PUCE Virtual

CLASE

8

# ESTRUCTURA DE DATOS (PROGRAMACIÓN III)

## Integración de estructuras y proyecto final

QS WORLD UNIVERSITY RANKINGS

Educación de calidad

# INTRODUCCIÓN DE LA CLASE

GRADO / POSGRADO / TECNOLOGÍAS

Estudia a tu tiempo
son interrupciones

EDUCACIÓN EN LÍNEA DE CALIDAD

En esta última semana de clases se abordará el desarrollo del proyecto final, enfocado en la integración de estructuras de datos y algoritmos dentro de un sistema funcional aplicado a un escenario realista de ciberseguridad. A lo largo de la asignatura, se han estudiado conceptos fundamentales como listas, árboles, grafos, algoritmos de búsqueda y ordenamiento, los cuales ahora deben ser combinados de manera coherente para resolver un problema concreto. El objetivo principal es que los estudiantes comprendan cómo estos elementos interactúan entre sí dentro de una arquitectura de software, permitiendo el análisis eficiente de datos y la detección de posibles amenazas en entornos tecnológicos.

Asimismo, se profundizará en aspectos clave como la validación técnica del diseño, el análisis algorítmico y la justificación de las decisiones tomadas durante el desarrollo del sistema. Se enfatizará la importancia de aplicar criterios de eficiencia computacional, escalabilidad y buenas prácticas de ingeniería de software. Además, los estudiantes se prepararán para la presentación y defensa de su proyecto, desarrollando habilidades de comunicación y argumentación técnica. Esta clase representa la culminación del proceso de aprendizaje, permitiendo consolidar conocimientos y aplicarlos en una solución práctica orientada a los desafíos actuales de la ciberseguridad.

**Semana 8:**
Seleccionar las estructuras de datos más adecuadas según los requerimientos y características de distintos escenarios de ciberseguridad.

## 8. Integración de estructuras y proyecto final

Para el desarrollo del proyecto final se requiere realizar una integración adecuada y coherente de las diferentes estructuras de datos estudiadas, tales como listas, árboles y grafos, enfocándose en la optimización de los algoritmos implementados. Este proceso implica no solo utilizar dichas estructuras, sino también seleccionar las más apropiadas según el problema de ciberseguridad planteado, considerando criterios como eficiencia computacional, uso de memoria y capacidad de escalabilidad.

Se espera que los estudiantes apliquen de manera práctica todos los conocimientos adquiridos a lo largo de la asignatura, incluyendo algoritmos de búsqueda y ordenamiento, análisis de complejidad y buenas prácticas de ingeniería de software. La integración debe reflejar un diseño estructurado, modular y justificado técnicamente, permitiendo construir un sistema funcional, eficiente y alineado con un escenario realista de ciberseguridad, capaz de analizar información, detectar amenazas y apoyar la toma de decisiones.

### 8.1. Integración de estructuras de datos

Proceso clave en el diseño de sistemas complejos, especialmente en el ámbito de la ciberseguridad, donde es necesario gestionar grandes volúmenes de información de manera eficiente. Esta integración implica combinar diferentes estructuras como listas, árboles, tablas hash y grafos para aprovechar sus ventajas específicas y cubrir distintas necesidades del sistema. Por ejemplo, las listas pueden utilizarse para almacenar eventos de seguridad, mientras que los grafos permiten modelar relaciones entre dispositivos o usuarios.

La correcta integración no solo mejora el rendimiento, sino que también facilita la organización y el análisis de los datos (Sánchez, 2020). Además, este enfoque permite diseñar soluciones más flexibles y adaptables, capaces de responder a escenarios dinámicos y cambiantes. Esto es fundamental para detectar amenazas en tiempo real y correlacionar eventos de distintas fuentes. Una integración adecuada también contribuye a reducir redundancias y optimizar el uso de recursos, lo que se traduce en sistemas más eficientes y escalables (Pérez & Gómez, 2021). En definitiva, la combinación estratégica de estructuras de datos permite construir soluciones robustas que mejoran la capacidad de análisis y respuesta ante incidentes de seguridad (Martínez, 2019).

#### 8.1.1. Integración de búsqueda, ordenamiento, árboles y grafos

La integración de algoritmos de búsqueda, ordenamiento y estructuras como árboles y grafos es fundamental para el desarrollo de sistemas eficientes en ciberseguridad. Los algoritmos de búsqueda permiten localizar información relevante dentro de grandes volúmenes de datos, mientras que los algoritmos de ordenamiento facilitan la organización de estos datos para un análisis más rápido y estructurado. Por su parte, los árboles permiten representar jerarquías,

como sistemas de permisos o clasificación de amenazas, y los grafos modelan relaciones complejas entre entidades, como redes de comunicación (Sánchez, 2020).

La combinación de estos elementos permite construir sistemas capaces de analizar eventos de seguridad desde múltiples perspectivas, mejorando la detección de patrones anómalos. Por ejemplo, se puede utilizar un árbol para clasificar tipos de ataques, mientras que un grafo permite identificar cómo se propagan dentro de la red. Esta integración también contribuye a optimizar el rendimiento del sistema, ya que cada componente cumple una función específica dentro del proceso de análisis (Pérez & Gómez, 2021).

En consecuencia, el uso conjunto de estas herramientas permite desarrollar soluciones más completas y eficientes, capaces de enfrentar los desafíos actuales en ciberseguridad (Martínez, 2019). La Tabla 1 muestra los componentes del proyecto final.

**Tabla 1. Componentes del proyecto final. Creación de autor Patricio Coba**

| Componente | Recomendación de Integración | Impacto en Ciberseguridad |
|---|---|---|
| **Estructuras de Datos** | **Árboles AVL** para gestionar diccionarios de contraseñas o listas blancas. Los árboles balanceados garantizan búsquedas en tiempo $O(\log n)$. | Evita ataques de "Algorithmic Complexity" donde un atacante satura el sistema con datos que degradan el rendimiento. |
| **Algoritmos de Ordenamiento** | **QuickSort o MergeSort** para organizar registros de logs por timestamp o nivel de severidad antes de analizarlos. | Facilita la detección de anomalías al tener datos estructurados y listos para procesos de auditoría. |
| **Grafos** | Modelar la red de infraestructura como un grafo donde los nodos son dispositivos y las aristas son conexiones. Usa **Dijkstra** para encontrar la ruta de datos más corta. | Permite identificar "puntos únicos de fallo" y analizar la propagación de malware en una red. |
| **Búsqueda** | Emplear **Búsqueda Binaria** sobre arreglos ordenados o tablas Hash para verificar firmas digitales o tokens de acceso de forma instantánea. | Reduce la latencia en procesos de autenticación, mejorando la experiencia de usuario sin sacrificar seguridad. |
| **Optimización** | Analizar la **Complejidad Temporal ($O$)**. Evita bucles anidados innecesarios ($O(n^2)$) en funciones críticas de cifrado o validación. | Un código optimizado es menos propenso a ataques de temporización (timing attacks) y consume menos recursos de hardware. |

### 8.1.2. Arquitectura del sistema de análisis de seguridad

La arquitectura de un sistema de análisis de seguridad define la forma en que se organizan e interactúan los diferentes componentes encargados de procesar, almacenar y analizar información relacionada con eventos de seguridad. Este tipo de arquitectura suele estar compuesta por módulos especializados, como recolección de datos, procesamiento, almacenamiento y análisis, los cuales trabajan de manera integrada para garantizar un funcionamiento eficiente.

Las estructuras de datos juegan un papel fundamental ya que permiten organizar la información de forma adecuada según su naturaleza y uso (Sánchez, 2020). Por ejemplo, las tablas hash pueden utilizarse para búsquedas rápidas, mientras que los grafos permiten analizar relaciones entre eventos o dispositivos. Una arquitectura bien diseñada facilita la detección de amenazas, la correlación de eventos y la toma de decisiones en tiempo real (Pérez & Gómez, 2021). Además, debe ser flexible y escalable para adaptarse a cambios en el entorno y al crecimiento de los datos.

Esto es especialmente importante debido a la constante evolución de las amenazas. Por tanto, una arquitectura adecuada no solo mejora el rendimiento del sistema, sino que también fortalece la capacidad de respuesta ante incidentes (Martínez, 2019).

Figura 1 presenta la arquitectura del sistema de análisis de seguridad.


**Figura 1. Arquitectura de un sistema de análisis de seguridad. Creación de autor Patricio Coba**

¿Qué es la arquitectura de seguridad? Página web de Palo Alto Networks que presenta conceptos, ventajas de la arquitectura de seguridad, sus marcos y normas, mejores prácticas, entre otras. Enlace: https://www.paloaltonetworks.lat/cyberpedia/what-is-security-architecture#benefits

### 8.1.3. Evaluación de eficiencia global

La evaluación de la eficiencia global de un sistema es un proceso que permite medir su rendimiento considerando factores como el tiempo de respuesta, el uso de memoria y la capacidad de procesamiento. Esta evaluación es crucial, ya que los sistemas deben analizar grandes volúmenes de datos en tiempo real sin comprometer la precisión en la detección de amenazas. Para ello, es necesario analizar el comportamiento de las estructuras de datos y algoritmos utilizados, identificando posibles cuellos de botella y oportunidades de mejora (Sánchez, 2020). Por ejemplo, el uso de grafos puede ofrecer un análisis más detallado de las relaciones, pero también puede aumentar la complejidad computacional.

Por otro lado, estructuras como tablas hash permiten búsquedas rápidas, optimizando el rendimiento del sistema. La evaluación de eficiencia también implica realizar pruebas y simulaciones que permitan validar el funcionamiento del sistema en diferentes escenarios (Pérez & Gómez, 2021). De esta manera, se pueden tomar decisiones informadas para optimizar el diseño y garantizar un equilibrio adecuado entre rendimiento y consumo de recursos. En definitiva, una evaluación adecuada contribuye a mejorar la eficacia del sistema y su capacidad para enfrentar amenazas (Martínez, 2019).
La Figura 2 muestra tipos de evaluación de eficiencia de algoritmos para estructuras de datos.

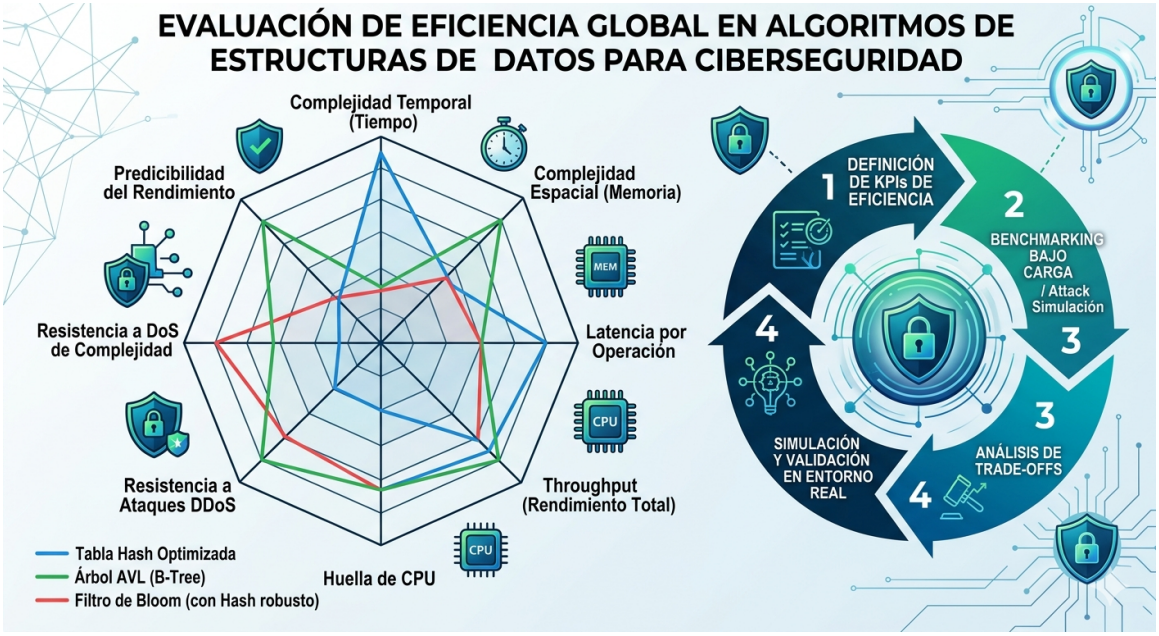



**Figura 2. Eficiencia de algoritmos. Creación de autor Patricio Coba**

### 8.1.4. Escalabilidad y mantenibilidad

La escalabilidad y mantenibilidad son aspectos fundamentales en el diseño de sistemas de ciberseguridad, ya que garantizan que la solución pueda adaptarse al crecimiento de los datos y a cambios en los requerimientos sin perder eficiencia. La escalabilidad se refiere a la capacidad del sistema para manejar un aumento en la carga de trabajo, mientras que la mantenibilidad implica la facilidad con la que el sistema puede ser modificado o actualizado (Sánchez, 2020). La elección de estructuras de datos y algoritmos influye directamente en ambos aspectos. Por ejemplo, estructuras eficientes como las tablas hash o los grafos bien diseñados permiten manejar grandes volúmenes de información sin degradar el rendimiento. Asimismo, un diseño modular facilita la implementación de mejoras y la corrección de errores (Pérez & Gómez, 2021).

Esto es especialmente importante debido a la evolución constante de las amenazas, lo que requiere sistemas flexibles y actualizables. Además, una buena mantenibilidad reduce costos y tiempos de desarrollo a largo plazo. Por lo tanto, considerar estos factores desde la etapa de diseño permite construir soluciones robustas, eficientes y sostenibles en el tiempo (Martínez, 2019).

La Figura 3 muestra aspectos de escalabilidad y mantenibilidad en el diseño de sistemas de ciberseguridad.


**Figura 3. Aspectos de escalabilidad y mantenibilidad en el diseño de sistemas de ciberseguridad. Creación de autor Patricio Coba**

Escalabilidad: bases y aplicación a sistemas, equipos y procesos | DevOpsDays Cáceres 2020 Video subido por DevOpsDays Cáceres que reproduce una parte de las conferencias DevOpsDays donde se presenta una introducción a las teorías que nos permiten entender la escalabilidad (ley universal de escalabilidad, teoría de colas, teoría de las restricciones, ley de Conway, ley de Andahl, etc.) y cómo afectan a nuestros sistemas (software y organización de equipos). Enlace: https://www.youtube.com/watch?v=sd_2VSL2vGs

### 8.1.5. Preparación del proyecto final

La preparación del proyecto final implica la aplicación práctica de los conocimientos adquiridos sobre estructuras de datos, algoritmos y ciberseguridad en el desarrollo de una solución funcional. En esta etapa, los estudiantes deben diseñar un sistema que integre diferentes estructuras, como listas, árboles y grafos, junto con algoritmos de búsqueda y ordenamiento, para resolver un problema realista de seguridad informática. Este proceso requiere no solo habilidades técnicas, sino también la capacidad de justificar cada decisión de diseño desde un enfoque teórico y práctico. Además, es importante considerar aspectos como la eficiencia, la escalabilidad y la mantenibilidad del sistema, asegurando que la solución sea viable a largo plazo. La documentación del proyecto también juega un papel clave, ya que permite explicar el funcionamiento del sistema y las razones detrás de cada elección. Este tipo de proyectos contribuye al desarrollo de competencias para enfrentar problemas reales. En consecuencia, la preparación adecuada del proyecto final permite consolidar los conocimientos y desarrollar soluciones innovadoras y eficientes.

## 8.2. Proyecto final y defensa

El proyecto final consiste en el desarrollo de un sistema funcional orientado a un escenario realista de ciberseguridad, en el cual se integran algoritmos de búsqueda, algoritmos de ordenamiento, árboles y grafos. Es fundamental que el sistema esté diseñado bajo criterios de eficiencia computacional, buenas prácticas de ingeniería de software y una adecuada justificación técnica de cada decisión. Además, la defensa del proyecto implica explicar claramente el funcionamiento del sistema, las tecnologías utilizadas y el impacto de la solución en la detección o prevención de amenazas. La Tabla 2 muestra las fases del proyecto final.

**Tabla 2. Fases del proyecto. Creación de autor Patricio Coba**

| Fase | Actividades | Entregable |
|---|---|---|
| **1. Definición del Escenario** | Identificar un problema de seguridad real. | Documento de alcance y caso de uso. |
| **2. Diseño de la Arquitectura** | Seleccionar las estructuras adecuadas: Listas, Árboles y Grafos. | Diagrama de flujo de datos y modelo de clases. |

| | | |
|---|---|---|
| **3. Selección de Algoritmos** | Elegir métodos de búsqueda y ordenamiento para procesar la información. | Pseudocódigo de los procesos críticos. |
| **4. Desarrollo** | Implementar las estructuras de datos seleccionadas. | Prototipo funcional del backend. |
| **5. Pruebas de Estrés y Seguridad** | Someter al sistema a cargas pesadas para verificar que los algoritmos de búsqueda no se degraden. | Matriz de resultados de pruebas y tiempos de ejecución. |
| **6. Documentación y Defensa** | Redactar la memoria técnica. | Presentación final y manual técnico. |

### 8.2.1. Integración completa del sistema

La integración completa del sistema implica unificar todos los componentes desarrollados en una solución coherente y funcional. En este contexto, los estudiantes deben asegurar que los algoritmos de búsqueda permitan localizar información relevante de manera eficiente, mientras que los algoritmos de ordenamiento optimicen la organización de los datos, como logs o eventos de seguridad.

Los árboles deben utilizarse para representar estructuras jerárquicas, como permisos o clasificación de amenazas, y los grafos para modelar relaciones complejas, como conexiones de red o rutas de ataque. Esta integración debe realizarse bajo un enfoque modular, garantizando que cada componente interactúe correctamente con los demás. Además, es importante validar que el sistema funcione de manera conjunta sin errores, asegurando consistencia, rendimiento y escalabilidad.

### 8.2.2. Validación técnica del diseño

La validación técnica del diseño consiste en verificar que las decisiones tomadas durante el desarrollo del sistema estén correctamente fundamentadas y cumplan con los objetivos del proyecto. Esto implica evaluar si las estructuras de datos seleccionadas son las más adecuadas para el problema planteado, considerando factores como eficiencia, consumo de memoria y facilidad de implementación. Asimismo, se debe comprobar que los algoritmos utilizados responden de manera efectiva a los requerimientos del sistema, como la detección de amenazas en tiempo real. La validación también incluye pruebas funcionales y de rendimiento, que permitan identificar posibles fallos o áreas de mejora. En este sentido, los estudiantes deben ser capaces de justificar técnicamente cada componente del sistema, demostrando que su diseño no es arbitrario, sino resultado de un análisis riguroso. La Tabla 3 muestra diferentes tipos de pruebas de software.

Tabla 3. Pruebas de software. Creación de autor Patricio Coba

| Prueba | Validación | Resultado Esperado |
|---|---|---|
| **Pruebas Unitarias** | Verificación de funciones individuales. | Cada estructura debe mantener su integridad. |
| **Pruebas de Límite** | Comportamiento con estructuras vacías, un solo nodo o valores duplicados. | El sistema no debe arrojar excepciones |
| **Pruebas de Carga** | Inserción masiva de datos. | El tiempo de respuesta debe crecer de forma logarítmica o lineal. |
| **Pruebas de Estrés** | Saturación de la memoria RAM mediante la creación de nodos en el grafo hasta el límite del sistema. | Identificar el punto de ruptura y asegurar que el sistema falle de forma segura. |
| **Pruebas de Inyección** | Intentar insertar datos malformados o scripts en los campos que alimentan las estructuras de datos. | Las estructuras deben rechazar entradas que no cumplan con el formato, evitando desbordamientos de memoria. |

### 8.2.3. Análisis algorítmico del sistema

El análisis algorítmico del sistema es una etapa clave en la que se evalúa el comportamiento de los algoritmos utilizados en términos de complejidad temporal y espacial. En el contexto del proyecto, esto implica analizar la eficiencia de los algoritmos de búsqueda y ordenamiento, así como el rendimiento de las estructuras de datos como árboles y grafos.

Los estudiantes deben identificar el costo computacional de las operaciones principales, como inserción, búsqueda y recorrido, y evaluar su impacto en el desempeño del sistema. Este análisis permite detectar posibles cuellos de botella y optimizar el funcionamiento del sistema, garantizando que pueda manejar grandes volúmenes de datos de manera eficiente. Además, contribuye a justificar las decisiones de diseño desde un enfoque técnico, demostrando que el sistema cumple con los criterios de eficiencia establecidos.

### 8.2.4. Presentación y defensa del proyecto

La presentación y defensa del proyecto es la etapa en la que los estudiantes exponen su solución ante un público o evaluador, demostrando tanto el funcionamiento del sistema como la lógica detrás de su diseño.  En esta fase, es fundamental explicar de manera clara y estructurada cómo se integraron los algoritmos y estructuras de datos, así como el problema de ciberseguridad que se busca resolver.

La defensa debe incluir una demostración del sistema, evidenciando su capacidad para detectar, analizar o prevenir amenazas. Asimismo, los estudiantes deben estar preparados para responder

preguntas técnicas, justificando sus decisiones y mostrando dominio del tema. Esta etapa no solo evalúa el producto desarrollado, sino también habilidades de comunicación, argumentación y pensamiento crítico.  La Tabla 4 muestra subfases de la presentación y defensa del proyecto final.

**Tabla 4. Subfases de la presentación y defensa del proyecto. Creación de autor Patricio Coba**

| Fase | Recomendación |
|---|---|
| **Introducción** | Definir el vector de ataque o el problema de seguridad que tu proyecto resuelve. |
| **Arquitectura** | Usar diagramas claros para mostrar cómo fluyen los datos a través de tus Árboles o Grafos. |
| **Demostración** | Realizar una "Prueba de Estrés" en vivo mostrando cómo el algoritmo maneja grandes volúmenes de datos. |
| **Análisis de Complejidad** | Incluir una diapositiva con la Notación Big O ($O(n \log n)$, $O(n)$, etc.) de las funciones críticas. |
| **Seguridad** | Explicar cómo se protegen las estructuras (ej. evitar desbordamientos de búfer o fugas de memoria). |
| **Conclusión** | Resaltar la escalabilidad: ¿Qué pasaría si el grafo tuviera 1 millón de nodos? |

### 8.2.5. Reflexión final desde la ciberseguridad

La reflexión final permite analizar el aprendizaje obtenido durante el desarrollo del proyecto, así como el impacto de la solución en el ámbito de la ciberseguridad. Los estudiantes deben evaluar qué tan efectiva fue su propuesta para abordar el problema planteado, identificando fortalezas, limitaciones y posibles mejoras.

Es importante reflexionar sobre la importancia de seleccionar adecuadamente estructuras de datos y algoritmos, ya que estas decisiones influyen directamente en la capacidad del sistema para detectar amenazas y responder a incidentes. Esta etapa también fomenta una visión crítica sobre los desafíos actuales de la ciberseguridad, como el manejo de grandes volúmenes de datos y la evolución constante de los ataques.

## Definición de los términos citados en la Semana 8

**Arquitectura de un sistema de análisis de seguridad.** Estructura organizada de componentes, herramientas y procesos que permiten recolectar, procesar, analizar y responder a eventos de seguridad dentro de una infraestructura tecnológica, con el fin de detectar amenazas, prevenir ataques y proteger la información.

**Escalabilidad y mantenibilidad.** Escalabilidad se refiere a la capacidad de un sistema de ciberseguridad para adaptarse al crecimiento en volumen de datos, usuarios o eventos sin perder rendimiento, permitiendo ampliar recursos o funcionalidades de forma eficiente, mientras que mantenibilidad es la facilidad con la que un sistema puede ser modificado, actualizado o corregido, asegurando que los cambios se realicen de manera rápida, segura y sin afectar su funcionamiento.

Educación de calidad

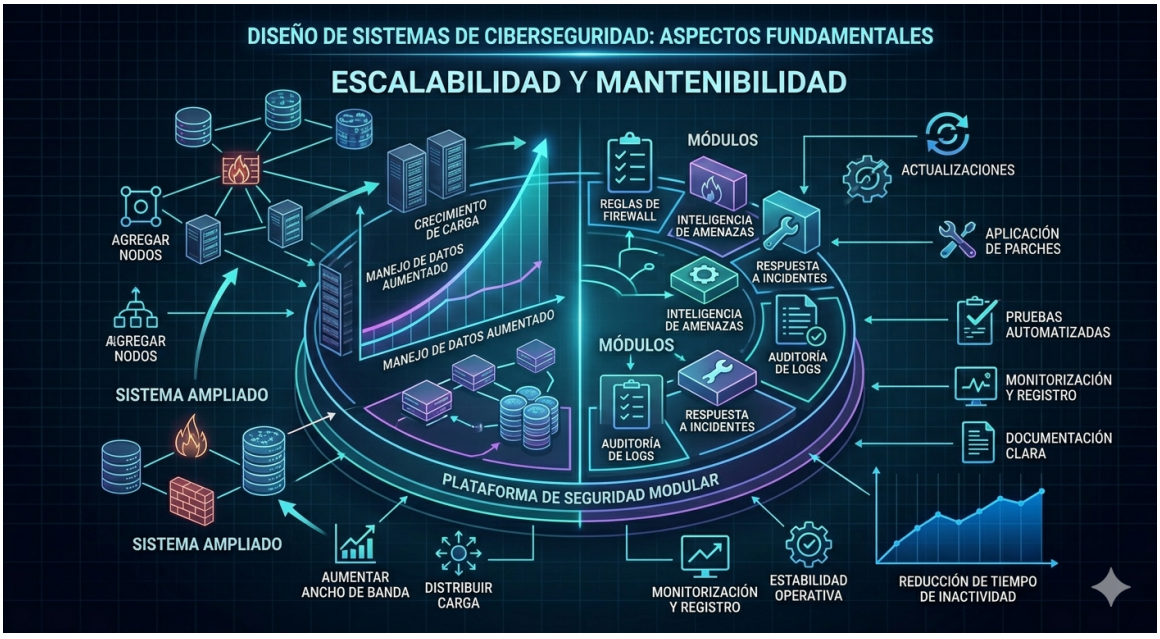
# REFERENCIAS


**Martínez, L. (2019).** Fundamentos de redes informáticas y seguridad. Editorial Alfaomega.

**Pérez, J., & Gómez, R. (2021).** Ciberseguridad: análisis y protección de infraestructuras digitales. Editorial Ra-Ma.

**Sánchez, M. (2020).** Estructuras de datos y algoritmos aplicados a redes. Editorial Paraninfo.


Diseño de sistemas de ciberseguridad: Aspectos fundamentales. Gráfico resumido sobre aspectos fundamentales de escalabilidad y mantenibilidad para el diseño de sistemas de ciberseguridad, incluye el aumento de carga para la escalabilidad y técnicas de mantenibilidad.



Metodología aplicada al proyecto final. Infografía que muestra la aplicación de la metodología aplicada al proyecto final con cada una de as actividades a desarrollar.

**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.

**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.

**Laboratorio de Contenido 3:  Análisis Algorítmico del Sistema**

**Explicación básica**

El análisis algorítmico evalúa la eficiencia de los algoritmos en términos de tiempo y uso de recursos. En ciberseguridad, esto es clave para procesar grandes volúmenes de datos (logs, eventos, redes). Se enfoca en algoritmos de búsqueda, ordenamiento y estructuras como árboles y grafos, analizando su rendimiento mediante conceptos como la complejidad temporal (Big-O).

**Objetivos**

- Comprender la eficiencia de algoritmos de búsqueda y ordenamiento.
- Analizar el rendimiento de estructuras de datos (árboles y grafos).
- Comparar diferentes enfoques algorítmicos.
- Aplicar estos conceptos en un contexto de ciberseguridad.

**Caso práctico**

Una empresa necesita analizar miles de eventos de seguridad para:

- Buscar accesos sospechosos rápidamente.
- Ordenar eventos por prioridad o fecha.
- Analizar conexiones entre dispositivos (red).

**Problema**

El sistema actual:

- Usa búsqueda lineal (lenta con muchos datos).
- No ordena eficientemente los eventos.
- No analiza relaciones entre dispositivos.

**Aplicación paso a paso**

Paso 1: Búsqueda lineal (ineficiente)

```
def busqueda_lineal(logs, usuario):
    for log in logs:
        if log["usuario"] == usuario:
            return log
    return None
```

Complejidad: **O(n)**, lenta con grandes volúmenes

Paso 2: Búsqueda binaria (optimizada)

Requiere datos ordenados:

```
def busqueda_binaria(logs, usuario):
    izquierda, derecha = 0, len(logs) - 1

    while izquierda <= derecha:
```

```
        medio = (izquierda + derecha) // 2
        if logs[medio]["usuario"] == usuario:
            return logs[medio]
        elif logs[medio]["usuario"] < usuario:
            izquierda = medio + 1
        else:
            derecha = medio - 1

    return None
```

Complejidad: **O(log n)**, mucho más rápida

Paso 3: Ordenamiento (eficiencia del sistema)

```
def ordenar_logs(logs):
    return sorted(logs, key=lambda x: x["fecha"])
```

Complejidad: **O(n log n)**, permite búsquedas más eficientes

Paso 4: Uso de árboles (búsqueda eficiente)

```
class Nodo:
    def __init__(self, valor):
        self.valor = valor
        self.izquierda = None
        self.derecha = None

def insertar(raiz, valor):
    if raiz is None:
        return Nodo(valor)
    if valor < raiz.valor:
        raiz.izquierda = insertar(raiz.izquierda, valor)
    else:
        raiz.derecha = insertar(raiz.derecha, valor)
    return raíz
```

Búsqueda promedio: **O(log n)**, puede degradarse a O(n) si no está balanceado

Paso 5: Uso de grafos (análisis de red)

Representamos conexiones entre dispositivos:

```
grafo = {
    "PC1": ["Router"],
    "Router": ["PC1", "Servidor"],
    "Servidor": ["Router"]
}
```

Búsqueda en anchura (BFS):

```
from collections import deque

def bfs(grafo, inicio):
    visitados = set()
    cola = deque([inicio])

    while cola:
        nodo = cola.popleft()
        if nodo not in visitados:
            print(nodo)
            visitados.add(nodo)
            cola.extend(grafo[nodo])
```

Complejidad: **O(V + E)**

Paso 6: Comparación de eficiencia

| Método | Complejidad | Uso en ciberseguridad |
|---|---|---|
| Búsqueda lineal | O(n) | Datos pequeños |
| Búsqueda binaria | O(log n) | Logs ordenados |
| Ordenamiento | O(n log n) | Preparar datos |
| Árboles | O(log n) | Índices de búsqueda |
| Grafos (BFS/DFS) | O(V+E) | Redes y ataques |

**Conclusiones**

- La eficiencia algorítmica es clave en sistemas de ciberseguridad.
- Elegir el algoritmo correcto mejora el rendimiento significativamente.
- Las estructuras como árboles y grafos permiten análisis más avanzados.
- Un mal diseño puede volver lento e ineficiente todo el sistema.

**Estudio de caso 1: Preparación de un Proyecto para implementar un sistema de Análisis de Seguridad**

**Contexto**

Una institución educativa en Quito ha incrementado el uso de plataformas digitales (aulas virtuales, sistemas administrativos y redes internas). Esto ha generado un aumento en la cantidad de datos y eventos de seguridad, sin contar con un sistema eficiente para su monitoreo y análisis.

**Problema**

La institución presenta varias debilidades:

- No existe un sistema centralizado de monitoreo de seguridad.
- Los registros (logs) no se analizan en tiempo real.
- Se han detectado accesos sospechosos sin una respuesta oportuna.
- El sistema actual no es escalable ni fácil de mantener.

**Solución Propuesta**

Diseñar e implementar un sistema de análisis de seguridad basado en:

- Recolección y centralización de logs.
- Análisis algorítmico eficiente (búsqueda, detección de patrones).
- Uso de grafos para representar relaciones en la red.
- Arquitectura modular que garantice escalabilidad y mantenibilidad.

**Estrategia:**

- Implementar el sistema de forma incremental (por módulos).
- Priorizar la detección de accesos no autorizados.
- Utilizar estructuras de datos eficientes (listas, árboles, grafos).
- Incorporar reglas dinámicas para facilitar actualizaciones.
- Validar cada fase mediante pruebas funcionales.

**Implementación**

Para la implementación de la solución se lo realiza en 6 fases:

**Fase 1:** Recolección de logs desde servidores y red.
**Fase 2:** Centralización de datos en un sistema único.
**Fase 3:** Análisis mediante algoritmos eficientes (búsqueda y detección).
**Fase 4:** Modelado de la red con grafos para identificar patrones de ataque.

**Fase 5:** Generación de alertas automáticas.
**Fase 6:** Monitoreo continuo y mejora del sistema.

Se utiliza **Python** para el desarrollo inicial, con código modular y escalable.
**Resultados e Impacto en ciberseguridad**

Una vez ejecutadas las fases de implementación se obtienen los siguientes resultados enfocados a la ciberseguridad.

- Reducción del tiempo de detección de incidentes.
- Identificación de accesos sospechosos en tiempo casi real.
- Mejor organización y análisis de los datos de seguridad.
- Mayor capacidad de adaptación ante nuevas amenazas.
- Base sólida para implementar soluciones más avanzadas (SIEM)..

**Reflexión Técnica**

Este proyecto demuestra que:

- Un buen diseño arquitectónico es clave para la eficiencia del sistema.
- La elección de algoritmos impacta directamente en el rendimiento.
- La escalabilidad y mantenibilidad son esenciales en entornos reales.
- La integración de teoría (algoritmos, estructuras) con práctica permite soluciones efectivas en ciberseguridad.

**ESTUDIO DE CASO 2: Implementación de un Sistema de Análisis de Seguridad en una Empresa Comercial**

**Contexto**

Una empresa mediana del sector comercial ha incrementado su infraestructura digital, incluyendo sistemas de ventas, bases de datos de clientes y servicios en la nube. Con este crecimiento, también ha aumentado su exposición a amenazas como accesos no autorizados, ataques de fuerza bruta y filtración de información.

**Problema**

La empresa enfrenta varias limitaciones:

- No cuenta con monitoreo continuo de eventos de seguridad.
- Los logs se almacenan, pero no se analizan.
- No existe detección temprana de ataques.
- El sistema es poco escalable ante el crecimiento de datos.

Esto genera riesgos de pérdida de información y afectación a la continuidad del negocio.

**Solución Propuesta**

Desarrollar un sistema de análisis de seguridad como proyecto final que incluya:

- Recolección y centralización de logs.
- Análisis algorítmico eficiente (búsqueda, ordenamiento).
- Detección de patrones sospechosos.
- Modelado de red mediante grafos.
- Arquitectura escalable y mantenible.

**Estrategia:**

- Crear el proyecto de ciberseguridad
- Implementación progresiva por módulos (recolección, análisis, respuesta).
- Uso de algoritmos eficientes para manejar grandes volúmenes de datos.
- Integración de reglas dinámicas para detectar amenazas.
- Pruebas constantes para validar el funcionamiento del sistema.

**Implementación**

**Fase 1:** Captura de logs de servidores y sistemas críticos.
**Fase 2:** Centralización de la información en una estructura organizada.
**Fase 3:** Aplicación de algoritmos de búsqueda y análisis para detectar anomalías.

**Fase 4:** Representación de la red mediante grafos para identificar relaciones y posibles rutas de ataque.
**Fase 5:** Generación de alertas automáticas ante eventos sospechosos.
**Fase 6:** Monitoreo continuo y mejora del sistema.

**Resultados**

- Detección temprana de intentos de intrusión.
- Reducción del tiempo de respuesta ante incidentes.
- Mejor control y visibilidad de la seguridad de la red.
- Optimización del uso de recursos tecnológicos.
- Mayor confianza en la protección de datos sensibles.

**Impacto en ciberseguridad**

Este tipo de proyectos permite a las empresas:

- Prevenir ataques antes de que causen daño.
- Tomar decisiones basadas en datos reales.
- Escalar sus sistemas de seguridad conforme crecen.
- Reducir costos asociados a incidentes de seguridad.

**Reflexión Técnica**

El desarrollo de un sistema de análisis de seguridad demuestra que:

- La combinación de algoritmos eficientes y buenas estructuras de datos mejora el rendimiento.
- La escalabilidad y mantenibilidad son claves en entornos empresariales.
- Los proyectos académicos pueden convertirse en soluciones reales aplicables a empresas.
- La ciberseguridad no solo protege sistemas, sino que también aporta valor estratégico al negocio.

PUCE Virtual

síguenos en:

www.pucevirtual.puce.edu.ec