

11

Entrada y salida de datos. Ficheros

En este capítulo veremos cómo Python interactúa con el mundo exterior: leer del usuario, escribir en pantalla, manejar ficheros locales en formatos de texto y binarios, y trabajar con representaciones estructuradas como JSON y CSV. Es un capítulo transversal: la mayor parte de los programas de tratamiento de datos (analizadores de *logs*, procesadores de tablas, conversores de formatos, gestores de configuración) consisten en gran medida en leer ficheros, transformarlos y escribir resultados. La ciberseguridad es uno de los dominios donde estas tareas resultan especialmente frecuentes, y la usaremos como ejemplo de aplicación en varios casos. Como el fichero suele ser el primer punto de contacto del programa con datos cuyo origen no siempre controlamos, dedicaremos también atención a las trampas más habituales y a las buenas prácticas para tratar datos de forma robusta: `eval`, `pickle`, *path traversal* y la gestión de ficheros temporales y permisos.

11.1. Entrada y salida estándar

Las funciones más usadas para interactuar con la consola son `input()` y `print()`. `input()` lee una línea del teclado y la devuelve como cadena (sin el salto de línea final); si necesitamos otro tipo, hay que convertir explícitamente con `int()`, `float()`, etc. `print()` imprime cualquier número de argumentos y admite parámetros como `sep` (separador entre argumentos), `end` (carácter final) y `file` (destino, por defecto `sys.stdout`).

```
nombre = input("Introduce tu nombre: ")
edad = int(input("Introduce tu edad: "))
print(f"Hola, {nombre}. Tienes {edad} años.")
```

```
# Personalizar separador y final
print("a", "b", "c", sep="-", end="!\n") # 'a-b-c!'

# Escribir en stderr
import sys
print("error", file=sys.stderr)
```

↪ A bajo nivel, los flujos estándar son los objetos `sys.stdin`, `sys.stdout` y `sys.stderr`. Trabajamos con ellos cuando, por ejemplo, queremos componer pipelines en línea de comandos.

11.2. La función `eval` y la alternativa segura

`eval(s)` evalúa la cadena `s` como una expresión Python y devuelve su resultado. Es muy potente y, justamente por eso, **muy peligrosa**.

```
print(eval("2 + 3 * 4")) # 14

x = 10
print(eval("x + 5")) # 15
```

Nunca uses `eval` con datos no confiables

`eval` ejecuta **cualquier** expresión Python que se le pase. Si la cadena viene del usuario, de la red, de un fichero o de una base de datos, un atacante puede convertir tu programa en una shell:

```
# Cadena maliciosa que un atacante podría suministrar:
entrada = "__import__('os').system('rm -rf /')"
eval(entrada) # Ejecuta el comando del sistema
```

Si lo que necesitas es interpretar literales (números, listas, diccionarios, cadenas, tuplas, booleanos, `None`), usa `ast.literal_eval`: solo acepta literales y **nunca** ejecuta código arbitrario.

```
import ast

print(ast.literal_eval("[1, 2, 3]")) # [1, 2, 3]
print(ast.literal_eval("{'a': 1}")) # {'a': 1}
ast.literal_eval("__import__('os')") # ValueError:
malformed node
```

Si lo que necesitas es evaluar fórmulas matemáticas con seguridad, considera bibliotecas como `sympy` o restringe el conjunto de operadores que admites con tu propio parser.

11.3. Apertura de ficheros: *open*, modos y codificación

La función `open(path, mode, encoding=...)` abre un fichero y devuelve un objeto sobre el que se realizan las operaciones de lectura/escritura. Los **modos** más utilizados se recogen en la Tabla 1.

Modo	Descripción
'r'	Lectura (por defecto). Falla si no existe.
'w'	Escritura. Trunca el fichero si existe.
'a'	Anexar al final. Crea el fichero si no existe.
'x'	Crear de forma exclusiva : falla si existe.
'r'+	Lectura y escritura.
'b'	Sufijo: modo binario (ej. 'rb', 'wb').
't'	Sufijo: modo texto (por defecto).

Tabla 1: Modos de apertura de ficheros en Python.

11.3.1. Codificación: el parámetro `encoding`

En modo texto, Python decodifica los bytes del fichero a `str` usando una **codificación**. Si no la especificas, Python usa la del sistema, que en macOS y Linux suele ser UTF-8 pero en Windows puede ser CP-1252 u otra. Esto es una fuente recurrente de *bugs* de portabilidad y de *bugs* con caracteres no ASCII.

Especifica siempre `encoding` en modo texto

Salvo que tengas un motivo concreto, usa `encoding="utf-8"` tanto para leer como para escribir:

```
with open("datos.txt", "r", encoding="utf-8") as f:
    contenido = f.read()
```

Si lees datos cuya codificación no controlas, el parámetro `errors` permite definir el comportamiento ante bytes inválidos: `"strict"` (lanza excepción, por defecto), `"replace"` (sustituye por ?), `"ignore"` (descarta). Usar `"ignore"` en datos potencialmente maliciosos es peligroso: puede ocultar contenido o ataques.

11.4. Rutas: `pathlib` y `os.path`

Para manipular nombres de ficheros y directorios, Python ofrece dos APIs:

- `os.path`: API tradicional, basada en cadenas. Sigue siendo común y compatible.

- `pathlib` (Python 3.4+): API orientada a objetos, más legible y portable. Es la opción recomendada hoy.

```
from pathlib import Path

raiz = Path.home() / "datos"          # ~/datos (operador /
↪   como composicion)
fichero = raiz / "logs" / "abr.log"

print(fichero.exists())               # True/False
print(fichero.suffix)                 # '.log'
print(fichero.stem)                   # 'abr'
print(fichero.parent)                 # Path('~ / datos / logs')
↪ print(fichero.read_text(encoding="utf-8")) # Lectura corta
    y comoda

raiz.mkdir(parents=True, exist_ok=True) # mkdir -p
↪ for f in raiz.glob("*.log"):          # Pattern
    print(f.name)
```

`pathlib` es portable: el operador `/` usa el separador correcto en cada sistema (Linux/macOS `/`, Windows `\`).

11.5. Ficheros de texto

11.5.1. Lectura

↪ Los métodos básicos sobre un fichero abierto en modo texto son `f.read([n])`, `f.readline()` y `f.readlines()`, pero la forma más *pythonica* de leer línea a línea es iterar directamente sobre el objeto fichero, lo que evita cargar todo el contenido en memoria:

```
with open("archivo.txt", "r", encoding="utf-8") as f:
    contenido = f.read()               # Lee todo de golpe (cuidado si
↪   es grande)

with open("archivo.txt", "r", encoding="utf-8") as f:
    for linea in f:                    # Iteracion linea a linea,
↪   eficiente
        print(linea, end="")
```

11.5.2. Escritura

Para escribir, se utiliza `f.write(s)` (o `f.writelines(iter)`). El modo `'w'` **trunca** el fichero antes de escribir; el modo `'a'` añade al final.

```
with open("salida.txt", "w", encoding="utf-8") as f:
    f.write("Primera línea.\n")
    f.write("Segunda línea.\n")
```

11.5.3. El bloque `with` y el cierre automático

El bloque `with` es la forma recomendada de manejar ficheros: garantiza que el fichero se cierra al salir del bloque, **incluso si ocurre una excepción**. Sin `with`, habría que llamar manualmente a `f.close()` y envolver todo en un `try/finally` para garantizar el cierre. `with` es más corto y más seguro.

11.5.4. Manejo de errores

Las operaciones con ficheros pueden fallar por muchas razones: el fichero no existe, no tenemos permisos, el disco está lleno, etc. Las excepciones más comunes son `FileNotFoundError`, `PermissionError` e `IsADirectoryError`, todas subclases de `OSError`. Conviene capturarlas explícitamente:

```
try:
    with open("datos.txt", "r", encoding="utf-8") as f:
        contenido = f.read()
except FileNotFoundError:
    print("Error: el archivo no existe.")
except PermissionError:
    print("Error: sin permisos para leer el archivo.")
except OSError as e:
    print(f"Error de E/S: {e}")
```

11.6. Formatos comunes: JSON y CSV

11.6.1. JSON

JSON (*JavaScript Object Notation*) es un formato de texto para intercambio de datos basado en pares clave-valor. Se mapea de forma natural a los tipos básicos de Python: objetos → `dict`, arrays → `list`, strings → `str`, etc. El módulo estándar `json` ofrece cuatro funciones principales:

- `json.dump(obj, f)`: serializa `obj` y lo escribe en el fichero `f`.
- `json.load(f)`: lee del fichero `f` y devuelve el objeto.
- `json.dumps(obj)`: como `dump`, pero devuelve la cadena.
- `json.loads(s)`: como `load`, pero parsea la cadena `s`.

```
1 import json
2
3 datos = {"nombre": "Juan", "edad": 30, "tags": ["admin", "
↪ user"]}
4
5 with open("datos.json", "w", encoding="utf-8") as f:
6     json.dump(datos, f, indent=2, ensure_ascii=False,
↪     sort_keys=True)
7
8 with open("datos.json", "r", encoding="utf-8") as f:
9     cargado = json.load(f)
10    print(cargado)
```

Tres parámetros que conviene conocer:

- `indent=2`: formatea con sangría humanamente legible.
- `ensure_ascii=False`: permite escribir caracteres no ASCII (“ñ”, acentos, emoji) tal cual, sin escaparlos como `\uXXXX`.
- `sort_keys=True`: ordena las claves; útil para diff, control de versiones y hashes reproducibles.

11.6.2. CSV y TSV con `csv`

CSV (*Comma-Separated Values*) y TSV (*Tab-Separated Values*) son ficheros de texto donde cada línea representa una fila y las celdas se separan por un delimitador. Para leerlos y escribirlos correctamente – manejando comillas, escapes, saltos de línea dentro de campos, etc. – conviene usar el módulo estándar `csv`, no hacerlo a mano.

```
1 import csv
2
3 # Escribir
4 with open("salida.csv", "w", newline="", encoding="utf-8") as
↪ f:
5     writer = csv.writer(f)
6     writer.writerow(["Nombre", "Edad"])
7     writer.writerow(["Juan", 30])
8     writer.writerow(["Ana", 25])
9
10 # Leer
11 with open("salida.csv", "r", newline="", encoding="utf-8") as
↪ f:
12     reader = csv.reader(f)
13     cabecera = next(reader)
14     for fila in reader:
15         print(fila)
```

Para trabajar con CSV con cabecera, la API basada en diccionarios es más cómoda:

```
import csv

# DictReader: cada fila es un dict {columna: valor}
with open("usuarios.csv", "r", newline="", encoding="utf-8")
↪ as f:
    for fila in csv.DictReader(f):
        print(fila["Nombre"], fila["Edad"])

# DictWriter
↪ with open("salida.csv", "w", newline="", encoding="utf-8") as
    f:
        writer = csv.DictWriter(f, fieldnames=["Nombre", "Edad"])
        writer.writeheader()
        writer.writerow({"Nombre": "Juan", "Edad": 30})
```

El parámetro `newline=""`

Al abrir ficheros CSV, conviene pasar `newline=""`. Internamente, el módulo `csv` maneja los saltos de línea (que pueden estar dentro de campos entre comillas) y deshabilitar la conversión universal de `open` evita que se dupliquen o pierdan saltos en Windows.

11.6.3. Ejemplo: gestión de contactos

A continuación se ilustra todo lo visto desarrollando un programa que mantiene un fichero de contactos. Dos opciones: añadir registros nuevos o leer y mostrar el contenido. Cada contacto se guarda en una línea con el formato "`<nombre>`, `<teléfono>`, `<email>`
↪ `>`". Se manejan los errores de fichero inexistente y de entrada inválida.

Código 11.1: Código de ejemplo: gestión de contactos.

```
1 import os
2
3 ARCHIVO_CONTACTOS = "contactos.txt"
4
```

```
5 def guardar_contacto():
6     """Solicita información del usuario y la guarda en el
    ↪     archivo de texto."""
7     with open(ARCHIVO_CONTACTOS, 'a') as f: # Abrir en modo
    ↪     'append' para agregar datos
8         while True:
9             nombre = input("Ingrese el nombre del contacto (o
    ↪             presione Enter para terminar): ").strip()
10            if not nombre:
11                break
12            telefono = input("Ingrese el teléfono: ").strip()
13            email = input("Ingrese el correo electrónico: ").
    ↪            strip()
14
15            f.write(f"{nombre}, {telefono}, {email}\n")
16            print("Contacto guardado con éxito.")
17
18 def leer_contactos():
19     """Lee y muestra los contactos almacenados en el archivo
    ↪     de texto."""
20     if not os.path.exists(ARCHIVO_CONTACTOS):
21         print("\nNo hay contactos guardados aún.")
22         return
23
24     print("\nLista de contactos guardados:\n")
25     with open(ARCHIVO_CONTACTOS, 'r') as f:
26         for linea in f:
27             nombre, telefono, email = linea.strip().split(',
    ↪             ')
28             print(f"Nombre: {nombre}, Teléfono: {telefono},
    ↪             Correo: {email}")
29
30 def menu():
31     """Muestra el menú de opciones y ejecuta las acciones
    ↪     correspondientes."""
32     while True:
33         print("\nMenú de opciones:")
34         print("1. Guardar un nuevo contacto")
35         print("2. Mostrar contactos guardados")
36         print("3. Salir")
37
```

```

38     opcion = input("\nSeleccione una opción: ")
39     if opcion == "1":
40         guardar_contacto()
41     elif opcion == "2":
42         leer_contactos()
43     elif opcion == "3":
44         print("\nSaliendo del programa.\n")
45         break
46     else:
47         print("Opción no válida, intente nuevamente.")
48
49 if __name__ == "__main__":
50     menu()

```

11.7. Ficheros binarios

Los ficheros binarios almacenan datos en su representación nativa, sin codificar como texto. Se utilizan para imágenes, vídeos, ejecutables, formatos de protocolo, capturas de tráfico, etc. Se abren con los modos `'rb'`, `'wb'`, `'ab'`, `'rb'+` (no se pasa `encoding` porque trabajamos con bytes, no con `str`).

11.7.1. Lectura y escritura básicas

```

with open("datos.bin", "wb") as f:
    f.write(b"Hola, esto es binario")      # Literal bytes
    f.write("Hola de nuevo".encode())      # Cadena codificada
    f.write(bytes([65, 66, 67]))           # Lista de octetos
    -> b'ABC'

with open("datos.bin", "rb") as f:
    primeros_10 = f.read(10)               # Lee 10 bytes
    resto = f.read()                       # Lee hasta EOF

```

Para procesar ficheros grandes **por bloques** (sin cargar todo en memoria), conviene usar el operador morsa `:=` con un tamaño de bloque típico de 4 KiB o 64 KiB:

```

with open("grande.bin", "rb") as f:
    while bloque := f.read(64 * 1024):    # 64 KiB por
        iteracion
        procesar(bloque)

```

11.7.2. Empaquetado con struct

El módulo `struct` traduce entre tipos de Python y representaciones binarias C. Es indispensable para parsear formatos de fichero y protocolos. `struct.pack(fmt, *args)` produce bytes; `struct.unpack(fmt, data)` los decodifica.

La cadena de formato indica los tipos ('i' entero, 'f' flotante, 'I' unsigned int, 'h' short, 's' bytes, etc.) y, opcionalmente, el **orden de bytes**:

Prefijo	Significado
<	Little-endian (<i>x86, x86-64</i>)
>	Big-endian
!	Network byte order (= big-endian)
=	Nativo, alineamiento estándar
@	Nativo, alineamiento nativo (por defecto)

Tabla 2: Prefijos de orden de bytes en `struct`.

```
1 import struct
2
3 # Empaquetar un entero y un flotante (little-endian explícito
↳ )
4 datos = struct.pack("<if", 42, 3.14159)
5 with open("datos.bin", "wb") as f:
6     f.write(datos)
7
8 # Desempaquetar
9 with open("datos.bin", "rb") as f:
10     contenido = f.read()
11     n, x = struct.unpack("<if", contenido)
12     print(n, x)      # 42 3.141590118408203
```

Especifica siempre el orden de bytes en formatos persistentes

Si guardas datos binarios en un fichero o los envías por la red, **usa siempre** un prefijo explícito (<, > o !). El formato por defecto (@) usa el orden de bytes y alineamiento del sistema, lo que rompe la portabilidad: un fichero generado en una máquina ARM podría no leerse correctamente en x86. Para protocolos de red, el estándar es ! (network byte order).

11.7.3. Hashes y resúmenes con hashlib

El módulo `hashlib` proporciona los algoritmos de hash criptográfico estándar: SHA-256, SHA-512, BLAKE2 y otros. Su patrón habitual con ficheros es leer por bloques y

actualizar el digest:

```
import hashlib

def sha256_de(path):
    h = hashlib.sha256()
    with open(path, "rb") as f:
        while bloque := f.read(64 * 1024):
            h.update(bloque)
    return h.hexdigest()

print(sha256_de("paquete.tar.gz"))
```

No uses hashlib.md5 o sha1 para seguridad

md5 y sha1 están **rotos** desde el punto de vista criptográfico (admiten colisiones prácticas). Aún se admiten en `hashlib` porque siguen siendo útiles como simples *checksums* (verificar integridad sin presencia de adversarios). Para cualquier uso de seguridad – comprobar integridad de ficheros descargados, firmar, etc. – usa sha256 o superior.

11.7.4. El módulo pickle

`pickle` serializa cualquier objeto Python (listas, diccionarios, instancias de clase, funciones, etc.) a una representación binaria, y permite reconstruirlo después. Es muy cómodo, pero conlleva un riesgo de seguridad enorme.

```
import pickle

datos = {"clave": "valor", "edad": 30}

with open("estado.pkl", "wb") as f:
    pickle.dump(datos, f)

with open("estado.pkl", "rb") as f:
    datos = pickle.load(f)
```

¡PELIGRO! `pickle.load` permite ejecución de código arbitrario

`pickle.load` **NO** es seguro frente a datos no confiables. Un fichero `.pkl` controlado por un atacante puede ejecutar código arbitrario en el momento de cargarlo, simplemente definiendo el método especial `__reduce__`. Esto es un **vector de ejecución remota de código** clásico.

Como regla absoluta: **nunca cargues con pickle datos cuyo origen no controles 100 %**. No los descargues de Internet, no los aceptes de un usuario, no los uses como formato de intercambio entre sistemas con confianzas distintas.

Alternativas seguras según el caso de uso:

- Para datos estructurados pero *plenamente serializables* (dict, list, str, int, float, bool, None): `json`.
- Para datos numéricos: `numpy.save` / `numpy.load` (con `allow_pickle=False` ↪).
- Para datos tabulares: `pandas` a `parquet` o `csv`.
- Para datos arbitrarios autenticados: serializa a JSON y **firma** el resultado con HMAC.

11.8. Robustez y buenas prácticas al manejar ficheros

Los ficheros son uno de los puntos más delicados de una aplicación: cualquier ruta o contenido cuyo origen no controlamos conviene tratarlo con cautela. Esta sección recoge las trampas más habituales y cómo evitarlas para escribir código robusto.

11.8.1. Path traversal

Un *path traversal* se produce cuando una ruta proporcionada por el usuario incluye `..` para escapar del directorio previsto y acceder a otros ficheros del sistema. El ejemplo clásico es un servidor web que sirve ficheros desde `«/var/www/files/»` y recibe una petición de `«../../etc/passwd»`.

La defensa estándar es **normalizar** la ruta y comprobar que sigue dentro del directorio permitido:

```
from pathlib import Path

def fichero_seguro(base, nombre_pedido):
    base = Path(base).resolve()
    pedido = (base / nombre_pedido).resolve()
    if base not in pedido.parents and base != pedido:
        raise ValueError("Acceso fuera del directorio")
```

```
↪         permitido")
    return pedido

contenido = fichero_seguro("/var/www/files", "../etc/passwd")
↪         # ValueError
```

11.8.2. Escritura atómica

Si un programa se interrumpe a media escritura, el fichero queda corrupto. La técnica habitual para escribir *atómicamente* es escribir a un fichero temporal y, una vez completado, renombrarlo al destino final con `os.replace` (que en POSIX es atómico):

```
import os
from pathlib import Path

def escribir_atómico(path, contenido):
    path = Path(path)
    tmp = path.with_suffix(path.suffix + ".tmp")
    tmp.write_text(contenido, encoding="utf-8")
    os.replace(tmp, path)    # Atómico en POSIX
```

11.8.3. Ficheros temporales seguros

Crear un fichero temporal con un nombre predecible (por ejemplo, `«/tmp/datos.tmp»`) es vulnerable a *race conditions*: otro proceso podría crear el fichero antes que el tuyo con un enlace simbólico a un fichero sensible. El módulo estándar `tempfile` resuelve el problema: genera nombres aleatorios y abre el fichero con permisos restrictivos:

```
import tempfile

↪ with tempfile.NamedTemporaryFile("w", delete=False, encoding=
    "utf-8") as tmp:
    tmp.write("datos sensibles")
    print(tmp.name)    # Nombre seguro y unico

# Para directorios temporales
↪ with tempfile.TemporaryDirectory() as d:
    print(d)    # Directorio que se autodestruye al salir del
    with
```

11.8.4. Permisos al escribir secretos

En sistemas POSIX, los ficheros se crean por defecto con permisos `0644` (lectura mundial). Si estás escribiendo claves, tokens o datos sensibles, restringe los permisos:

```
import os

# Crear con permisos 0600 (solo el propietario lee/escribe)
fd = os.open("clave.key", os.O_WRONLY | os.O_CREAT | os.
↪ O_TRUNC, 0o600)
with os.fdopen(fd, "w") as f:
    f.write(clave_secreta)
```

11.9. Ejemplo: almacenamiento binario de usuarios

Para cerrar el capítulo, un programa interactivo que guarda y recupera registros de usuarios en un fichero binario. Cada usuario tiene un identificador entero, un nombre de longitud fija (20 bytes) y una edad entera. El menú permite añadir, listar y salir; la entrada termina cuando el identificador introducido es 0. Se manejan errores de entrada y la posible inexistencia del fichero.

Código 11.2: Código de ejemplo: almacenamiento y recuperación de datos en un archivo binario.

```
1 import struct
2 import os
3
4 # Definir el formato de los registros binarios: int (4 bytes)
↪   + string de 20 bytes + int (4 bytes)
5 FORMATO = 'i20si'
6 TAMANYO_REGISTRO = struct.calcsize(FORMATO)
7 ARCHIVO_BINARIO = 'usuarios.bin'
8
```

```

9 def guardar_usuarios():
10     usuarios = []
11     while True:
12         try:
13             id_usuario = int(input("Ingrese ID del usuario (0
↳             para terminar): "))
14             if id_usuario == 0:
15                 break
16             nombre = input("Ingrese nombre del usuario (má
↳             ximo 20 caracteres): ").strip()
17             nombre_bytes = nombre.encode('utf-8').ljust(20, b
↳             '\x00') # Convertir a bytes y rellenar
18             edad = int(input("Ingrese edad del usuario: "))
19             usuarios.append((id_usuario, nombre_bytes, edad))
20         except ValueError:
21             print("Error: Entrada inválida. Intente de nuevo.
↳             ")
22
23     with open(ARCHIVO_BINARIO, 'wb') as f:
24         for usuario in usuarios:
25             f.write(struct.pack(FORMATO, *usuario))
26     print(f"\nDatos guardados en {ARCHIVO_BINARIO}.\n")
27
28 def leer_usuarios():
29     if not os.path.exists(ARCHIVO_BINARIO):
30         print(f"\nNo se encontró el archivo {ARCHIVO_BINARIO
↳         }. Primero guarde datos.\n")
31         return
32
33     print("\nLista de usuarios guardados:\n")
34     with open(ARCHIVO_BINARIO, 'rb') as f:
35         while chunk := f.read(TAMANYO_REGISTRO): # Leer en
↳         bloques de tamaño fijo
36             id_usuario, nombre_bytes, edad = struct.unpack(
↳             FORMATO, chunk)
37             nombre = nombre_bytes.decode('utf-8').strip('\x00
↳             ') # Eliminar padding
38             print(f"ID: {id_usuario}, Nombre: {nombre}, Edad:
↳             {edad}")
39

```

```
40 def menu():
41     while True:
42         print("\nMenú de opciones:")
43         print(f"1. Guardar usuarios en archivo binario ({'{'
↪         ARCHIVO_BINARIO}}\')")
44         print(f"2. Leer y mostrar usuarios desde archivo
↪         binario ({'{'ARCHIVO_BINARIO}}\')")
45         print("3. Salir")
46
47         opcion = input("\nSeleccione una opción: ")
48         if opcion == "1":
49             guardar_usuarios()
50         elif opcion == "2":
51             leer_usuarios()
52         elif opcion == "3":
53             print("\nSaliendo del programa.\n")
54             break
55         else:
56             print("Opción no válida, intente nuevamente.")
57
58 if __name__ == "__main__":
59     menu()
```

11.10. Ejercicios

EJERCICIO 11.1

Escribe un programa en Python que procese un archivo de logs de acceso web y determine las 10 IPs más frecuentes que han accedido al sistema. El archivo de logs tiene el siguiente formato:

```
192.168.1.1 - - [10/Mar/2025:12:00:01 +0100] "GET /index.html HTTP/1.1" 200 1234
192.168.1.10 - - [10/Mar/2025:18:48:33 +0100] "GET /login HTTP/1.1" 200 2681
192.168.1.10 - - [12/Mar/2025:07:50:51 +0100] "GET /about.html HTTP/1.1" 404 1962
192.168.1.5 - - [05/Mar/2025:23:56:58 +0100] "DELETE /index.html HTTP/1.1" 500 9110
192.168.1.3 - - [09/Mar/2025:23:40:36 +0100] "GET /index.html HTTP/1.1" 301 4393
192.168.1.9 - - [10/Mar/2025:04:28:46 +0100] "GET /contact.html HTTP/1.1" 200 3286
192.168.1.10 - - [08/Mar/2025:11:46:54 +0100] "DELETE /services.html HTTP/1.1" 301 4174
192.168.1.2 - - [10/Mar/2025:12:00:02 +0100] "POST /login HTTP/1.1" 200 3456
...
```

El programa debe leer el archivo de logs, contar la cantidad de accesos por cada IP y generar un archivo de salida con nombre `top_ips.txt` con las 10 IPs con más accesos, junto con su número de accesos.

Se requiere que el programa use `collections.Counter` para contar las ocurrencias de las IPs, sea capaz de manejar archivos grandes de logs mediante la lectura línea por línea y genere los datos en el archivo de salida de forma ordenada en función de la frecuencia.

Solución (girar para leer)

```

1 import re
2 from collections import Counter
3
4 def procesar_logs(archivo_log, archivo_salida):
5     # Expresión regular para extraer las IPs
6     patron_ip = r'(\d+\.\d+\.\d+\.\d+)'
7
8     # Contador para las IPs
9     contador_ips = Counter()
10
11     with open(archivo_log, 'r') as f:
12         for linea in f:
13             # Buscar la IP en cada línea
14             match = re.match(patron_ip, linea)
15             if match:
16                 ip = match.group(1)
17                 contador_ips[ip] += 1
18
19     # Obtener las 10 IPs con más accesos
20     top_ips = contador_ips.most_common(10)
21
22     # Escribir los resultados en el archivo de salida
23     with open(archivo_salida, 'w') as salida:
24         for ip, cantidad in top_ips:
25             salida.write(f"{ip} {cantidad}\n")
26
27     # Llamar a la programación
28     procesar_logs('access.log', 'top-ips.txt')

```

Código 11.3: Solución al Ejercicio 11.1.

EJERCICIO 11.2

↳ Escribe un programa en Python que realice una copia incremental de archivos grandes, donde sólo se copien aquellos archivos que hayan cambiado desde la última copia de seguridad. El programa debe tomar un directorio de origen (`/origen`) y un directorio de destino (`/backup`) y realizar lo siguiente:

1. Comparar los archivos en `/origen` con los de `/backup`.
2. Si un archivo en `/origen` no está en `/backup` o ha cambiado (comparando los hashes SHA256 de los archivos), debe copiarlo al directorio de destino.

El programa debe generar un archivo de registro `«copia_log.txt»` que contenga los nombres de los archivos copiados, el lugar de destino de la copia, y su fecha de copia.

Se requiere que el programa use el módulo `hashlib` para calcular los hashes SHA256 de los archivos, compare los archivos por su hash y tamaño para determinar si han cambiado y utilice `os.walk()` para recorrer los directorios y archivos.

Solución (girar para leer)

```

1 import os
2 import hashlib
3 import shutil
4 from datetime import datetime
5
6 def calcular_hash(archivo):
7     """Calcula el hash SHA256 de un archivo."""
8     sha256 = hashlib.sha256()
9     with open(archivo, 'rb') as f:
10         while chunk := f.read(8192):
11             sha256.update(chunk)
12     return sha256.hexdigest()
13
14 def copiar_incremental(origen, destino, archivo_log):
15     """Realiza una copia incremental de archivos desde
16     un directorio de origen a un destino."""
17     os.makedirs(destino)
18     with open(archivo_log, 'a') as log:
19         for carpeta, _, archivos in os.walk(origen):
20             for archivo in archivos:
21                 archivo_origen = os.path.join(carpeta,
22                                                 archivo)
23                 archivo_destino = os.path.join(destino,
24                                                 os.path.relpath(archivo_origen,
25                                                                 origen))
26                 # Si el archivo no existe en destino o
27                 # ha cambiado
28                 if not os.path.exists(archivo_destino) or \
29                    calcular_hash(archivo_origen) != \
30                    calcular_hash(archivo_destino):
31                     # Crear la estructura de
32                     # directorios si no existe
33                     os.makedirs(os.path.dirname(
34                         archivo_destino), exist_ok=True)
35                     # Copiar el archivo
36                     shutil.copy2(archivo_origen,
37                                   archivo_destino)
38                     # Registrar la copia
39                     fecha_modificacion = datetime.now()
40                     strftime = "%Y-%m-%d %H:%M:%S"
41                     log.write(f"{archivo_destino} copiado a {
42                               fecha_modificacion}\n")
43                     print(f"Archivo {archivo_destino} copiado.")
44
45 # Llamada a la función
46 copiar_incremental('.', './backup', 'copia_log.txt')

```

EJERCICIO 11.3

Escribe un programa en Python que implemente un sistema de indexación de texto para archivos `.txt`. El programa debe crear un índice invertido, que almacene, para cada palabra en el archivo, la lista de líneas en las que aparece. El archivo de texto de entrada tiene el siguiente formato:

Esto es un archivo de ejemplo.
Este archivo tiene varias líneas de texto.
Cada línea contiene varias palabras.

El programa debe almacenar el índice invertido en un archivo denominado «`indice.json`», donde cada palabra clave es una clave del diccionario y el valor asociado es una lista de números de línea en las que aparece esa palabra.

El programa ha de leer el archivo `.txt` de entrada línea por línea, almacenar el índice en formato JSON, implementar una búsqueda rápida de palabras clave que devuelva las líneas donde aparecen y controlar los posibles errores de ejecución que pudieran suceder.

Solución (girar para leer)

```

1 import json
2 import re
3
4 def crear_indice(archivo_entrada, archivo_salida):
5     indice = {}
6
7     try:
8         with open(archivo_entrada, 'r') as f:
9             for num_linea, linea in enumerate(f, start
10                 =1):
11                 palabras = re.findall(r'\b\w+\b', linea
12                     .lower()) # Extrae palabras y las
13                     pasa a minúsculas
14                 for palabra in palabras:
15                     if palabra not in indice:
16                         indice[palabra] = []
17                 indice[palabra].append(num_linea)
18
19         with open(archivo_salida, 'w') as f_out:
20             json.dump(indice, f_out, indent=4)
21             print(f"Índice invertido guardado en {
22                 archivo_salida}")
23     except FileNotFoundError:
24         print(f"Error: El archivo {archivo_entrada} no
25             se encontró.")
26     except IOError as e:
27         print(f"Error de entrada/salida: {e}")
28     except Exception as e:
29         print(f"Ocurrió un error inesperado: {e}")
30
31 def buscar_palabra(archivo_indice, palabra):
32     try:
33         with open(archivo_indice, 'r') as f:
34             indice = json.load(f)
35             palabra = palabra.lower()
36             if palabra in indice:
37                 return indice[palabra]
38             else:
39                 return []
40     except FileNotFoundError:
41         print(f"Error: El archivo {archivo_indice} no
42             se encontró.")
43     return []

```

Solución (girar para leer)

```

39 except json.JSONDecodeError:
40     print(f"Error: El archivo {archivo_indice} no
    ← contiene un formato JSON válido.")
41     return []
42 except Exception as e:
43     print(f"Ocurrió un error inesperado: {e}")
44     return []
45
46 # Llamada a las funciones
47 crear_indice('archivo.txt', 'indice.json')
48
49 # Ejemplo de búsqueda
50 resultado = buscar_palabra('indice.json', 'archivo')
51 if resultado:
52     print(f'Las palabras se encuentran en las líneas: {
    ← resultado}')
53 else:
54     print('Palabra no encontrada o error en la búsqueda
    ← .')

```

Código 11.6: Solución al Ejercicio 11.3 – continuación.

EJERCICIO 11.4

Escribe un programa en Python que procese un archivo CSV con registros de usuarios y realice las siguientes validaciones y transformaciones:

1. Si el campo “Nombre” está vacío, eliminar el registro.
2. Si “Edad” es negativa, establecerla en `None`.
3. Si “Email” no tiene un formato válido, reemplazarlo por “email_invalido”.
4. Si “FechaRegistro” no es válida, establecerla en `None`.

El archivo de entrada «`usuarios.csv`» tiene el siguiente formato:

Ejemplo de ejecución esperada

```
ID,Nombre,Edad,Email,FechaRegistro
1,Juan,25,juan@example.com,2025-03-10
2,,30,ana@ejemplo,2025-02-30
3,Carlos,-5,carlos@example.com,2024-12-01
```

El programa debe generar un nuevo archivo «`usuarios_limpios.csv`» con los datos validados y transformados. El programa ha de usar el módulo `csv` para leer y escribir el archivo CSV, validar el correo electrónico utilizando una expresión regular (módulo `re`) y usar `datetime.strptime()` para validar las fechas.

Solución (girar para leer)

```

1 import csv
2 import re
3 from datetime import datetime
4
5 # Función para validar el formato de email
6 def validar_email(email):
7     patron = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
8     if re.match(patron, email):
9         return email
10    else:
11        return 'email_invalido'
12
13 # Función para validar la fecha
14 def validar_fecha(fecha):
15     try:
16         return datetime.strptime(fecha, '%Y-%m-%d').
17         strftime('%Y-%m-%d')
18    except ValueError:
19        return None
20
21 # Procesar archivo CSV
22 def procesar_csv(archivo_entrada, archivo_salida):
23     with open(archivo_entrada, 'r', newline='') as infle, open(archivo_salida, 'w', newline='') as outfle:
24         reader = csv.DictReader(infle)
25         writer = csv.DictWriter(outfle, fieldnames=
26             reader.fieldnames)
27         writer.writeheader()
28         for row in reader:
29             # Validar y transformar los datos
30             if not row['Nombre']: # Eliminar registros
31                 con Nombre vacio
32             continue
33             if int(row['Edad']) > 0: # Establecer Edad
34                 en None si es negativa
35                 row['Edad'] = None
36             else:
37                 row['Edad'] = int(row['Edad'])
38             # Validar Email
39             row['Email'] = validar_email(row['Email'])
40             # Validar FechaRegistro
41             row['FechaRegistro'] = validar_fecha(row['
42                 FechaRegistro'])
43             # Escribir la fila validada
44             writer.writerow(row)
45
46 # Llamada a la función
47 procesar_csv('usuarios.csv', 'usuarios_limpios.csv')

```

EJERCICIO 11.5

Escribe un programa en Python que implemente el cifrado y descifrado de archivos utilizando el algoritmo AES en modo CBC. El programa debe realizar las siguientes operaciones:

1. Cifrar un archivo de texto dado con una clave proporcionada por el usuario.
2. Guardar el archivo cifrado en formato binario.
3. Permitir el descifrado del archivo cifrado usando la misma clave.

El archivo de entrada debe ser un archivo de texto, y el archivo de salida debe ser un archivo binario. Como requisitos, en este programa se establecen los siguientes: (I) usar el módulo `pycryptodome` para la implementación de AES; (II) solicitar la clave de cifrado al usuario de forma segura utilizando `getpass` (del módulo `getpass`); y (III) gestionar adecuadamente el relleno de bloques para cifrado en modo CBC.

Solución (girar para leer)

```

1 from Crypto.Cipher import AES
2 from Crypto.Util.Padding import pad, unpad
3 from Crypto.Random import get_random_bytes
4 import getpass
5 import os
6
7 # Función para cifrar un archivo
8 def cifrar_archivo(archivo-entrada, archivo-salida,
9 clave):
10     # Asegurar que la clave tenga una longitud adecuada
11     # (16, 24 o 32 bytes para AES)
12     clave = clave.encode('utf-8')
13     clave = clave.ljust(32, b'\0')[:32] # Ajustar a 32
14     # bytes si es necesario
15     # Generar un IV aleatorio
16     iv = get_random_bytes(AES.block-size)
17     # Crear el cifrador AES en modo CBC
18     cipher = AES.new(clave, AES.MODE_CBC, iv)
19     # Leer el archivo de entrada
20     with open(archivo-entrada, 'rb') as f:
21         datos = f.read()
22     # Rellenar los datos para que su tamaño sea un
23     # múltiplo de 16 bytes
24     datos-rellenados = pad(datos, AES.block-size)
25     # Cifrar los datos
26     datos-cifrados = cipher.encrypt(datos-rellenados)
27     # Guardar el IV y los datos cifrados en el archivo
28     # binario de salida
29     with open(archivo-salida, 'wb') as f:
30         f.write(iv + datos-cifrados)

```

Código 11.8: Solución al Ejercicio 11.5.

Solución (girar para leer)

```

32
33 # Función para descifrar un archivo
34 def descifrar_archivo(archivo_entrada, archivo_salida,
35                        clave):
36     # Asegurar que la clave tenga una longitud adecuada
37     # (16, 24 o 32 bytes para AES)
38     clave = clave.encode('utf-8')
39     clave = clave.ljust(32, b'\0')[:32] # Ajustar a 32
40     bytes si es necesario
41     # Leer el archivo cifrado
42     with open(archivo_entrada, 'rb') as f:
43         iv = f.read(AES.block_size) # Leer el IV
44         datos_cifrados = f.read() # Leer los datos
45     # Crear el cifrador AES en modo CBC
46     cipher = AES.new(clave, AES.MODE_CBC, iv)
47     # Descifrar los datos
48     datos_descifrados = unpad(cipher.decrypt(
49         datos_cifrados), AES.block_size)
50     # Guardar los datos descifrados en el archivo de
51     # salida
52     with open(archivo_salida, 'wb') as f:
53         f.write(datos_descifrados)
54     # Solicitar la clave al usuario de forma segura
55     clave = getpass.getpass("Introduce la clave de cifrado:
56     ")
57     # Llamada a la función de cifrado
58     cifrar_archivo(archivo_entrada, 'archivo_cifrado.bin',
59                   clave)
60     # Llamada a la función de descifrado
61     descifrar_archivo('archivo_cifrado.bin', '
62     archivo_descifrado.txt', clave)
63     print("Cifrado y descifrado completados.")

```

Código 11.9: Solución al Ejercicio 11.5 – continuación.

EJERCICIO 11.6

Escribe un programa en Python que permita detectar archivos maliciosos en un sistema utilizando una lista negra de hashes. El programa debe leer un archivo de logs o directorio con archivos a analizar, calcular el hash SHA256 para cada archivo, comparar el hash calculado con una lista negra de hashes almacenada en un archivo «`hashes_maliciosos.txt`», y, si el hash coincide con alguno de la lista negra, alertar al usuario y registrar el hallazgo en un archivo denominado «`registro_alertas.log`».

El programa ha de usar el módulo `hashlib` para calcular los hashes SHA256.

Solución (girar para leer)

```

1 import hashlib
2 import os
3
4 # Función para calcular el hash SHA256 de un archivo
5 def calcular_hash(archivo):
6     try:
7         sha256 = hashlib.sha256()
8         with open(archivo, 'rb') as f:
9             while chunk := f.read(8192):
10                 sha256.update(chunk)
11         return sha256.hexdigest()
12     except FileNotFoundError:
13         print(f"Error: El archivo {archivo} no se
14             encontró.")
15     except IOError as e:
16         print(f"Error de entrada/salida al leer {
17             archivo}: {e}")
18     except Exception as e:
19         print(f"Ocurrió un error inesperado al calcular
20             el hash de {archivo}: {e}")
21         return None
22
23 # Función para cargar la lista negra de hashes
24 # maliciosos
25 def cargar_lista_negra(archivo_lista):
26     try:
27         with open(archivo_lista, 'r') as f:
28             return set(line.strip() for line in f)
29     except FileNotFoundError:
30         print(f"Error: El archivo {archivo_lista} no se
31             encontró.")
32     except IOError as e:
33         print(f"Error de entrada/salida al leer {
34             archivo_lista}: {e}")
35     except Exception as e:
36         print(f"Ocurrió un error inesperado al cargar
37             la lista negra: {e}")
38         return set()

```

Código 11.10: Solución al Ejercicio 11.6.

Solución (girar para leer)

```

32
33 # Función para procesar los archivos y detectar los
34 def detectar_archivos_maliciosos(directorio,
35                                lista_negra, archivo_registro):
36     try:
37         for carpeta, -, archivos in os.walk(
38             directorio):
39             for archivo in archivos:
40                 archivo_completo = os.path.join(
41                     carpeta, archivo)
42                 hash_calculado = calcular_hash(
43                     archivo_completo)
44                 # Si el hash es válido y está en la
45                 # lista negra
46                 if hash_calculado and
47                 hash_calculado in lista_negra:
48                     alerta = f"ALERTA: Archivo
49                     malicioso detectado: {
50                         archivo_completo}, Hash: {
51                         hash_calculado}\n"
52                     print(alerta)
53                     log.write(alerta)
54                     except Exception as e:
55                         print(f"Ocurrió un error inesperado al procesar
56                             el directorio: {e}")
57                         # Cargar lista negra de hashes maliciosos
58                         lista_negra = cargar_lista_negra('hashes-maliciosos.txt')
59                     # Llamada a la función para detectar archivos
60                     # maliciosos
61                     if lista_negra:
62                         detectar_archivos_maliciosos('.', lista_negra,
63                                                         registro_alertas.log')
64     print("Detección de archivos maliciosos completada.")

```

Código 11.11: Solución al Ejercicio 11.6 – *continuación*.

EJERCICIO 11.7

Escribe un programa en Python que escanee una lista de URLs en busca de posibles vulnerabilidades de seguridad, como XSS o SQL Injection. El programa debe tomar una lista de URLs de entrada desde un archivo «`urls.txt`» y realizar las siguientes verificaciones:

1. Verificar si una URL contiene parámetros susceptibles a SQL Injection (por ejemplo, `id=1`).
2. Verificar si una URL puede estar vulnerable a ataques de XSS, al detectar que los parámetros no están adecuadamente escapados.

El programa debe generar un archivo «`resultados_vulnerabilidades.txt`» que contenga la URL escaneada y el tipo de vulnerabilidad encontrada. En este caso, se requiere que el programa use expresiones regulares para detectar posibles vulnerabilidades y lea las URLs desde un archivo de texto (llamado «`urls.txt`»), escribiendo los resultados en un archivo de salida (llamado «`resultados_vulnerabilidades.txt`»).

Ejemplo de ejecución esperada

URL: `http://example.com?id=1`

Vulnerabilidad detectada: SQL Injection

URL: `http://example.com?name=<script>alert('XSS')</script>`

Vulnerabilidad detectada: XSS

Solución (girar para leer)

1	<code>import re</code>	43
2		42
3	<code>def funcion_escanear_urls(url):</code>	41
4	<code> # Función para verificar vulnerabilidades de XSS</code>	40
5	<code> try:</code>	39
6	<code> # Patrón para detectar parámetros susceptibles</code>	38
7	<code> a SQL Injection</code>	37
8	<code> patron_sql_injection = r'[\?&]id=\d+'</code>	36
9	<code> if re.search(patron_sql_injection, url):</code>	35
10	<code> return "SQL Injection"</code>	34
11	<code> except Exception as e:</code>	33
12	<code> print(f"Error en la verificación de SQL Injection para la URL {url}: {e}")</code>	32
13	<code> return None</code>	31
14	<code> # Función para verificar vulnerabilidades de XSS</code>	30
15	<code> def verificar_xss(url):</code>	29
16	<code> try:</code>	28
17	<code> # Patrón para detectar parámetros que podrían</code>	27
18	<code> ser vulnerables a XSS</code>	26
19	<code> patron_xss = r'[\?&]name=.*<script.*>.*</script>'</code>	25
20	<code> if re.search(patron_xss, url, re.IGNORECASE):</code>	24
21	<code> return "XSS"</code>	23
22	<code> except Exception as e:</code>	22
23	<code> print(f"Error en la verificación de XSS para la URL {url}: {e}")</code>	21
24	<code> return None</code>	20
25	<code> # Función para escanear URLs desde un archivo de texto</code>	19
26	<code> def escanear_urls(archivo_urls, archivo_resultados):</code>	18
27	<code> try:</code>	17
28	<code> with open(archivo_urls, 'r') as f_in, open(</code>	16
29	<code> archivo_resultados, 'w') as f_out:</code>	15
30	<code> for url in f_in:</code>	14
31	<code> url = url.strip()</code>	13
32	<code> vulnerabilidad = None</code>	12
33	<code> # Verificar SQL Injection</code>	11
34	<code> vulnerabilidad = verificar_xss(url)</code>	10
35	<code> if not vulnerabilidad:</code>	9
36	<code> # Verificar XSS si no se detectó</code>	8
37	<code> vulnerabilidad = verificar_xss(url)</code>	7

Solución (girar para leer)

```

38 # Si se detecta alguna vulnerabilidad,
39     escribir la en el archivo de
40     resultados
41     if vulnerabilidad:
42         f_out.write(f"URL: {url}\n"
43                     "vulnerabilidad detectada: {url}\n")
44     else:
45         f_out.write(f"URL: {url}\n"
46                     "vulnerabilidad detectada: Ninguna\n")
47     except FileNotFoundError as e:
48         print(f"Error: El archivo no se encontró: {e}")
49     except IOError as e:
50         print(f"Error de entrada/salida: {e}")
51     except Exception as e:
52         print(f"Ocurrió un error inesperado: {e}")
53 # llamada a la función para escanear las URLs
54 try:
55     escanear_urls('urls.txt', 'resultados_vulnerabilidades.txt',
56                  guardar en 'resultados_vulnerabilidades.txt',
57                  print("Escaneo completado. Los resultados se
58                  guardar en 'resultados_vulnerabilidades.txt'.")
59     except Exception as e:
60         print(f"Error al ejecutar el programa: {e}")

```

Código 11.13: Solución al Ejercicio 11.7 – continuación.

EJERCICIO 11.8

Escribe un programa en Python que analice archivos binarios en busca de patrones que podrían indicar la presencia de malware. El programa debe realizar lo siguiente:

1. Leer un archivo binario de entrada.
2. Buscar patrones sospechosos, como cadenas que podrían corresponder a URLs maliciosas, direcciones IP o comandos de sistema.
3. Generar un archivo «`informes_malware.txt`» con los hallazgos encontrados, indicando el patrón y su ubicación en el archivo.

El programa ha de usar expresiones regulares para identificar cadenas de texto sospechosas y guardar los resultados en un archivo de texto (llamado «`informes_malware.txt`»). Respecto a las expresiones regulares, considera las siguientes:

```
r'http://[a-zA-Z0-9./?=&_-]+' # URL maliciosas
r'\b(?:\d{1,3}\.){3}\d{1,3}\b' # Direcciones IP
r'\b(?:rm\s+-rf\s+/\s+shutdown\s+-h\s+now)\b' # Comandos
de sistema peligrosos
```

Ejemplo de ejecución esperada

Patrón detectado: `http://malicious-website.com`
Ubicación: 1024 bytes

Patrón detectado: `192.168.1.1`
Ubicación: 2048 bytes

Solución (girar para leer)

```

1 import re
2
3 # Función para buscar patrones sospechosos en un
4 def buscar_patrones(archivo):
5     patrones = [
6         r'http://[a-zA-Z0-9./?=&~+]', # URL
7         r'\b(?:\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})', # Direcciones
8         r'\b(?:\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})', # IP
9         r'\b(?:\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})', # Comandos de sistema peligrosos
10    ]
11    resultados = []
12    try:
13        with open(archivo, 'rb') as f:
14            datos = f.read()
15            for patron in patrones:
16                for match in re.finditer(patron, str(
17                    datos)):
18                    resultados.append((match.group(0),
19                        match.start()))
20    except FileNotFoundError:
21        print(f"Error: El archivo {archivo} no se
22            encontró.")
23    except IOError as e:
24        print(f"Error de entrada/salida al leer {
25            archivo}: {e}")
26    except Exception as e:
27        print(f"Ocurrió un error inesperado: {e}")
28    return resultados
29
30 # Función para guardar los resultados en un archivo de
31 texto
32 def guardar_resultados(resultados, archivo_resultados):
33    try:
34        with open(archivo_resultados, 'w') as f:
35            for patron, ubicacion in resultados:
36                f.write(f"Patrón detectado: {patron}\n
37                    ubicación: {ubicacion}\n bytes\n\n")
38            print(f"Resultados guardados en {
39                archivo_resultados}")

```

Solución (girar para leer)

```

36 except IOError as e:
37     print(f"Error al guardar el archivo {
38         archivo_resultados}: {e}")
39     except Exception as e:
40         print(f"Ocurrió un error inesperado al guardar
41             los resultados: {e}")
42         # llamada a las funciones
43         resultados = buscar_patrones('archivo.bin')
44         if resultados:
45             guardar_resultados(resultados, 'informes_malware.
46                 txt')
47         else:
48             print(f"No se encontraron patrones sospechosos.")

```

Código 11.15: Solución al Ejercicio 11.8 – *continuación*.

EJERCICIO 11.9

Escribe un programa en Python que analice los paquetes de red capturados en un archivo de registro de tráfico (.pcap). El programa debe realizar lo siguiente:

1. Cargar el archivo .pcap y extraer los paquetes de red.
2. Analizar los paquetes para detectar posibles ataques, como DDoS (Denegación de Servicio Distribuida) y escaneos de puertos.
3. Generar un archivo « `analisis_trafico.log` » que contenga los paquetes sospechosos detectados, con detalles como la IP origen, IP destino, protocolo y el tipo de ataque detectado.

El programa ha de usar el módulo `scapy` para cargar y analizar los paquetes de red.

Ejemplo de ejecución esperada

```
IP Origen: 192.168.1.10, IP Destino: 192.168.1.20, Protocolo: TCP,  
Ataque: DDoS  
IP Origen: 192.168.1.10, IP Destino: 192.168.1.30, Protocolo: UDP,  
Ataque: Escaneo de puertos
```

Solución (girar para leer)

```

1  from scapy.all import rdpcap, IP, TCP, UDP
2  from collections import defaultdict
3  import logging
4
5  # Configurar el logger para guardar los resultados en
6  logging.basicConfig(filename='análisis-trafico.log',
7                        level=logging.INFO, format='%(message)s')
8  # Función para detectar DDoS
9  def detectar_ddos(packets, umbral=100):
10     origen_ips = defaultdict(int)
11     for packet in packets:
12         if IP in packet:
13             origen_ips[packet[IP].src] += 1
14
15     ataques_ddos = []
16     for ip, count in origen_ips.items():
17         if count > umbral:
18             return ataques_ddos
19
20     # Función para detectar escaneos de puertos
21     def detectar_escaneo_puertos(packets, umbral=20):
22         puerto_origen = defaultdict(int)
23         for packet in packets:
24             if IP in packet and (TCP in packet or UDP in
25                                   packet):
26                 puerto_origen[packet[IP].src, packet[IP].
27                               puerto] += 1
28
29     escaneos = [(ip_src, ip_dst, puerto, count) for
30                  (ip_src, ip_dst, puerto, count) in
31                  puerto_origen.items()]
32     if count > umbral:
33         return escaneos
34
35     # Función para analizar el tráfico
36     def analizar_trafico(archivo_pcap):
37         try:
38             packets = rdpcap(archivo_pcap)
39             except Exception as e:
40                 print(f"Error al leer el archivo {archivo_pcap}
41                       {e}")
42         return

```

Código 11.16: Solución al Ejercicio 11.9.

Solución (girar para leer)

```

35
36 # Detectar DDoS
37 ddos_ips = detectar-ddos(packets)
38 for ip in ddos_ips:
39     logging.info(f"IP Origen: {ip}, Ataque: DDoS")
40
41 # Detectar escaneo de puertos
42 escaneos = detectar-escaneo-puertos(packets)
43 for ip-src, ip-dst, puerto, count in escaneos:
44     logging.info(f"IP Origen: {ip-src}, IP Destino:
45         {ip-dst}, Protocolo: {'TCP' if count > 0
46         else 'UDP'}, Ataque: Escaneo de puertos")
47     # Llamada a la función
48     analizar-trafico('trafico.pcap')
49     print("Análisis completado. Los resultados se guardaron
50         en 'analisis-trafico.log'.")

```

Código 11.17: Solución al Ejercicio 11.9 – continuación.

EJERCICIO 11.10

Escribe un programa en Python que implemente un sistema de autenticación en dos pasos utilizando un código temporal generado por la biblioteca `pyotp`. El sistema debe realizar lo siguiente:

1. Solicitar al usuario su nombre de usuario y contraseña.
2. Verificar las credenciales ingresadas con un archivo «`usuarios.csv`» que contiene las credenciales de los usuarios.
3. Si las credenciales son correctas, generar un código temporal (OTP) con `pyotp` y enviarlo al usuario (simulando el envío por correo o mensaje de texto).
4. Solicitar al usuario el código OTP y verificar si es correcto.

El programa ha de usar `pyotp` para generar el código OTP y asumir que las contraseñas se almacenan de forma segura (mediante un hash, usa el módulo `hashlib` en el programa).

Ejemplo de ejecución esperada

```
Introduce tu nombre de usuario: juan
Introduce tu contraseña: password123
Introduce el código OTP enviado: 123456
Autenticación exitosa.
```

Solución (girar para leer)

```

1  import csv
2  import pyotp
3  import hashlib
4  import getpass
5  import time
6
7  # Función para verificar las credenciales en el archivo
8  def verificar_credenciales(usuario, contraseña,
9                               archivo_usuarios):
10     try:
11         reader = csv.DictReader(f)
12         for row in reader:
13             if row['usuario'] == usuario:
14                 # Comparar la contraseña con el
15                 # hash almacenado
16                 if hashlib.sha256(contraseña.encode(
17                     'utf-8')).hexdigest() == row['
18                     contraseña']:
19                     return True
20             except FileNotFoundError:
21                 print(f"Error: El archivo {archivo_usuarios} no
22                     se encontró.")
23             except IOError e:
24                 print(f"Error de entrada/salida al leer {
25                     archivo_usuarios}: {e}")
26             except Exception e:
27                 print(f"Ocurrió un error inesperado al
28                     verificar las credenciales: {e}")
29         return False
30
31     # Función para generar y enviar el código OTP
32     def generar_otp():
33         # Usar una clave secreta para generar un código OTP
34         secret = pyotp.random_base32() # Genera una clave
35         secreta_alatoria
36         totp = pyotp.TOTP(secret)
37         print(f"Clave secreta (simulada para ejemplo): {
38             secret}")
39         # Solo para fines de demostración
40         otp = totp.now() # Generar el OTP
41         return otp, totp, secret

```

Código 11.18: Solución al Ejercicio 11.10.

Solución (girar para leer)

```

34 # Función principal para realizar la autenticación en
35     dos pasos
36 def autenticacion-dos-pasos():
37     try:
38         usuario = input("Introduce tu nombre de usuario
39         : ")
40         contrasena = getpass.getpass("Introduce tu
41         contrasena: ")
42         # Verificar las credenciales con el archivo '
43         usuarios.csv'
44         if not verificar-credenciales(usuario,
45         contrasena, 'usuarios.csv'):
46             print("Credenciales incorrectas.")
47             return
48         # Generar el OTP
49         otp, totp, secret = generar-otp()
50         # Simular el envío del código OTP al usuario
51         print(f"Código OTP enviado (simulado): {otp}")
52         tiempo-expiracion = time.time() + 30 # El OTP
53         es válido por 30 segundos
54         # Solicitar el código OTP
55         otp-usuario = input("Introduce el código OTP
56         enviado: ")
57         # Verificar si el código OTP es correcto y no
58         ha expirado
59         if time.time() > tiempo-expiracion:
60             print("El código OTP ha expirado.")
61         elif totp.verify(otp-usuario):
62             print("Autenticación exitosa.")
63         else:
64             print("Código OTP incorrecto.")
65         except Exception as e:
66             print(f"Ocurrió un error inesperado durante el
67             proceso de autenticación: {e}")
68         # Llamar a la función de autenticación
69         autenticacion-dos-pasos()

```

Código 11.19: Solución al Ejercicio 11.10 – continuación.