


Computación Centrada en Redes

# Seguridad de redes

## Amenazas, firewalls y VPNs

**Gerson Garrido**

Semana 15

UNIVERSIDAD NACIONAL DE INGENIERÍA

# Hoy

## Sesión A: amenazas y defensas

1. Modelo de amenazas
2. Ataques por capa: un repaso del curso
3. Firewalls sin estado y con estado
4. Zonas, DMZ y zero trust
5. IPsec: AH, ESP, túnel y transporte
6. WireGuard: identidad por clave
7. Qué protege una VPN y qué no

## Sesión B: proyectos

8. Presentación de P5 (`wasi` sobre `rimay`)
9. Falla de enlace en vivo
10. Repaso para el examen final

**Lectura**

Kurose y Ross, secs. 1.6, 8.1, 8.7 y 8.9. Artículo de WireGuard, secs. 1 y 2. Sesión B: releer el enunciado de P5. Complementarias: sec. 8.8 y RFC 4301.

# La idea de la semana

**No es una capa nueva**

Es releer todo el curso con otra pregunta: en cada capa, ¿qué puede hacer alguien que quiere hacer daño, y qué defensa existe?

# La idea de la semana

**No es una capa nueva**

Es releer todo el curso con otra pregunta: en cada capa, ¿qué puede hacer alguien que quiere hacer daño, y qué defensa existe?

- ARP no autentica (semana 3). IP deja falsificar el origen (semanas 10, 12). BGP cree a su vecino (semana 7). DNS responde sin firma (semana 12).

# La idea de la semana

**No es una capa nueva**

Es releer todo el curso con otra pregunta: en cada capa, ¿qué puede hacer alguien que quiere hacer daño, y qué defensa existe?

- ARP no autentica (semana 3). IP deja falsificar el origen (semanas 10, 12). BGP cree a su vecino (semana 7). DNS responde sin firma (semana 12).
- Casi ningún ataque es exótico: sale de una decisión de diseño que en su momento fue razonable (no autenticar, no cifrar, confiar).

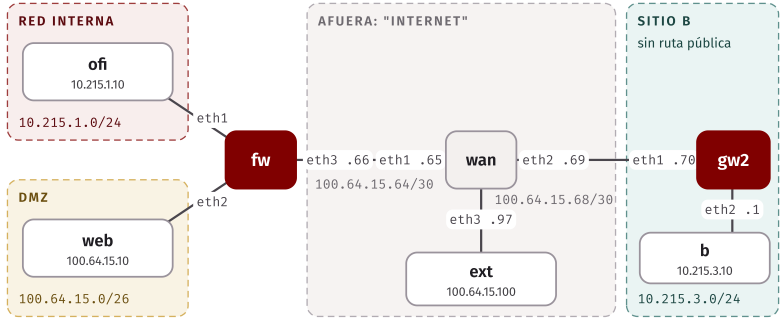
# La idea de la semana

**No es una capa nueva**

Es releer todo el curso con otra pregunta: en cada capa, ¿qué puede hacer alguien que quiere hacer daño, y qué defensa existe?

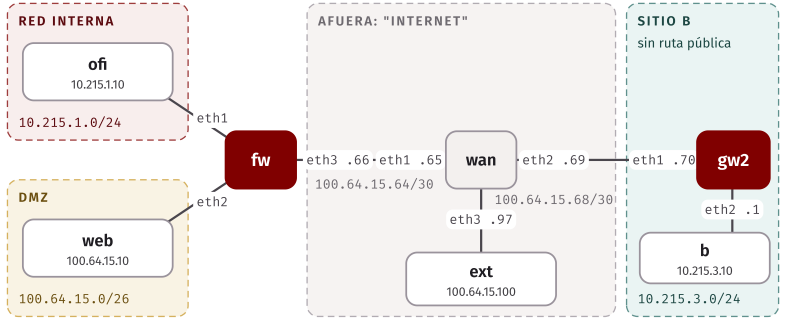
- ARP no autentica (semana 3). IP deja falsificar el origen (semanas 10, 12). BGP cree a su vecino (semana 7). DNS responde sin firma (semana 12).
- Casi ningún ataque es exótico: sale de una decisión de diseño que en su momento fue razonable (no autenticar, no cifrar, confiar).
- Y casi ninguna defensa cambia el protocolo: lo **envuelve** (TLS), lo **valida** desde afuera (RPKI, DNSSEC) o lo **filtra** (firewall, BCP 38).

# La red de esta semana




Sin NAT, a propósito: las capturas muestran las direcciones reales. wan sí sabe llegar a la red interna de fw, así que lo que la protege es el firewall y no la falta de una ruta.

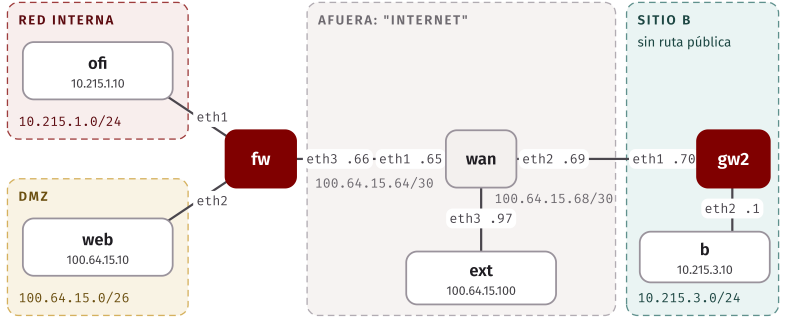
# La red de esta semana




Sin NAT, a propósito: las capturas muestran las direcciones reales. wan sí sabe llegar a la red interna de fw, así que lo que la protege es el firewall y no la falta de una ruta.

- `fw` separa tres zonas: red interna, DMZ (pública) y afuera.

# La red de esta semana




Sin NAT, a propósito: las capturas muestran las direcciones reales. wan sí sabe llegar a la red interna de fw, así que lo que la protege es el firewall y no la falta de una ruta.

- `fw` separa tres zonas: red interna, DMZ (pública) y afuera.
- La red de `b` no tiene ruta pública: solo se alcanza por el túnel.

SECCIÓN 01

# Modelo de amenazas

01



# Contra quién

**Modelo de amenazas**

Quién es el adversario, qué puede **observar** y qué puede **modificar**.

# Contra quién

**Modelo de amenazas**

Quién es el adversario, qué puede **observar** y qué puede **modificar**.

### Fuera del camino (off-path)

- Envía y falsifica, pero **no ve** el tráfico.
- Tiene que **adivinar** (ID de DNS, puerto): los trucos de la sem. 12 lo frenan.

# Contra quién

**Modelo de amenazas**

Quién es el adversario, qué puede **observar** y qué puede **modificar**.

### Fuera del camino (off-path)

- Envía y falsifica, pero **no ve** el tráfico.
- Tiene que **adivinar** (ID de DNS, puerto): los trucos de la sem. 12 lo frenan.

### En el camino (on-path)

- **Ve** el tráfico (Wi-Fi abierto, router comprometido).
- No adivina: solo lo frena la criptografía.

# Contra quién

**Modelo de amenazas**

Quién es el adversario, qué puede **observar** y qué puede **modificar**.

### Fuera del camino (off-path)

- Envía y falsifica, pero **no ve** el tráfico.
- Tiene que **adivinar** (ID de DNS, puerto): los trucos de la sem. 12 lo frenan.

### En el camino (on-path)

- **Ve** el tráfico (Wi-Fi abierto, router comprometido).
- No adivina: solo lo frena la criptografía.

**No existe “seguro” a secas**

Toda medida protege contra un adversario con ciertas capacidades. ¿Contra quién?

# Tres metas, tres familias de ataque

| Meta | Se rompe con | Ejemplo del curso |
|---|---|---|
| Confidencialidad | sniffing | leer una consulta DNS en claro |
| Integridad / autenticación | spoofing, MitM | ARP spoofing, hijack de BGP |
| Disponibilidad | DoS / DDoS | SYN flood, amplificación |

**Man-in-the-middle**

Activo y on-path a la vez: se pone en el medio y habla con cada lado haciéndose pasar por el otro.

SECCIÓN 02

# Ataques por capa

02



# Un repaso del curso, capa por capa

| Capa | Ataque | Defensa y su límite |
|---|---|---|
| Enlace | ARP spoofing (sem. 3) | VLANs, DHCP snooping; solo en redes administradas |
| Red | IP spoofing, amplificación (sem. 12) | BCP 38 / rp_filter; depende de *otras* redes |
| Transporte | SYN flood, DoS (sem. 10) | SYN cookies; no protegen el ancho de banda |
| Aplicación | envenenar DNS (sem. 12) | puerto al azar, DNSSEC, DoH; cada uno protege algo |
| Routing | hijack de BGP (sem. 7) | filtros de prefijos, RPKI; RPKI no valida el AS_PATH |

**Patrón**

Casi todo ataque sale de no autenticar o no cifrar; casi toda defensa envuelve, valida o filtra sin cambiar el protocolo.

# IP spoofing y BCP 38

- IP no verifica el origen: quien controla una máquina escribe el que quiera.

fw

```
$ sysctl net.ipv4.conf.all.rp_filter
net.ipv4.conf.all.rp_filter = 0     # con BCP 38 el operador lo pone en 1
```

# IP spoofing y BCP 38

- IP no verifica el origen: quien controla una máquina escribe el que quiera.
- De ahí la **amplificación**: pregunta chica con el origen de la víctima, respuesta grande hacia ella. En la sem. 12: 43 B de pregunta, 384 B de respuesta.

fw

```
$ sysctl net.ipv4.conf.all.rp_filter
net.ipv4.conf.all.rp_filter = 0     # con BCP 38 el operador lo pone en 1
```

# IP spoofing y BCP 38

- IP no verifica el origen: quien controla una máquina escribe el que quiera.
- De ahí la **amplificación**: pregunta chica con el origen de la víctima, respuesta grande hacia ella. En la sem. 12: 43 B de pregunta, 384 B de respuesta.
- **BCP 38** (RFC 2827): cada red descarta en su borde lo que no tenga un origen suyo. En Linux, el *reverse path filter*.

fw

```
$ sysctl net.ipv4.conf.all.rp_filter
net.ipv4.conf.all.rp_filter = 0     # con BCP 38 el operador lo pone en 1
```

# IP spoofing y BCP 38

- IP no verifica el origen: quien controla una máquina escribe el que quiera.
- De ahí la **amplificación**: pregunta chica con el origen de la víctima, respuesta grande hacia ella. En la sem. 12: 43 B de pregunta, 384 B de respuesta.
- **BCP 38** (RFC 2827): cada red descarta en su borde lo que no tenga un origen suyo. En Linux, el *reverse path filter*.

fw

```
$ sysctl net.ipv4.conf.all.rp_filter
net.ipv4.conf.all.rp_filter = 0     # con BCP 38 el operador lo pone en 1
```

Ningún servicio distingue, por sí solo, un origen falso de uno legítimo: se arregla **cerca del atacante**.

# ARP spoofing: man-in-the-middle en la capa 2

**Lo vimos en la semana 3**

ARP no autentica: cualquier nodo responde “esa IP soy yo”.

# ARP spoofing: man-in-the-middle en la capa 2

> **Lo vimos en la semana 3**
>
> ARP no autentica: cualquier nodo responde “esa IP soy yo”.

- Un atacante en la misma red responde a las consultas del gateway con su MAC, y al revés: queda en el medio de todo el tráfico local.

# ARP spoofing: man-in-the-middle en la capa 2

> **Lo vimos en la semana 3**
> ARP no autentica: cualquier nodo responde “esa IP soy yo”.

- Un atacante en la misma red responde a las consultas del gateway con su MAC, y al revés: queda en el medio de todo el tráfico local.
- Es un man-in-the-middle que **se hace** on-path, sin salir de la capa de enlace.

# ARP spoofing: man-in-the-middle en la capa 2

> **Lo vimos en la semana 3**
>
> ARP no autentica: cualquier nodo responde “esa IP soy yo”.

- Un atacante en la misma red responde a las consultas del gateway con su MAC, y al revés: queda en el medio de todo el tráfico local.
- Es un man-in-the-middle que **se hace** on-path, sin salir de la capa de enlace.
- Defensa estructural: separar dominios de broadcast con **VLANs**; en switches administrados, DHCP snooping y Dynamic ARP Inspection.

Es la razón por la que en la sem. 3 separamos oficinas de servidores con VLANs distintas.

# Disponibilidad: SYN flood, DoS y DDoS

- **SYN flood** (sem. 10): muchos SYN a medio completar llenan la cola del servidor. Las **SYN cookies** dejan de guardar estado y lo codifican en el número de secuencia.

# Disponibilidad: SYN flood, DoS y DDoS

- **SYN flood** (sem. 10): muchos SYN a medio completar llenan la cola del servidor. Las **SYN cookies** dejan de guardar estado y lo codifican en el número de secuencia.
- Pero las SYN cookies protegen la **memoria**, no el **ancho de banda**.

# Disponibilidad: SYN flood, DoS y DDoS

- **SYN flood** (sem. 10): muchos SYN a medio completar llenan la cola del servidor. Las **SYN cookies** dejan de guardar estado y lo codifican en el número de secuencia.
- Pero las SYN cookies protegen la **memoria**, no el **ancho de banda**.
- Un **DoS por volumen** —y un **DDoS** desde miles de máquinas— satura el enlace antes de llegar al servidor.

# Disponibilidad: SYN flood, DoS y DDoS

- **SYN flood** (sem. 10): muchos SYN a medio completar llenan la cola del servidor. Las **SYN cookies** dejan de guardar estado y lo codifican en el número de secuencia.
- Pero las SYN cookies protegen la **memoria**, no el **ancho de banda**.
- Un **DoS por volumen** —y un **DDoS** desde miles de máquinas— satura el enlace antes de llegar al servidor.
- Ahí la única defensa es **filtrar antes**: en el proveedor o en un servicio de mitigación. Por eso los firewalls viven en el borde.

# Aplicación y routing: lo que ya vimos

**DNS** (sem. 12)

- ID y puerto al azar: frenan al off-path.
- DNSSEC **autentica** con firmas.
- DoT / DoH **cifran** el canal.

**BGP** (sem. 7)

- Hijack: originar o propagar un prefijo ajeno.
- Filtros de prefijos (RFC 7454).
- RPKI valida quién origina cada prefijo.

**La seguridad no es una caja al final**

Cada ataque sale de una decisión de diseño; la defensa envuelve, valida o filtra. Ninguna defensa lo cubre todo: el examen pregunta los **límites**.

SECCIÓN 03

# Firewalls

03

# Sin estado y con estado

**Firewall:** un punto donde el tráfico se acepta o se descarta según una **política**.

### Sin estado

- Decide cada paquete por sus campos.
- Dejar volver respuestas obliga a aceptar todo lo que *parezca* respuesta… y eso abre la puerta.

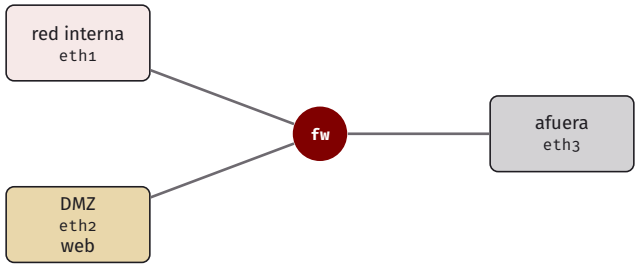
### Con estado

- Recuerda las conexiones (`conntrack`).
- Deja volver un paquete solo si es de una conexión que **vio salir**.
- `ct state`: `new`, `established`, `related`, `invalid`.

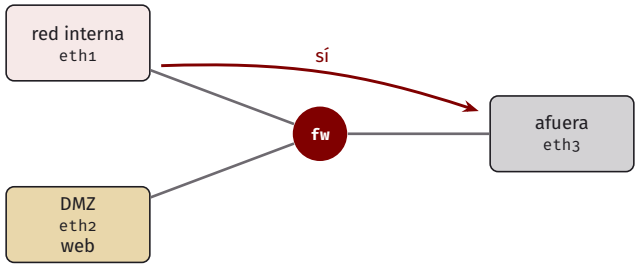
**La regla que acepta va primero**

`ct state established,related accept` antes de filtrar. Al revés (`!= ... drop`) se descartan también los `new` legítimos.

# Zonas y DMZ

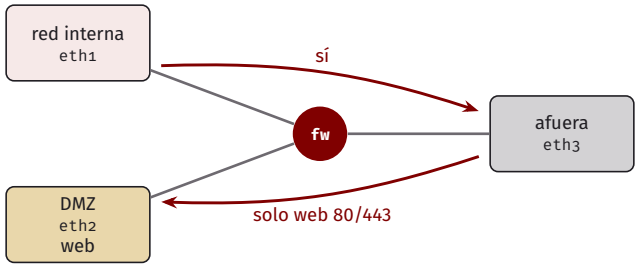


# Zonas y DMZ



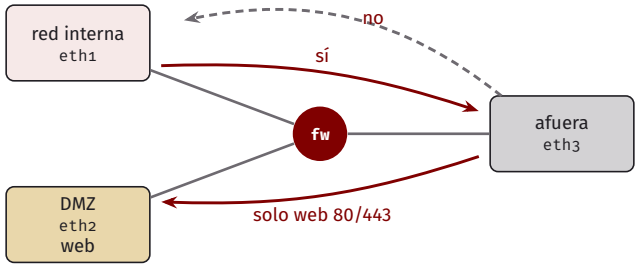

- Red interna y DMZ **inician** conexiones hacia afuera.

# Zonas y DMZ




- Red interna y DMZ **inician** conexiones hacia afuera.
- De afuera hacia adentro: **solo** el servidor web de la DMZ, en 80 y 443.

# Zonas y DMZ




- Red interna y DMZ **inician** conexiones hacia afuera.
- De afuera hacia adentro: **solo** el servidor web de la DMZ, en 80 y 443.
- Si comprometen el servidor público, queda en la DMZ, con el firewall todavía entre él y las oficinas.

# El esqueleto, y la regla que importa

forward: el esqueleto es siempre igual

```
chain forward {
    type filter hook forward priority filter; policy drop;
    ct state established,related accept   # lo establecido, primero
    ct state invalid drop
    ...                                   # una regla por camino de la tabla
    iifname "eth3" oifname "eth2" ip daddr 100.64.15.10 tcp dport { 80, 443 } accept
}
```

- La única puerta de afuera hacia adentro se ata a la dirección del servidor y a sus puertos: es la superficie expuesta.

# El esqueleto, y la regla que importa

forward: el esqueleto es siempre igual

```
chain forward {
    type filter hook forward priority filter; policy drop;
    ct state established,related accept   # lo establecido, primero
    ct state invalid drop
    ...                                   # una regla por camino de la tabla
    iifname "eth3" oifname "eth2" ip daddr 100.64.15.10 tcp dport { 80, 443 } accept
}
```

- La única puerta de afuera hacia adentro se ata a la dirección del servidor y a sus puertos: es la superficie expuesta.
- No olviden la cadena `input`: protege al **propio router** (su SSH, el UDP de WireGuard). `forward` pasa a través; `input` termina en la máquina.

El firewall completo es el laboratorio: ustedes lo escriben.

# Antes y después

```
python3 scenarios.py firewall
```

```
Antes del firewall (fw solo hace forwarding): afuera llega a todo.
    ext -> web (DMZ): 200
    ext -> ofi (red interna): 200
Después del firewall:
    ext -> web (DMZ)   : 200
    ext -> ofi (interna): falla en 0.00 s
    ofi -> web (interna a DMZ): 200
```

# Antes y después

```
python3 scenarios.py firewall

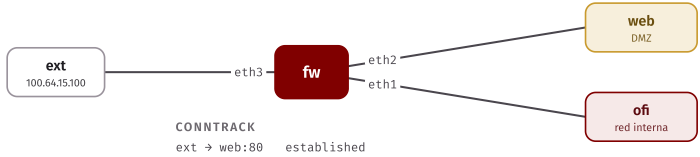
Antes del firewall (fw solo hace forwarding): afuera llega a todo.
    ext -> web (DMZ): 200
    ext -> ofi (red interna): 200
Después del firewall:
    ext -> web (DMZ)   : 200
    ext -> ofi (interna): falla en 0.00 s
    ofi -> web (interna a DMZ): 200
```

```
el tiempo cuenta la historia

    ext -> web:443 (aceptado por fw, sin servidor): no conecta, 0.04 s
    ext -> web:22  (sin regla, drop en forward) : no conecta, 3.04 s
    ext -> ofi:8080 (red interna, reject)       : no conecta, 1.06 s
```

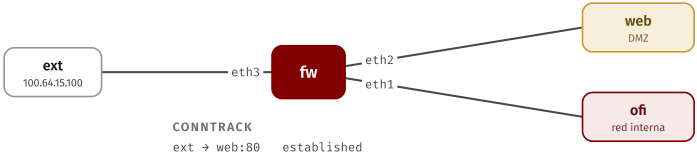
443: RST del servidor. 22: silencio. Hacia adentro: ICMP “admin prohibited”; `nc` reintenta una vez.

# Regla por regla



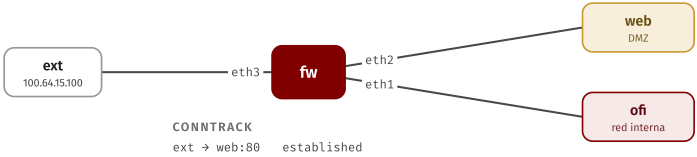

El SYN de un cliente no es established ni invalid, y calza con la única puerta de afuera hacia adentro. conntrack lo anota. El SYN-ACK y el resto de la conexión calzan con la primera regla: no hace falta una regla de vuelta.

# Regla por regla



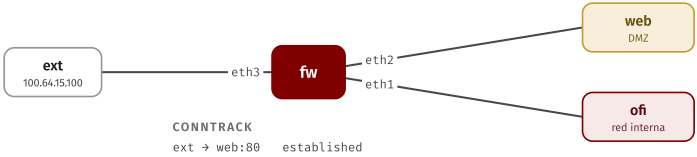

Puerto 443: fw lo acepta, pero no hay servidor. Es web quien contesta con un RST, y falla al instante.

# Regla por regla




Puerto 22: hay servidor, pero no hay regla. Cae en policy drop y ext no recibe nada: retransmite el SYN hasta que nc se rinde. El contador lo prueba: 3 paquetes, 180 bytes.

# Regla por regla




Hacia la red interna, fw rechaza con un ICMP desde su propia dirección. nc retransmite una vez y recibe otro ICMP. El tiempo mide también al cliente, no solo al firewall.

# Zero trust y detección

**Zero trust**

No confiar en alguien solo por **estar dentro** de la red. Cada acceso se autentica por su cuenta.

# Zero trust y detección

**Zero trust**

No confiar en alguien solo por **estar dentro** de la red. Cada acceso se autentica por su cuenta.

- El perímetro duro y el interior blando fallan con un solo equipo comprometido adentro.

# Zero trust y detección

**Zero trust**

No confiar en alguien solo por **estar dentro** de la red. Cada acceso se autentica por su cuenta.

- El perímetro duro y el interior blando fallan con un solo equipo comprometido adentro.
- Empuja la autenticación a cada servicio: TLS mutuo, identidad por dispositivo, micro-segmentación. WireGuard encaja: identifica por clave, no por dirección.

# Zero trust y detección

**Zero trust**

No confiar en alguien solo por **estar dentro** de la red. Cada acceso se autentica por su cuenta.

- El perímetro duro y el interior blando fallan con un solo equipo comprometido adentro.
- Empuja la autenticación a cada servicio: TLS mutuo, identidad por dispositivo, micro-segmentación. WireGuard encaja: identifica por clave, no por dirección.
- **IDS/IPS**: mira el tráfico buscando patrones y avisa (IDS) o bloquea (IPS). El costo son los falsos positivos y negativos. `nftables` ya trae `counter`, `limit` y `meter`; un IDS real agrega reensamblado y miles de firmas.

SECCIÓN 04

# IPsec

04

# IPsec: cifrado en la capa de red

**Una VPN es la idea de túnel de la sem. 4, con criptografía**

Un túnel cifrado hace que dos redes separadas se comporten como una sola, privada, sobre una red pública.

# IPsec: cifrado en la capa de red

**Una VPN es la idea de túnel de la sem. 4, con criptografía**

Un túnel cifrado hace que dos redes separadas se comporten como una sola, privada, sobre una red pública.

- **AH**: autentica, **no** cifra. **ESP**: cifra y autentica (lo usual).

# IPsec: cifrado en la capa de red

**Una VPN es la idea de túnel de la sem. 4, con criptografía**

Un túnel cifrado hace que dos redes separadas se comporten como una sola, privada, sobre una red pública.

- **AH**: autentica, **no** cifra. **ESP**: cifra y autentica (lo usual).
- **SA**: el estado de una dirección del túnel (algoritmo, clave, SPI). **Una SA por sentido**.

# IPsec: cifrado en la capa de red

**Una VPN es la idea de túnel de la sem. 4, con criptografía**

Un túnel cifrado hace que dos redes separadas se comporten como una sola, privada, sobre una red pública.

- **AH**: autentica, **no** cifra. **ESP**: cifra y autentica (lo usual).
- **SA**: el estado de una dirección del túnel (algoritmo, clave, SPI). **Una SA por sentido**.
- **Modo transporte**: protege la carga, entre dos hosts. **Modo túnel**: protege el datagrama entero y oculta las direcciones internas, entre dos sitios.

# IPsec: cifrado en la capa de red

**Una VPN es la idea de túnel de la sem. 4, con criptografía**

Un túnel cifrado hace que dos redes separadas se comporten como una sola, privada, sobre una red pública.

- **AH**: autentica, **no** cifra. **ESP**: cifra y autentica (lo usual).
- **SA**: el estado de una dirección del túnel (algoritmo, clave, SPI). **Una SA por sentido**.
- **Modo transporte**: protege la carga, entre dos hosts. **Modo túnel**: protege el datagrama entero y oculta las direcciones internas, entre dos sitios.
- Las claves las negocia **IKEv2**; aquí las ponemos a mano con `ip xfrm` para ver las piezas.

# ESP cifra, AH no

```
python3 scenarios.py ipsec

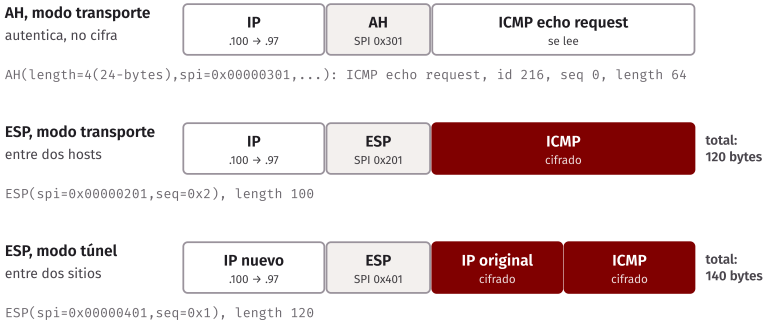
ESP transporte (cifra la carga):
  100.64.15.100 > 100.64.15.97: ESP(spi=0x00000201,seq=0x2), length 100
  100.64.15.97 > 100.64.15.100: ESP(spi=0x00000202,seq=0x2), length 100
AH transporte (autentica, NO cifra): el ICMP se lee
  100.64.15.100 > 100.64.15.97: AH(spi=0x00000301,...): ICMP echo request, id 216, seq 0
```

**AH autentica, ESP cifra**

Con ESP, en el medio solo hay bytes cifrados y un SPI por sentido. Con AH, el `ping` se lee: solo garantiza que nadie lo cambió.

En modo túnel el datagrama entero va cifrado dentro de uno nuevo: el paquete externo es más grande.

# Modo transporte y modo túnel




Direcciones: 100.64.15.100 y 100.64.15.97. En granate, lo cifrado: lo que quien mira en el medio no puede leer.

"VPN IPsec sitio a sitio" casi siempre quiere decir ESP en modo túnel: oculta las direcciones internas.

SECCIÓN 05

# WireGuard

05



# WireGuard: menos piezas

**La observación del artículo**

IPsec está *correctamente* diseñado y aun así es tan complejo que implementarlo sin errores es prohibitivo. WireGuard: menos de 4000 líneas en el kernel.

# WireGuard: menos piezas

**La observación del artículo**

IPsec está *correctamente* diseñado y aun así es tan complejo que implementarlo sin errores es prohibitivo. WireGuard: menos de 4000 líneas en el kernel.

- Una **interfaz virtual** `wg0`, no una capa de transformación. Se configura con `ip` y `wg`.

# WireGuard: menos piezas

**La observación del artículo**

IPsec está *correctamente* diseñado y aun así es tan complejo que implementarlo sin errores es prohibitivo. WireGuard: menos de 4000 líneas en el kernel.

- Una **interfaz virtual** `wg0`, no una capa de transformación. Se configura con `ip` y `wg`.
- **Identidad por clave pública** Curve25519 (32 B), al estilo OpenSSH. Sin certificados ni autoridades.

# WireGuard: menos piezas

**La observación del artículo**

IPsec está *correctamente* diseñado y aun así es tan complejo que implementarlo sin errores es prohibitivo. WireGuard: menos de 4000 líneas en el kernel.

- Una **interfaz virtual** `wg0`, no una capa de transformación. Se configura con `ip` y `wg`.
- **Identidad por clave pública** Curve25519 (32 B), al estilo OpenSSH. Sin certificados ni autoridades.
- **Cryptokey routing**: cada peer tiene `allowed-ips`. Al enviar, la clave sale del destino; al recibir, solo se acepta si el origen está en la lista.

# WireGuard: menos piezas

**La observación del artículo**

IPsec está *correctamente* diseñado y aun así es tan complejo que implementarlo sin errores es prohibitivo. WireGuard: menos de 4000 líneas en el kernel.

- Una **interfaz virtual** `wg0`, no una capa de transformación. Se configura con `ip` y `wg`.
- **Identidad por clave pública** Curve25519 (32 B), al estilo OpenSSH. Sin certificados ni autoridades.
- **Cryptokey routing**: cada peer tiene `allowed-ips`. Al enviar, la clave sale del destino; al recibir, solo se acepta si el origen está en la lista.
- **Sin agilidad de algoritmos**: un solo conjunto. Elimina los ataques de *downgrade*.

# WireGuard: menos piezas

**La observación del artículo**

IPsec está *correctamente* diseñado y aun así es tan complejo que implementarlo sin errores es prohibitivo. WireGuard: menos de 4000 líneas en el kernel.

- Una **interfaz virtual** `wg0`, no una capa de transformación. Se configura con `ip` y `wg`.
- **Identidad por clave pública** Curve25519 (32 B), al estilo OpenSSH. Sin certificados ni autoridades.
- **Cryptokey routing**: cada peer tiene `allowed-ips`. Al enviar, la clave sale del destino; al recibir, solo se acepta si el origen está en la lista.
- **Sin agilidad de algoritmos**: un solo conjunto. Elimina los ataques de *downgrade*.
- Sobre UDP, un puerto, 1-RTT (Noise). Desde afuera parece **sin estado**.

# Cryptokey routing

```
wg show wg0 allowed-ips

2PtwHBSu04WEnQ7/rNNZUPmpw3W18K012xjpKAVapRI=  10.215.1.0/24 10.215.9.1/32
```

# Cryptokey routing

```
wg show wg0 allowed-ips

2PtwHBSu04WEnQ7/rNNZUPmpw3W18K012xjpKAVapRI=  10.215.1.0/24 10.215.9.1/32
```

- **Al enviar:** el destino elige la clave (qué peer cubre esa dirección).

# Cryptokey routing

```
wg show wg0 allowed-ips

2PtwHBSu04WEnQ7/rNNZUPmpw3W18K012xjpKAVapRI=  10.215.1.0/24 10.215.9.1/32
```

- **Al enviar:** el destino elige la clave (qué peer cubre esa dirección).
- **Al recibir:** se descifra y **solo se acepta** si el origen está en la lista del peer que firmó.

# Cryptokey routing

```
wg show wg0 allowed-ips

2PtwHBSu04WEnQ7/rNNZUPmpw3W18K012xjpKAVapRI=  10.215.1.0/24 10.215.9.1/32
```

- **Al enviar:** el destino elige la clave (qué peer cubre esa dirección).
- **Al recibir:** se descifra y **solo se acepta** si el origen está en la lista del peer que firmó.
- La asociación clave $\leftrightarrow$ direcciones es toda la configuración de routing del túnel.

# Levantar el túnel

fw

```
$ wg genkey | tee fw.key | wg pubkey > fw.pub
fw#  wg set wg0 private-key fw.key listen-port 51820
fw#  wg set wg0 peer <gw2.pub> endpoint 100.64.15.70:51820 \
       allowed-ips 10.215.3.0/24,10.215.9.2/32 persistent-keepalive 25
fw#  ip addr add 10.215.9.1/30 dev wg0 ; ip link set wg0 up
fw#  ip route add 10.215.3.0/24 dev wg0
```

- En `gw2`, el espejo: peer `fw`, `allowed-ips 10.215.1.0/24`.

# Levantar el túnel

fw

```
$ wg genkey | tee fw.key | wg pubkey > fw.pub
fw#  wg set wg0 private-key fw.key listen-port 51820
fw#  wg set wg0 peer <gw2.pub> endpoint 100.64.15.70:51820 \
       allowed-ips 10.215.3.0/24,10.215.9.2/32 persistent-keepalive 25
fw#  ip addr add 10.215.9.1/30 dev wg0 ; ip link set wg0 up
fw#  ip route add 10.215.3.0/24 dev wg0
```

- En `gw2`, el espejo: peer `fw`, `allowed-ips 10.215.1.0/24`.
- `wg show` reporta el handshake, y la oficina alcanza `b`, que no tiene ruta pública.

# Qué protege: la captura a los dos lados

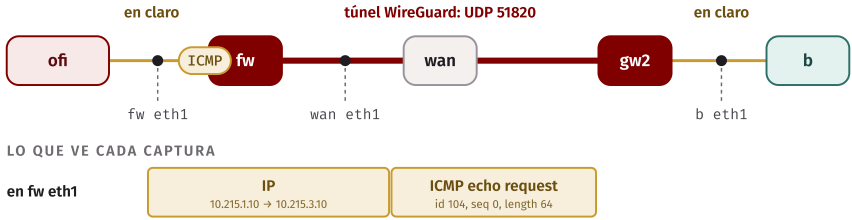
tres puntos de un ping ofi -> b

```
  en fw eth1 (interno): ICMP en claro
IP 10.215.1.10 > 10.215.3.10: ICMP echo request, id 104, seq 0, length 64
  en wan eth1 (externo): UDP 51820, cifrado
IP 100.64.15.66.51820 > 100.64.15.70.51820: UDP, length 128
IP 100.64.15.70.51820 > 100.64.15.66.51820: UDP, length 128
  en b eth1 (interno, destino): ICMP en claro otra vez
IP 10.215.1.10 > 10.215.3.10: ICMP echo request, id 104, seq 0, length 64
```

**El error común de las VPN**

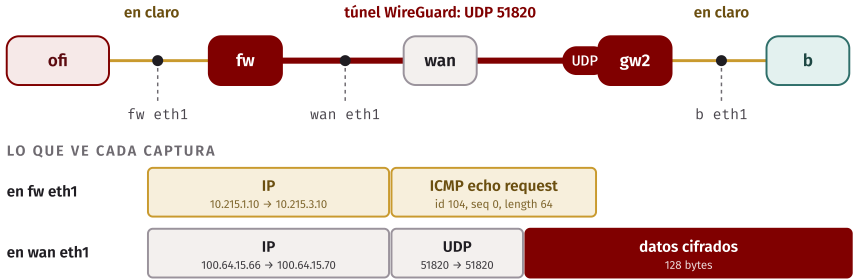
Protege **el tramo entre sus dos extremos**. En el medio solo se ve UDP cifrado entre las direcciones públicas. Fuera del túnel, todo va en claro: no hace anónimo ni seguro a todo.

# El mismo ping, tramo por tramo



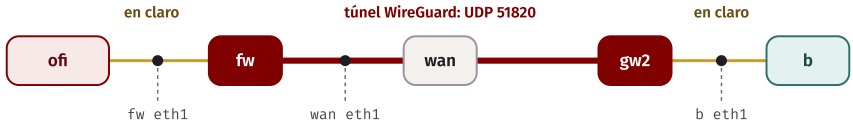

En fw eth1, el lado interno, el ICMP va en claro. fw tiene ruta hacia 10.215.3.0/24 por wg0.

# El mismo ping, tramo por tramo



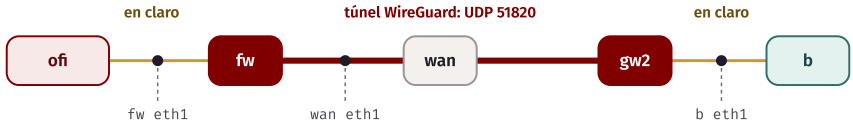

Al salir por wg0, WireGuard cifra el datagrama entero y lo mete en un UDP entre las direcciones públicas.
En wan solo se ve eso: 128 bytes de carga (84 rellenados a 96, más 32 de encabezado y etiqueta).

# El mismo ping, tramo por tramo




gw2 descifra, comprueba que el origen esté en las allowed-ips de fw y lo reinyecta. En b eth1 es otra vez el ICMP original.

# El mismo ping, tramo por tramo




Quien mira wan no ve 10.215.1.10, ni 10.215.3.10, ni que es un ping. La VPN protege el tramo entre sus dos extremos. Afuera de ese tramo, todo va en claro, como siempre.

# El costo: MTU y throughput

**MTU.** `wg0` = 1420 (1500 − 80).

```
python3 scenarios.py overhead

    -s 56: interno 84, afuera UDP, length 128
    -s 68: interno 96, afuera UDP, length 128
    -s 69: interno 97, afuera UDP, length 144
```

Relleno a múltiplo de 16, +16 encabezado +16 etiqueta; por fuera, IP+UDP. PMTUD propaga la MTU:

```
ofi

$ ip route get 10.215.3.10
10.215.3.10 via 10.215.1.1 dev eth1 src 10.215.1.10 uid 0
    cache expires 596sec mtu 1420
```

**Throughput**, mismo camino:

```
python3 scenarios.py cost

    sin cifrar    7479 Mbits/sec
    por el túnel  1407 Mbits/sec
    sin cifrar    6823 Mbits/sec
    por el túnel  1158 Mbits/sec
```

Los encabezados cuestan cerca de 5 %; el resto lo atribuimos a la CPU. Depende de la máquina.

**PMTUD y el firewall:** si el firewall bloquea los ICMP “fragmentation needed”, PMTUD se rompe y las conexiones grandes se cuelgan.

# Cryptokey routing, en los dos sentidos




**ALLOWED-IPS EN FW**

peer: gw2

- `10.215.3.0/24`
- `10.215.9.2/32`

**EL PAQUETE INTERNO**

`fw → 10.215.3.10`

**ALLOWED-IPS EN GW2**

peer: fw

- `10.215.1.0/24`
- `10.215.9.1/32`

| LO QUE SE CONTÓ | UDP del túnel en gw2 | ICMP en b |
|---|---|---|
| `origen 100.64.15.66` | · | · |
| `origen 10.215.9.1` | · | · |

Al enviar, fw mira el destino: 10.215.3.10 está en las allowed-ips del peer gw2, y con eso elige su clave.
La lista hace de tabla de routing del túnel.

# Cryptokey routing, en los dos sentidos




**ALLOWED-IPS EN FW**

peer: gw2

`10.215.3.0/24`

`10.215.9.2/32`

**EL PAQUETE INTERNO**

`100.64.15.66 → b`

**no está en la lista: se descarta**

**ALLOWED-IPS EN GW2**

peer: fw

`10.215.1.0/24`

`10.215.9.1/32`

| LO QUE SE CONTÓ | UDP del túnel en gw2 | ICMP en b | |
|---|---|---|---|
| `origen 100.64.15.66` | 2 | 0 | perdió los 2 pings |
| `origen 10.215.9.1` | · | · | |

Ping desde fw con origen 100.64.15.66, su eth3. gw2 lo recibe cifrado, lo descifra y ve que ese origen no le corresponde a fw. Lo descarta: los 2 pedidos llegan como UDP y a b no llega nada.

# Cryptokey routing, en los dos sentidos




**ALLOWED-IPS EN FW**

peer: gw2

`10.215.3.0/24`

`10.215.9.2/32`

**EL PAQUETE INTERNO**

`10.215.9.1 → b`

**ALLOWED-IPS EN GW2**

peer: fw

`10.215.1.0/24`

`10.215.9.1/32`

| LO QUE SE CONTÓ | UDP del túnel en gw2 | ICMP en b | |
|---|---|---|---|
| `origen 100.64.15.66` | 2 | 0 | perdió los 2 pings |
| `origen 10.215.9.1` | 4 | 4 | 2 pedidos y 2 respuestas |

Con origen 10.215.9.1, que sí está en la lista, cruza: 2 pedidos y 2 respuestas, 4 paquetes en cada punto.
La clave da identidad, y la identidad solo autoriza ciertas direcciones.

# Identidad por clave como defensa

**allowed-ips es también un filtro**

`gw2` solo acepta de `fw` paquetes cuyo origen esté en `10.215.1.0/24`. Un origen fuera de la lista se descarta **después** de descifrar.

# Identidad por clave como defensa

**allowed-ips es también un filtro**

`gw2` solo acepta de `fw` paquetes cuyo origen esté en `10.215.1.0/24`. Un origen fuera de la lista se descarta **después** de descifrar.

- La clave da identidad; la identidad solo autoriza ciertas direcciones.

# Identidad por clave como defensa

**allowed-ips es también un filtro**

`gw2` solo acepta de `fw` paquetes cuyo origen esté en `10.215.1.0/24`. Un origen fuera de la lista se descarta **después** de descifrar.

- La clave da identidad; la identidad solo autoriza ciertas direcciones.
- “Cualquier paquete que llegue por `wg0` tiene un origen auténtico” (sec. 2 del artículo).

# Identidad por clave como defensa

**allowed-ips es también un filtro**

`gw2` solo acepta de `fw` paquetes cuyo origen esté en `10.215.1.0/24`. Un origen fuera de la lista se descarta **después** de descifrar.

- La clave da identidad; la identidad solo autoriza ciertas direcciones.
- “Cualquier paquete que llegue por `wg0` tiene un origen auténtico” (sec. 2 del artículo).
- Sobre eso el firewall del otro sitio **puede** confiar en la dirección de origen de lo que sale de `wg0`, algo que nunca puede hacer con el tráfico de Internet.

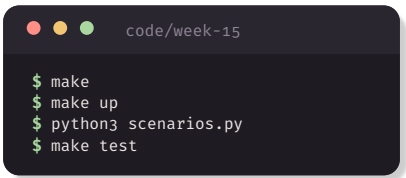
SECCIÓN 06

# Laboratorio

06

# Laboratorio: firewall y WireGuard

1. Completen `lab/firewall.nft` (quiten los `TODO`).
2. Prueben cada regla con `nc` y `curl`: anoten esperado vs ocurrido.
3. Levanten WireGuard entre `fw` y `gw2`; confirmen con `wg show`.
4. Capturen en `fw`, `wan` y `b`: comparen interno con externo.
5. Midan con `iperf3` el costo; revisen la MTU con `ip route get`.
6. Entreguen **P5**.




Las pruebas del firewall se saltan hasta que quiten el `TODO`. La solución no se publica.

# Sesión B: presentación de P5

**Cada equipo muestra `wasi` sobre `rimay` en Qhapaq Ñan**

Desde un host de otro AS, cruzando BGP, con su nombre en DNS.

# Sesión B: presentación de P5

**Cada equipo muestra `wasi` sobre `rimay` en Qhapaq Ñan**

Desde un host de otro AS, cruzando BGP, con su nombre en DNS.

- El docente **corta un enlace** con otro camino disponible mientras se baja un archivo grande: la transferencia debe terminar bien.

# Sesión B: presentación de P5

**Cada equipo muestra `wasi` sobre `rimay` en Qhapaq Ñan**

Desde un host de otro AS, cruzando BGP, con su nombre en DNS.

- El docente **corta un enlace** con otro camino disponible mientras se baja un archivo grande: la transferencia debe terminar bien.
- Ahí se ve si `rimay` recupera la pérdida y si la red encuentra otro camino.

# Sesión B: presentación de P5

**Cada equipo muestra `wasi` sobre `rimay` en Qhapaq Ñan**

Desde un host de otro AS, cruzando BGP, con su nombre en DNS.

- El docente **corta un enlace** con otro camino disponible mientras se baja un archivo grande: la transferencia debe terminar bien.
- Ahí se ve si `rimay` recupera la pérdida y si la red encuentra otro camino.
- Preguntas sobre las decisiones de diseño y las mediciones (`rimay` vs TCP), no sobre el código línea por línea.

Enunciado y reglas: `projects/p5-wasi/README.md`.

# Repaso para el examen final

- **Seguir un paquete con túneles:** el `ping` de la oficina a `b` —ICMP en claro, UDP 51820 en el medio, ICMP otra vez— y qué cambia con un NAT.
- Transporte y congestión (9–11): GBN, SR, `cwnd` de Reno, RTO; leer una captura.
- Aplicaciones (12–13): DNS de la raíz; qué autentica DNSSEC y qué cifra DoH; TLS 1.3; por qué QUIC usa UDP.
- Redes modernas (14–15): Clos y ECMP; qué separa SDN; un ataque de cada capa y su defensa.

**Qué NO protege cada defensa**

RPKI no valida el AS_PATH; DNSSEC no da privacidad; TLS no oculta el SNI; una VPN no protege más allá de su extremo; las SYN cookies no protegen el ancho de banda.

# Resumen

- Toda defensa es contra un adversario con ciertas capacidades: **en el camino** o **fuera**. “Seguro” sin decir contra quién no significa nada.
- Ataques por capa: ARP spoofing, IP spoofing / amplificación, SYN flood, envenenar DNS, hijack de BGP. Casi todos por no autenticar o no cifrar.
- Firewall con estado: denegar por defecto, aceptar lo establecido primero, política por zonas. Proteger `input` además de `forward`.
- IPsec: AH autentica, ESP cifra; transporte entre hosts, túnel entre sitios; IKE negocia.
- WireGuard: `wg0`, identidad por clave, cryptokey routing, 1-RTT. Simple de auditar.
- Una VPN protege **el tramo entre sus extremos**, no más. Cuesta MTU y throughput.

# Para la próxima clase

- **Sesión B**: presentación de P5. `wasi` sobre `rimay` desde otro AS, con DNS; el docente corta un enlace durante una descarga grande y la descarga debe terminar.
- **Examen final** (semana 16): todo el curso, con énfasis en las semanas 9 a 15. Formato en `exams/README.md`.
- Repasen qué **no** protege cada defensa: RPKI no valida el AS_PATH; DNSSEC no da privacidad; TLS no oculta el SNI; una VPN no protege más allá de su extremo.

**Pregunta**

Un `ping` de la oficina a `b` cruza un NAT y luego el túnel WireGuard. Dibujen, salto por salto, qué encabezados tiene el paquete y qué direcciones quedan visibles en cada tramo.