UNIVERSIDAD DE CALDAS
LUMINA SPARGO ®

FACULTAD DE
INGENERÍAS

Segundo Parcial

Ciberseguridad
Segundo parcial

Over Haider Castrillón Valencia

Facultad de Inteligencia Artificial e Ingenierías
Facultad de Inteligencia Artificial e Ingenierías
Ingeniería en Sistemas y Computación

Manizales Caldas, octubre 28 de 2025

This page intentionally left blank.

FACULTAD DE INGENERÍAS

Segundo Parcial

Over Haider Castrillón Valencia

Docente: Gustavo Isaza Echeverry

Facultad de Inteligencia Artificial e Ingenierías

Facultad de Inteligencia Artificial e Ingenierías

Ingeniería en Sistemas y Computación

Manizales Caldas, octubre 28 de 2025

This page intentionally left blank.

Índice general

This page intentionally left blank.

1 Planteamientos

1.1. Problema 1 (40 %)

Con base en la arquitectura definida en la Figura, modelar un conjunto de reglas vía NetFilter/IPTables, así:

1.1.1. Análisis de la Arquitectura de Red

Analizando la arquitectura mostrada, parece que se presenta la topología clásica de **defensa en profundidad** con las tres zonas de seguridad claramente diferenciadas:

- **Internet (Zona no confiable):** IP pública del router: 80.37.120.42
- **DMZ (Zona semi-confiable):** Red 212.194.89.150/30
  - Gateway: 212.194.89.151 (interfaz eth2 del firewall)
  - Servidor expuesto: 212.194.89.152
  - Broadcast: 212.194.89.153
- **LAN Interna (Zona confiable):** Red 192.168.10.0/24
  - Gateway: 192.168.10.1 (interfaz eth1 del firewall)

Interfaces del Firewall:

- `eth0`: 80.37.120.43 (interfaz externa hacia Internet)

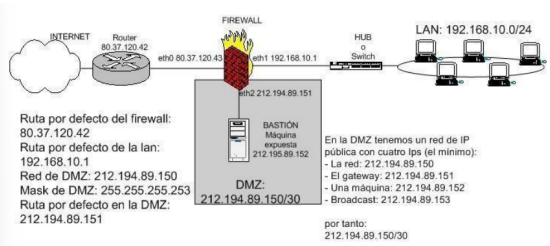

Figura 1.1: *Topología de red*

- `eth1`: 192.168.10.1 (interfaz interna hacia LAN)
- `eth2`: 212.194.89.151 (interfaz DMZ)

1.1.2. Literal a

Permitir el tráfico de los clientes LAN a los Web Servers de la DMZ (*.151, *.152):

Para permitir el acceso desde la red LAN hacia los servidores web en la DMZ, implementamos las siguientes reglas:

```
# Permitir HTTP (puerto 80) desde LAN hacia servidor DMZ .151
iptables -A FORWARD -s 192.168.10.0/24 -d 212.194.89.151 \
         -p tcp --dport 80 -m state --state NEW,ESTABLISHED -j
            ACCEPT

# Permitir HTTP (puerto 80) desde LAN hacia servidor DMZ .152
iptables -A FORWARD -s 192.168.10.0/24 -d 212.194.89.152 \
         -p tcp --dport 80 -m state --state NEW,ESTABLISHED -j
            ACCEPT

# Permitir HTTPS (puerto 443) desde LAN hacia servidor DMZ .151
iptables -A FORWARD -s 192.168.10.0/24 -d 212.194.89.151 \
         -p tcp --dport 443 -m state --state NEW,ESTABLISHED -j
            ACCEPT

# Permitir HTTPS (puerto 443) desde LAN hacia servidor DMZ .152
iptables -A FORWARD -s 192.168.10.0/24 -d 212.194.89.152 \
         -p tcp --dport 443 -m state --state NEW,ESTABLISHED -j
            ACCEPT

# Permitir DNS TCP desde LAN hacia servidores DMZ
iptables -A FORWARD -s 192.168.10.0/24 -d 212.194.89.151 \
         -p tcp --dport 53 -m state --state NEW,ESTABLISHED -j
            ACCEPT
iptables -A FORWARD -s 192.168.10.0/24 -d 212.194.89.152 \
         -p tcp --dport 53 -m state --state NEW,ESTABLISHED -j
            ACCEPT

# Permitir DNS UDP desde LAN hacia servidores DMZ
iptables -A FORWARD -s 192.168.10.0/24 -d 212.194.89.151 \
         -p udp --dport 53 -m state --state NEW,ESTABLISHED -j
            ACCEPT
iptables -A FORWARD -s 192.168.10.0/24 -d 212.194.89.152 \
         -p udp --dport 53 -m state --state NEW,ESTABLISHED -j
            ACCEPT
```

Listing 1.1: *Reglas para tráfico LAN hacia DMZ*

Justificación técnica:

- Utilizamos el módulo `state` con `NEW,ESTABLISHED` para permitir conexiones nuevas y establecidas.
- Se incluyen tanto HTTP (80) como HTTPS (443) para soportar tráfico web seguro.
- DNS (53) se permite en TCP y UDP para resolución de nombres completa.
- Las reglas son específicas por dirección IP de destino para mayor control.

1.1.3. Literal b

Permitir el tráfico desde Internet hacia los servidores de la DMZ WWW & email Server

Para exponer los servicios públicos de la DMZ a Internet:

```
# === SERVICIOS WEB ===
# Permitir HTTP desde Internet hacia ambos servidores web DMZ
iptables -A FORWARD -i eth0 -d 212.194.89.151 \
         -p tcp --dport 80 -m state --state NEW,ESTABLISHED -j
            ACCEPT
iptables -A FORWARD -i eth0 -d 212.194.89.152 \
         -p tcp --dport 80 -m state --state NEW,ESTABLISHED -j
            ACCEPT

# Permitir HTTPS desde Internet hacia ambos servidores web DMZ
iptables -A FORWARD -i eth0 -d 212.194.89.151 \
         -p tcp --dport 443 -m state --state NEW,ESTABLISHED -j
            ACCEPT
iptables -A FORWARD -i eth0 -d 212.194.89.152 \
         -p tcp --dport 443 -m state --state NEW,ESTABLISHED -j
            ACCEPT

# === SERVICIOS DE EMAIL (asumiendo .152 como servidor de correo)
   ===
# SMTP est ndar (puerto 25)
iptables -A FORWARD -i eth0 -d 212.194.89.152 \
         -p tcp --dport 25 -m state --state NEW,ESTABLISHED -j
            ACCEPT

# SMTP seguro - SMTPS (puerto 465)
iptables -A FORWARD -i eth0 -d 212.194.89.152 \
         -p tcp --dport 465 -m state --state NEW,ESTABLISHED -j
            ACCEPT

# SMTP con STARTTLS (puerto 587)
iptables -A FORWARD -i eth0 -d 212.194.89.152 \
         -p tcp --dport 587 -m state --state NEW,ESTABLISHED -j
            ACCEPT

```

```
# IMAP (puerto 143) e IMAPS (puerto 993)
iptables -A FORWARD -i eth0 -d 212.194.89.152 \
         -p tcp --dport 143 -m state --state NEW,ESTABLISHED -j
            ACCEPT
iptables -A FORWARD -i eth0 -d 212.194.89.152 \
         -p tcp --dport 993 -m state --state NEW,ESTABLISHED -j
            ACCEPT

# POP3 (puerto 110) y POP3S (puerto 995)
iptables -A FORWARD -i eth0 -d 212.194.89.152 \
         -p tcp --dport 110 -m state --state NEW,ESTABLISHED -j
            ACCEPT
iptables -A FORWARD -i eth0 -d 212.194.89.152 \
         -p tcp --dport 995 -m state --state NEW,ESTABLISHED -j
            ACCEPT
```

Listing 1.2: *Reglas para servicios públicos en DMZ*

Justificación técnica:

- Se especifica la interfaz de entrada `-i eth0` para asegurar que el tráfico provenga de Internet.
- Se incluyen todos los protocolos de correo estándar (SMTP, IMAP, POP3) y sus versiones seguras.
- Las reglas utilizan el módulo de estado para mantener el seguimiento de conexiones.

1.1.4. Literal c

Proteger el Router de la DMZ y de la LAN para tráfico ICMP y Telnet:
Para proteger el router de accesos no autorizados:

```
# === BLOQUEO DE TELNET ===
# Bloquear Telnet desde DMZ hacia el router
iptables -A FORWARD -s 212.194.89.150/30 -d 80.37.120.42 \
         -p tcp --dport 23 -j DROP

# Bloquear Telnet desde LAN hacia el router
iptables -A FORWARD -s 192.168.10.0/24 -d 80.37.120.42 \
         -p tcp --dport 23 -j DROP

# === PROTECCI N ICMP ===
# Limitar ICMP echo-request (ping) desde DMZ - Anti-DoS
iptables -A FORWARD -s 212.194.89.150/30 -d 80.37.120.42 \
         -p icmp --icmp-type echo-request \
         -m limit --limit 1/s --limit-burst 3 -j ACCEPT

# Limitar ICMP echo-request (ping) desde LAN - Anti-DoS
iptables -A FORWARD -s 192.168.10.0/24 -d 80.37.120.42 \
```

```
18          -p icmp --icmp-type echo-request \
19          -m limit --limit 1/s --limit-burst 3 -j ACCEPT
20
21 # Permitir ICMP echo-reply desde DMZ
22 iptables -A FORWARD -s 212.194.89.150/30 -d 80.37.120.42 \
23          -p icmp --icmp-type echo-reply -j ACCEPT
24
25 # Permitir ICMP echo-reply desde LAN
26 iptables -A FORWARD -s 192.168.10.0/24 -d 80.37.120.42 \
27          -p icmp --icmp-type echo-reply -j ACCEPT
28
29 # Bloquear todos los dem s tipos de ICMP hacia el router
30 iptables -A FORWARD -d 80.37.120.42 -p icmp -j DROP
31
32 # === LOGGING PARA AN LISIS FORENSE ===
33 # Registrar intentos de Telnet bloqueados
34 iptables -A FORWARD -d 80.37.120.42 -p tcp --dport 23 \
35          -j LOG --log-prefix "TELNET_BLOCKED_TO_ROUTER: " \
36          --log-level 4
```

Listing 1.3: *Reglas de protección del router*

Justificación técnica:

- **Bloqueo de Telnet:** Se deniega completamente el acceso Telnet (puerto 23) por ser un protocolo inseguro.
- **Limitación ICMP:** Se implementa rate limiting (1 ping/segundo) para prevenir ataques DoS.
- **Logging:** Se registran intentos de acceso no autorizado para análisis posterior.

1.1.5. Literal d

Precisar semánticamente una regla en de Detección de Intrusiones (basada en Snort) que permita alertar intentos de elevar privilegios (“sudo -i”) a cualquier servidor de la DMZ:

Regla Principal

```
1 # Regla principal para detectar "sudo -i" en tr fico hacia
     servidores DMZ
2 alert tcp any any -> 212.194.89.150/30 any (
3     msg:"PRIVILEGE_ESCALATION: Intento de elevacion de
         privilegios detectado (sudo -i)";
4     content:"sudo -i";
5     nocase;
6     classtype:attempted-admin;
7     priority:1;
8     sid:1000001;
```

```
    rev:1;
    reference:url,cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3156;
)
```

Listing 1.4: *Regla Snort para detectar sudo -i*

Análisis Semántico de la Regla

La regla Snort implementada tiene los siguientes componentes semánticos:

- **Encabezado de la regla:**
  - `alert`: Acción a tomar cuando se detecta la coincidencia.
  - `tcp`: Protocolo a monitorear.
  - `any any`: Cualquier IP y puerto de origen.
  - `->212.194.89.150/30 any`: Destino específico en la red DMZ, cualquier puerto.
- **Opciones de detección:**
  - `content:"sudo -i"`: Busca la cadena específica en el payload.
  - `nocase`: Hace la búsqueda insensible a mayúsculas/minúsculas.
  - `classtype:attempted-admin`: Categoriza el evento como intento administrativo.
  - `priority:1`: Asigna máxima prioridad (crítico).
  - `sid:1000001`: Identificador único de la regla.
  - `rev:1`: Versión de la regla.
  - `reference`: Enlace a CVE relacionado con vulnerabilidades de sudo.

Reglas Complementarias

```
# Detecci n de variantes de sudo con expresiones regulares
alert tcp any any -> 212.194.89.150/30 any (
    msg:"PRIVILEGE_ESCALATION: Comando sudo con opciones
       privilegiadas";
    content:"sudo";
    nocase;
    pcre:"/sudo\s+(-i|--login|-s|--shell|-u\s+root)/i";
    classtype:attempted-admin;
    priority:2;
    sid:1000002;
    rev:1;
)

# Detecci n de comando su (cambio a root)
alert tcp any any -> 212.194.89.150/30 any (
    msg:"PRIVILEGE_ESCALATION: Intento de cambio a usuario root";
    content:"su -";
```

```
    nocase;
    classtype:attempted-admin;
    priority:2;
    sid:1000003;
    rev:1;
)

# Detecci n de modificaci n de sudoers
alert tcp any any -> 212.194.89.150/30 any (
    msg:"PRIVILEGE_ESCALATION: Modificacion de archivo sudoers";
    content:"visudo";
    nocase;
    classtype:attempted-admin;
    priority:1;
    sid:1000004;
    rev:1;
)

# Regla avanzada con control de flujo y l mite de detecci n
alert tcp any any -> 212.194.89.150/30 any (
    msg:"PRIVILEGE_ESCALATION: M ltiples intentos de sudo
       detectados";
    content:"sudo";
    nocase;
    flow:to_server,established;
    threshold:type limit, track by_src, count 3, seconds 60;
    classtype:attempted-admin;
    priority:1;
    sid:1000005;
    rev:1;
)
```

Listing 1.5: *Reglas adicionales para detección completa*

Limitaciones y Consideraciones

1. **Evasión por codificación:** Los atacantes pueden codificar comandos en base64 o usar ofuscación.
2. **Tráfico cifrado:** No detectará comandos dentro de sesiones SSH o HTTPS.
3. **Falsos positivos:** Puede alertar sobre uso legítimo de sudo por administradores.
4. **Rendimiento:** Las reglas con PCRE pueden impactar el rendimiento en redes de alto tráfico.

1.1.6. Conclusiones

La implementación presentada proporciona:

1. **Segmentación efectiva:** Separación clara entre zonas de seguridad con control granular del tráfico.
2. **Principio de menor privilegio:** Solo se permite el tráfico estrictamente necesario.
3. **Defensa en profundidad:** Múltiples capas de seguridad con firewall stateful y IDS (Snort).
4. **Monitoreo y auditoría:** Logging detallado para análisis forense y detección de incidentes.
5. **Protección contra ataques comunes:** Anti-spoofing, rate limiting, y bloqueo de protocolos inseguros.

Esta configuración debe complementarse con:

- Actualización regular de reglas según nuevas amenazas
- Integración con SIEM para correlación de eventos
- Pruebas periódicas de penetración
- Hardening adicional de los sistemas en la DMZ
- Implementación de VPN para acceso remoto seguro

1.2. Problema 2 (25 %)

2. (25

1.2.1. Literal a

Brevemente describa el funcionamiento (fases) de SSL/TLS:

Descripción General del Protocolo

SSL/TLS (Secure Socket Layer/Transport Layer Security) es un protocolo criptográfico que proporciona comunicación segura sobre Internet. El protocolo establece una conexión cifrada entre un cliente (navegador) y un servidor, garantizando confidencialidad, integridad y autenticación de los datos transmitidos.

Fases del Protocolo SSL/TLS

Fase 1: Handshake (Protocolo de Enlace)
El handshake SSL/TLS es un proceso de negociación que establece los parámetros de seguridad para la comunicación:

1. **Client Hello:** El cliente inicia enviando un mensaje que incluye:
    - Versión de TLS soportada
    - Valor aleatorio del cliente (client random)
    - Lista de cipher suites disponibles
    - Extensiones soportadas

2. **Server Hello:** El servidor responde con:
   - Cipher suite seleccionado
   - Valor aleatorio del servidor (server random)
   - Certificado SSL/TLS con clave pública
   - Solicitud opcional de certificado del cliente
3. **Verificación del Certificado:** El cliente autentica el certificado del servidor verificando:
   - Validez temporal del certificado
   - Coincidencia con el dominio solicitado
   - Firma de una Autoridad de Certificación (CA) confiable
   - Estado de revocación del certificado
4. **Intercambio de Claves:** Generación del secreto compartido:
   - El cliente genera un pre-master secret
   - Lo cifra con la clave pública del servidor
   - El servidor lo descifra con su clave privada
   - Ambos derivan las claves de sesión usando client random, server random y pre-master secret
5. **Finalización:** Confirmación del handshake:
   - Cliente envía mensaje "Finishedçifrado con clave de sesión
   - Servidor envía mensaje "Finishedçifrado
   - Se establece el canal seguro

Fase 2: Capa de Registro (Record Layer)
Una vez establecido el handshake, la capa de registro maneja el cifrado simétrico de los datos:

- **Fragmentación:** Los datos se dividen en bloques manejables
- **Compresión:** Aplicación opcional de algoritmos de compresión
- **Autenticación:** Generación de MAC (Message Authentication Code)
- **Cifrado:** Aplicación de cifrado simétrico con las claves de sesión
- **Transmisión:** Envío de los datos cifrados a través de la red

1.2.2. Literal b

Describa las principales diferencias entre un modelo de certificación basado en PKI con certificados digitales y el estándar PGP:

Modelo PKI (Public Key Infrastructure)

PKI es una infraestructura jerárquica centralizada que gestiona certificados digitales:
Características principales:

- **Autoridad Central:** Las Autoridades de Certificación (CA) actúan como terceros confiables
- **Estructura Jerárquica:** Modelo de árbol con CA raíz y CA subordinadas
- **Certificados X.509:** Formato estándar que incluye clave pública, identidad y firma de la CA
- **Validación Centralizada:** La confianza se basa en la verificación de la cadena de certificación
- **Gestión Corporativa:** Ideal para organizaciones con políticas de seguridad centralizadas

Proceso de certificación PKI:

1. Usuario genera par de claves (pública/privada)
2. Solicita certificado a la CA con su clave pública
3. CA verifica identidad del solicitante
4. CA firma el certificado con su clave privada
5. Certificado se distribuye y puede ser verificado por terceros

Modelo PGP (Pretty Good Privacy)

PGP implementa un modelo de red de confianza"descentralizado:
Características principales:

- **Red de Confianza:** No existe autoridad central, la confianza se construye peer-to-peer
- **Firmas Cruzadas:** Los usuarios firman las claves de otros usuarios para validar identidades
- **Niveles de Confianza:** Sistema graduado de confianza basado en relaciones personales
- **Autogestión:** Cada usuario decide en quién confiar sin intermediarios
- **Flexibilidad Algorítmica:** Soporta múltiples algoritmos (RSA/MD5, DH/SHA-1)

Proceso de certificación PGP:

1. Usuario A genera par de claves y las publica
2. Usuario B conoce personalmente a A y verifica su identidad
3. Usuario B firma la clave pública de A con su clave privada
4. La firma se agrega al certificado de A
5. Otros usuarios pueden confiar en A basándose en la firma de B

Comparación Detallada

1.2.3. Literal c

¿Por qué las funciones de hashing son el núcleo de la firmas digitales y del blockchains?

Cuadro 1.1: *PKI vs PGP: Comparación de Modelos*

Aspecto	**PKI**	**PGP**
Estructura	Jerárquica centralizada	Red de confianza descentralizada
Autoridad	CA como tercero confiable	Sin autoridad central
Validación	Cadena de certificación	Firmas cruzadas de usuarios
Escalabilidad	Alta para organizaciones	Limitada por relaciones personales
Costo	Alto (infraestructura CA)	Bajo (sin infraestructura central)
Gestión	Centralizada y automatizada	Descentralizada y manual
Revocación	CRL/OCSP centralizados	Difícil sin autoridad central
Uso típico	Empresas, e-commerce	Comunicaciones personales

1.2.4. Literal C: Funciones Hash como Núcleo de Firmas Digitales y Blockchain

Propiedades Fundamentales de las Funciones Hash

Las funciones hash son algoritmos matemáticos que transforman datos de cualquier tamaño en una cadena de longitud fija:
Propiedades criptográficas esenciales:

- **Determinismo:** El mismo input siempre produce el mismo hash
- **Eficiencia:** Cálculo rápido del hash para cualquier input
- **Irreversibilidad:** Imposible reconstruir el input original desde el hash
- **Resistencia a colisiones:** Extremadamente difícil encontrar dos inputs con el mismo hash
- **Efecto avalancha:** Pequeños cambios en el input causan cambios drásticos en el hash

Rol en las Firmas Digitales

Las funciones hash son fundamentales en las firmas digitales por las siguientes razones:
1. Eficiencia Computacional

- Firmar directamente documentos grandes sería computacionalmente costoso
- El hash reduce cualquier documento a una huella de tamaño fijo (ej: 256 bits para SHA-256)
- Se firma el hash en lugar del documento completo, optimizando el proceso

2. Integridad de Datos

- El hash actúa como "huella digital"única del documento
- Cualquier alteración del documento cambia completamente el hash
- La verificación de la firma detecta automáticamente modificaciones

3. Proceso de Firma Digital

1. Se calcula el hash del mensaje original
2. El hash se cifra con la clave privada del firmante

3. El resultado cifrado es la firma digital
4. Para verificar: se descifra la firma con la clave pública y se compara con el hash del mensaje recibido

Rol en Blockchain

En blockchain, las funciones hash son el núcleo de la seguridad y funcionalidad:
1. Integridad de Bloques

- Cada bloque contiene el hash de todas sus transacciones
- Cualquier modificación en una transacción cambia el hash del bloque completo
- Esto hace que la manipulación sea inmediatamente detectable

2. Enlace de Bloques

- Cada bloque incluye el hash del bloque anterior
- Esto crea una cadena criptográficamente enlazada
- Modificar un bloque histórico requiere recalcular todos los bloques posteriores

3. Prueba de Trabajo (Proof of Work)

- Los mineros buscan un nonce que produzca un hash con características específicas
- Bitcoin usa SHA-256 para este proceso
- La dificultad computacional protege la red contra ataques

4. Merkle Trees

- Estructura de datos que usa hashes para organizar transacciones
- Permite verificación eficiente de transacciones sin descargar todo el bloque
- El hash raíz representa todas las transacciones del bloque

1.2.5. Literal d

Diagrame y defina el pipeline para firmar y verificar un mensaje usando sha256withRSA:

1.2.6. Literal D: Pipeline SHA256withRSA - Firma y Verificación

Descripción del Algoritmo

SHA256withRSA es un esquema de firma digital que combina la función hash SHA-256 con el algoritmo de clave pública RSA. Este método implementa el estándar RSASSA-PKCS1-v1_5 definido en RFC 3447.

Diagrama del Pipeline

Proceso Detallado de Firma

Paso 1: Generación del Hash

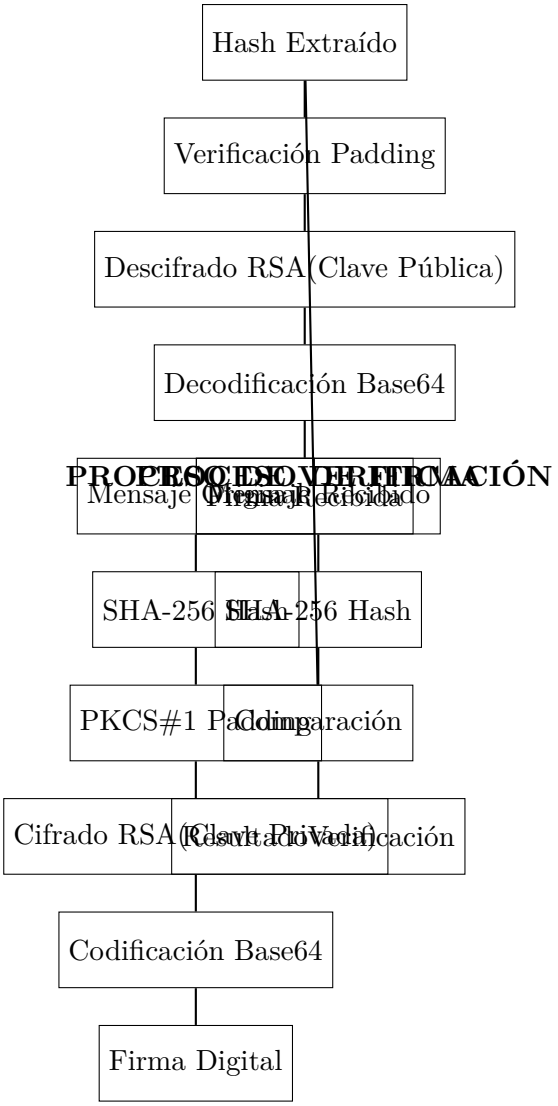

Figura 1.2: *Pipeline SHA256withRSA - Firma y Verificación*

```
FUNCTION generateHash(message):
    // Aplicar funci n hash SHA-256
    hash = SHA256(message)
    // Resultado: 32 bytes (256 bits)
    RETURN hash
END FUNCTION
```

Listing 1.6: *Cálculo del hash SHA-256*

Paso 2: Aplicación de Padding PKCS#1

```
FUNCTION applyPKCS1Padding(hash, keySize):
    // Identificador de algoritmo SHA-256
    algorithmId = "3031300d060960864801650304020105000420"

    // Construir estructura ASN.1
    digestInfo = algorithmId + hash
```

```
7
8       // Calcular padding necesario
9       paddingLength = keySize - length(digestInfo) - 3
10      padding = "FF" repetido paddingLength veces
11
12      // Construir mensaje con padding
13      paddedMessage = "0001" + padding + "00" + digestInfo
14
15      RETURN paddedMessage
16  END FUNCTION
```

Listing 1.7: *Padding PKCS#1 v1.5*

Paso 3: Firma RSA

```
1   FUNCTION signRSA(paddedMessage, privateKey):
2       // Convertir mensaje a entero
3       m = bytesToInteger(paddedMessage)
4
5       // Operaci n RSA: s = m^d mod n
6       // donde d es el exponente privado y n es el m dulo
7       signature = modularExponentiation(m, privateKey.d,
            privateKey.n)
8
9       // Convertir resultado a bytes
10      signatureBytes = integerToBytes(signature, privateKey.keySize)
11
12      RETURN signatureBytes
13  END FUNCTION
```

Listing 1.8: *Operación de firma RSA*

Paso 4: Codificación Base64

```
1   FUNCTION encodeSignature(signatureBytes):
2       // Codificar en Base64 para transmisi n
3       base64Signature = base64Encode(signatureBytes)
4       RETURN base64Signature
5   END FUNCTION
```

Listing 1.9: *Codificación final*

Proceso Detallado de Verificación

Paso 1: Decodificación y Verificación RSA

```
1   FUNCTION verifyRSA(signature, publicKey):
2       // Decodificar Base64
3       signatureBytes = base64Decode(signature)
4
```

```
    // Convertir a entero
    s = bytesToInteger(signatureBytes)

    // Operaci n RSA inversa: m = s^e mod n
    // donde e es el exponente p blico
    decrypted = modularExponentiation(s, publicKey.e, publicKey.n)

    // Convertir resultado a bytes
    decryptedBytes = integerToBytes(decrypted, publicKey.keySize)

    RETURN decryptedBytes
END FUNCTION
```

Listing 1.10: *Verificación de firma RSA*

Paso 2: Verificación de Padding y Extracción de Hash

```
FUNCTION extractHash(decryptedBytes):
    // Verificar formato PKCS#1
    IF decryptedBytes[0] != 0x00 OR decryptedBytes[1] != 0x01 THEN
        RETURN ERROR("Invalid PKCS#1 format")
    END IF

    // Buscar separador 0x00
    separatorIndex = findByte(decryptedBytes, 0x00, startIndex=2)

    // Extraer DigestInfo
    digestInfo = decryptedBytes[separatorIndex+1:]

    // Verificar identificador de algoritmo SHA-256
    expectedAlgId = "3031300d060960864801650304020105000420"
    IF NOT startsWith(digestInfo, expectedAlgId) THEN
        RETURN ERROR("Invalid algorithm identifier")
    END IF

    // Extraer hash ( ltimos  32 bytes)
    extractedHash = digestInfo[length(expectedAlgId):]

    RETURN extractedHash
END FUNCTION
```

Listing 1.11: *Extracción del hash original*

Paso 3: Comparación Final

```
FUNCTION verifySignature(message, signature, publicKey):
    // Calcular hash del mensaje recibido
    messageHash = SHA256(message)

    // Verificar firma y extraer hash
```

```
    extractedHash = extractHash(verifyRSA(signature, publicKey))

    // Comparaci n segura de hashes
    IF secureCompare(messageHash, extractedHash) THEN
        RETURN "SIGNATURE_VALID"
    ELSE
        RETURN "SIGNATURE_INVALID"
    END IF
END FUNCTION
```

Listing 1.12: *Verificación de integridad*

Consideraciones de Seguridad

Fortalezas del Esquema:

- **Resistencia criptográfica:** SHA-256 proporciona 128 bits de seguridad efectiva
- **Estándar probado:** PKCS#1 v1.5 es ampliamente utilizado y auditado
- **Interoperabilidad:** Compatible con múltiples plataformas y librerías

Vulnerabilidades Potenciales:

- **Ataques de padding:** PKCS#1 v1.5 es susceptible a ataques de padding oracle
- **Tamaño de clave:** RSA requiere claves de al menos 2048 bits para seguridad actual
- **Implementación:** Errores en la implementación pueden comprometer la seguridad

Mejores Prácticas:

- Usar claves RSA de 2048 bits o superiores
- Implementar verificación temporal de certificados
- Considerar migración a PSS (Probabilistic Signature Scheme) para mayor seguridad
- Implementar protecciones contra ataques de canal lateral

1.3. Problema 3 (35 %)

Desarrollo Seguro y DevSecOps:

1.3.1. Literal a

Brevemente describa el funcionamiento (fases) de SSL/TLS:

Funcionalidad de la Inyección SQL

La **inyección SQL** es una técnica de ataque que explota vulnerabilidades en la capa de datos de una aplicación, permitiendo a un atacante ejecutar comandos SQL arbitrarios en la base de datos. Esta vulnerabilidad ocurre cuando la aplicación construye consultas SQL dinámicamente concatenando entrada del usuario sin validación adecuada, haciendo que los datos proporcionados por el usuario se interpreten como código SQL en lugar de datos literales.
El **mecanismo de funcionamiento** se basa en la inserción directa de código malicioso en variables especificadas por el usuario que se concatenan con comandos SQL y se ejecutan. El proceso típico involucra:

1. **Finalización prematura de cadenas:** El atacante termina una cadena de texto existente
2. **Anexión de comandos maliciosos:** Se inserta código SQL no autorizado
3. **Uso de comentarios:** Se emplean marcas como - para omitir el resto de la consulta original

Ejemplo de vulnerabilidad:

```
-- C digo original vulnerable
consulta := "SELECT * FROM usuarios WHERE nombre = '" +
    nombreUsuario + "';"

-- Si el atacante ingresa: Alicia'; DROP TABLE usuarios; --
-- Resulta en:
SELECT * FROM usuarios WHERE nombre = 'Alicia';
DROP TABLE usuarios; --';
```

Listing 1.13: *Consulta vulnerable por concatenación*

Tipos principales de inyección SQL:

- **In-band (Clásica):** Los resultados se obtienen por el mismo canal de comunicación
  - *Error-based:* Extrae información mediante mensajes de error de la base de datos
  - *Union-based:* Combina resultados usando el operador UNION SQL
- **Inferential (Blind):** No se obtienen datos directamente, requiere inferencia
  - *Boolean-based:* Deduce información según respuestas verdadero/falso
  - *Time-based:* Usa delays temporales para inferir información
- **Out-of-band:** Exfiltra datos por un canal diferente (HTTP, DNS, etc.)

Contramedidas contra Inyección SQL

Contramedida 1: Consultas Parametrizadas (Prepared Statements)

Las consultas parametrizadas separan completamente el código SQL de los datos, haciendo imposible la inyección. Esta técnica utiliza marcadores de posición que asignan mecánicamente valores de entrada a sentencias SQL preparadas previamente.

```
-- Estructura segura con par metros
PREPARE stmt FROM 'SELECT * FROM users WHERE username = ? AND
    password = ?';
SET @username = 'valor_usuario';
SET @password = 'valor_password';
EXECUTE stmt USING @username, @password;
DEALLOCATE PREPARE stmt;
```

Listing 1.14: *Implementación segura con consultas parametrizadas*

Ventajas técnicas:

- Separación absoluta entre código y datos
- Prevención automática de inyección SQL
- Mejor rendimiento por caché de planes de ejecución
- Portabilidad entre diferentes motores de base de datos

Contramedida 2: Procedimientos Almacenados (Stored Procedures)
Los procedimientos almacenados encapsulan la lógica SQL en el servidor de base de datos, proporcionando una capa adicional de seguridad. Ejecutan varios comandos SQL en una sola sentencia sin permitir acceso directo al servidor a través de campos de entrada.

```
-- Crear procedimiento almacenado con validaci n
CREATE PROCEDURE sp_authenticate_user
    @Username NVARCHAR(50),
    @Password NVARCHAR(255)
AS
BEGIN
    -- Validaci n de entrada dentro del procedimiento
    IF LEN(@Username) > 50 OR LEN(@Password) > 255
    BEGIN
        RAISERROR('Invalid input length', 16, 1)
        RETURN
    END

    -- Consulta segura con par metros
    SELECT user_id, username, role
    FROM users
    WHERE username = @Username
    AND password_hash = HASHBYTES('SHA2_256', @Password + salt)
END
```

Listing 1.15: *Procedimiento almacenado seguro*

1.3.2. Literal b

Describa las principales diferencias entre un modelo de certificación basado en PKI con certificados digitales y el estándar PGP:

Control 1: Gestión Segura de Autenticación y Sesiones

Descripción: Implementar mecanismos robustos de autenticación que protejan las credenciales y gestionen las sesiones de forma segura.
Elementos clave:

- Hash de contraseñas con algoritmos seguros (BCrypt, Argon2)
- Protección contra ataques de fuerza bruta mediante rate limiting
- Regeneración de ID de sesión para prevenir session fixation
- Implementación de tokens CSRF únicos por sesión
- Timeout de sesiones y logout seguro
- Logging de eventos de autenticación para auditoría

```
FUNCTION authenticateUser(username, password, request):
    // 1. Validar entrada
    IF NOT isValidInput(username, password) THEN
        RETURN AuthResult.failure()
    END IF

    // 2. Verificar intentos fallidos (anti brute-force)
    IF isAccountLocked(username) THEN
        logSecurityEvent("LOCKED_ACCOUNT_ACCESS", username)
        THROW AccountLockedException
    END IF

    // 3. Verificar contrase a con hash seguro
    user = getUserByUsername(username)
    IF user == NULL OR NOT verifyPassword(password,
       user.passwordHash) THEN
        incrementFailedAttempts(username)
        RETURN AuthResult.failure()
    END IF

    // 4. Generar sesi n segura
    session = createSecureSession(user)
    regenerateSessionId(session)
    setCsrfToken(session)

    RETURN AuthResult.success(user, session)
END FUNCTION
```

Listing 1.16: *Pseudocódigo para autenticación segura*

Control 2: Validación y Sanitización de Entrada

Descripción: Toda entrada externa debe ser validada contra reglas de negocio específicas y sanitizada antes de su procesamiento.
Principios fundamentales:

- **Nunca confiar en datos de entrada:** Todo debe ser verificado para garantizar que corresponde a lo esperado
- **Validación por whitelist:** Definir explícitamente qué caracteres y formatos son permitidos
- **Sanitización contextual:** Aplicar escapado específico según el contexto de uso (HTML, JavaScript, SQL, etc.)
- **Verificación de longitud y formato:** Establecer límites máximos y patrones válidos
- **Detección de patrones peligrosos:** Identificar y bloquear secuencias maliciosas conocidas

```
FUNCTION validateInput(input, validationContext):
    // 1. Verificar nulos y longitud
    IF input == NULL OR length(input) >
       validationContext.maxLength THEN
        RETURN ValidationResult.error("Invalid input")
    END IF

    // 2. Aplicar reglas de formato
    IF NOT matches(input, validationContext.pattern) THEN
        RETURN ValidationResult.error("Invalid format")
    END IF

    // 3. Verificar patrones peligrosos
    dangerousPatterns = ["<script", "javascript:", "union
       select", "../"]
    FOR EACH pattern IN dangerousPatterns DO
        IF contains(input, pattern) THEN
            logSecurityEvent("DANGEROUS_PATTERN", input)
            RETURN ValidationResult.error("Forbidden pattern")
        END IF
    END FOR

    // 4. Sanitizar seg n contexto
    sanitized = sanitizeForContext(input,
       validationContext.outputContext)
    RETURN ValidationResult.success(sanitized)
END FUNCTION
```

Listing 1.17: *Framework de validación de entrada*

Control 3: Cifrado de Datos Sensibles

Descripción: Implementar cifrado tanto en tránsito como en reposo para proteger información confidencial.

Componentes esenciales:

- **Cifrado en tránsito:** Uso de protocolos seguros (TLS 1.3, HTTPS) para todas las comunicaciones
- **Cifrado en reposo:** Protección de datos almacenados usando algoritmos fuertes (AES-256)
- **Gestión segura de claves:** Implementación de HSM o sistemas de gestión de claves dedicados
- **Derivación de claves contextual:** Uso de técnicas como HKDF para generar claves específicas
- **Rotación regular de claves:** Establecer políticas de renovación periódica

```
FUNCTION encryptSensitiveData(plaintext, context):
    // 1. Derivar clave espec fica del contexto
    contextKey = deriveKey(masterKey, context)

    // 2. Generar IV aleatorio
    iv = generateRandomBytes(12) // Para AES-GCM

    // 3. Configurar cifrado AES-GCM
    cipher = initializeCipher("AES-GCM", contextKey, iv)

    // 4. Agregar datos adicionales autenticados
    aad = context + ":" + currentTimestamp()
    cipher.setAAD(aad)

    // 5. Cifrar datos
    ciphertext = cipher.encrypt(plaintext)

    // 6. Retornar objeto con metadatos
    RETURN EncryptedData(ciphertext, iv, algorithm, keyVersion,
       context, aad)
END FUNCTION
```

Listing 1.18: *Cifrado de datos sensibles*

Control 4: Manejo Seguro de Errores y Logging

Descripción: Los errores deben manejarse sin exponer información sensible, manteniendo registros detallados para análisis forense.

Características principales:

- **Mensajes de error genéricos:** Evitar revelar detalles técnicos o estructura interna
- **Logging estructurado:** Registro detallado de eventos de seguridad en formato JSON
- **Sanitización de logs:** Redactar información sensible antes del registro
- **Correlación de eventos:** Asignar IDs únicos para trazabilidad de incidentes
- **Alertas automáticas:** Notificación inmediata para eventos críticos de seguridad

```
FUNCTION handleException(exception, request):
    // 1. Generar ID  nico  para trazabilidad
    errorId = generateUUID()

    // 2. Clasificar el error
    classification = classifyError(exception)

    // 3. Log detallado (solo interno)
    logContext = {
        "errorId": errorId,
        "timestamp": currentTimestamp(),
        "exception": exception.type,
        "message": exception.message,
        "remoteAddr": request.remoteAddress,
        "userAgent": sanitize(request.userAgent),
        "requestUri": request.uri,
        "parameters": sanitizeParameters(request.parameters)
    }
    securityLogger.error(toJSON(logContext))

    // 4. Respuesta gen rica al cliente
    clientResponse = createSafeResponse(errorId, classification)

    // 5. Notificar si es cr tico
    IF classification.isCritical() THEN
        notifySecurityTeam(errorId, exception)
    END IF

    RETURN clientResponse
END FUNCTION
```

Listing 1.19: *Manejo seguro de errores*

1.3.3. Literal c

¿Por qué las funciones de hashing son el núcleo de la firmas digitales y del blockchains?

Shift Left: Seguridad Temprana en el Desarrollo

Definición: El enfoque Shift Left consiste en incorporar controles de seguridad en las primeras etapas del ciclo de vida del desarrollo de software (SDLC). Se basa en el principio de que es más eficiente y económico detectar y corregir vulnerabilidades durante las fases de diseño y desarrollo que en producción.
Características principales:

- **Integración temprana:** La seguridad se considera desde la planificación y diseño
- **Automatización de pruebas:** Implementación de análisis estático y dinámico en el pipeline CI/CD
- **Desarrollo dirigido por seguridad:** Los desarrolladores reciben herramientas y capacitación en prácticas seguras
- **Detección proactiva:** Identificación de vulnerabilidades antes del despliegue

Etapas de aplicación en el SDLC:

- **Planificación:** Definición de requisitos de seguridad y modelado de amenazas
- **Diseño:** Revisión de arquitectura de seguridad y selección de patrones seguros
- **Desarrollo:** Análisis estático de código (SAST) y revisiones de código
- **Pruebas:** Análisis dinámico (DAST) y pruebas de penetración automatizadas

Ejemplo de herramienta: SonarQube - Plataforma de análisis estático que se integra en el pipeline CI/CD para detectar vulnerabilidades, code smells y problemas de calidad durante la fase de desarrollo, proporcionando feedback inmediato a los desarrolladores.

```
# Pipeline CI/CD con an lisis est tico
stages:
  - build
  - security-scan
  - test
  - deploy

sonarqube-scan:
  stage: security-scan
  script:
    - sonar-scanner
        -Dsonar.projectKey=myproject
        -Dsonar.sources=src/
        -Dsonar.host.url=$SONAR_HOST_URL
        -Dsonar.login=$SONAR_TOKEN
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'

quality-gate:
  stage: security-scan
  script:
```

```
    - sonar-quality-gate-check
  dependencies:
    - sonarqube-scan
  allow_failure: false  # Bloquea el pipeline si hay
     vulnerabilidades cr ticas
```

Listing 1.20: *Integración de SonarQube en pipeline CI/CD*

Shift Right: Seguridad Continua en Producción

Definición: El enfoque Shift Right se centra en la supervisión continua del comportamiento, rendimiento e indicadores de seguridad durante la etapa de producción. Su objetivo es verificar el funcionamiento real del software y detectar amenazas en tiempo real.

Características principales:

- **Monitoreo continuo:** Supervisión 24/7 de aplicaciones en producción
- **Observabilidad:** Recolección y análisis de métricas, logs y trazas en tiempo real
- **Respuesta a incidentes:** Capacidad de detección y mitigación automática de amenazas
- **Feedback loop:** Información de producción que mejora el desarrollo futuro

Etapas de aplicación en el SDLC:

- **Despliegue:** Monitoreo de la implementación y configuración
- **Producción:** Supervisión continua de seguridad y rendimiento
- **Mantenimiento:** Análisis de patrones de uso y detección de anomalías
- **Evolución:** Retroalimentación para mejorar versiones futuras

Ejemplo de herramienta: Splunk SIEM - Sistema de gestión de eventos e información de seguridad que correlaciona logs y eventos de múltiples fuentes en tiempo real, detecta patrones anómalos y genera alertas automáticas para respuesta inmediata a incidentes de seguridad en producción.

```
# B squeda SPL para detectar intentos de inyecci n SQL
index=webapp sourcetype=access_log
| regex _raw="(?i)(union|select|insert|update|delete|drop|exec)"
| eval threat_level=case(
    match(_raw, "(?i)union.*select"), "HIGH",
    match(_raw, "(?i)drop.*table"), "CRITICAL",
    match(_raw, "(?i)(insert|update|delete)"), "MEDIUM",
    1=1, "LOW"
  )
| stats count by src_ip, threat_level, _time
| where count > 5
| sort -threat_level, -count
| eval alert_message="Potential SQL injection from " + src_ip
```

```
14 | outputlookup sql_injection_alerts.csv
```

Listing 1.21: *Consulta Splunk para detección de anomalías*

Complementariedad de Ambos Enfoques

Los enfoques Shift Left y Shift Right no son competitivos sino **complementarios**. Una estrategia DevSecOps efectiva requiere la implementación de ambos para lograr:

- **Cobertura completa del ciclo de vida:** Desde el diseño hasta la operación continua
- **Feedback bidireccional:** Los hallazgos en producción informan mejoras en desarrollo
- **Reducción de riesgos:** Detección temprana y monitoreo continuo minimizan la exposición
- **Cultura de seguridad:** Responsabilidad compartida entre todos los equipos

Cuadro 1.2: *Comparación Shift Left vs Shift Right*

Aspecto	**Shift Left**	**Shift Right**
Enfoque	Prevención	Detección y Respuesta
Timing	Pre-producción	Post-producción
Costo de remediación	Bajo	Alto
Tipo de pruebas	SAST, DAST, Code Review	Monitoreo, RASP, Chaos Engineering
Objetivo principal	Evitar vulnerabilidades	Detectar amenazas reales
Feedback	Inmediato en desarrollo	Continuo en operación

La **combinación óptima** permite crear un sistema de defensa multicapa donde la seguridad es una consideración constante, no un checkpoint aislado, estableciendo un ciclo virtuoso de mejora continua en la postura de seguridad organizacional.

Muchas gracias por la atención.
Over V.