PUCE Virtual

DOCUMENTO DE PROFUNDIZACIÓN

Estructura de Datos

Estudio de caso 1: Simulación de Propagación de Malware (BFS)

Estudio de caso 1: Simulación de Propagación de Malware (BFS)

Contexto

La empresa **"TechLogistics"** posee una infraestructura de red híbrida con 100 servidores interconectados. El departamento de seguridad quiere poner a prueba su estrategia de segmentación. Para ello, necesitan simular el "radio de explosión" (*blast radius*) de un nuevo tipo de ransomware que explota una vulnerabilidad en el protocolo SMB para saltar de un equipo a otro.

Problema

Si un solo terminal de ventas en una oficina remota es infectado, ¿cuántos saltos (*hops*) le toma al malware llegar al servidor de base de datos central? ¿Qué equipos se verán comprometidos en la primera, segunda y tercera hora si el malware tarda 1 hora en infectar un nodo vecino?

Solución Propuesta

Utilizar el algoritmo **BFS (Breadth-First Search)** para simular la propagación nivel por nivel. Dado que el malware se propaga a todos los vecinos disponibles simultáneamente, BFS representa perfectamente este crecimiento radial, a diferencia de DFS que solo seguiría una línea de infección.

Estrategia:

1. **Modelado:** Los nodos son dispositivos (PCs, Servidores, Switch) y las aristas son las conexiones físicas o lógicas.
2. **Estado de Infección:** Mantener una lista de nodos "sanos", "infectados" y "en proceso".
3. **Análisis por Capas:** Cada nivel del BFS representará una "Ola de Infección" (Tiempo).
4. **Detección de Cortafuegos:** Si una arista conecta con una VLAN protegida, se le asigna una probabilidad de bloqueo (en esta simulación básica, asumiremos paso libre para medir el peor escenario).

Implementación

```python
from collections import deque

def simular_propagacion(red, inicio):
    cola = deque([(inicio, 0)])  # (Nodo actual, Hora de infección)
    infectados = {inicio: 0}

    print(f"--- Inicio de Simulación: Paciente Cero [{inicio}] ---")

    while cola:
        nodo_actual, hora = cola.popleft()

        # Simulamos la propagación a vecinos no infectados
        for vecino in red.get(nodo_actual, []):
            if vecino not in infectados:
                infectados[vecino] = hora + 1
                cola.append((vecino, hora + 1))
                print(f"Hora {hora + 1}: El nodo [{vecino}] ha sido comprometido por [{nodo_actual}]")

    return infectados
# Mapa de la Red Corporativa
red_empresa = {
    "PC_Ventas_01": ["Switch_Oficina", "PC_Ventas_02"],
    "PC_Ventas_02": ["PC_Ventas_01", "Switch_Oficina"],
    "Switch_Oficina": ["Router_Principal", "PC_Ventas_01", "PC_Ventas_02"],
    "Router_Principal": ["Switch_Oficina", "Firewall_Core"],
    "Firewall_Core": ["Router_Principal", "Servidor_AD", "Switch_DataCenter"],
    "Switch_DataCenter": ["Firewall_Core", "DB_Produccion", "Backup_Server"],
    "Servidor_AD": ["Firewall_Core"],
    "DB_Produccion": ["Switch_DataCenter"],
    "Backup_Server": ["Switch_DataCenter"]
}

# Ejecución de la simulación
log_infeccion = simular_propagacion(red_empresa, "PC_Ventas_01")
```

Resultados e Impacto en ciberseguridad

- **Identificación de Nodos Críticos:** La simulación muestra que el Firewall_Core es el punto de inflexión. Si este nodo cae, el malware accede al Data Center en solo 1 salto adicional.
- **Cálculo del Tiempo de Respuesta:** Si el equipo de IT tarda 3 horas en reaccionar, la simulación revela que para la **Hora 3**, el Firewall_Core ya habría sido alcanzado, comprometiendo toda la red.
- **Estrategia de Contención:** Gracias al BFS, se pueden identificar "puntos de estrangulamiento" (*choke points*) donde colocar sistemas de detección de intrusos (IDS) para romper la cadena de infección.

Reflexión Técnica

BFS es la herramienta estándar para el análisis de alcance en redes. Mientras que Dijkstra busca la ruta más rápida, BFS nos da la expansión total.

En un escenario real de ciberseguridad, esta simulación se refina convirtiéndola en un BFS Estocástico, donde la infección de un vecino no es segura (100%), sino que depende de una probabilidad basada en las vulnerabilidades detectadas en cada nodo.