Ejercicio 6: Network Load Balancer (NLB) y Gateway Load Balancer (GWLB)

Instrucciones:

- **Explica qué es un Network Load Balancer (NLB) y cuándo sería más adecuado utilizarlo en lugar de otros tipos de balanceadores de carga.**

  Es un tipo de balanceador que destruye el tráfico de red de manera eficiente en múltiples destinos como servidor o instancias virtuales dentro de una red

  Cuando utilizar NLB:

  Alta capacidad de manejo de tráfico

  Aplicaciones y servicios sensibles a la dirección IP de origen

  Es mejor utilizar los balanceadores de carga cuando se necesita manejar tráfico de red a nivel de capa 4 (protocolos TCP/UDP), ya que opera a ese nivel de la pila de protocolos**.**

- **Describe el concepto de Gateway Load Balancer (GWLB) y sus aplicaciones en una arquitectura de red.**

  Es un servicio o dispositivo que combina las funcionalidades de un balanceador de carga, que distribuye el tráfico de red entrante entre múltiples instancias de destino para asegurar que ninguna instancia individual esté sobrecargada, y un gateway de red, que actúa como un punto de conexión entre dos redes diferentes, permitiendo la comunicación y transferencia de datos entre ellas.

  El Gateway Load Balancer (GWLB) tiene aplicaciones clave en seguridad y cumplimiento (firewalls de próxima generación, IDS/IPS, filtrado de contenido), optimización de red (distribución de tráfico, arquitecturas híbridas), monitoreo y análisis (monitoreo de tráfico, análisis de rendimiento), y desempeño y escalabilidad (alta disponibilidad, escalado automático).

- **Proporciona un caso de uso donde un NLB y un GWLB trabajen juntos para mejorar la disponibilidad y seguridad de una aplicación**

  Caso:
  Una empresa de comercio electrónico necesita asegurar la alta disponibilidad y seguridad de su aplicación web. La aplicación es crítica para el negocio, y cualquier tiempo de inactividad o brecha de seguridad puede resultar en pérdidas significativas.

  Network Load Balancer (NLB): Proporciona alta disponibilidad distribuyendo el tráfico entrante entre múltiples servidores de aplicaciones.

  Gateway Load Balancer (GWLB): Añade una capa de seguridad, inspeccionando y filtrando el tráfico antes de que llegue a los servidores de aplicaciones.

Ejercicio 6: Simulación de un sistema de firewall para aplicaciones Web (WAF)

Objetivo: Implementar un sistema de firewall para aplicaciones web.

Instrucciones:

- Implementa una clase WebApplicationFirewall que permita definir y aplicar reglas de protección.
- Define reglas para bloquear solicitudes basadas en IP, prevenir ataques de SQL injection y XSS.
- Implementa una función que aplique el WAF a un conjunto de servidores detrás de un balanceador de carga.

```
class WebApplicationFirewall:
  def __init__(self):
      self.rules = []
  def add_rule(self, rule, name=None): #Añade la regla a la lista de reglas
      if name is None:
          name = rule.__name__
      self.rules.append((rule, name))
  def apply_rules(self, request): #Examina si la consulta pasa todas las
reglas
      for rule, name in self.rules:
          if not rule(request):
              print(f"Request blocked by rule: {name}")
              return False
      return True

def block_ip_rule(ip, blocked_ips): #Verifica si la ip de la consulta no
pertenece a la lista de ips bloqueadas
  return ip not in blocked_ips

def sql_injection_rule(query): #verifica que en las consultas no se encuentre
ninguna de esas ordenes
  if any(keyword in query.lower() for keyword in ["select", "drop", "insert",
"delete"]):
      return False
  return True

def xss_rule(content): #Verifica que la consulta no sea un condigo
  if "<script>" in content.lower():
      return False
  return True
```

```
# Ejemplo de servers
servers = ["Server 1", "Server 2", "Server 3"]

def load_balancer(request): # Distribucion de la carga entre cada servido
  server = servers[load_balancer.counter % len(servers)]
  load_balancer.counter += 1
  return server

load_balancer.counter = 0
blocked_ips = []
# WAF instance
waf = WebApplicationFirewall()
waf.add_rule(lambda req: block_ip_rule(req["ip"], blocked_ips),
"block_ip_rule")
waf.add_rule(lambda req: sql_injection_rule(req["query"]),
"sql_injection_rule")
waf.add_rule(lambda req: xss_rule(req["content"]), "xss_rule")

# Ejemplo de consultas
requests = [
  {"ip": "192.168.0.2", "query": "SELECT * FROM users", "content": "Hello
World"},
  {"ip": "192.168.0.1", "query": "DROP TABLE users", "content": "Hello
World"},
  {"ip": "192.168.0.3", "query": "UPDATE users SET name='admin'", "content":
"Hello <script>alert('xss')</script>"},
  {"ip": "192.168.0.4", "query": "INSERT INTO users (name) VALUES ('user')",
"content": "Safe content"},
  {"ip": "192.168.0.4", "query": "INTO users (name) VALUES ('user')",
"content": "Safe content"}
]

# Iniciar consultas
for request in requests:
  if waf.apply_rules(request):
      server = load_balancer(request)
      print(f"Request passed through WAF and is being handled by {server}")
  else:
      blocked_ips.append(request['ip'])
      print("Request blocked by WAF")
```

segundo mio

```
#Constructor, inicializa el atributo reglas que es una lista vacia
class CortafuegosAplicacionWeb:
    def __init__(self):
#Almacena todas las funciones de reglas que se agregan para filtrar
solicitudes
        self.reglas = []

#Permite agregar nuevas reglas al cortafuegos y se agrega a la lista
    def agregar_reglas(self,regla):
        self.reglas.append(regla)

#Se verifica que la regla este correcta para poder aplicarla de lo
contrario da un mensaje de solicitud bloqueadas
    def aplicar_reglas(self, solicitud):
        for regla in self.reglas:
            if not regla(solicitud):
                print(f"Solicitud bloqueada por la regla::
{(regla.__name__)}")
                return False
            return True

#
def regla_bloqueo_ip(ip):
    ips_bloqueadas=["192.168.0.1"]
    return ip not in ips_bloqueadas

def regla_inyeccion_sql(query):
    palabras_clave = ["selct", "drop", "insert", "delete"]
    if any(palabras_clave in query.lower()for palabras_clave in
palabras_clave):
        return False
    return True

def reglas_xss(contenido):
    if "<script>" in contenido.lower():
        return False
    return True

#Example usage
waf = CortafuegosAplicacionWeb()
waf.agregar_reglas(lambda req: regla_bloqueo_ip(req["ip"]))
waf.agregar_reglas(lambda req: regla_inyeccion_sql(req["query"]))
waf.agregar_reglas(lambda req: reglas_xss(req["contenido"]))
```

```
solicitud = {"ip":"192.168.0.2","query":"SELECT * FROM users",
"content":"Hello World"}
if waf.aplicar_reglas(solicitud):
    print("La solicitud pasó a través del WAF")
```