DOCUMENTO DE PROFUNDIZACIÓN

Estructura de Datos

Laboratorio de Contenido 1: Arquitectura de un Sistema de Análisis de Seguridad

Laboratorio de Contenido 1: Arquitectura de un Sistema de Análisis de Seguridad

Explicación básica

La arquitectura de un sistema de análisis de seguridad es el conjunto de componentes que permiten recopilar, procesar, analizar y responder a eventos de seguridad. Incluye herramientas como sistemas de monitoreo, recolección de logs, análisis de datos y respuesta ante incidentes. Su objetivo es detectar amenazas y proteger la infraestructura tecnológica de manera eficiente.

Objetivos

- Comprender la estructura de un sistema de análisis de seguridad.
- Identificar sus componentes principales (recolección, análisis y respuesta).
- Aplicar conceptos teóricos en un entorno práctico.
- Analizar eventos de seguridad para detectar posibles amenazas.

Caso práctico

Una empresa detecta accesos sospechosos a su servidor web fuera del horario laboral. Se requiere implementar un sistema que permita monitorear la red, analizar eventos y generar alertas ante posibles intrusiones.

Problema

La organización no cuenta con un sistema centralizado de análisis de seguridad, lo que dificulta:

- Detectar accesos no autorizados.
- Analizar registros de eventos (logs).
- Responder de manera oportuna a incidentes.

Aplicación paso a paso

Ejemplo de Logs iniciales (log.txt):

```
2026-03-20 22:15:32 - LOGIN - user1 - SUCCESS
2026-03-20 02:45:10 - LOGIN - admin - FAILED
2026-03-20 03:10:05 - LOGIN - admin - FAILED
2026-03-20 03:15:20 - LOGIN - admin - SUCCESS
```

Paso 1: Recolección de datos

- Configurar la captura de logs (servidores, firewall, red).

Ejemplo: Simulamos la lectura de logs desde un archivo:

```
def recolectar_logs(ruta):
  with open(ruta, "r") as archivo:
    logs = archivo.readlines()
  return logs

logs = recolectar_logs("logs.txt")
print(logs)
```

Paso 2: Centralización

- Integrar los registros en un sistema central (ej: SIEM).

Ejemplo: Convertimos los logs en una estructura organizada (lista de diccionarios):

```
def centralizar_logs(logs):
  datos = []
  for log in logs:
    partes = log.strip().split(" - ")
    datos.append({
      "fecha": partes[0],
      "evento": partes[1],
      "usuario": partes[2],
      "estado": partes[3]
    })
  return datos

logs_centralizados = centralizar_logs(logs)
print(logs_centralizados)
```

Paso 3: Análisis

- Definir reglas para detectar comportamientos anómalos (ej: accesos fuera de horario).

Ejemplo: Detectamos accesos fuera de horario (00:00 a 06:00):

```
from datetime import datetime

def analizar_logs(logs):
    sospechosos = []
    for log in logs:
        hora = datetime.strptime(log["fecha"], "%Y-%m-%d %H:%M:%S").hour
        if hora >= 0 and hora < 6:
            sospechosos.append(log)
```

```
    return sospechosos

eventos_sospechosos = analizar_logs(logs_centralizados)
print(eventos_sospechosos)
```

Paso 4: Detección

- Identificar patrones sospechosos o intentos de intrusión.

Ejemplo: Detectamos múltiples intentos fallidos de login.

```
def detectar_ataques(logs):
    intentos = {}
    alertas = []

    for log in logs:
        if log["estado"] == "FAILED":
            user = log["usuario"]
            intentos[user] = intentos.get(user, 0) + 1

            if intentos[user] >= 2:
                alertas.append(f"Posible ataque a usuario: {user}")

    return alertas

alertas = detectar_ataques(logs_centralizados)
print(alertas)
```

Paso 5: Respuesta

- Generar alertas y tomar acciones (bloqueo de IP, notificación).

Ejemplo: Simulamos una acción automática (bloquear usuario):

```
def responder(alertas):
    for alerta in alertas:
        print("ALERTA:", alerta)
        print("Acción: Bloquear acceso temporal\n")
```

```
    responder(alertas)
```

Paso 6: Monitoreo continuo

- Supervisar constantemente el sistema para prevenir nuevos ataques.

Ejemplo: Simulación de monitoreo en tiempo real:

```
import time

def monitoreo_continuo():
  while True:
    logs = recolectar_logs("logs.txt")
    logs_centralizados = centralizar_logs(logs)
    alertas = detectar_ataques(logs_centralizados)

    if alertas:
      responder(alertas)

    time.sleep(10)  # revisar cada 10 segundos

# monitoreo_continuo()  # (descomentar para ejecutar)
```

Conclusiones

- Una arquitectura bien definida permite detectar amenazas de forma oportuna.
- La centralización y análisis de datos son claves en la ciberseguridad.
- La automatización mejora la capacidad de respuesta ante incidentes.
- Implementar estos sistemas fortalece la protección de la información y reduce riesgos.