PUCE Virtual

CLASE

5

ESTRUCTURA DE DATOS (PROGRAMACIÓN III)

Aplicación de árboles y grafos

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 la informática y la ciberseguridad, es fundamental contar con herramientas que permitan representar, analizar y comprender la estructura de los sistemas y redes. Entre estas herramientas destacan las estructuras de datos como los árboles y los grafos, las cuales permiten modelar relaciones entre diferentes elementos de manera organizada y comprensible. Estas estructuras no solo se utilizan en el desarrollo de software y algoritmos, sino también en el análisis de amenazas y en la protección de infraestructuras tecnológicas.

En esta clase se estudiará la aplicación de árboles y grafos en el ámbito de la ciberseguridad, analizando cómo estas estructuras pueden ayudar a representar escenarios de ataque, evaluar riesgos y comprender el comportamiento de una red informática. Los árboles permiten modelar decisiones, jerarquías y posibles rutas de ataque mediante estructuras organizadas que facilitan el análisis de amenazas. Por otro lado, los grafos permiten representar redes complejas, mostrando cómo se conectan diferentes dispositivos, sistemas o usuarios dentro de una infraestructura tecnológica.

A lo largo del tema se abordarán conceptos fundamentales como árboles de decisión, árboles de ataque, modelado de amenazas, análisis de rutas de ataque y evaluación de nodos críticos, así como los principios básicos de los grafos y sus tipos. Estos conocimientos permitirán comprender mejor cómo se analizan las vulnerabilidades y cómo se diseñan estrategias para fortalecer la seguridad de los sistemas informáticos.

Semana 5:
Seleccionar las estructuras de datos más adecuadas según los requerimientos y características de distintos escenarios de ciberseguridad.

5. Aplicación de árboles y grafos.

En ciberseguridad, árboles y grafos facilitan la comprensión de cómo interactúan los componentes de una red y cómo pueden surgir vulnerabilidades o amenazas. Mediante su uso es posible modelar escenarios de ataque, identificar puntos críticos de acceso y analizar rutas que podrían ser explotadas por un atacante.

Árboles se emplean principalmente para representar decisiones, jerarquías y estrategias de ataque, mientras que grafos permiten modelar relaciones complejas entre sistemas, dispositivos o usuarios dentro de una red informática (Stallings, 2018).

5.1. Árboles aplicados a la ciberseguridad.

En ciberseguridad, los árboles se utilizan para analizar posibles escenarios de ataque, evaluar riesgos asociados a diferentes vulnerabilidades, representar decisiones relacionadas con estrategias de defensa y modelar amenazas potenciales dentro de un sistema informático. Los analistas pueden visualizar cómo se desarrollan los ataques y qué medidas pueden aplicarse para mitigarlos. Entre los modelos más utilizados destacan los árboles de decisión y los árboles de ataque, ampliamente empleados en el análisis de seguridad (Amoroso, 2012).

5.1.1. Árboles de decisión.

Árbol de decisión es un modelo utilizado en informática y análisis de datos que representa decisiones y sus posibles consecuencias mediante una estructura jerárquica similar a un árbol. Aquí, cada nodo interno representa una condición o prueba, cada rama representa el resultado de esa decisión, y cada nodo final u hoja corresponde a una posible conclusión o acción.

Ampliamente utilizados en áreas como la inteligencia artificial, el aprendizaje automático y el análisis de datos, ya que permiten clasificar información y predecir resultados a partir de un conjunto de variables. En ciberseguridad, estos árboles pueden emplearse para detectar actividades sospechosas, clasificar tráfico de red o apoyar sistemas de detección de intrusiones. Gracias a su estructura clara e interpretable, los árboles de decisión facilitan la comprensión de cómo se toman las decisiones dentro de un sistema de análisis de seguridad (Tan et al., 2019).

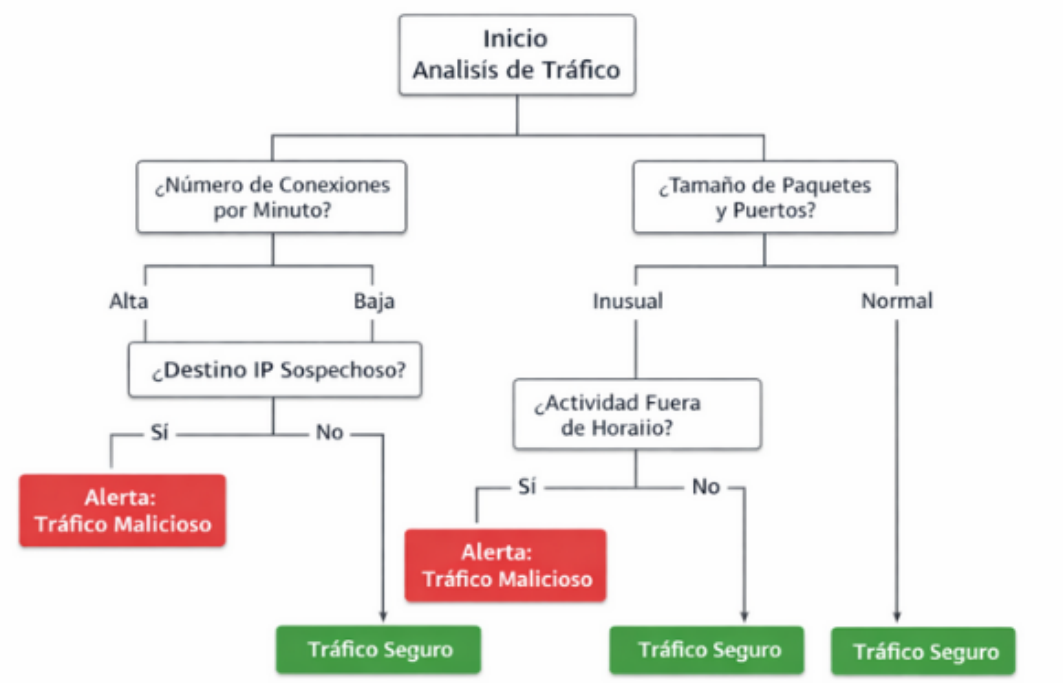


Figura 1. Ejemplo de Grafo. Creación de autor Patricio Coba

La figura 1 muestra un ejemplo de un grafo aplicado a la ciberseguridad.

Utilizadas en ciberseguridad como herramientas de análisis para identificar patrones en los datos y apoyar la detección de amenazas. Permiten evaluar diferentes variables de un sistema o de una red y tomar decisiones basadas en reglas definidas o aprendidas mediante técnicas de aprendizaje automático.

Una de sus aplicaciones más comunes es la clasificación de tráfico de red malicioso. El árbol analiza características del tráfico, como la cantidad de paquetes enviados, la frecuencia de las conexiones o el origen de las direcciones IP, y determina si el comportamiento corresponde a tráfico normal o a una posible actividad sospechosa.

Utilizados también para detectar intrusiones dentro de una red informática. Los sistemas de detección de intrusos (IDS) pueden emplear árboles de decisión para analizar registros de actividad y reconocer patrones asociados con ataques conocidos, como múltiples intentos fallidos de inicio de sesión o explotación de vulnerabilidades.

Otra aplicación importante es la identificación de comportamientos anómalos, analizando el comportamiento habitual de usuarios o dispositivos y detectando desviaciones, como accesos en horarios inusuales o transferencias de datos excesivas. Además, estos modelos permiten analizar eventos registrados en firewalls, servidores o aplicaciones para determinar si corresponden a un incidente de seguridad o a un comportamiento normal del sistema (Stallings, 2018).

Ejemplo básico en Python:

```
from sklearn.tree import DecisionTreeClassifier

# Datos simples: [numero_intentos_login, ip_sospechosa]
X = [
[1, 0],
[2, 0],
[5, 1],
[7, 1]
]
# 0 = normal, 1 = posible ataque
y = [0, 0, 1, 1]
modelo = DecisionTreeClassifier()
modelo.fit(X, y)
prediccion = modelo.predict([[6,1]])
print("¿Es un ataque?:", prediccion)
```

Tutorial de clasificación mediante árboles de decisión en Python. Página Web de DataCamp que permite aprender sobre árboles de decisión con ejemplos de código en Python. Enlace: https://www.datacamp.com/es/tutorial/decision-tree-classification-python

5.1.2. Árboles de ataque (Attack Trees).

Diagramas jerárquicos utilizados para representar de forma estructurada las diferentes estrategias o caminos que un atacante podría seguir para comprometer un sistema, una red o una aplicación. Permite descomponer un objetivo de ataque complejo en varios pasos más pequeños y comprensibles, lo que facilita el análisis de vulnerabilidades y la planificación de mecanismos de defensa.

En un árbol de ataque, el nodo raíz representa el objetivo principal del atacante, por ejemplo, obtener acceso no autorizado a un sistema, robar información confidencial o interrumpir un servicio. A partir de este nodo se generan nodos hijos, que describen las distintas acciones o métodos que podrían emplearse para lograr ese objetivo. Cada uno de estos nodos puede dividirse en subpasos más específicos, mostrando diferentes rutas posibles de ataque.

Además, los árboles de ataque pueden incluir relaciones lógicas entre nodos, como condiciones AND y OR, que indican si el atacante debe cumplir varias acciones simultáneamente o si existen diferentes alternativas para alcanzar el mismo objetivo. Gracias a esta representación, los analistas de seguridad pueden identificar vulnerabilidades, priorizar controles de protección y evaluar el impacto de posibles amenazas (Schneier, 1999).

La figura 2 muestra un ejemplo de un árbol de ataque

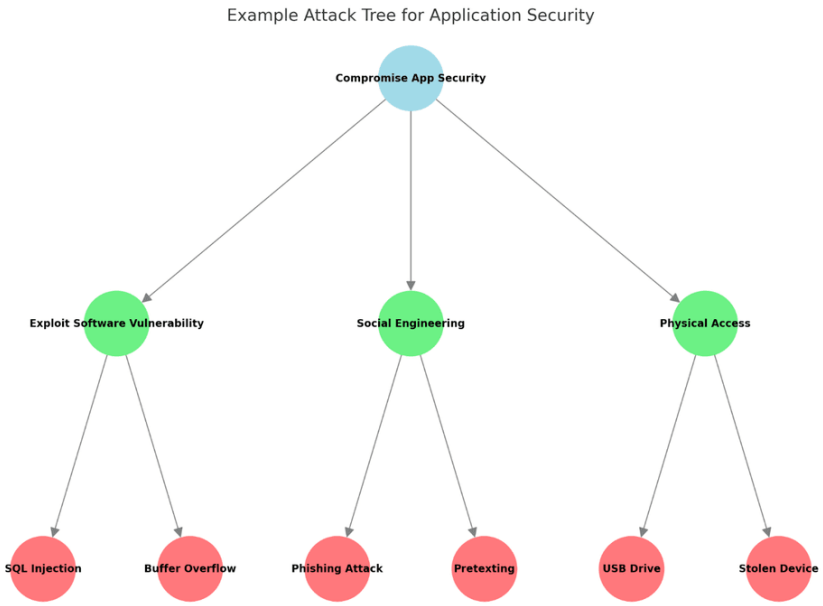


Figura 2. ejemplo de un árbol de ataque. Tomado de: https://www.exploresec.com/attack-tree-example

Aplicación en ciberseguridad

Se utilizan para:

- Analizar vulnerabilidades
- Diseñar estrategias de defensa
- Evaluar riesgos
- Simular escenarios de ataque

Ejemplo en Python

```
class NodoAtaque:
 def __init__(self, nombre):
 self.nombre = nombre
 self.hijos = []

def agregar(self, nodo):
 self.hijos.append(nodo)

def mostrar(nodo, nivel=0):
 print(" " * nivel + nodo.nombre)
 for h in nodo.hijos:
  mostrar(h, nivel + 2)

  raiz = NodoAtaque("Comprometer servidor")

raiz.agregar(NodoAtaque("Fuerza bruta"))
raiz.agregar(NodoAtaque("Exploit vulnerabilidad"))
raiz.agregar(NodoAtaque("Phishing administrador"))
```

```
mostrar(raiz)
```

5.1.3. Modelado de amenazas.

Proceso que consiste en identificar, analizar y evaluar posibles amenazas que podrían afectar a un sistema informático antes de que estas ocurran. Su objetivo principal es anticipar riesgos y diseñar estrategias de protección que reduzcan la probabilidad de ataques o minimicen su impacto.

Se puede identificar activos críticos, como bases de datos, servidores o información sensible, que requieren mayor protección. También permite analizar vectores de ataque, es decir, los métodos que un atacante podría utilizar para explotar vulnerabilidades. Además, facilita el diseño de controles de seguridad adecuados. En este contexto, los árboles y grafos ayudan a visualizar cómo una amenaza puede evolucionar dentro de un sistema y cuáles podrían ser sus posibles rutas de ataque. (Shostack, 2014).

Ejemplo en Python

```
amenazas = {
"Robo de datos": ["SQL Injection", "Acceso no autorizado", "Malware"]
}

for amenaza, vectores in amenazas.items( ):
  print("Amenaza:", amenaza)
  for v in vectores:
    print("-", v)
```

5.1.4. Análisis de rutas de ataque.

Consiste en identificar y estudiar los diferentes caminos que un atacante podría seguir para comprometer un sistema informático o una red. Este análisis permite comprender cómo una vulnerabilidad inicial puede conducir a otros sistemas o recursos críticos. Para representar estas posibles rutas se utilizan comúnmente grafos o árboles, ya que estas estructuras permiten visualizar las conexiones entre sistemas, usuarios y dispositivos, facilitando así la identificación de puntos débiles y la planificación de medidas de seguridad adecuadas (Stallings, 2018).

Ejemplo Python (búsqueda de ruta)

```
grafo = {
   "Internet": ["Firewall"],
   "Firewall": ["Servidor"],
   "Servidor": ["BaseDatos"],
   "BaseDatos": []
```

```
}

def dfs(nodo, visitados):
    if nodo not in visitados:
        print(nodo)
        visitados.add(nodo)
        for vecino in grafo[nodo]:
            dfs(vecino, visitados)

dfs("Internet", set())
```

5.1.5. Evaluación de nodos críticos.

Los nodos críticos son elementos cuya falla comprometería todo el sistema.

Ejemplos:

- Servidor de autenticación
- Firewall principal
- Base de datos central

En ciberseguridad identificar estos nodos permite priorizar controles de seguridad.
En la Tabla 1 se muestra un ejemplo de nodos críticos en una red y su impacto.

Tabla 1. Ejemplo de nodos críticos en una red

Nodo	Riesgo	Impacto
Firewall	Alto	Permite acceso a red interna
Servidor Web	Medio	Acceso a servicios
Base de datos	Muy alto	Exposición de información

5.2. Introducción a grafos.

Estructura matemática que está formada por un conjunto de vértices o nodos y un conjunto de aristas o conexiones que enlazan dichos nodos. Esta estructura permite representar relaciones entre diferentes elementos de forma clara y organizada. En un grafo, los nodos pueden representar entidades como computadoras, servidores, usuarios o dispositivos de red, mientras que las aristas representan las conexiones, comunicaciones o interacciones que existen entre ellos.

Grafos se pueden utilizar para analizar y comprender la estructura y el comportamiento de las redes informáticas. Mediante grafos es posible visualizar cómo están conectados los diferentes sistemas dentro de una infraestructura tecnológica y analizar el flujo de información entre ellos.

Además, los grafos permiten identificar patrones de comunicación, detectar comportamientos anómalos y estudiar posibles vulnerabilidades dentro de una red. Por ejemplo, un aumento inusual de conexiones hacia un servidor podría indicar un ataque de denegación de servicio. También se emplean para modelar la propagación de malware, mostrando cómo una amenaza puede extenderse entre dispositivos conectados (Tanenbaum & Wetherall, 2011).

La Figura 3 muestra un ejemplo básico de un grafo.

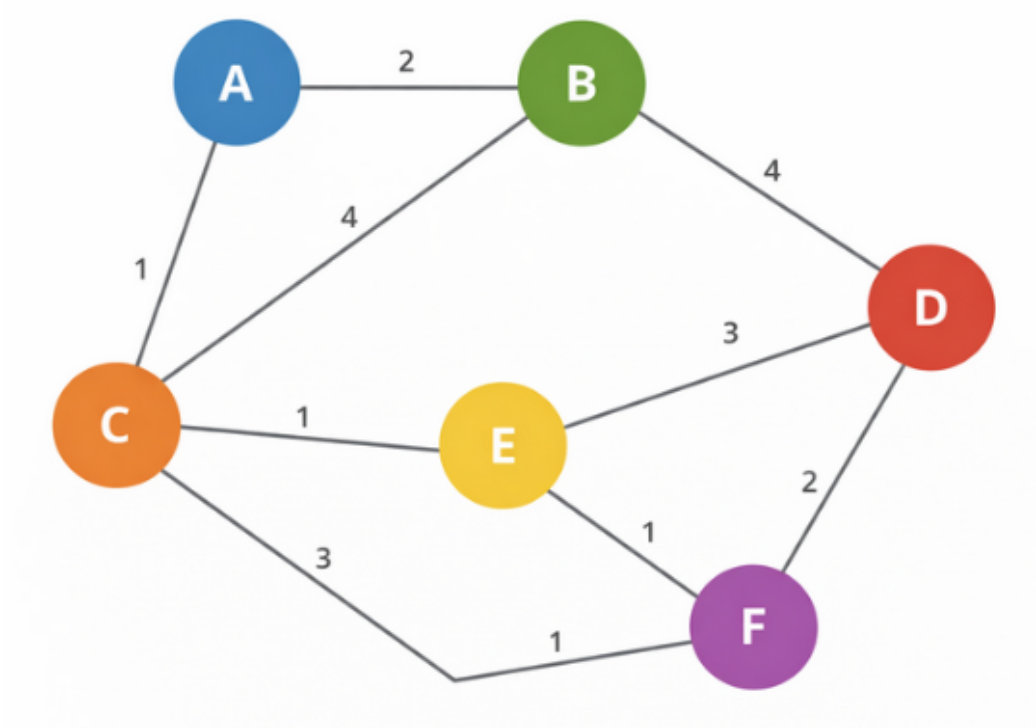


Figura 3. Ejemplo de grafo. Creación de autor Patricio Coba

5.2.1. Definición de grafo

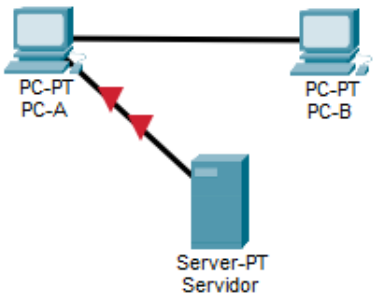
Formalmente un grafo se define como:

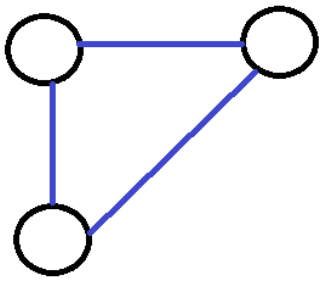
$$G = (V, E)$$

donde:

- **V** = conjunto de vértices
- **E** = conjunto de aristas

Ejemplo:





Estructuras de Grafos en Python: Conceptos, Implementaciones y Buenas Prácticas. Página web de Asimov donde se tiene una guía para entender, crear y manipular grafos en Python usando implementaciones propias y librerías como NetworkX, con ejemplos reales, comparativas, optimización y troubleshooting. Enlace: https://asimov.cloud/blog/programacion-5/estructuras-de-grafos-en-python-conceptos-implementaciones-y-buenas-practicas-436

5.2.2. Grafos dirigidos y no dirigidos

Son dirigidos o no dirigidos, dependiendo de si las conexiones entre los nodos tienen o no una dirección definida. Esta diferencia es fundamental para representar distintos tipos de relaciones en sistemas informáticos y redes.

Dirigidos
También llamado digrafo, es aquel en el que las aristas poseen una dirección específica, es decir, la conexión va de un nodo origen hacia un nodo destino. Aquí, la relación entre los nodos no necesariamente es bidireccional.

Ejemplo, si existe una conexión desde el nodo A hacia el nodo B, esto no implica necesariamente que exista una conexión desde B hacia A. Las aristas en los grafos dirigidos suelen representarse mediante flechas, indicando el sentido de la relación.

En ciberseguridad, los grafos dirigidos son útiles para representar:

- Flujo de tráfico de red entre dispositivos
- Rutas de propagación de malware
- Secuencia de eventos en un ataque informático
- Dependencias entre procesos o servicios

Por ejemplo, un grafo dirigido puede representar el flujo de datos desde un cliente hacia un servidor, o la secuencia de pasos que sigue un atacante al comprometer distintos sistemas dentro de una red.

No dirigidos
Sus aristas no tienen dirección, lo que significa que la relación entre dos nodos es bidireccional. En este caso, si un nodo A está conectado con B, también se asume que B está conectado con A.

Las aristas en este tipo de grafo se representan simplemente como líneas entre nodos, sin flechas de dirección.

En redes informáticas y ciberseguridad, los grafos no dirigidos se utilizan comúnmente para representar:

- Topologías de red
- Conexiones físicas entre dispositivos
- Relaciones entre usuarios dentro de un sistema
- Infraestructura de comunicaciones

Ejemplo, en una red local, si dos computadoras están conectadas mediante un switch, la relación puede representarse mediante un grafo no dirigido, ya que ambas pueden comunicarse entre sí.

La Figura 4 muestra un ejemplo de un grafo no dirigido y un grafo dirigido con cuatro nodos.

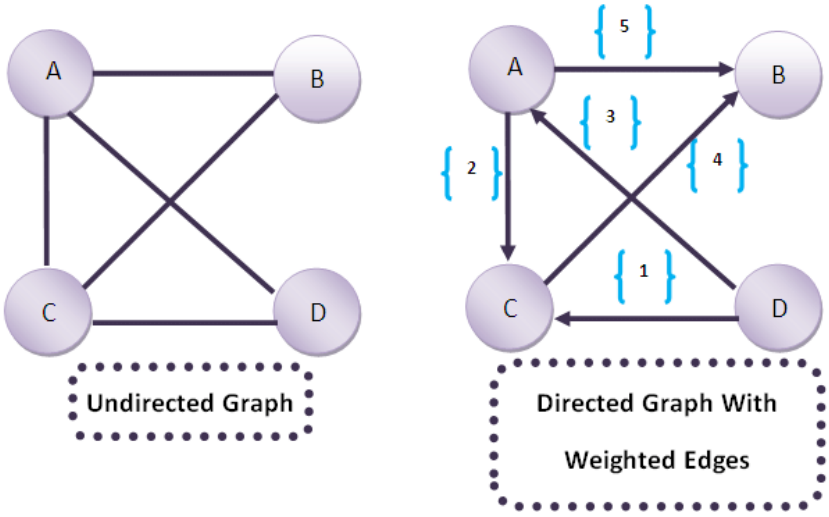


Figura 4. Ejemplo de un grafo dirigido y no dirigido. Tomado de: https://www.researchgate.net/figure/Example-of-an-undirected-graph-and-a-directed-graph-with-weighted-edges_fig4_254062494

La tabla 2 muestra una descripción general y un ejemplo de los grafos dirigidos y no dirigidos.

Tabla 2. Descripción y ejemplo de grafos dirigidos y no dirigidos. Creación de autor Patricio Coba

Tipo	Descripción	Ejemplo
Grafo dirigido	Las conexiones tienen dirección	Flujo de red
Grafo no dirigido	Conexión bidireccional	Conexión entre dispositivos

5.2.3. Grafos ponderados y no ponderados

Dependiendo de si las conexiones entre los nodos poseen o no un valor asociado. Esta característica permite representar diferentes tipos de relaciones y medir propiedades específicas dentro de una red o sistema.

Ponderados
Cada arista tiene asignado un peso o valor numérico. Este peso puede representar distintas magnitudes según el contexto, como distancia, tiempo, costo, capacidad o nivel de riesgo. Gracias a estos valores, los grafos ponderados permiten realizar análisis más detallados sobre las relaciones entre los nodos.

Los grafos ponderados se pueden utilizar para representar aspectos como la latencia entre dispositivos, el nivel de riesgo de una conexión, el número de intentos de acceso o el costo asociado a una posible ruta de ataque. Por ejemplo, en el análisis de rutas de ataque, cada conexión entre sistemas puede tener un peso que indique la dificultad o probabilidad de explotación de una vulnerabilidad.

No ponderados
Es aquel en el que las aristas no tienen un valor asociado, por lo que todas las conexiones se consideran equivalentes. En este tipo de grafo solo se representa la existencia de una relación entre nodos, sin especificar características adicionales sobre esa conexión.

Son útiles cuando únicamente interesa conocer si existe o no una conexión entre los elementos de un sistema. En redes informáticas, se pueden utilizar para representar la topología básica de una red, mostrando qué dispositivos están conectados entre sí sin considerar factores como velocidad, distancia o prioridad de la conexión. En la Tabla 3 mostrada a continuación se muestra una comparación entre grafos ponderados y no ponderados

Tabla 3. Descripción y ejemplo de grafos ponderados y no ponderados. Creación de autor Patricio Coba

Tipo	Característica	Ejemplo
Ponderado	Cada arista tiene peso	latencia o riesgo
No ponderado	Todas las conexiones iguales	topología simple

Ejemplo:

$$\text{Servidor A} \rightarrow \text{Servidor B (peso = tiempo de acceso)}$$

5.2.4. Representación mediante listas y matrices

A través de listas y matrices se pueden representar grafos. La Tabla 2 se muestra la comparación (ventajas y desventajas entre las principales formas de representar grafos.

Tabla 4. Comparación de representaciones de grafos

Método	Ventaja	Desventaja
Lista de adyacencia	Menor memoria	Búsqueda más lenta
Matriz de adyacencia	Búsqueda rápida	Mayor memoria

Ejemplo en Python

```
Lista de adyacencia:
grafo = {
   "A": ["B", "C"],
   "B": ["C"],
   "C": ["A"]
}

for nodo in grafo:
   print(nodo, "->", grafo[nodo])
```

5.2.5. Grafos en modelado de redes

Se utilizan para modelar redes informáticas, ya que permiten representar de manera clara y estructurada la relación entre los diferentes dispositivos que forman parte de una infraestructura tecnológica.

Los grafos facilitan el análisis de la topología de la red, permitiendo comprender cómo se encuentran interconectados los dispositivos y cómo circula la información dentro del sistema. Además, esta representación resulta especialmente útil para estudiar aspectos como el flujo de datos, la disponibilidad de rutas de comunicación, la detección de fallos o la identificación de posibles vulnerabilidades.

En ciberseguridad, modelar redes mediante grafos permite analizar cómo podría propagarse un ataque o un malware dentro de una red, identificar nodos críticos cuya falla afectaría al sistema completo y estudiar posibles rutas de acceso que un atacante podría aprovechar para comprometer distintos dispositivos. Así, los grafos se convierten en una herramienta fundamental para el diseño, monitoreo y protección de redes informáticas (Tanenbaum & Wetherall, 2011).

Aplicaciones en ciberseguridad:

- detección de propagación de malware
- análisis de tráfico
- análisis de vulnerabilidades
- detección de anomalías

Ejemplo en Python usando NetworkX

```
import networkx as nx

G = nx.Graph()

G.add_edge("Router","Firewall")
G.add_edge("Firewall","Servidor")
G.add_edge("Servidor","BaseDatos")

print(G.edges())
```

Definición de los términos citados en la Semana 5

Árbol de decisión. Modelo utilizado en informática, minería de datos y aprendizaje automático que representa un proceso de toma de decisiones mediante una estructura jerárquica similar a un árbol. En esta estructura, cada nodo interno representa una prueba o condición sobre un atributo, cada rama corresponde al resultado de esa prueba y cada nodo hoja indica una decisión final o clasificación. Los árboles de decisión se utilizan para analizar datos, identificar patrones y predecir resultados a partir de diferentes variables.

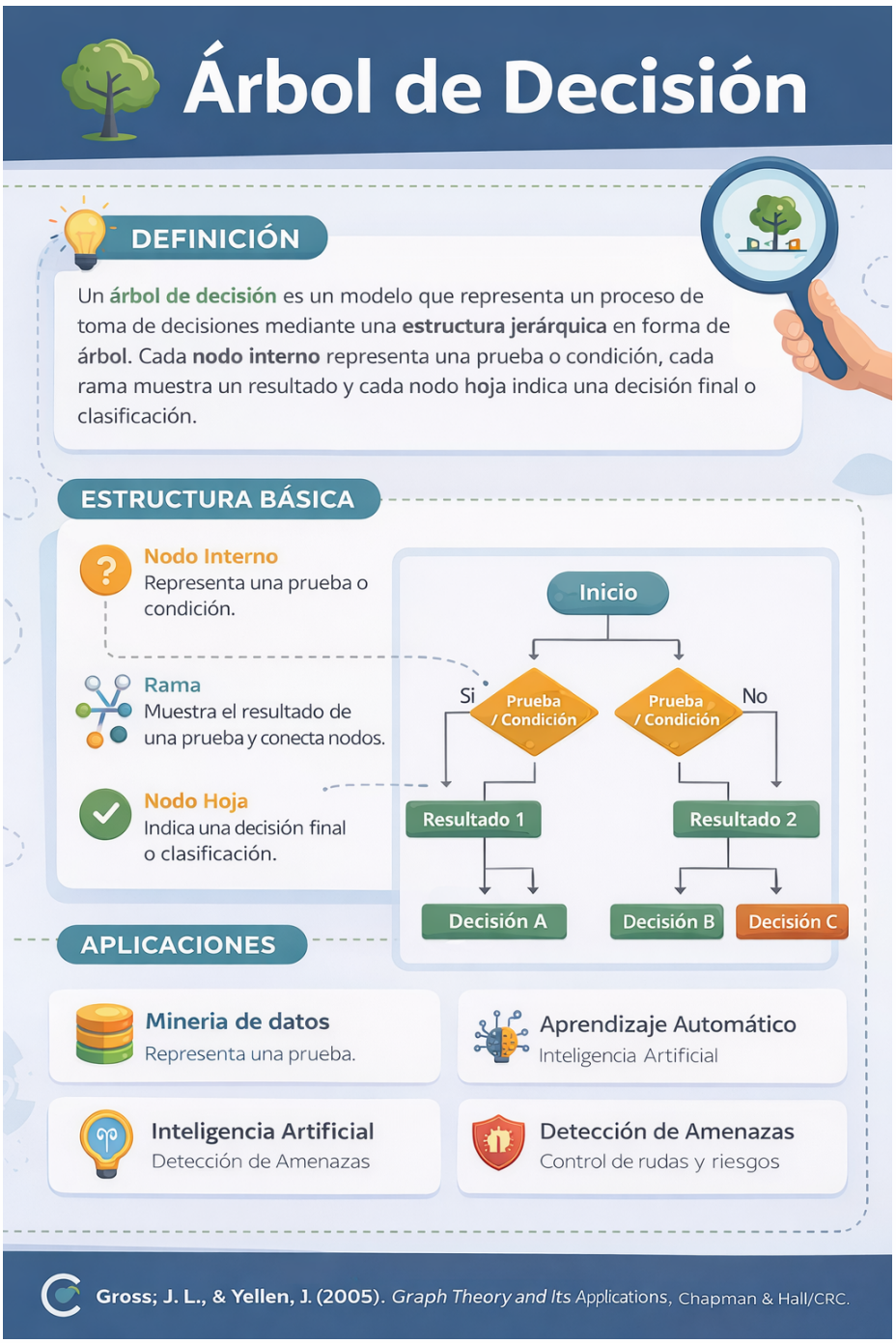
Grafo. Estructura matemática utilizada en informática y en la teoría de grafos que está formada por un conjunto de vértices o nodos y un conjunto de aristas o enlaces que conectan dichos nodos. Esta estructura permite representar relaciones o interacciones entre diferentes elementos de un sistema. En un grafo, los nodos pueden representar entidades como computadoras, usuarios o dispositivos, mientras que las aristas representan las conexiones o relaciones entre ellos.

Educación de calidad

REFERENCIAS

Amoroso, E. (2012). Fundamentos de seguridad informática. Prentice Hall.
Tan, P. N., Steinbach, M., & Kumar, V. (2019). Introducción a la minería de datos. Pearson.
Schneier, B. (1999). Attack Trees: Modeling Security Threats. Dr. Dobb's Journal.
Shostack, A. (2014). Modelado de amenazas: diseño de seguridad para sistemas. Wiley.
Stallings, W. (2018). Seguridad informática: principios y práctica. Pearson.
Tanenbaum, A. S., & Wetherall, D. (2011). Redes de computadoras. Pearson.

Árbol de decisión. Infografía que contiene la definición, estructura, aplicaciones y ejemplo de un árbol de decisión.



Qué es un grafo. Infografía que muestra en resumen lo que es un grafo, su estructura , ejemplos y aplicaciones en ciberseguridad.

Laboratorio de Contenido 1: Árbol de decisión aplicado a ciberseguridad.

Explicación básica

Un árbol de ataque es un modelo utilizado en ciberseguridad para representar de forma estructurada las distintas formas en que un atacante puede comprometer un sistema. En este modelo, el objetivo principal del atacante se coloca en el nodo raíz del árbol, mientras que los nodos hijos representan las acciones o pasos necesarios para alcanzar ese objetivo.

Los árboles de ataque permiten descomponer un ataque complejo en pasos más pequeños y analizables. Además, ayudan a identificar vulnerabilidades, priorizar medidas de seguridad y evaluar riesgos dentro de una infraestructura informática.

Objetivos

- Comprender el concepto de árbol de ataque.
- Identificar los componentes principales de un árbol de ataque.
- Analizar escenarios básicos de ataques informáticos.
- Representar un árbol de ataque mediante una estructura de datos.
- Implementar un ejemplo sencillo de árbol de ataque en Python.

Caso práctico

Una empresa posee un servidor web interno que almacena información importante. El equipo de seguridad desea analizar cómo un atacante podría comprometer dicho servidor.

El objetivo del atacante es: Comprometer el servidor web

Las posibles formas de lograrlo pueden ser:

1. Ataque de fuerza bruta para obtener credenciales.
2. Explotación de una vulnerabilidad del sistema.
3. Phishing dirigido al administrador del sistema.

Esto puede representarse mediante un árbol de ataque.

- Comprometer servidor web
- Fuerza bruta
- Exploit vulnerabilidad
- Phishing al administrador

Problema

El departamento de seguridad desea modelar las posibles rutas de ataque contra su servidor web para identificar riesgos potenciales.

Se requiere:

- Representar el árbol de ataque mediante una estructura de datos.
- Mostrar los posibles caminos que un atacante podría utilizar.
- Implementar una simulación simple utilizando Python.

Aplicación paso a paso

Paso 1: Definir el objetivo del ataque

El nodo raíz será:

Comprometer servidor web

Paso 2: Identificar los métodos de ataque

Se identifican tres métodos principales:

- Fuerza bruta
- Exploit de vulnerabilidad
- Phishing

Paso 3: Representar el árbol

El árbol puede representarse mediante nodos y subnodos.

Comprometer servidor

* Fuerza bruta
    └── Adivinar contraseña
* Exploit vulnerabilidad
    └── Vulnerabilidad en software
* Phishing
    └── Engañar administrador

Paso 4: Modelar la estructura en Python

Cada nodo del árbol representará una acción dentro del ataque.

Implementación en Python

```python
class NodoAtaque:
    def __init__(self, nombre):
        self.nombre = nombre
        self.hijos = []

    def agregar_hijo(self, nodo):
        self.hijos.append(nodo)

def mostrar_arbol(nodo, nivel=0):
    print(" " * nivel + "- " + nodo.nombre)
    for hijo in nodo.hijos:
        mostrar_arbol(hijo, nivel + 4)
```

```python
# Crear nodo raíz
raiz = NodoAtaque("Comprometer servidor web")

# Crear nodos de ataque
fuerza_bruta = NodoAtaque("Ataque de fuerza bruta")
exploit = NodoAtaque("Exploit de vulnerabilidad")
phishing = NodoAtaque("Phishing al administrador")

# Subataques
adivinar = NodoAtaque("Adivinar contraseña")
vulnerabilidad = NodoAtaque("Explotar vulnerabilidad del software")
engaño = NodoAtaque("Enviar correo de phishing")

# Construir árbol
raiz.agregar_hijo(fuerza_bruta)
raiz.agregar_hijo(exploit)
raiz.agregar_hijo(phishing)

fuerza_bruta.agregar_hijo(adivinar)
exploit.agregar_hijo(vulnerabilidad)
phishing.agregar_hijo(engaño)

# Mostrar árbol
mostrar_arbol(raiz)
```

Conclusiones

Los árboles de ataque son herramientas muy útiles para analizar amenazas dentro de un sistema informático. Permiten representar de forma clara los distintos caminos que un atacante podría seguir para comprometer una infraestructura tecnológica.

A través de este laboratorio se pudo observar cómo un escenario de ataque puede dividirse en diferentes etapas y cómo estas pueden modelarse mediante una estructura jerárquica. Además, se demostró que es posible implementar este tipo de modelos utilizando Python, lo que facilita la simulación y análisis de escenarios de seguridad.

El uso de árboles de ataque ayuda a los analistas de seguridad a identificar vulnerabilidades, evaluar riesgos y diseñar estrategias de defensa más efectivas, contribuyendo a mejorar la protección de los sistemas informáticos.

Laboratorio de Contenido 2:

Explicación básica

En ciberseguridad, los grafos dirigidos son especialmente útiles para representar flujos de comunicación, rutas de ataque y dependencias entre sistemas. Por ejemplo, pueden modelar cómo un atacante se mueve dentro de una red, comenzando desde un equipo comprometido y avanzando hacia servidores o bases de datos.

Este tipo de representación facilita el análisis de posibles vulnerabilidades, la identificación de rutas críticas y la comprensión del comportamiento de la red.

Objetivos

- Comprender el concepto de grafo dirigido.
- Identificar los componentes principales de un grafo.
- Representar relaciones entre sistemas mediante grafos dirigidos.
- Analizar posibles rutas de ataque en una red.
- Implementar un grafo dirigido utilizando Python.

Caso práctico

Una organización desea analizar cómo un atacante podría desplazarse dentro de su red interna.

La infraestructura simplificada es la siguiente:

- Internet
- Firewall
- Servidor Web
- Servidor de Aplicaciones
- Base de Datos

El flujo de acceso dentro de la red puede representarse de la siguiente forma:

Internet → Firewall → Servidor Web → Servidor de Aplicaciones → Base de Datos

Cada conexión tiene una **dirección**, ya que el acceso ocurre siguiendo un flujo específico dentro de la red.

Problema

El equipo de seguridad desea analizar las posibles rutas que un atacante podría seguir dentro de la red si logra comprometer un punto inicial.

Se requiere:

- Representar la infraestructura como un grafo dirigido.
- Analizar las rutas posibles desde un nodo inicial.
- Simular el recorrido del atacante dentro del sistema.

Aplicación paso a paso

Paso 1: Identificar los nodos

Los nodos del grafo serán:

- Internet
- Firewall
- Servidor Web
- Servidor de Aplicaciones
- Base de Datos

Paso 2: Identificar las conexiones dirigidas

Las conexiones representan el flujo de acceso dentro de la red.

Internet → Firewall

Firewall → Servidor Web

Servidor Web → Servidor Aplicaciones

Servidor Aplicaciones → Base de Datos

Paso 3: Representar el grafo

El grafo puede representarse mediante una **lista de adyacencia**, que es una estructura común en programación.

Implementación en Python

```python
grafo = {
    "Internet": ["Firewall"],
    "Firewall": ["Servidor Web"],
    "Servidor Web": ["Servidor Aplicaciones"],
    "Servidor Aplicaciones": ["Base de Datos"],
    "Base de Datos": []
}

# Función para recorrer el grafo (búsqueda en profundidad)
def dfs(nodo, visitados):
    if nodo not in visitados:
        print(nodo)
        visitados.add(nodo)

        for vecino in grafo[nodo]:
            dfs(vecino, visitados)

# Simulación del recorrido de un atacante
print("Ruta potencial de ataque:")

dfs("Internet", set())
```

Conclusiones

En ciberseguridad, los grafos facilitan la comprensión de cómo un atacante podría desplazarse dentro de una infraestructura tecnológica.

A través de este laboratorio se observó cómo una red puede representarse mediante un grafo dirigido y cómo es posible analizar rutas de acceso utilizando algoritmos de recorrido.

La implementación en Python demuestra que estas estructuras pueden utilizarse para simular escenarios de ataque y estudiar posibles vulnerabilidades.

El uso de grafos dirigidos ayuda a los analistas de seguridad a identificar rutas críticas, detectar posibles puntos de intrusión y mejorar las estrategias de defensa, fortaleciendo así la seguridad de los sistemas informáticos.

Laboratorio de Contenido 3:

Explicación básica

En ciberseguridad, los grafos ponderados se utilizan para analizar redes informáticas y evaluar posibles rutas de ataque. En este caso, el peso de una conexión puede representar el nivel de vulnerabilidad, la dificultad del ataque o el impacto potencial de comprometer un sistema. Gracias a estos valores, los analistas de seguridad pueden identificar cuáles son los caminos más probables o peligrosos que un atacante podría utilizar para desplazarse dentro de una red.

Objetivos

- Comprender el concepto de grafo ponderado.
- Identificar cómo se representan los pesos en las aristas.
- Analizar rutas dentro de una red informática.
- Determinar rutas de ataque con menor o mayor costo.
- Implementar un grafo ponderado utilizando Python.

Caso práctico

Una empresa desea analizar la seguridad de su infraestructura de red. Los analistas han identificado diferentes sistemas y el nivel de dificultad que tendría un atacante para moverse entre ellos.

Los sistemas identificados son:

- Internet
- Firewall
- Servidor Web
- Servidor de Aplicaciones
- Base de Datos

Cada conexión tiene un peso que representa la dificultad del ataque (entre menor sea el número, más fácil sería comprometer el sistema).

Conexión	**Dificultad del ataque**
Internet → Firewall	3
Firewall → Servidor Web	2
Servidor Web → Servidor Aplicaciones	2
Servidor Aplicaciones → Base de Datos	4

Problema

El equipo de seguridad desea analizar cuál sería la ruta más sencilla para que un atacante comprometa la base de datos si logra entrar desde Internet.

Para ello se requiere:

- Representar la red como un grafo ponderado.

- Calcular el costo total de las rutas posibles.
- Identificar la ruta de menor dificultad para el atacante.

Aplicación paso a paso

Paso 1: Identificar los nodos
Los nodos representan los sistemas de la red:

- Internet
- Firewall
- Servidor Web
- Servidor de Aplicaciones
- Base de Datos

Paso 2: Definir las conexiones con pesos
Cada conexión tendrá un valor que representa el nivel de dificultad del ataque.

Ejemplo:

Internet → Firewall (3)
Firewall → Servidor Web (2)
Servidor Web → Servidor Aplicaciones (2)
Servidor Aplicaciones → Base de Datos (4)

Paso 3: Representar el grafo en Python
El grafo puede representarse utilizando **diccionarios con pesos asociados**.

Implementación en Python

```python
import heapq

# Grafo ponderado representado como diccionario
grafo = {
    "Internet": {"Firewall": 3},
    "Firewall": {"Servidor Web": 2},
    "Servidor Web": {"Servidor Aplicaciones": 2},
    "Servidor Aplicaciones": {"Base de Datos": 4},
    "Base de Datos": {}
}
```

```python
def dijkstra(grafo, inicio):
    distancias = {nodo: float('inf') for nodo in grafo}
    distancias[inicio] = 0
    cola = [(0, inicio)]

    while cola:
        distancia_actual, nodo_actual = heapq.heappop(cola)

        for vecino, peso in grafo[nodo_actual].items():
            distancia = distancia_actual + peso

            if distancia < distancias[vecino]:
                distancias[vecino] = distancia
                heapq.heappush(cola, (distancia, vecino))

    return distancias

resultado = dijkstra(grafo, "Internet")

print("Costo mínimo para llegar a cada sistema:")
for nodo, costo in resultado.items():
    print(nodo, ":", costo)
```

Conclusiones

Los grafos ponderados permiten representar redes informáticas donde las conexiones tienen diferentes niveles de costo, riesgo o dificultad. En ciberseguridad, esta característica es muy útil para analizar rutas de ataque potenciales y comprender qué sistemas podrían ser más vulnerables dentro de una infraestructura.

En este laboratorio se observó cómo una red puede modelarse mediante un grafo ponderado y cómo es posible calcular rutas utilizando algoritmos como Dijkstra. Este tipo de análisis permite a los especialistas en seguridad identificar los caminos más críticos que un atacante podría aprovechar.

El uso de grafos ponderados facilita la evaluación de riesgos, la priorización de medidas de seguridad y el diseño de estrategias de defensa, contribuyendo a mejorar la protección de los sistemas y redes informáticas.

ESTUDIO DE CASO 1: Clasificación Automatizada de Tráfico para la Detección de Intrusos (IDS)

Contexto

La empresa "SecureNet Data Solutions" gestiona un centro de datos que procesa millones de solicitudes por hora. Con el auge del teletrabajo y la interconectividad de dispositivos IoT, el volumen de logs de red se ha vuelto inmanejable para los analistas humanos del SOC (Security Operations Center).

Problema

El sistema de detección de intrusos tradicional, basado en firmas (reglas fijas), no logra identificar ataques de **"Día Cero"** o variaciones sutiles de malware. El equipo de seguridad se enfrenta a dos grandes problemas:

- **Fatiga de alertas:** Demasiados falsos positivos que distraen de las amenazas reales.
- **Latencia:** El análisis manual de paquetes sospechosos tarda minutos, mientras que una brecha de datos ocurre en milisegundos.

Solución Propuesta

Implementar un modelo de Árbol de Decisión entrenado para clasificar el tráfico de red en tiempo real. A diferencia de las "cajas negras" como las redes neuronales profundas, los árboles de decisión ofrecen interpretabilidad, lo que permite a los analistas entender *por qué* una conexión fue marcada como maliciosa.

Estrategia:

Se propone un modelo de clasificación binaria basado en el algoritmo CART (Classification and Regression Trees). El árbol analizará características específicas del flujo de red (protocolo, duración, bytes enviados, estado de los flags TCP) para categorizar cada evento como "Tráfico Legítimo" o "Ataque Potencial".

Implementación

La estrategia se divide en tres fases técnicas:

1. Selección de Atributos: Identificar las variables con mayor "ganancia de información" (usando medidas como la Impureza de Gini).
2. Poda del Árbol (Pruning): Limitar la profundidad del árbol para evitar el *overfitting* (sobreajuste), asegurando que el modelo generalice bien ante ataques nuevos.
3. Validación Cruzada: Probar el modelo con datasets históricos como el NSL-KDD para ajustar la precisión antes del despliegue.

```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt

# 1. Creación de un Dataset Sintético de Ciberseguridad
# Características: [Duración(seg), Tamaño_Paquete(kb), Intentos_Login]
data = {
    'duracion': [0.1, 0.2, 10, 0.5, 30, 0.1, 0.2, 45, 0.3, 0.1, 60],
    'tamano_paquete': [100, 150, 5000, 120, 8000, 90, 110, 7500, 130, 95, 9000],
    'intentos_login': [1, 1, 5, 1, 10, 1, 1, 12, 1, 1, 15],
    'es_intrusion': [0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1] # 1 es Ataque (ej. Brute Force o DoS)
}

df = pd.DataFrame(data)

# 2. Separación de características (X) y etiqueta (y)
X = df.drop('es_intrusion', axis=1)
y = df['es_intrusion']

# Dividimos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```python
# 3. Construcción del Árbol de Decisión
# Usamos 'entropy' para medir la ganancia de información
clf = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=42)
clf.fit(X_train, y_train)

# 4. Predicción y Evaluación
y_pred = clf.predict(X_test)

print("--- Reporte de Ciberseguridad ---")
print(f"Precisión del Modelo: {accuracy_score(y_test, y_pred) * 100}%")
print("\nMatriz de Clasificación:")
print(classification_report(y_test, y_pred))

# 5. Visualización de la Lógica del Árbol
plt.figure(figsize=(12, 8))
plot_tree(clf, feature_names=X.columns, class_names=['Legítimo', 'Intrusión'], filled=True)
plt.title("Lógica de Decisión del IDS (Intrusion Detection System)")
plt.show()
```

Resultados

El modelo se despliega mediante un script en Python utilizando la librería scikit-learn. Se integra en el flujo de datos del firewall:

- **Entrada:** Datos vectorizados del encabezado de los paquetes.
- **Procesamiento:** El árbol evalúa condiciones (ej: ¿Es el puerto de destino 80? → ¿El tamaño del paquete es > 5000 bytes?).
- **Salida:** Una etiqueta de clase y una probabilidad de confianza.

Tras un mes de implementación, los resultados comparados con el sistema anterior fueron:

- **Reducción de falsos positivos:** 45%.
- **Precisión (Accuracy):** 92.5% en la detección de escaneos de puertos y ataques DoS.
- **Velocidad:** Tiempo de inferencia inferior a 0.01 segundos por registro.

Impacto en Ciberseguridad

El uso de árboles de decisión transformó la postura de seguridad de la empresa de **reactiva a proactiva**:

- **Respuesta Automatizada:** El sistema puede bloquear IPs automáticamente si la confianza del árbol supera el 95%.
- **Auditoría:** En caso de un incidente, el camino tomado por el árbol sirve como evidencia forense clara para explicar la lógica de la detección.

Reflexión Técnica

Aunque los árboles de decisión son excelentes por su claridad y rapidez, presentan una debilidad: son sensibles a pequeñas variaciones en los datos de entrenamiento (alta varianza). Para una robustez total en entornos de alta criticidad, la evolución natural de este caso de estudio sería la implementación de un Random Forest (un bosque de árboles), que reduciría la varianza mediante el ensamblaje de múltiples decisiones independientes.

Estudio de caso 2: Análisis de Movimiento Lateral mediante Grafos No Ponderados

Contexto
La multinacional "TechGlobal Corp" posee una infraestructura de red híbrida con miles de terminales (endpoints), servidores y dispositivos móviles. En este entorno, las relaciones de comunicación no son lineales, sino que forman una malla compleja de interacciones diarias.

Problema
Tras una auditoría, se detectó que una vez que un atacante logra comprometer una terminal de usuario (punto de entrada), puede realizar movimiento lateral para alcanzar servidores críticos de bases de datos. El problema es que las listas de control de acceso (ACL) tradicionales no permiten visualizar el "camino de menor resistencia" que un atacante podría seguir aprovechando las confianzas existentes entre máquinas.

Solución Propuesta

Modelar la red completa como un **Grafo No Ponderado**.

- **V (Vértices):** Representan cada dispositivo o usuario en la red.
- **E (Aristas):** Representan una conexión o permiso de comunicación activo entre dos nodos. Al ser un grafo no ponderado, nos enfocamos en la **conectividad y la topología**, tratando todas las conexiones como igualmente posibles para un atacante.

Implementar un sistema de análisis de **Centralidad y Alcance** basado en grafos. El objetivo es identificar "nodos articuladores" o "puentes" que, si son comprometidos, darían al atacante acceso a gran parte de la red de manera inmediata.

Estrategia:

La estrategia utiliza algoritmos de recorrido de grafos:

1. Mapeo de Relaciones: Extraer logs de Active Directory y flujos de red para construir el grafo.
2. Cálculo de Centralidad de Intermediación (Betweenness Centrality): Identificar qué nodos actúan como puentes críticos entre diferentes segmentos de red.
3. Análisis de Componentes Conectados: Determinar si la red está demasiado "abierta", permitiendo que cualquier nodo llegue a cualquier otro en pocos saltos.

Implementación

Se utiliza la librería NetworkX en Python para procesar el grafo de la infraestructura:

- **Algoritmo BFS (Breadth-First Search):** Para calcular la distancia mínima (en saltos) desde cualquier nodo infectado hasta el servidor central.
- **Detección de Comunidades:** Para segmentar la red lógicamente y verificar que los grupos (ej. Finanzas vs. Invitados) no tengan aristas (conexiones) no autorizadas entre ellos.

Implementación en Python

```python
import networkx as nx
import matplotlib.pyplot as plt

# 1. Creación del Grafo No Ponderado
G = nx.Graph()

# Definimos los nodos (Dispositivos/Usuarios)
nodos = [
    "PC-Recepcion", "PC-Ventas", "PC-RRHH",
    "Switch-Core", "Server-BasesDeDatos", "Server-Web",
    "Admin-IT", "PC-Contabilidad"
]
G.add_nodes_from(nodos)

# 2. Definición de Aristas (Conexiones/Confianzas)
# Al ser no ponderado, solo indicamos que la conexión EXISTE
conexiones = [
    ("PC-Recepcion", "Switch-Core"),
    ("PC-Ventas", "Switch-Core"),
    ("PC-RRHH", "Switch-Core"),
    ("Switch-Core", "Server-Web"),
    ("Switch-Core", "Admin-IT"),
    ("Admin-IT", "Server-BasesDeDatos"), # Camino legítimo
    ("PC-Contabilidad", "Server-BasesDeDatos"), # ¡Vulnerabilidad detectada!
    ("PC-Contabilidad", "Switch-Core")
]
G.add_edges_from(conexiones)
```

```python
# 3. Análisis de Movimiento Lateral
# Supongamos que "PC-Recepcion" es comprometida por un Ransomware
inicio = "PC-Recepcion"
objetivo = "Server-BasesDeDatos"

try:
    # Algoritmo BFS para encontrar la ruta con menos saltos
    ruta_ataque = nx.shortest_path(G, source=inicio, target=objetivo)
    print(f"⚠️ Alerta: Ruta de movimiento lateral detectada ({len(ruta_ataque)-1} saltos):")
    print(" -> ".join(ruta_ataque))
except nx.NetworkXNoPath:
    print("✅ El servidor crítico está aislado de este segmento.")
```

```python
# 4. Cálculo de Importancia (Centralidad)
# Identifica qué nodos son críticos para la conectividad de la red
centralidad = nx.betweenness_centrality(G)
nodo_critico = max(centralidad, key=centralidad.get)
print(f"\n🔍 Nodo estructuralmente más crítico: {nodo_critico}")

# 5. Visualización
plt.figure(figsize=(10, 6))
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='lightblue', node_size=2000, font_size=10)

# Resaltar la ruta del ataque en rojo
path_edges = list(zip(ruta_ataque, ruta_ataque[1:]))
nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color='r', width=3)

plt.title("Visualización de Ruta de Movimiento Lateral en la Red")
plt.show()
```

Resultados

Identificación de Riesgos: Se descubrió que el 15% de las estaciones de trabajo tenían acceso directo a la red de servidores debido a configuraciones de legado.

Optimización de Microsegmentación: Se eliminaron aristas innecesarias en el grafo, reduciendo el "diámetro" del grafo de la red y limitando el radio de explosión de un ataque.

Simulación de Ataques: El equipo azul (Blue Team) pudo predecir las rutas más probables de un atacante antes de que ocurriera un incidente.

Impacto en Ciberseguridad

El enfoque de grafos permite una visión **holística y relacional**:

- **Visualización de la Superficie de Ataque:** Los administradores pueden ver literalmente "los caminos" que el malware seguiría (gusanos de red).
- **Endurecimiento de Red (Hardening):** Priorización de parches en los nodos con mayor grado de centralidad, ya que su compromiso es más peligroso para la organización.

Reflexión Técnica

En un grafo no ponderado, la métrica clave es la distancia geodésica (el número mínimo de saltos). A diferencia de los modelos ponderados que miden latencia o ancho de banda, aquí lo que importa es la topología de la confianza. Si existe una arista entre un usuario y un servidor, la seguridad depende enteramente de la autenticación; si esa arista no debería existir, el grafo revela una vulnerabilidad de diseño estructural que los firewalls perimetrales suelen pasar por alto.

PUCE Virtual

síguenos en:

www.pucevirtual.puce.edu.ec