9zaXRlcy5nb29nbGUuY29tL3NpdGUvbXlnaXJsZnJpZW5kd2FyZ2FtZTAx L3JldG8vcmV0by56aXA=

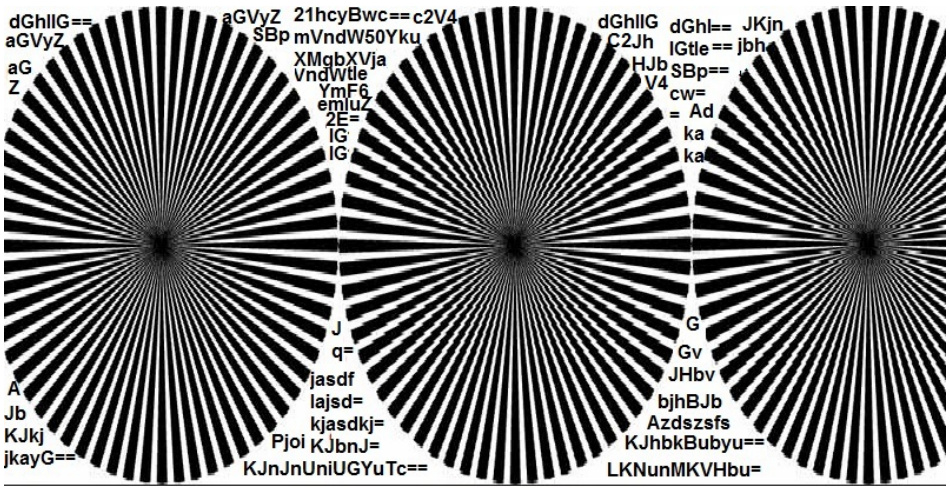
800qy-Fbsuinbq1000y500 wb fo npefm
iuuqt:// tjuft.hpphmf.dpn/tjuf/s0b0e0j0d0b0m0w0j0s0v0t0/lpnf/jnqpsubouf!!.ajq”**

Podemos distinguir fácilmente dos partes que son el texto codificado con base 64: “aHR0cHM6Ly9zaXRlcy5nb29nbGUuY29tL3NpdGUvbXlnaXJsZnJpZW5kd2FyZ2FtZTAxL3JldG8vcmV0by56aXA” y la cadena que sigue un patrón como si fuera una sustitución mono alfabética: “800qy-Fbsuinbq1000y500 wb fo npefm
iuuqt://tjuft.hpphmf.dpn/tjuf/s0b0e0j0d0b0m0w0j0s0v0t0/lpnf/jnqpsubouf!!.ajq”.

Haciendo la respectiva decodificación el primer texto en claro producido es: “**https://sites.google.com/site/mygirlfriendwargame01/reto/reto.zip**”

Para la sustitución mono alfabética se utilizó SNEAK usando el método de fuerza bruta para César. El resultado es: “**800px-Earthmap1000x500 va en model <aq>https://sites.google.com/site/r0a0d0i0c0a0l0v0i0r0u0s0/Home/importante!!.zip**”

El primer comprimido contiene modelos y datos usados por el software Blender. El segundo y más importante contiene una imagen faltante para el modelado y finalmente la respuesta:



No era necesario descargar Blender para encontrar la respuesta. En la anterior imagen

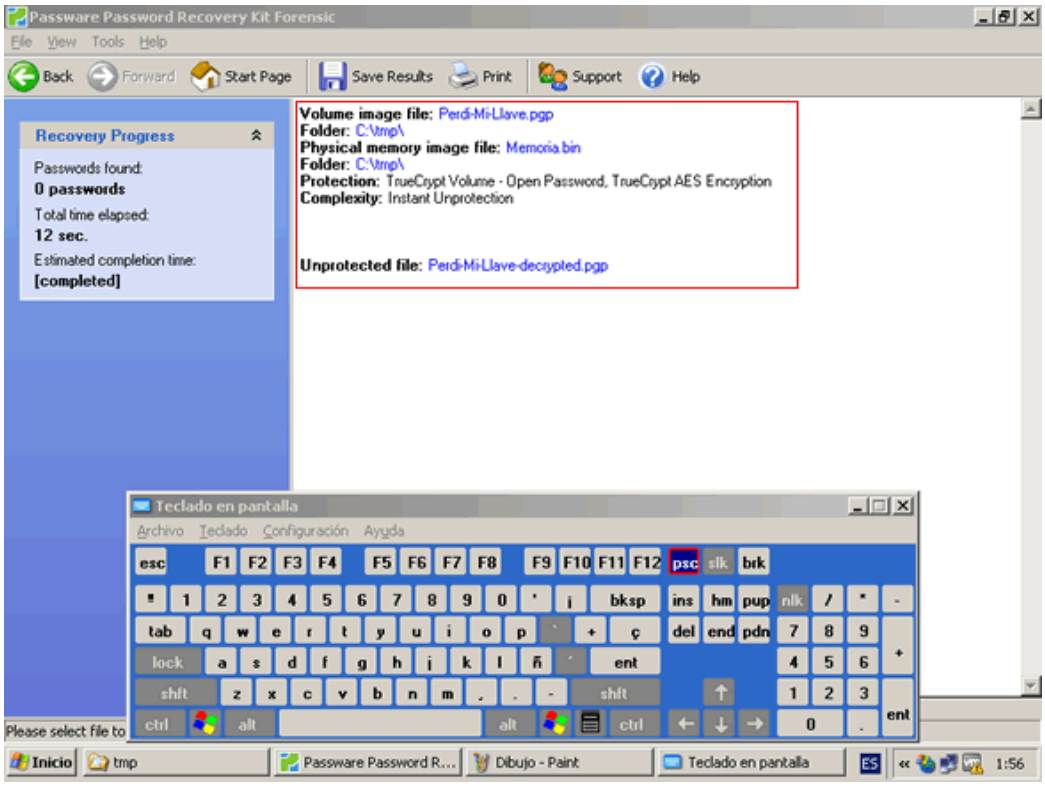
podemos ver de nuevo varias cadenas codificadas en base 64, una de ellas contenía la respuesta: “YmF6emluZ2E=”, que tiene como texto claro: “bazzinga”.

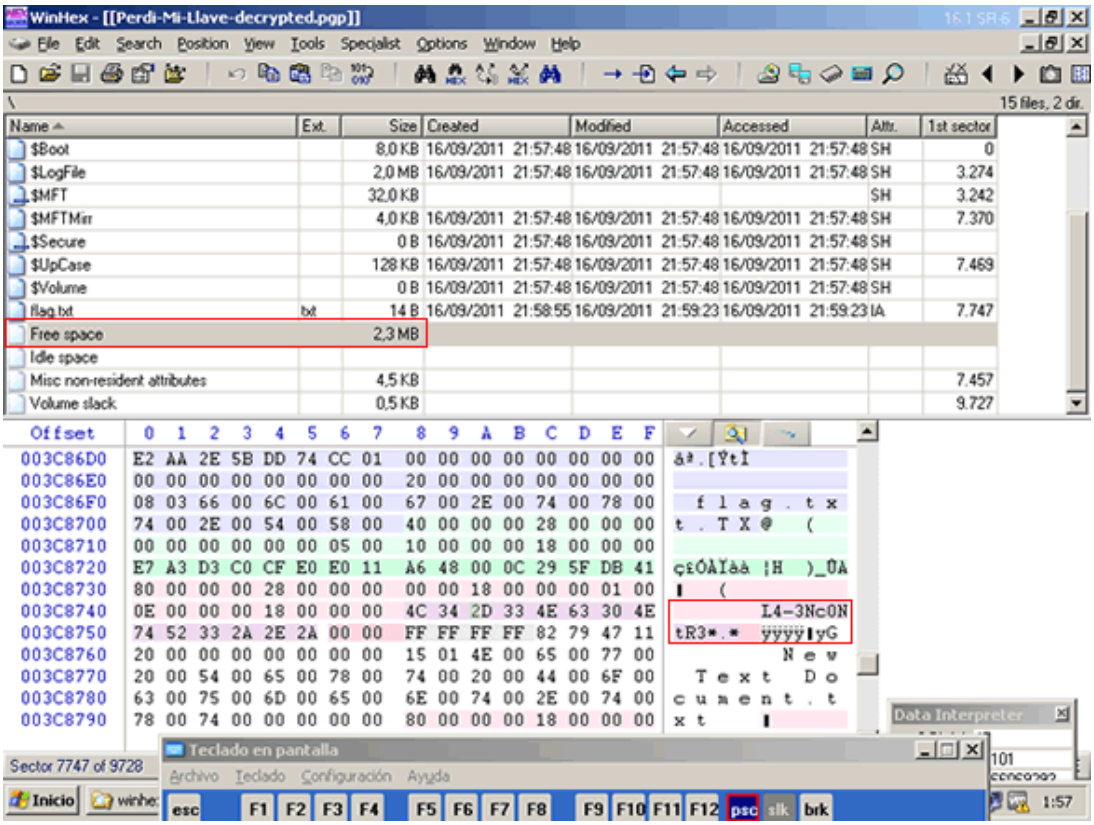
Respuesta
bazzinga

Reto 7

Descripción
Que Tengas Suerte Hijo Mio

Solución
Usando Passware Password Recovery Kit Forensic y mediante la opción para encontrar volúmenes TrueCrypt podemos montar la unidad cifrada y con Winhex encontrar en el espacio libre del disco el archivo eliminado.





Respuesta
L4-3Nc0NtR3*.*

Reto 8

Descripción
Con esas Nalgas Quien No Vende :P

Solución
En el archivo comprimido encontramos dos archivos: una imagen y un jar. La respuesta se encontraba codificada con base 64 al final de la imagen: “Ny4xNjEuNTYx”

Respuesta
7.161.561

Reto 10

Descripción
Simón el Bobito llamó al pastelero:
"¡A ver los pasteles! ¡los quiero probar!"
Sí, repuso el otro, pero antes yo quiero
ver ese cuartillo con que has de pagar.

Buscó en los bolsillos el buen Simoncito

y dijo: ¡De veras! no tengo ni unito.

concatena las dos palabras con _ , ejemplo A_B

Solución
Abrir el archivo comprimido RETO_10.zip y en la carpeta “kagure” leer la clave que se encuentra en el archivo “FLAG-Reto-2.txt”.

Respuesta
BOBITO_RULES