Módulo 6: Seguridad de Sistemas y Redes

Lección 2: Defensas y Controles de Seguridad en Redes

Objetivos de la Lección

Al finalizar, el estudiante podrá:

1. Configurar **Listas de Control de Acceso (ACL)** y **firewalls** para proteger servicios de red.
2. Aplicar **TLS 1.3** para asegurar la confidencialidad e integridad de las comunicaciones.
3. Diferenciar los mecanismos de detección y bloqueo de **IDS/IPS**.
4. Implementar **honeypots**, **tarpits** y **CAPTCHA** como defensas complementarias.
5. Diseñar estrategias de **defensa en profundidad** y **Zero Trust** en redes modernas.
6. Aplicar buenas prácticas de **autenticación segura** usando *salt & pepper* y **Argon2id**.

Introducción a la Lección

Después de estudiar los principales tipos de ataques en red, el siguiente paso es aprender a **defender los sistemas**. La seguridad efectiva no depende de una sola herramienta, sino de **capas de protección** que funcionan juntas: filtrado, cifrado, detección, autenticación y monitoreo. Esta lección muestra cómo combinar controles técnicos (ACL, TLS, IDS/IPS) y medidas proactivas (honeypots, MFA, Zero Trust) para construir una red resiliente ante ataques reales.

Desarrollo del Tema

Listas de Control de Acceso (ACL)

Concepto general

Una **ACL (Access Control List)** define qué tráfico puede entrar o salir de una red, basándose en criterios como dirección IP, puerto o protocolo.
Actúa como un **filtro lógico** entre redes o interfaces.

Tipos de ACL

Tipo	Nivel	Ejemplo	Aplicación
Estándar	Capa 3 (IP)	Permite o bloquea por dirección IP origen.	Redes internas.
Extendida	Capas 3–4	Filtra por IP, puerto y protocolo.	Control de servicios específicos.
Dinámica	Basada en sesión	Se activa temporalmente.	VPNs o accesos programados.

Ejemplo práctico (Cisco IOS)

```
ip access-list extended WEBONLY
  permit tcp 192.168.10.0 0.0.0.255 any eq 443
  deny  ip any any
interface g0/0
  ip access-group WEBONLY in
```

Explicación

- `ip access-list extended WEBONLY`
    - Se crea la ACL extendida. Las ACL extendidas permiten filtrar por **IP origen, IP destino, protocolo y puerto** (más granular).

- permit tcp 192.168.10.0 0.0.0.255 any eq 443
    - Protocolo: TCP
    - Origen: 192.168.10.0/24
    - (wildcard 0.0.0.255 = permite hosts del .1 al .254)
    - Destino: cualquier IP (any)
    - Puerto destino: 443 (HTTPS)
        - Permite que esta red local solo acceda a HTTPS.
- `deny ip any any`
    - Aquí se bloquea **todo el tráfico que no sea lo permitido**.
- interface g0/0

  ip access-group WEBONLY in
    - Aplica la ACL WEBONLY (WEBONLY es el nombre de la ACL extendida)
    - En la interfaz g0/0
    - En dirección IN (tráfico que entra a esa interfaz)

¿**Qué hace esta configuración**?

- Permite solo tráfico HTTPS desde la red interna.
- Todo lo demás es bloqueado (regla implícita “deny all”).

Ejemplo en Linux con nftables

```
nft add rule inet filter input tcp dport {22,443} ct state
new,established accept

nft add rule inet filter input counter drop
```

Explicación

- `nft add rule inet filter input tcp dport {22,443} ct state new,established accept`
    - nft (nftables - son **contenedores lógicos** donde se organizan las reglas del firewall en Linux.)

    - En la tabla *inet*, dentro de la cadena *filter/input*, se procesan las reglas que controlan el tráfico entrante del sistema.
    - Permite TCP hacia los puertos 22 (SSH) y 443 (HTTPS), esto solo si la conexión está en estado new o established.
    - Acción: accept
- `nft add rule inet filter input counter drop`
    - Agrega un contador (para estadísticas)
    - **Bloquea todo lo demás** que llegue a input

Filtra únicamente puertos **SSH (22)** y **HTTPS (443)**, bloqueando los demás.

Buenas prácticas

- Documentar todas las reglas y justificar su existencia.
- Habilitar *logging* de intentos denegados.
- Revisar las ACL trimestralmente.
- Evitar “permit any any” en ambientes de producción.

Firewalls: Primera Línea de Defensa

Definición

Un **firewall** controla el flujo de tráfico entre redes, aplicando políticas de seguridad definidas.

Existen varios tipos:

Tipo	Descripción	Ejemplo
Packet Filter (https://www.welivesecurity.com/es/recursos-herramientas/tcp-ip-protocol-perspectiva-ciberseguridad/**)**	Evalúa encabezados IP/TCP (stateless).	iptables, nftables.

Tipo	Descripción	Ejemplo
Stateful Firewall (https://www.checkpoint.com/es/cyber-hub/network-security/what-is-firewall/what-is-a-stateful-firewall/stateful_vs_stateless_firewall/**)**	Este es un firewall que mantiene un "estado" o almacena información sobre las conexiones de red activas. Cuando se abre una conexión, el firewall comienza a rastrearla y actualiza su estado interno a medida que el firewall inspecciona y procesa nuevos paquetes.	pfSense, Cisco ASA.
NGFW (Next-Gen Firewall) (https://www.cisco.com/site/us/en/learn/topics/security/what-is-a-next-generation-firewall.html**)**	Un firewall de nueva generación (NGFW) es un dispositivo de seguridad de red que, además de la inspección con control de estado típica de un firewall tradicional, integra reconocimiento y control de aplicaciones, prevención de intrusiones y capacidades de inteligencia de amenazas basadas en la nube para ofrecer una protección superior.	FortiGate, Palo Alto.

Ejemplo nftables (moderno)

```
table inet filter {
  chain input {
    type filter hook input priority 0;
    tcp dport {22,443} ct state new,established accept
```

```
    ip saddr 10.0.0.0/8 drop
    ct state established,related accept
    counter drop
  }
}
```

Bloquea redes privadas externas y permite solo tráfico controlado.

Tendencias 2025

- Integración con **IA/ML** para detección de anomalías.
- Firewalls “as-a-service” en nubes híbridas (AWS Network Firewall).
- Auditorías automáticas con *Terraform* + *Checkov* (Infra-as-Code).

HTTPS y TLS 1.3 — Cifrado en Tránsito

Concepto

TLS (Transport Layer Security) cifra la comunicación entre cliente y servidor, garantizando:

- **Confidencialidad:** los datos no pueden ser interceptados.
- **Integridad:** los datos no pueden alterarse.
- **Autenticidad:** el servidor (y opcionalmente el cliente) se valida mediante certificados.

Configuración básica en Nginx

```
ssl_protocols TLSv1.3;
ssl_ciphers TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256;
ssl_certificate     /etc/letsencrypt/live/site/fullchain.pem;
```

```
ssl_certificate_key /etc/letsencrypt/live/site/privkey.pem;

add_header Strict-Transport-Security "max-age=63072000;
includeSubDomains";
```

Explicación

Esta configuración hace que Nginx (NGINX = “Engine-X”) use solo TLS 1.3, cifrados modernos, certificados de Let’s Encrypt y políticas HSTS, ofreciendo un HTTPS seguro y actualizado**.**

Buenas prácticas:

- Usar **TLS 1.3** (más rápido y seguro que 1.2).
- Configurar **HSTS (HTTP Strict Transport Security)**.
- Activar **OCSP Stapling** para validar certificados.
- Automatizar la renovación con Let’s Encrypt:
- certbot renew --quiet --deploy-hook "systemctl reload nginx"

Herramientas de prueba:

- testssl.sh — evalúa seguridad del servidor HTTPS.
- ssllabs.com — califica la fortaleza del cifrado.

IDS e IPS — Detección y Prevención de Intrusiones

Definiciones:

Tipo	Función	Ejemplo
IDS (Intrusion Detection System)	Detecta y alerta sobre actividad sospechosa.	Suricata IDS, Zeek, Snort3.

Tipo	Función	Ejemplo
IPS (Intrusion Prevention System)	Bloquea tráfico malicioso en tiempo real.	Suricata en modo inline, pfSense IPS.

Ejemplo de regla en Suricata

```
alert http any any -> any any (msg:"XSS attempt"; content:"<script"; nocase; sid:100001;)
```

Detecta intentos de inyección de código XSS.

Comparación técnica

Aspecto	IDS	IPS
Función	Detección y registro	Detección + bloqueo
Latencia	Baja	+1–3 ms
Falsos positivos	Tolerables	Críticos, puede bloquear tráfico legitimo
Implementación	SPAN (Permite que un IDS vea el tráfico sin estar en el camino del flujo.) / TAP (Es un dispositivo físico insertado en el cable de red que duplica el tráfico eléctricamente, no pierde trafico)	Bridge o inline

Buenas prácticas

- Integrar logs con **Elastic SIEM** o **Wazuh**.
- Actualizar firmas (ET Open, Abuse.ch).
- Crear reglas personalizadas según políticas internas.

Honeypots y Tarpits — Detección Temprana y Análisis

Concepto

Sistemas señuelo diseñados para atraer, registrar y analizar el comportamiento de atacantes.

Tipos de honeypots

Tipo	Descripción	Ejemplo
HIH (High-Interaction)	Imitan sistemas reales (Windows, Linux).	VirtualBox con Windows Server honeypot.
LIH (Low-Interaction)	Emulan servicios comunes.	Cowrie (SSH/Telnet), Dionaea (malware).

Ejemplo con Cowrie

```
docker run -d -p 2222:2222 cowrie/cowrie
```

Explicación:

- *-d* = lanza contenedro en segundo plano
- *2222:2222* = Expone el puerto 2222 del contenedor en el puerto 2222 del host. Esto significa que los atacantes podrán conectarse a tu honeypot SSH falso a través de:ssh usuario@IP-del-servidor -p 2222
- *Cowrie/Cowrie* = Cowrie es un honeypot de alta interacción para SSH y Telnet.

El pasado ejemplo lo que se quiere es capturar intentos de acceso SSH falsos y registrar comandos ejecutados.

Tarpit

Utiliza reglas de firewall para **ralentizar** los ataques:

```
iptables -A INPUT -p tcp --syn -j TARPIT
```

Hace que los escáneres de puertos permanezcan atrapados, reduciendo el impacto.

Autenticación Segura — “Salt & Pepper” y Argon2id

Concepto

- **Salt & Pepper:**
  - **Salt:** Valor aleatorio único para cada contraseña que se mezcla antes del hash para evitar ataques de tablas rainbow.

  - **Pepper:** Clave secreta global, almacenada fuera de la base de datos, que se añade adicionalmente al hash para dificultar ataques incluso si la BD es comprometida.
- **Argon2id:**
  - **Algoritmo moderno** de hashing de contraseñas, resistente a ataques de GPU/ASIC, que combina defensa contra ataques basados en CPU (Argon2i) y ataques de side-channel (Argon2d).

Problema

Las contraseñas almacenadas con hashes débiles (MD5, SHA-1) pueden ser descifradas mediante **diccionarios o rainbow tables**.

Solución moderna

Usar **Argon2id**, un algoritmo resistente a ataques de GPU y CPU paralelas.

```
import os, argon2id


salt   = os.urandom(16)

pepper = os.getenv("PEPPER")

hashed = argon2.low_level.hash_secret(

    (b"password123" + salt + pepper.encode()),

    salt,
```

```
    time_cost=3,
    memory_cost=64*1024,
    parallelism=4,
    type=argon2.low_level.Type.ID
)
```

Explicación:

El código genera un salt, obtiene un pepper secreto, mezcla ambos con la contraseña, y genera un hash Argon2id altamente seguro usando parámetros recomendados para proteger contraseñas modernas.

Buenas prácticas

- Almacenar el “pepper” en un **HSM** o gestor de secretos.
- Re-hashear contraseñas al iniciar sesión si se actualiza el algoritmo.
- Implementar **MFA (Multi-Factor Authentication)** para usuarios críticos.
- Aplicar bloqueo progresivo tras varios intentos fallidos.

CAPTCHA y Control de Bots

Concepto
Un **CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart)** es un mecanismo de seguridad que sirve para distinguir si quien interactúa con un sitio web es un humano o un bot.

Propósito: impedir que bots automáticos realicen acciones masivas (login, registro, spam).

Tipos comunes

Tipo	Descripción	Ejemplo
ReCAPTCHA v3	Asigna puntuación según comportamiento.	Google reCAPTCHA.
hCaptcha	Basado en imágenes, compatible con privacidad.	hcaptcha.com.
Friendly-Captcha	**Friendly-Captcha** es un sistema de verificación anti-bots que reemplaza los CAPTCHAs tradicionales sin requerir que el usuario “demuestre” nada (no imágenes, no clics, no puzzles visuales). Se usan pruebas critpograficas del lado del navegador, Privacidad por Diseño y Brinda accesibilidad e invisibilidad (el usuario no nota que hay un CAPTCHA).	Implementación local.

Buenas prácticas:

- Mostrar CAPTCHA solo tras detección de IP sospechosa.
- Ofrecer versión accesible (audio).
- Integrar con políticas de riesgo adaptativo.

Defensa en Profundidad y Zero Trust

Defensa en Profundidad **¿Qué es la Defensa en Profundidad?**

Es una estrategia de ciberseguridad que utiliza **múltiples capas de protección** (red, host, aplicación, usuarios, monitoreo) para que, si una capa falla, otras continúen deteniendo o mitigando el ataque.

Estrategia que combina múltiples capas de protección:

1. **Perímetro:** Firewalls, ACL.
2. **Cifrado:** TLS 1.3, VPN.

3. **Detección:** IDS/IPS, honeypots.
4. **Autenticación:** Argon2id, MFA.
5. **Monitoreo:** SIEM, alertas en tiempo real.

Ventaja: si una capa falla, otra la respalda.

Zero Trust (ZTNA – Zero Trust Network Access)

Principio moderno basado en *"nunca confiar, siempre verificar"*.
Cada usuario o dispositivo debe autenticarse y autorizarse para cada acción, sin asumir confianza interna.

Ejemplo de implementación:

- Autenticación continua (identity tokens + MFA).
- Micro-segmentación con **Calico** o **Istio**.
- Políticas OPA (Open Policy Agent) que definen quién puede hacer qué:

```
allow {
  input.user.role == "admin"
  input.action == "read"
}
```

Relación con Otros Conceptos

- Las ACL y firewalls aplican los principios de **control de acceso** vistos en módulos anteriores.
- TLS y cifrado se conectan con los temas de **criptografía aplicada**.
- IDS/IPS y honeypots complementan los procesos de **detección y respuesta a incidentes**.
- Argon2id y MFA refuerzan la autenticación discutida en la sección de **seguridad de sistemas**.

Resumen de la Lección

- Las ACL y firewalls filtran tráfico no autorizado desde la capa de red.
- TLS 1.3 asegura confidencialidad y autenticidad en las comunicaciones.
- IDS e IPS ofrecen visibilidad y bloqueo de ataques en tiempo real.
- Honeypots y tarpits permiten observar comportamientos de atacantes.
- Argon2id, sal y pepper protegen las contraseñas contra cracking masivo.
- CAPTCHA y Zero Trust fortalecen las defensas frente a bots y accesos internos.
- En conjunto, estos mecanismos conforman una **defensa en profundidad** efectiva.

Actividad de la Lección — “Construye tu mini-DMZ” (120 min)

Objetivo: aplicar controles de red y defensa en profundidad en un entorno práctico.

Escenario: máquina virtual Ubuntu Server + contenedor Nginx.

Tareas (por equipos de 2–3 estudiantes):

1. Crear reglas de firewall (iptables o nftables) que permitan solo **SSH (2222)** y **HTTPS (443)**.
2. Generar un certificado con **Let’s Encrypt** e implementar **TLS 1.3**.
3. Instalar **Suricata IDS** y configurar una regla personalizada (XSS).
4. Desplegar **Cowrie** como honeypot SSH en el puerto 2223.
5. Configurar autenticación con **Argon2id + salt/pepper** y **reCAPTCHA v3** en /login.

6. Documentar evidencias (capturas, logs, alertas IDS, conexión al honeypot).

Entregable:

- Script de configuración (setup_dmz.sh o Ansible playbook).
- Documento con capturas y análisis de resultados (PDF o Word).

Referencias Recomendadas

- Pfleeger, C. P., Pfleeger, S. L., & Coles-Kemp, L. (2023). *Security in Computing* (6th ed.).
- OWASP (2025). *Server-Side TLS Cheat Sheet & Top 10 Security Controls.*
- NIST SP 800-94 (2023). *Guide to Intrusion Detection and Prevention Systems.*
- Cisco (2024). *Configuring Access Control Lists in IOS-XE.*
- Project Honeypot (2024). *Cowrie and Modern Honeypots Whitepaper.*
- ENISA (2024). *Network Security Best Practices for SMEs.*