Tecnología Digital IV: Redes de Computadoras

Clase 24: Seguridad en Redes de Computadoras - Parte 3

Lucio Santi & Emmanuel Iarussi

Licenciatura en Tecnología Digital
Universidad Torcuato Di Tella

17 de junio de 2025

Protocolos seguros en Internet
TLS

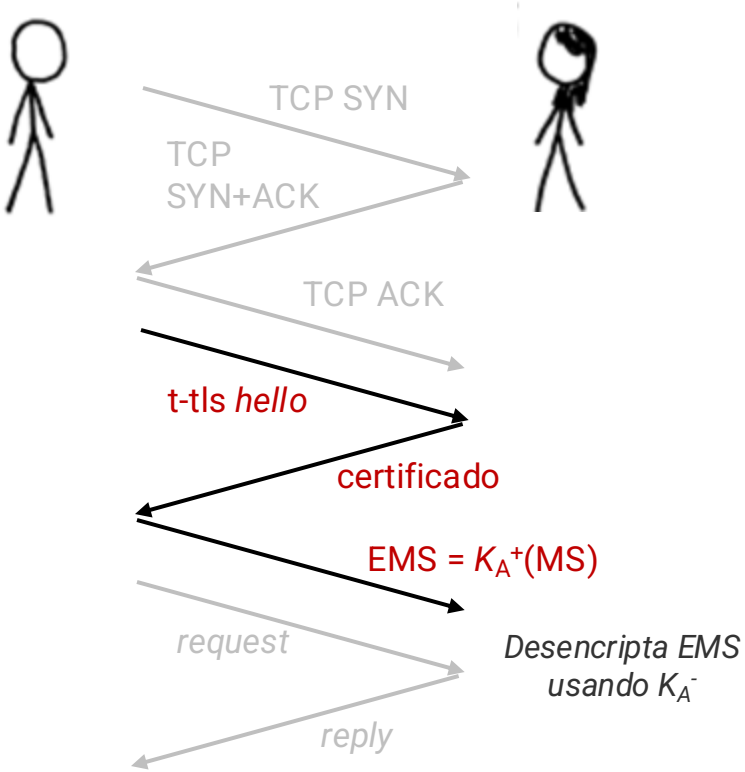
Transport Layer Security (TLS)

- Protocolo seguro que opera sobre la capa de transporte del stack TCP/IP
- Soportado por prácticamente todos los navegadores y servidores web: **HTTPS** (puerto 443)
- Ofrece:
  - Confidencialidad: vía cifrado simétrico
  - Integridad: vía hashes criptográficos
  - Autenticación: vía criptografía de clave pública
- Historia:
  - Orígenes en los ‘80 (investigaciones iniciales sobre APIs seguras similares a los sockets)
  - Netscape introdujo el protocolo SSL (*Secure Sockets Layer*) en la década del ‘90 (SSL 3.0 se deprecó en 2015)
  - SSL evolucionó a TLS (la versión actual 1.3 data de 2018; RFC 8846)

Motivación: *t-tls*

- Veamos qué necesitamos en el protocolo mediante una simplificación del mismo: **t-tls** (*Toy TLS*)
- Ya estudiamos las distintas componentes:
  - *Handshaking*: los interlocutores emplean sus certificados y claves privadas para autenticarse mutuamente; luego intercambian o generan un ***secreto*** compartido por ambos
  - Derivación de claves: los interlocutores generan una serie de claves a partir del secreto compartido
  - Transferencia de datos: el *stream* de datos se divide en registros (*records*) acompañados de *digests* y posteriormente encriptados
  - Cierre de conexión: mensajes de control especiales para cerrar de forma segura la conexión

Handshake de t-tls



Handshake de t-tls:

- Bob establece una conexión TCP con Alice
- Bob comprueba la identidad de Alice mediante su certificado
- Bob genera un secreto (*Master Secret*, MS) y se lo envía a Alice cifrado con su clave pública (de aquí se derivarán luego todas las claves de sesión)
- Se necesitan 3 RTTs antes de que los interlocutores puedan intercambiar datos

Claves de sesión

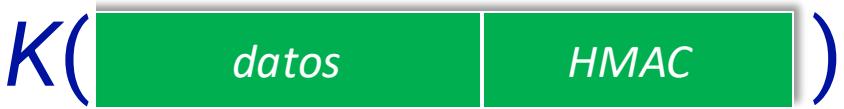
- Se generan claves distintas para cada propósito: dos para cifrado de datos y dos para los mecanismos de integridad
  - Es más seguro que tener una única clave

  - K_c : clave de cifrado de datos cliente → servidor
  - M_c : clave HMAC para datos cliente → servidor
  - K_s : clave de cifrado de datos servidor → cliente
  - M_s : clave HMAC para datos servidor → cliente

- **HMAC** (*Hash-based Message Authentication Code*, RFC 2104) es un *digest* calculado a partir de una función de hash y una clave
- Estas claves se generan mediante la función KDF (*Key Derivation Function*)
  - Utiliza el *master secret* y datos aleatorios

Cifrado de datos

- TCP ofrece un *stream* de bytes a los protocolos de aplicación
- Si quisiéramos encriptar datos del *stream* a medida que se inyectan en el socket,
  - ¿Dónde iría el HMAC? Si se calcula al final, no tendríamos control de integridad hasta no recibir todos los datos
  - Por esta razón se fragmenta el *stream* en una secuencia de registros (*records*)
  - Cada registro incluye un HMAC (calculado a partir de la clave de sesión *M*)
- Cada registro se cifra con la clave simétrica *K* y luego se entrega al socket TCP subyacente:

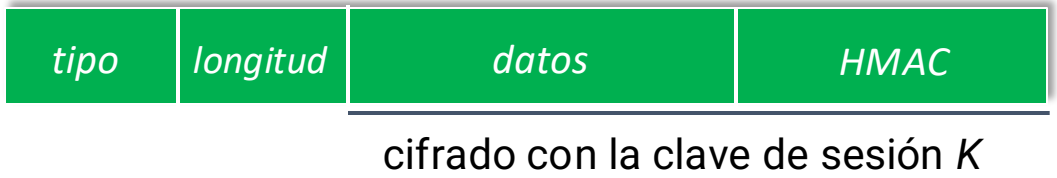


Cifrado de datos

- ¿Es posible ejecutar ataques sobre el *stream* de datos?
  - Reordenamiento: intercepción de segmentos TCP (vía m*an-in middle*) y reordenamiento de los mismos (cambiando los #SEQs en los headers TCP sin cifrar)
  - *Replay*: reenviar paquetes previamente observados
- No es posible: TLS emplea números de secuencia y *nonces*
  - Los números de secuencia son “implícitos”
  - Los interlocutores mantienen cuentas separadas de los mensajes transmitidos y recibidos; ese valor se incorpora dentro del cálculo del HMAC
  - Los *nonces* son valores aleatorios elegidos durante el *handshake* que impactan en la derivación de claves de sesión

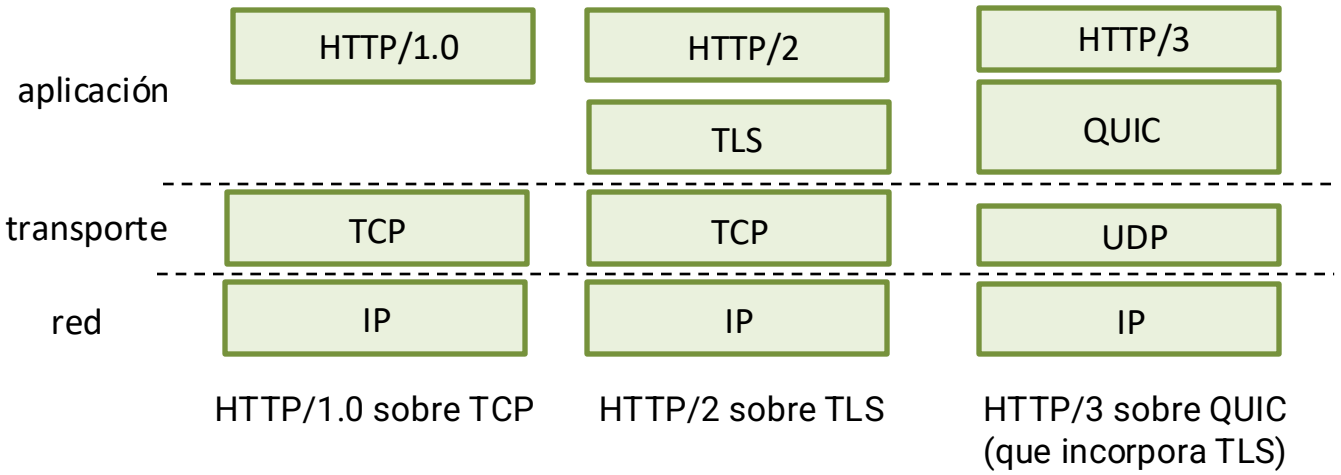
Cierre de conexión TLS

- ¿Qué ocurriría si un atacante falsificara un segmento TCP para cerrar la conexión subyacente a una sesión TLS legítima? (ataque de *truncamiento*)
- TLS contempla tipos de registro e incluye uno para notificar el cierre de la sesión
- Si bien el tipo va en texto plano, queda protegida su integridad por el HMAC del registro



TLS desde la óptica de HTTP

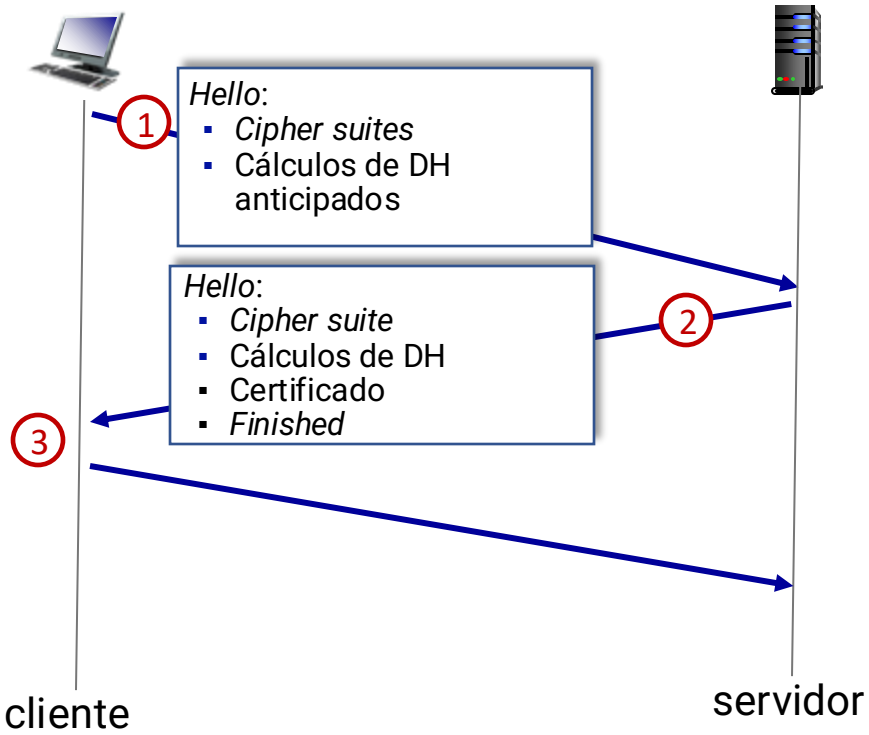
- TLS expone una API utilizable por cualquier protocolo de aplicación
- Desde la vista de HTTP,



TLS 1.3

- Surgió con el objetivo de mejorar la performance y la seguridad de las versiones anteriores
- Recorta considerablemente la suite de cifrado (*cipher suite*), dejando sólo 5 opciones respecto de las 37 de TLS 1.2
  - La *cipher suite* es una paleta de algoritmos criptográficos para generar claves, cifrar datos y calcular MACs
  - En el *handshake* se llega a un acuerdo de cuáles de estos utilizar
- Sólo permite el uso de **Diffie-Hellman** (DH) para generar claves, dejando de lado la opción de RSA
  - Es **muy difícil** implementar RSA correctamente (e.g. ataque de Bleichenbacher)
  - DH es un algoritmo criptográfico que permite que los interlocutores generen una clave compartida sin conocimiento previo mutuo
  - El uso de DH permite, además, comprimir el *handshake* a 1 RTT

Handshake de TLS 1.3



1. *Hello* del cliente
   - Indica qué algoritmos criptográficos soporta
   - "Adivina" la parametrización de DH del servidor y envía sus cálculos
2. *Hello* del servidor
   - *Cipher suite* seleccionada
   - Cálculos de DH
   - Certificado
   - *Finished*: hash encriptado de todo el *handshake*
3. El cliente:
   - Verifica el certificado
   - Genera claves
   - Envía su *Finished*
   - Puede enviar datos de aplicación

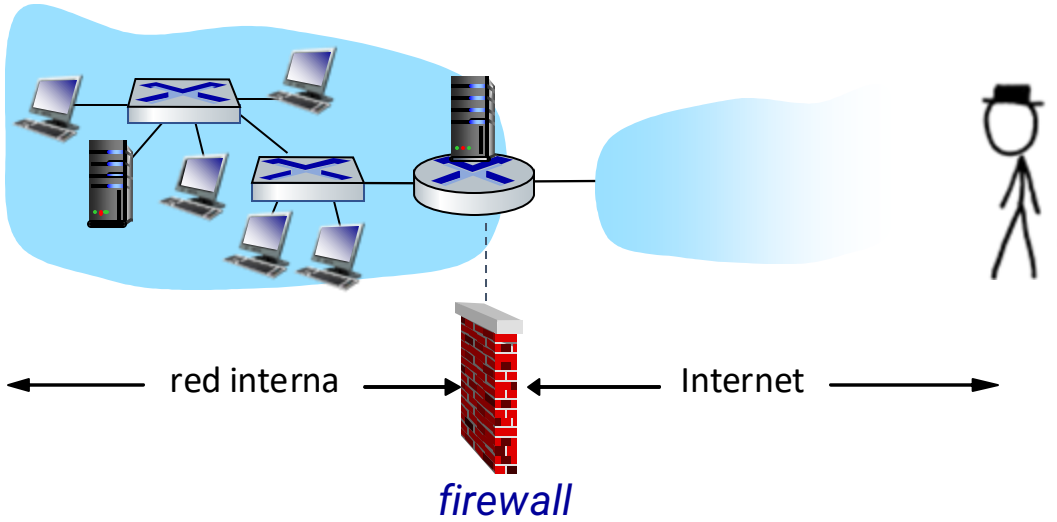
Handshake de TLS 1.3: resumir en 0-RTT

- Optimización adicional inspirada por QUIC
- Permite que los clientes envíen datos cifrados en su primer mensaje (elimina *delays* adicionales)
- Se da cuando el cliente se conecta a un servidor con el que ya intercambió datos
  - Ambos generan un secreto (*resumption main secret*) con el que posteriormente se pueden encriptar datos en futuras sesiones
- Vulnerable a ataques de **replay**
  - Puede ser útil en casos de requests sin efectos en el servidor (e.g. GETs de HTTP)

Firewalls

Firewalls

- Dispositivos que permiten aislar de Internet la red interna de una organización
- Implementan políticas para permitir ciertos paquetes y descartar otros



Funciones de los *firewalls*

Prevenir acceso/manipulación de datos privados

- e.g., un atacante reemplaza la página web de la UTDT con otra cosa (*defacement*)

Permitir acceso autorizado a la red interna

- Conjunto de usuarios/hosts autenticados

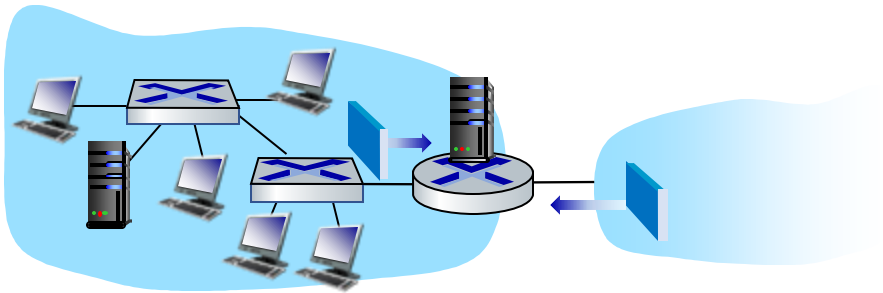
Prevenir ataques DoS (*denial of service*):

- SYN *flooding*: un atacante abre muchas conexiones TCP falsas, comprometiendo los recursos del sistema para establecer conexiones legítimas

Tres tipos de *firewalls*:

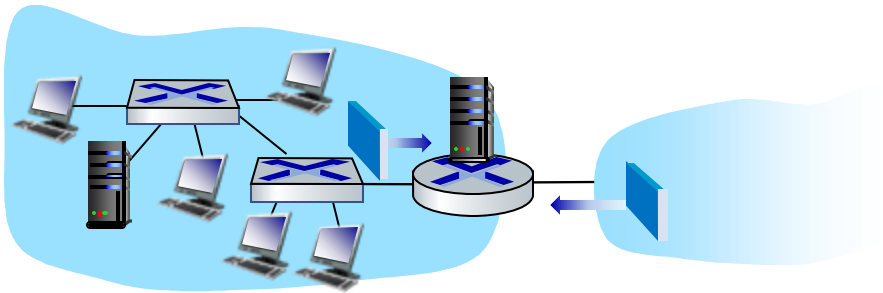
- Filtrado sin estado (***stateless packet filters***)
- Filtrado con estado (***stateful packet filters***)
- *Gateways* de aplicación (***application gateways***)

Filtrado *stateless*



- El *firewall* filtra paquete a paquete, tomando decisiones a partir de:
  - Direcciones IP origen y destino
  - Puertos origen y destino y protocolo de transporte (TCP/UDP)
  - Tipo de mensaje ICMP
  - *Flags* de TCP

Filtrado *stateless*: ejemplos



- Ejemplo 1: bloquear datagramas entrantes y salientes con puerto de transporte 23 (origen o destino)
  - No se permiten conexiones vía telnet (protocolo inseguro de login remoto)
- Ejemplo 2: bloquear segmentos TCP entrantes sin flag de *ACK*
  - No permite que se generen conexiones TCP desde el exterior, pero sí que se generen desde la red interna hacia afuera

Listas de control de acceso (ACLs)

Tabla de reglas que se aplican en orden a los paquetes (*match + action* “a la” OpenFlow)

acción	IP src	IP dst	proto	puerto src	puerto dst	*flags*
allow	222.22/16	fuera de 222.22/16	TCP	> 1023	80	*
allow	fuera de 222.22/16	222.22/16	TCP	80	> 1023	ACK
allow	222.22/16	fuera de 222.22/16	UDP	> 1023	53	---
allow	fuera de 222.22/16	222.22/16	UDP	53	> 1023	----
deny	*	*	*	*	*	*

Filtrado *stateful*

- Traza el estado de cada conexión TCP
  - Registra el establecimiento y el fin de las conexiones y puede determinar si los paquetes intercambiados son consistentes
  - En la ACL se indica si se debe verificar el estado de la conexión antes de admitir los paquetes

acción	IP src	IP dst	proto	puerto src	puerto dst	*flags*	conexión
allow	222.22/16	fuera de 222.22/16	TCP	> 1023	80	*	
allow	fuera de 222.22/16	222.22/16	TCP	80	> 1023	ACK	X

Ejemplo de *firewall stateful* en Linux: **iptables**

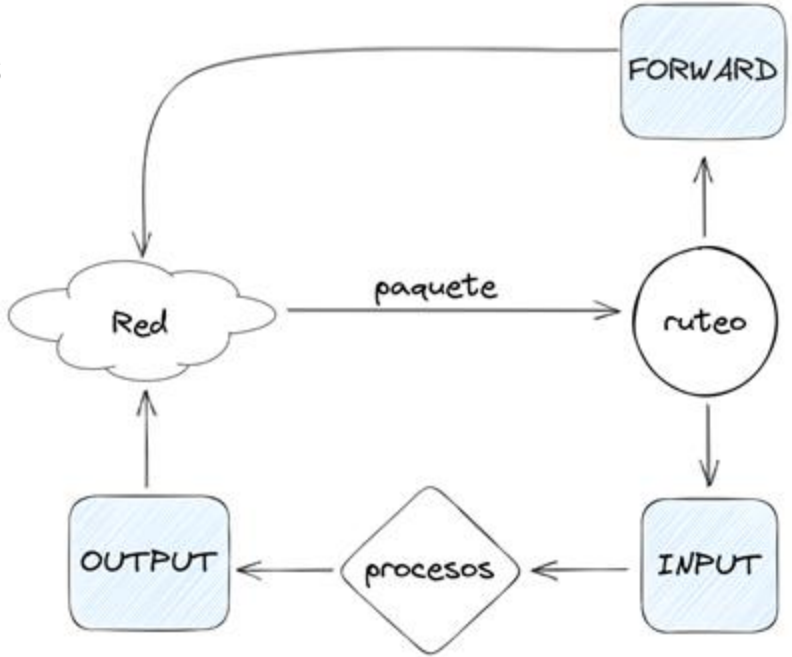
```
iptables -A INPUT -p tcp --dport 443 --m state --state NEW,ESTABLISHED -
j ACCEPT
```

Firewalling en Linux: iptables

- **iptables** es una herramienta de filtrado de paquetes de red en el kernel de Linux
  - Permite configurar y administrar reglas para controlar el tráfico de red entrante y saliente
- Utiliza reglas para definir cómo se debe manejar el tráfico: pueden especificar acciones como permitir (**ACCEPT**) o descartar (**DROP**) paquetes
  - Los paquetes se evalúan según distintos criterios (e.g. direcciones IP, puertos y protocolos)
- Las reglas se organizan en tablas y cadenas (**chains**)
  - Tabla: agrupamiento de reglas según su función
  - Cadena: secuencia ordenada de reglas asociada a una tabla
- Las tablas más comunes son *filter* (para filtrado de paquetes) y *nat* (para traducciones NAT)

iptables: flujo de paquetes

- Tres cadenas vinculadas a puntos de intercepción de tráfico (tabla *filter*):
  - **INPUT**: antes de entregar paquetes a los procesos del sistema operativo
  - **FORWARD**: antes de reenviar paquetes hacia otra interfaz (si corresponde)
  - **OUTPUT**: después de que los procesos generan paquetes
- Las reglas de las cadenas se evalúan en orden:
  - Si hay coincidencia, se ejecuta la acción
  - Si no la hay, se continúa con la siguiente
- Existe una política por defecto (*base policy*) para los paquetes que no coincidan con ninguna regla



Demo!

- Veamos cómo usar iptables para configurar un *firewall* sencillo:
  - Inspeccionar las *chains* de la tabla *filter*. ¿Existen reglas en ellas? ¿Cuáles son las *policies* por defecto de cada una?
  - Cambiar la *policy* de INPUT a DROP. ¿Qué consecuencias trae esta acción?
  - Agregar una regla para denegar el tráfico ICMP saliente al host 8.8.8.8. ¿Qué ocurre al ejecutar `ping` a dicho host?
  - Repetir el punto anterior pero denegando el tráfico ICMP entrante
  - Agregar una regla para descartar los segmentos TCP con flag SYN provenientes del servidor web de UTDT. ¿Qué consecuencias trae esta acción?