PUCE Virtual

DOCUMENTO DE PROFUNDIZACIÓN

Estructura de Datos

Laboratorio de Contenido 2: Escalabilidad y Mantenibilidad en Sistemas de Ciberseguridad

Laboratorio de Contenido 2: Escalabilidad y Mantenibilidad en Sistemas de Ciberseguridad

Explicación básica

La escalabilidad permite que un sistema crezca (más usuarios, datos o eventos) sin perder rendimiento. La mantenibilidad asegura que el sistema pueda modificarse, actualizarse y corregirse fácilmente. En ciberseguridad, ambos conceptos son clave para adaptarse a nuevas amenazas y grandes volúmenes de información.

Objetivos

- Comprender los conceptos de escalabilidad y mantenibilidad.
- Diseñar un sistema adaptable y fácil de modificar.
- Aplicar buenas prácticas en código.
- Simular un sistema de análisis de seguridad eficiente.

Caso práctico

Una empresa comienza con pocos registros de seguridad, pero con el tiempo crece y necesita:

- Procesar miles de logs por minuto.
- Modificar reglas de detección fácilmente.
- Mantener el sistema sin afectar su funcionamiento.

Problema

El sistema actual:

- No soporta grandes volúmenes de datos.
- Tiene código rígido (difícil de modificar).
- No separa responsabilidades (todo está en un solo bloque).

Aplicación paso a paso

Paso 1: Diseño no escalable (problema inicial)

Código monolítico (difícil de mantener):

```
def analizar_logs(logs):
    for log in logs:
        if "FAILED" in log and "admin" in log:
            print("Alerta: posible ataque")
```

Problemas:

- No es reutilizable
- Difícil de escalar
- Difícil de modificar

Paso 2: Separación de responsabilidades (mantenibilidad)

```
def es_fallido(log):
    return "FAILED" in log

def es_admin(log):
    return "admin" in log

def generar_alerta():
    print("Alerta: posible ataque")

def analizar_logs(logs):
    for log in logs:
        if es_fallido(log) and es_admin(log):
            generar_alerta()
```

Mejora:

- Código modular
- Fácil de mantener

Paso 3: Uso de estructuras escalables
Convertimos logs en datos estructurados:

```
def parsear_log(log):
    partes = log.split(" - ")
    return {
        "fecha": partes[0],
        "evento": partes[1],
        "usuario": partes[2],
        "estado": partes[3]
    }
```

Paso 4: Procesamiento eficiente (escalabilidad)
Uso de procesamiento por lotes:

```
def procesar_lote(logs):
    return [parsear_log(log) for log in logs]
```

Paso 5: Reglas dinámicas (mantenibilidad + escalabilidad)

```
def regla_intentos_fallidos(log):
    return log["estado"] == "FAILED"

def regla_usuario_admin(log):
    return log["usuario"] == "admin"

reglas = [regla_intentos_fallidos, regla_usuario_admin]

def evaluar_log(log):
    return all(regla(log) for regla in reglas)
```

Paso 6: Escalabilidad con procesamiento concurrente

```
from concurrent.futures import ThreadPoolExecutor

def analizar_log(log):
    if evaluar_log(log):
        return f"Alerta en usuario {log['usuario']}"

def procesar_logs_concurrente(logs):
    with ThreadPoolExecutor() as executor:
        resultados = list(executor.map(analizar_log, logs))
    return [r for r in resultados if r]
```

Paso 7: Monitoreo y fácil actualización

Agregar nueva regla sin cambiar todo el sistema:

```
def regla_horario(log):
    hora = int(log["fecha"].split(" ")[1].split(":")[0])
    return hora < 6

reglas.append(regla_horario)
```

Conclusiones

- La escalabilidad se logra con procesamiento eficiente y estructuras adecuadas.
- La mantenibilidad se consigue con código modular y reglas desacopladas.
- Separar responsabilidades permite crecer sin rehacer el sistema.
- Un buen diseño facilita adaptarse a nuevas amenazas en ciberseguridad.