FP.057 - (P) Seguridad, procesos y zócalos

AA3. Programación de comunicaciones en red

Alumno: JAVIER DE LA VEGA EDER

Consultor: Lean Zabala Iglesias
CFGS en Desarrollo de Aplicaciones Multiplataforma(ICB02)
FP UOC - Fundació Jesuïtes Educació

Índice:

1.- Explicar el funcionamiento básico de un Malware tipo troyano. Explica el tipo de comunicación y conexión que se realiza.

Un **troyano es un tipo de malware que se camufla como un programa legítimo para engañar al usuario y ganar acceso no autorizado al sistema**. El troyano no se replica por sí mismo, pero una vez que el usuario lo ejecuta, puede abrir una puerta trasera en el sistema. La comunicación se establece a través de una conexión remota, permitiendo al atacante controlar la computadora infectada.

Esta comunicación puede ser realizada mediante la transmisión de datos a través de Internet o redes locales, permitiendo al atacante enviar comandos y recibir información sin el conocimiento del usuario.

2.- Explica la diferencia entre un Troyano y un Ransomware, pon al menos un ejemplo de cada uno.

Troyano: Un troyano **engaña al usuario haciéndose pasar por un programa legítimo**. Una vez ejecutado, permite a un atacante acceder al sistema y tomar control sin el conocimiento del usuario.

Ejemplo: **DarkComet** es un troyano de acceso remoto (RAT) que permite a un atacante controlar de manera remota la computadora infectada. Puede realizar diversas acciones, como capturar la pantalla, grabar el sonido, acceder a archivos y directorios, y ejecutar comandos arbitrarios en el sistema infectado. DarkComet ha sido utilizado con propósitos de espionaje y control malicioso.

Ransomware: El ransomware **cifra los archivos del usuario y exige un rescate para restaurar el acceso**. No busca el control total del sistema, sino el bloqueo de archivos cruciales.

Ejemplo: **CryptoLocker** fue uno de los primeros ransomware que se propagó masivamente. Descubierto en 2013, cifraba archivos en la computadora del usuario y exigía un pago en un plazo específico para proporcionar la clave de descifrado. Utilizaba técnicas de cifrado fuertes, lo que hacía extremadamente difícil recuperar los archivos sin pagar el rescate. CryptoLocker se distribuía principalmente a través de correos electrónicos de phishing y sitios web maliciosos.

3.- Explicar la diferencia entre multiproceso y multihilo.

Multiproceso: Implica la **ejecución simultánea de varios procesos independientes en un sistema operativo**. Cada proceso tiene su propio espacio de memoria y recursos, y la comunicación entre procesos se realiza a través de mecanismos de comunicación interprocesos (IPC).

Multihilo: Se refiere a la **ejecución simultánea de varios hilos dentro de un mismo proceso**. Los hilos comparten el mismo espacio de memoria y recursos, lo que facilita la comunicación entre ellos. Sin embargo, también puede llevar a problemas de concurrencia si no se manejan adecuadamente.

4.- Explicar la diferencia entre un thread daemon y non-daemon. Acompaña la explicación con un breve código de ejemplo.

Thread Daemon: Un thread daemon es un hilo que no evita que el programa finalice si todavía está en ejecución. En otras palabras, no impide que el programa principal termine su ejecución. Pueden realizarse tareas en segundo plano con threads daemon
.
Thread Non-Daemon: Un thread non-daemon, por otro lado, impide que el programa principal finalice hasta que este hilo haya completado su ejecución. Los threads no daemon son útiles cuando es necesario esperar a que ciertas tareas finalicen antes de que el programa principal termine.

Ejemplo:

```python
import threading
import time

def daemon_thread():
    while True:
        print("Daemon thread is running...")
        time.sleep(2)

def non_daemon_thread():
    for i in range(5):
        print(f"Non-daemon thread: {i}")
        time.sleep(1)

# Crear threads
daemon_thread = threading.Thread(target=daemon_thread)
non_daemon_thread = threading.Thread(target=non_daemon_thread)

# Marcar el thread como daemon
daemon_thread.daemon = True

# Iniciar los threads
daemon_thread.start()
non_daemon_thread.start()

# Esperar a que el thread no daemon termine antes de salir del programa principal
non_daemon_thread.join()

print("Done.")
```

```
javi@jvePython:~$ /bin/python3 /home/javi/Documents/Zocalos/Sample3.py
Daemon thread is running...
Non-daemon thread: 0
Non-daemon thread: 1
Daemon thread is running...
Non-daemon thread: 2
Non-daemon thread: 3
Daemon thread is running...
Non-daemon thread: 4
Done.
javi@jvePython:~$
```

Hilo Daemon:

El hilo daemon imprimirá continuamente el mensaje "Daemon thread is running..." cada 2 segundos en la consola.

Como este hilo está marcado como daemon, el programa principal no esperará a que termine antes de salir. Por lo tanto, cuando el programa principal termine, el hilo daemon también se detendrá automáticamente, incluso si estaba en medio de su ejecución.

Hilo No Daemon:

El hilo no daemon imprimirá "Non-daemon thread: 0", "Non-daemon thread: 1", ..., hasta "Non-daemon thread: 4" con un intervalo de 1 segundo entre cada impresión.

El programa principal esperará a que este hilo no daemon termine su ejecución antes de imprimir "Programa principal finalizado." y finalizar el programa.

5.- Ejecutar el programa "ACT3_EJ1_v2.py" disponible en el área de recursos del aula y explicar el funcionamiento del programa. Añadir un mecanismo de sincronización que permita a los procesos compartir el mismo espacio de memoria. Podéis utilizar la función "lock" y "acquire".

Script original:

Documents > Zocalos > ACT3_EJ1_v2.py > ...

```python
import time
import multiprocessing

def entrada(coches):
    #simulamos la entrada de 200 coches
    for i in range(200):
        time.sleep(0.01)
        coches.value = coches.value + 1

def salida(coches):
    #simulamos la salida de 200 coches
    for i in range(200):
        time.sleep(0.01)
        coches.value = coches.value - 1

if __name__ == '__main__':
    coches = multiprocessing.Value('i', 300)
    entrada_coche = multiprocessing.Process(target=entrada, args=(coches,))
    salida_coche = multiprocessing.Process(target=salida, args=(coches,))

    entrada_coche.start()
    salida_coche.start()

    entrada_coche.join()
    salida_coche.join()
```

Funciones de entrada y salida:
La función **entrada** simula la entrada de coches, incrementando el valor de la variable **coches** en 1 en un bucle que simula la entrada de 200 coches.
La función **salida** simula la salida de coches, decrementando el valor de la variable **coches** en 1 en un bucle que simula la salida de 200 coches.

Creación de procesos:
Se crea un objeto coches de tipo multiprocessing.Value con un valor inicial de 300.
Se crean dos procesos, uno para la entrada (**entrada_coche**) y otro para la salida (**salida_coche**), que ejecutarán las funciones respectivas y tendrán acceso a la variable coches compartida.

Inicio y espera de procesos:
Se inician ambos procesos. El programa principal espera a que ambos procesos (**entrada_coche** y **salida_coche**) terminen su ejecución mediante join.

Script modificado para printar resultados:

Documents > Zocalos > ACT3_EJ1_v2.py > ...

```python
import time
import multiprocessing

def entrada(coches):
    #simulamos la entrada de 200 coches
    for i in range(200):
        time.sleep(0.01)
        coches.value = coches.value + 1
        print(f"Entrada - Coches: {coches.value}")

def salida(coches):
    #simulamos la salida de 200 coches
    for i in range(200):
        time.sleep(0.01)
        coches.value = coches.value - 1
        print(f"Salida - Coches: {coches.value}")

if __name__ == '__main__':
    coches = multiprocessing.Value('i', 300)

    entrada_coche = multiprocessing.Process(target=entrada, args=(coches,))
    salida_coche = multiprocessing.Process(target=salida, args=(coches,))

    entrada_coche.start()
    salida_coche.start()

    entrada_coche.join()
    salida_coche.join()
```

Al ejecutar este código, vemos que los mensajes de impresión muestran el proceso de entrada y salida de coches. Sin embargo, debido a la falta de sincronización (**Lock**), podría haber condiciones de carrera que lleven a resultados impredecibles. Podemos observar valores inconsistentes si ambos procesos intentan modificar **coches.value** al mismo tiempo.

```
javi@jvePython:~$ /bin/python3 /home/javi/Documents/Zocalos/ACT3_EJ1_v2.py
Entrada - Coches: 301
Salida - Coches: 300
Entrada - Coches: 301
Salida - Coches: 300
Entrada - Coches: 301
Salida - Coches: 300
Entrada - Coches: 301
Salida - Coches: 300
Salida - Coches: 299
Entrada - Coches: 300
Entrada - Coches: 301
Salida - Coches: 300
Entrada - Coches: 301
Salida - Coches: 300
Salida - Coches: 299
Entrada - Coches: 300
Entrada - Coches: 301
Salida - Coches: 300
Salida - Coches: 299
Entrada - Coches: 300
Entrada - Coches: 301
Salida - Coches: 300
```

Código haciendo uso del contexto 'with lock':

Documents > Zocalos > ACT3_EJ1_v2.py > salida

```python
import time
import multiprocessing

def entrada(coches, lock):
    for i in range(200):
        time.sleep(0.05)
        with lock:
            coches.value += 1
            print(f"Entrada - Coches: {coches.value}")

def salida(coches, lock):
    for i in range(200):
        time.sleep(0.05)
        with lock:
            coches.value -= 1
            print(f"Salida - Coches: {coches.value}")

if __name__ == '__main__':
    coches = multiprocessing.Value('i', 300)
    lock = multiprocessing.Lock()

    entrada_coche = multiprocessing.Process(target=entrada, args=(coches, lock))
    salida_coche = multiprocessing.Process(target=salida, args=(coches, lock))

    entrada_coche.start()
    salida_coche.start()

    entrada_coche.join()
    salida_coche.join()
```

Con tal de garantizar una correcta simulación, también he incrementado el tiempo de espera de 0.01 a 0.05 y así poder visualizar el resultado correctamente:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Salida - Coches: 300
Entrada - Coches: 301
Salida - Coches: 300
Entrada - Coches: 301
Salida - Coches: 300
Entrada - Coches: 301
Salida - Coches: 300
Entrada - Coches: 301
Salida - Coches: 300
```

Con esta implementación, el **Lock** garantiza que solo un proceso pueda acceder y modificar **coches.value** en un momento dado, evitando condiciones de carrera y asegurando que los resultados sean consistentes y esperados.