

Securizarea rețelei

Până acum am construit o rețea care funcționează. Cursul acesta pornește de la observația neplăcută că Internetul nu este un loc prietenos, și adaugă trei lucruri care lipsesc. Întâi nivelul transport - pentru că aproape toate atacurile și aproape toate apărările se poartă acolo, la nivelul porturilor și al flag-urilor TCP. Apoi filtrarea cu iptables, adică ce se întâmplă când firewall-ul nu mai e un ruter Cisco, ci o mașină Linux. Și, la final, criptografia din spatele unei simple sesiuni SSH: cum reușesc două calculatoare care nu s-au întâlnit niciodată să se pună de acord asupra unei chei secrete, în văzul tuturor.

Din capitolul precedent.

- Un ACL este o listă **ordonată**, citită până la **prima potrivire**.
- La final există un `deny any` **invizibil**.
- Lista trebuie **aplicată** pe o interfață și o direcție ca să existe.
- Listele extinse pot filtra după **protocol și port** - adică după nivelul 4.
- Un ACL este **fără stare**: nu ține minte că am întrebat noi primii.

OBIECTIVELE CAPITOLULUI

- Să explicați ce adaugă nivelul transport peste IP și de ce există porturi
- Să alegeți între TCP și UDP pentru o aplicație dată, cu argumente
- Să citiți o strângere de mână în trei pași și să spuneți ce face fiecare flag
- Să recunoașteți cele trei familii de atacuri și să explicați cum funcționează un SYN flood
- Să scrieți reguli iptables corecte, pe lanțul potrivit
- Să construiți un firewall complet, cu politică implicită DROP
- Să explicați ce garantează, separat, autentificarea, confidențialitatea și integritatea
- Să descrieți schimbul Diffie-Hellman și să spuneți ce nu rezolvă el

Ultimul punct este exact locul unde ne oprimem, nemulțumiți. Astăzi vedem ce înseamnă un echipament care **ține minte** sesiunile - dar, ca să înțelegem ce anume ține minte, trebuie mai întâi să înțelegem ce este o sesiune.

8.1 Nivelul transport: de ce există porturi

Problema pe care o rezolvă nivelul 4. Până acum, un pachet ajunge la **calculatorul** potrivit. Adresa IP face exact atât: identifică mașina.

Dar pe mașina aceea rulează, în același timp, un browser, un client de mail, un joc și trei actualizări în fundal. Când sosește un pachet, sistemul de operare trebuie să știe **cărei aplicații** îi aparține. Adresa IP nu spune nimic despre asta.

De aceea nivelul 4 adaugă încă un număr: **portul**. Fiecare nivel introduce un nou tip de adresă - nivelul 2 adresa MAC, nivelul 3 adresa IP, nivelul 4 portul. Fără porturi, un calculator ar putea rula o singură aplicație de rețea la un moment dat.

DEFINIȚIA 8.1 (SOCKET). Perechea **adresă IP + număr de port**, scrisă 192.168.1.10:49152. O conexiune este identificată complet de **patru** numere: socketul sursă și socketul destinație. Două conexiuni pot avea aceeași destinație și aceeași adresă sursă, dar porturi sursă diferite - și de aceea puteți deschide zece file spre același site fără să se amestece.

Tabelul 8.1 Nivelul transport: de ce există porturi.

Rol al nivelului transport	Ce înseamnă	TCP	UDP
Segmentare	împarte fluxul aplicației în bucăți care încap într-un pachet	da	da
Adresare	identifică aplicația prin numărul de port	da	da
Multiplexare	mai multe aplicații pe aceeași adresă IP	da	da
Siguranța transmisiei	retransmite ce s-a pierdut, reordonează ce a sosit anapoda	da	nu
Controlul fluxului	împiedică un emițător rapid să înecă un receptor lent	da	nu
Inițierea conexiunilor	stabilește o sesiune înainte de transferul propriu-zis	da	nu

UDP nu este „TCP stricat”. Tabelul de mai sus enumeră ce *poate* face nivelul transport. UDP face doar primele trei - și o face deliberat. La un apel video, un pachet pierdut retransmis peste 200 de milisecunde este inutil: momentul lui a trecut. Mai bine lipsește. Uneori viteza contează mai mult decât garanțiile.

TCP - Transmission Control Protocol.

- Orientat pe conexiune: se stabilește o sesiune înainte de transfer
- Sigur: datele ajung garantat și în ordine
- Control de flux, de congestie și de eroare
- Antet de **minimum 20 de octeți**
- SSH, HTTP, SMTP, transfer de fișiere, baze de date

UDP - User Datagram Protocol

- Fără conexiune: se trimite și gata
- Nesigur: segmentele se pot pierde sau sosi în altă ordine
- Fără control de flux sau congestie
- Antet de **8 octeți** - sursă, destinație, lungime, sumă de control
- DNS, DHCP, VoIP, IPTV, jocuri, streaming

Intervalele de porturi

Tabelul 8.2 Intervalele de porturi.

Interval	Denumire	Cine le folosește
0 – 1023	well-known	servicii standard: 22 SSH, 25 SMTP, 53 DNS, 80 HTTP, 443 HTTPS. Pe Linux, doar procesele privilegiate pot asculta aici.
1024 – 49151	înregistrate	aplicații cunoscute, dar neoficiale: 3306 MySQL, 3389 RDP, 8080 proxy HTTP
49152 – 65535	dinamice / efemere	porturile sursă alese automat de sistemul de operare la fiecare conexiune nouă

Consecința pentru filtrare. Când stația dumneavoastră deschide o pagină web, conexiunea arată așa: 192.168.1.10:51423 → 93.184.216.34:80. Portul **destinație** este cunoscut (80), dar portul **sursă** este imprevizibil.

De aceea o regulă de firewall pentru traficul de întoarcere nu poate specifica portul destinație - el se schimbă la fiecare conexiune. Trebuie fie *established*, fie urmărirea sesiunilor. Aceasta e, în concret, limitarea de care ne-am lovit la finalul cursului trecut.



Figura 8.1 Terminal: porturi și socketuri.

Priviți a doua comandă. Cele trei conexiuni au **aceeași adresă locală**, dar porturi sursă diferite - și tocmai asta le ține separate. Iar ultima linie e o conexiune în celălalt sens: cineva e conectat prin SSH la mașina noastră, deci portul local este 22, cel binecunoscut.

8.2 Antetul TCP, câmp cu câmp

Aproape toate mecanismele discutate mai jos - retransmisia, controlul de flux, strângerea de mână, atacul SYN flood, opțiunea *established* - sunt scrise, la propriu, în acest antet de 20 de octeți.



Figura 8.2 Antetul TCP - apăsați pe câmpuri.

Tabelul 8.3 Antetul TCP, câmp cu câmp.

Flag	Rol	Unde îl întâlniți
SYN	cere sincronizarea numerelor de secvență - deschide conexiunea	SYN flood, <code>--syn</code> în iptables
ACK	confirmă datele primite; activează câmpul de confirmare	<i>established</i> în ACL-uri
FIN	anunță că emițătorul a terminat de transmis	închiderea normală
RST	închide brutal conexiunea; răspunsul la un pachet nevalid	REJECT în iptables

8.3 Deschiderea și închiderea unei conexiuni

O conexiune TCP se deschide printr-un schimb de **trei** segmente, în care fiecare capăt își comunică numărul de secvență inițial și îl confirmă pe al celuilalt. Se închide, în schimb, prin **patru** - și motivul acestei asimetrii spune ceva despre cum este gândit TCP.



Figura 8.3 Strângerea de mână în trei pași.

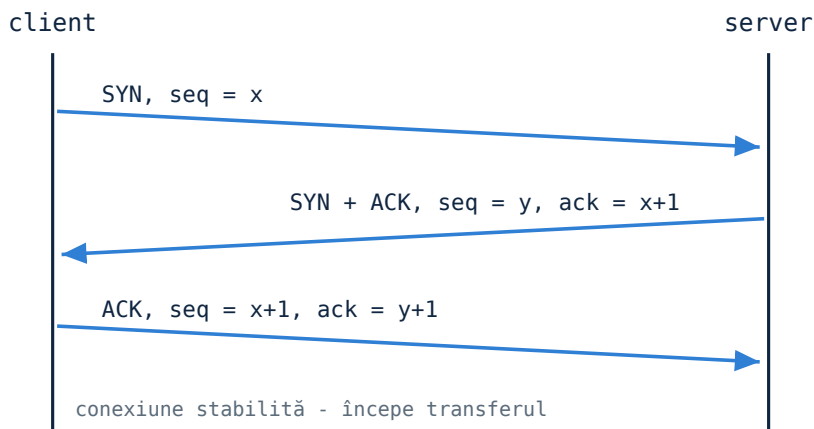


Figura 8.4 Cele trei segmente ale deschiderii unei conexiuni TCP, în forma clasică de diagramă temporală. Panta liniilor reprezintă timpul de propagare.

De ce numărul de secvență inițial este aleatoriu. Dacă ar fi previzibil, un atacator care cunoaște adresele și porturile ar putea fabrica segmente care par să facă parte din conversație - fără să vadă traficul. Se numește **TCP sequence prediction** și a fost o vulnerabilitate reală în anii '90, când multe sisteme porneau numerotarea de la o valoare crescută cu un pas fix.

8.4 Cele trei familii de atacuri

Orice rețea conectată la Internet este, permanent, ținta unor tentative automatizate. Nu e nevoie să fiți interesant: scanările sunt continue și oarbe. Atacurile se împart în trei familii, după scop.

Tabelul 8.4 Cele trei familii de atacuri.

Familie	Scop	Exemple	Ce ajută
De recunoaștere	aflarea a ce există și ce rulează	ping sweep, port scan, sniffing, interogări DNS și whois	filtrarea ICMP, DROP în loc de REJECT
De blocare (DoS/DDoS)	indisponibilizarea serviciului	SYN flood, smurf, amplificare DNS	limitarea ratei, SYN cookies
De acces	obținerea de drepturi nemeritate	spargere de parole, buffer overflow, man-in-the-middle	autentificare puternică, criptare

De ce DROP e mai bun decât REJECT în fața Internetului. REJECT aruncă pachetul și trimite un răspuns: „port închis”. Politicos - și extrem de util pentru cel care scanează, pentru că îi confirmă că mașina există.

DROP aruncă pachetul tăcut. Scanerul nu primește nimic și trebuie să aștepte expirarea temporizatorului pentru fiecare port. O scanare care ar dura o secundă durează minute.

În interiorul rețelei, însă, REJECT este preferabil: aplicațiile proprii primesc o eroare imediată în loc să aștepte 30 de secunde degeaba.

Cum funcționează un SYN flood



Figura 8.5 SYN flood, pas cu pas.

DEFINIȚIA 8.2 (FIREWALL). Una sau mai multe mașini al căror scop este prevenirea accesului neautorizat la o rețea. Controlează accesul la servicii atât dinspre exterior spre interior, cât și invers. Poate fi un dispozitiv dedicat, o funcție a unui ruter sau un program pe o stație.

Tabelul 8.5 Cum funcționează un SYN flood.

Tip	Până la ce nivel se uită	Ține minte sesiunile?	Exemplu
Stateless	3–4	nu - fiecare pachet este judecat izolat	ACL-uri pe ruter
Stateful	3–4, cu tabelă de conexiuni	da	iptables cu conntrack, Cisco ASA
De nivel aplicație (proxy)	până la 7	da, plus conținutul	proxy HTTP, WAF

Un firewall nu rezolvă toate cele trei familii, dar taie eficient primele două: poate bloca porturile vulnerabile, poate opri inițierea conexiunilor din exterior, poate refuza răspunsul la ICMP echo, poate limita numărul de sesiuni pe jumătate deschise și poate opri broadcast-urile direcționate care fac posibil atacul smurf. Împotriva atacurilor de acces ajută mai puțin: acolo lucrează criptografia și autentificarea, din secțiunea 10.

8.5 iptables: tabele și lanțuri

iptables este utilitarul din spațiul utilizator care configurează **Netfilter**, cadrul de filtrare din nucleul Linux. Permite unei mașini Linux să filtreze pachete, să translateze adrese și să rescrie câmpuri - adică să fie, simultan, ruter și firewall.

Analogie Netfilter e ca o clădire cu mai multe puncte de control, dispuse pe traseul obligatoriu al oricărui vizitator. La fiecare punct de control există un teanc de instrucțiuni.

Lanțul este punctul de control - *unde* pe traseu. **Tabela** este tipul de instrucțiuni - *ce fel de decizie* se ia acolo: filtrare, traducere de adrese sau modificare. Un punct de control poate avea mai multe teancuri.

Tabelul 8.6 iptables: tabele și lanțuri.

Tabelă	Ce conține
filter	regulile care decid ce trece și ce se aruncă; tabela implicită , folosită dacă nu scrieți -t
nat	regulile de traducere a adreselor - subiectul cursului 9
mangle	alterarea specializată a pachetelor: TTL, marcaje, ToS
raw	excepții de la urmărirea conexiunilor

Lanțurile sunt punctele din traseul unui pachet în care se aplică reguli. Cheia înțelegerii lui iptables este să știți **prin ce lanțuri trece** fiecare tip de pachet - și sunt doar trei tipuri.

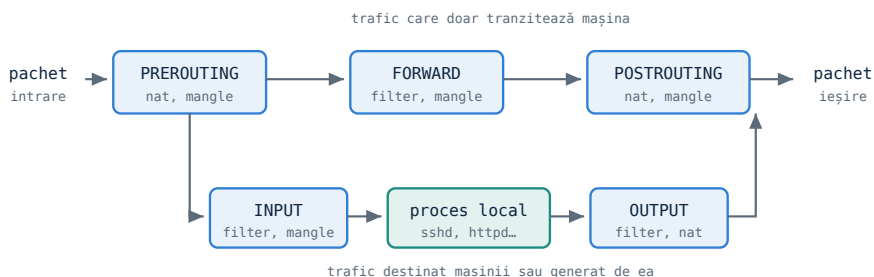


Figura 8.6 Traseul pachetelor prin lanțurile Netfilter. Un pachet care tranzitează mașina trece prin FORWARD; unul destinat ei trece prin INPUT; unul generat de ea trece prin OUTPUT.

Confuzia INPUT / FORWARD. Este cea mai frecventă sursă de reguli care nu au niciun efect. Întrebarea de pus, de fiecare dată: **cine este destinatarul din antetul IP?**

Dacă destinatarul este chiar mașina Linux (o sesiune SSH către ea, o pagină servită de ea) → **INPUT**.

Dacă destinatarul este altcineva, iar mașina doar rutează → **FORWARD**.

O regulă pusă pe INPUT pentru trafic care tranzitează nu se va aplica niciodată - și nici nu veți primi vreo eroare.



Figura 8.7 Ce lanț folosiți?.



Figura 8.8 Traseul unui pachet prin Netfilter.

8.6 Anatomia unei reguli iptables

O regulă are întotdeauna două părți: un **șablon** - ce condiții trebuie să îndeplinească pachetul - și o **țintă** - ce se întâmplă cu el.

```

iptables -t filter -A INPUT -s 10.0.0.0/8 -p icmp -j DROP
#      L  tabela  L op    L lant  L_____  condiții  _____  L tinta
#
# -t filter   tabela (implicită, se poate omite)
# -A INPUT    adauga la finalul lantului INPUT
# -s 10...    sursa: aici masca este NORMALA, nu wildcard!
# -p icmp     protocolul
# -j DROP     jump: ce se face cu pachetul
  
```

Listingul 8.1 o regulă completă, descompusă copiază

Atenție la mască. La Cisco am folosit **wildcard** (0.0.0.255). În iptables se folosește notația **CIDR normală** (/24) sau masca clasică. Este una dintre puținele locuri unde trecerea de la un ecosistem la altul chiar produce greșeli tăcute.

Tabelul 8.7 Anatomia unei reguli iptables.

Operație	Formă lungă	Ce face
-A	--append	adaugă o regulă la finalul lanțului
-I	--insert	inserează la o poziție dată (implicit prima)
-D	--delete	șterge o regulă
-L -n -v --line-numbers	--list	afișează regulile, cu contoare și numere de linie
-F	--flush	golește lanțul
-P	--policy	schimbă politica implicită

Tabelul 8.8 Anatomia unei reguli iptables.

Condiție	Ce verifică
-i eth0 / -o eth1	interfața de intrare / de ieșire
-s / -d	adresa sursă / destinație, în notație CIDR
-p tcp --dport 22	protocolul și portul destinație
-p tcp --syn	doar segmentele de inițiere a conexiunii
-m state --state ESTABLISHED,RELATED	starea sesiunii, din tabela conntrack
-m limit --limit 25/s	rata de sosire - pentru apărarea anti-flood
! -s 10.0.0.0/8	negarea oricărei condiții, cu semnul exclamării

Tabelul 8.9 Anatomia unei reguli iptables.

Țintă	Ce face
ACCEPT	lasă pachetul să treacă; evaluarea lanțului se oprește
DROP	aruncă tăcut - expeditorul nu află nimic
REJECT	aruncă și trimite un ICMP sau un RST TCP
LOG	înregistrează în jurnalul de sistem și continuă evaluarea

LOG este singura țintă care nu oprește parcurgerea. Toate celelalte sunt **terminale**: odată aplicate, pachetul a primit un verdict și lanțul nu se mai citește. La fel ca la ACL-urile Cisco.

LOG nu decide nimic - doar notează și lasă pachetul să meargă mai departe prin listă. De aceea o regulă de jurnalizare se pune **imediat deasupra** celei de DROP pe care vrea să o documenteze, cu exact aceleași condiții.



Figura 8.9 Terminal: inspectarea unui firewall Linux.

Priviți prima comandă. Coloana pkts este echivalentul exact al contorului de potriviri de la ACL-urile Cisco, și se folosește la fel: regula 4 din INPUT are **0 pachete**, ceea ce înseamnă că nimeni nu a accesat încă serverul web - sau că o regulă de deasupra prinde deja traficul.

Regulile iptables nu supraviețuiesc repornirii. Sunt ținute în memoria nucleului. Fără iptables-save (sau serviciul netfilter-persistent), la următorul boot mașina pornește cu lanțurile goale și politicile pe ACCEPT - adică fără firewall.

Partea bună: dacă v-ați blocat singuri, o repornire vă salvează. Partea proastă: dacă ați crezut că ați configurat un firewall, nu l-ați configurat.

8.7 Politica implicită și un firewall complet

Fiecare lanț predefinit are o **politică** - acțiunea aplicată pachetelor care nu s-au potrivit cu nicio regulă. Este echivalentul lui `deny any` implicit de la Cisco, cu o diferență importantă: aici este configurabilă, iar implicit este `ACCEPT`.

Politica `ACCEPT` - permisivă

- „Permite tot ce nu e interzis explicit"
- Trebuie să anticipați fiecare lucru rău
- Orice serviciu pornit din greșeală e expus
- Un port uitat rămâne deschis pentru totdeauna

Politica `DROP` - restrictivă.

- „Interzice tot ce nu e permis explicit"
- Trebuie să enumerați ce vreți să meargă
- Un serviciu nou nu merge până nu îl deschideți - **și e bine așa**
- Singura abordare acceptabilă în fața Internetului

Ordinea comenzilor, la propriu. Setați **întâi** regulile de `ACCEPT` și abia apoi politica `DROP`. Dacă faceți invers și lucrați prin SSH, prima comandă vă deconectează instantaneu de la mașină.

Lanțurile create de utilizator nu pot avea politică implicită - evaluarea lor se întoarce, pur și simplu, în lanțul apelant.

```
# 1. Curatam si pornim de la zero
iptables -F
iptables -X

# 2. Traficul pe interfata de bucla - mereu primul, altfel se strica servicii locale
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# 3. Regula care rezolva TOT traficul de intoarcere, dintr-o linie
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

# 4. Serviciile pe care le expunem noi
iptables -A INPUT -i eth1 -s 192.168.1.0/24 -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -j ACCEPT

# 5. LAN-ul are voie spre Internet
iptables -A FORWARD -i eth1 -o eth0 -s 192.168.1.0/24 -j ACCEPT

# 6. Jurnalizam ce urmeaza sa aruncam (LOG nu opreste parcurgerea)
iptables -A INPUT -m limit --limit 5/min -j LOG --log-prefix "IPTABLES-DROP: "

# 7. ABIA ACUM politicile
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# 8. Salvam, altfel totul dispare la repornire
iptables-save > /etc/iptables/rules.v4
```

Listingul 8.2 un firewall complet, pentru un ruter Linux copiază

Linia care merită reținută. `-m state --state ESTABLISHED,RELATED -j ACCEPT` este exact ce ne lipsea la ACL-urile Cisco. Nucleul ține o tabelă de conexiuni (conntrack); orice pachet care aparține unei sesiuni deja stabilite este acceptat automat, indiferent de port.

`RELATED` merge mai departe: acceptă și traficul **înrudit** - de exemplu conexiunea de date a unui FTP activ, sau mesajele ICMP de eroare legate de o sesiune existentă. Aceasta este, în concret, diferența dintre `stateless` și `stateful`.

Exercițiu rezolvat - antispoofing 8.1. O rețea privată are un ruter Linux. Configurați o politică antispoofing: pachetele cu adrese private nu au voie nici să intre din Internet, nici să iasă spre el.

antispoofing

```
iptables -A FORWARD -i eth0 -s 192.168.0.0/16 -j DROP
iptables -A FORWARD -i eth0 -s 172.16.0.0/12 -j DROP
iptables -A FORWARD -i eth0 -s 10.0.0.0/8 -j DROP
iptables -A FORWARD -o eth0 -d 192.168.0.0/16 -j DROP
iptables -A FORWARD -o eth0 -d 172.16.0.0/12 -j DROP
iptables -A FORWARD -o eth0 -d 10.0.0.0/8 -j DROP
```

Rezolvare. Regulile se pun pe `FORWARD`, pentru că este vorba de trafic care **tranzitează** ruterul.

Condiția de interfață este obligatorie: `-i eth0` înseamnă „venit dinspre Internet”. Fără ea, ați bloca și traficul intern legitim dintre două subrețele private - adică exact traficul pe care ruterul trebuie să îl ducă.

Exercițiu propus - patru bilete de la utilizatori 8.2. Topologie: un ruter Linux cu eth0 spre Internet, eth1 spre un server (142.31.16.9) și eth2 spre LAN (142.31.16.128/25). Administratorul lucrează de la 214.13.177.2, din Internet. Configurația de plecare este: INPUT: politică ACCEPT, regula -i eth0 -j DROP; FORWARD: politică DROP, regula -i eth0 -j ACCEPT.

Rezolvare. #1 - Stațiile din LAN nu ajung la server. Cauza: politica lanțului FORWARD este DROP, iar singura regulă permite doar traficul venit pe eth0 - adică din Internet.

soluția

```
iptables -P FORWARD ACCEPT
```

#2 - Doar stațiile din LAN au voie la HTTP-ul serverului.

soluția

```
iptables -F FORWARD
iptables -A FORWARD -s 142.31.16.128/25 -d 142.31.16.9 -p tcp --dport 80 -j ACCEPT
iptables -A FORWARD -d 142.31.16.9 -p tcp --dport 80 -j DROP
```

Ordinea este obligatorie: regula permisivă, mai specifică, trebuie să fie deasupra celei restrictive. Exact aceeași regulă ca la ACL-uri.

#3 - Doar administratorul are voie la SSH pe ruter. Fiind trafic *destinat ruterului*, lanțul este INPUT, nu FORWARD.

soluția

```
iptables -F INPUT
iptables -A INPUT -s 214.13.177.2 -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j DROP
```

#4 - Sesiunile TCP să poată fi inițiate doar dinspre interior. Prima variantă, `iptables -A FORWARD -p tcp --syn -j DROP`, este greșită: blochează inclusiv conexiunile pornite din LAN. Trebuie restrânsă la traficul care *intră* dinspre Internet:

soluția

```
iptables -A FORWARD -i eth0 -p tcp --syn -j DROP
```

Observați că soluția modernă ar fi cu totul alta - o singură linie cu `--state ESTABLISHED,RELATED` - și că ea acoperă și UDP-ul, și ICMP-ul, pe care `--syn` nu le atinge deloc.

8.8 Arhitectura perimetrului

Unde anume se pune firewall-ul față de ruterul de margine? Decizia este, cel mai adesea, impusă de furnizorul de Internet, dar principiile sunt clare.

Tabelul 8.10 Arhitectura perimetrului.

Arhitectură	Când se folosește	Observații
Ruter în față, firewall în spate	rețele medii și mari	cea mai frecventă: ruterul acoperă cerințele de rutare pe care firewall-ul nu le poate îndeplini
Firewall în față, ruter în spate	cerințe de securitate foarte stricte	ruterul intern este protejat, dar firewall-ul trebuie să facă și rutare
Un singur echipament	rețele mici, rutare statică	simplu și ieftin; punct unic de defectare

IDS și IPS

Un firewall decide pe baza **antetelor**. Pentru inspecția **conținutului** se adaugă un echipament separat, iar diferența dintre cele două variante este exclusiv de poziționare.

IPS - Intrusion Prevention System.

- Plasat **în calea** traficului, în serie
- Poate **opri** pachetele periculoase
- Devine un potențial punct de gâtuire
- Dacă pică, poate întrerupe tot traficul
- Un fals pozitiv taie trafic legitim

IDS - Intrusion Detection System

- Primește o **copie** a traficului, în paralel
- Doar **semnalează**; nu poate opri nimic
- Nu influențează performanța rețelei
- Dacă pică, traficul curge nestingherit
- Un fals pozitiv generează doar o alarmă

Alegerea nu este „care e mai bun”, ci „ce se întâmplă când greșește”. Într-o rețea unde disponibilitatea contează mai mult decât riscul, IDS. Într-una unde un singur pachet greșit e inacceptabil, IPS.

8.9 SSH și cele trei concepte de securitate

Telnet transmite totul în clar, inclusiv parola. Oricine ascultă traficul o obține - nu e nevoie de niciun efort, doar de acces la un port de switch oglindit sau la o rețea wireless. SSH rezolvă problema pe trei planuri distincte, care merită înțelese **separat**, pentru că se regăsesc, cu alte nume, în orice discuție de securitate.

Tabelul 8.11 SSH și cele trei concepte de securitate.

Concept	Întrebarea la care răspunde	Mecanismul din SSH
Autentificare	Sunt cei doi cine pretind că sunt?	parolă sau chei asimetrice
Confidențialitate	Poate altcineva să citească mesajul?	criptare simetrică (AES, ChaCha20)
Integritate	A fost mesajul modificat pe drum?	MAC - cod de autentificare a mesajului

De ce sunt trei probleme și nu una. Sunt independente, iar rezolvarea uneia nu o atinge pe cealaltă.

Un mesaj poate fi **criptat** perfect și totuși trimis de un impostor - criptarea nu spune cine scrie. Un mesaj poate fi **semnat** corect și totuși citit de toată lumea. Iar un mesaj criptat poate fi **modificat** în tranzit fără ca atacatorul să înțeleagă ce modifică, ceea ce e suficient ca să facă rău.

De aceea orice protocol serios le tratează pe toate trei, cu mecanisme diferite.

Confidențialitatea: problema cheii comune

Criptarea simetrică - aceeași cheie la ambele capete - este rapidă și eficientă. Are însă o problemă care pare imposibilă: cum ajung cele două capete să dețină aceeași cheie, dacă singurul canal disponibil este chiar cel pe care vor să îl protejeze?

Răspunsul este **schimbul Diffie-Hellman**, publicat în 1976, și e una dintre cele mai elegante idei din informatică: cele două capete construiesc o cheie comună **fără ca aceasta să fie transmisă vreodată**.

Analogie cu vopseluri Doi oameni vor o culoare secretă comună, dar tot ce își trimit este văzut de toată lumea.

Se pun de acord public asupra unei culori de bază - galben. Fiecare adaugă în secret o culoare proprie și trimite amestecul. Un observator vede cele două amestecuri, dar **nu poate separa** culorile din ele.

Fiecare adaugă apoi propria culoare secretă în amestecul primit. Ambii ajung la exact aceeași culoare finală - galben plus secretul lui plus secretul ei. Observatorul are galbenul și cele două amestecuri, dar nu poate produce culoarea finală, pentru că i-ar trebui unul dintre secrete.



Figura 8.10 Schimbul Diffie-Hellman.

Jucați-vă cu valorile. Schimbați secretul clientului: cheia comună se schimbă - dar rămâne **identică** pe ambele capete, deși cele două secrete nu au părăsit niciodată calculatoarele respective. Ce circulă pe canal sunt doar p , g , A și B .

Securitatea stă în faptul că, deși $A = g^a \bmod p$ se calculează instantaneu, drumul invers - aflarea lui a din A - este **problema logaritmului discret**, pentru care nu se cunoaște niciun algoritm rapid. Cu numerele de jucărie de mai sus se sparge prin încercări; cu un p de 2048 de biți, nu.

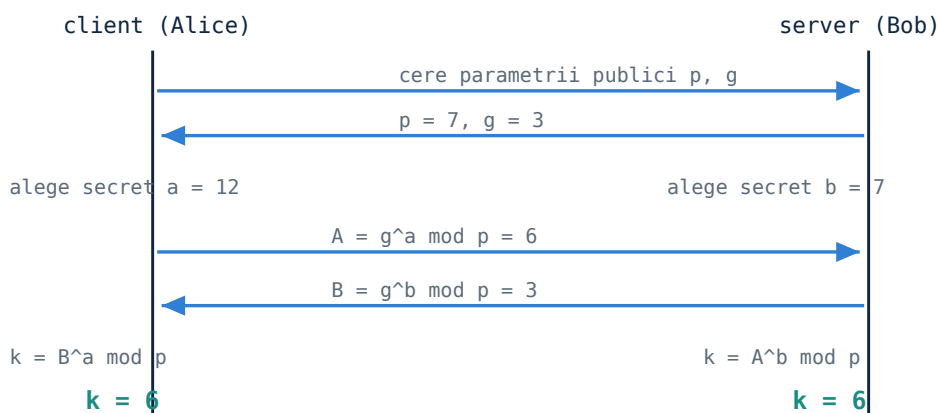


Figura 8.11 Schimbul Diffie-Hellman ca diagramă temporală. Cheia comună k nu circulă niciodată pe canal; ea este calculată independent, la ambele capete.

Ce NU rezolvă Diffie-Hellman. Schimbul garantează că nimeni **care doar ascultă** nu află cheia. Nu garantează absolut nimic despre **cine** este la celălalt capăt.

Un atacator plasat în mijloc poate face două schimburi DH separate - unul cu clientul, altul cu serverul - și să decripteze, citească și recripteze tot traficul. Este atacul **man-in-the-middle**, iar soluția nu ține de DH, ci de **autentificare**: de aceea SSH vă întreabă, la prima conectare, dacă recunoașteți amprenta cheii serverului. Mesajul acela pe care toată lumea îl acceptă fără să citească este exact apărarea împotriva acestui atac.

Autentificarea: parolă sau cheie

Autentificarea are loc **după** ce canalul criptat există. Pare să rezolve problema parolilor - dar nu complet:

- serverul trebuie să vadă parola ca să o valideze; dacă serverul este compromis, parola este compromisă - și, de regulă, e refolosită și altundeva;

- parolele suficient de sigure sunt greu de reținut, deci utilizatorii aleg parole slabe;
- o parolă poate fi ghicită prin încercări repetate, iar Internetul are răbdare infinită.

De aceea se preferă **cheile asimetrice**. O pereche de chei are proprietatea că ceea ce este criptat cu una poate fi decriptat doar cu cealaltă: $K^+(K^-(M)) = M$ și $K^-(K^+(M)) = M$. Clientul își păstrează cheia privată - care nu părăsește niciodată calculatorul lui - iar cheia publică este configurată pe server.

1. Se stabilește sesiunea sigură prin Diffie-Hellman.
2. Clientul cere autentificarea pentru un anumit utilizator.
3. Serverul trimite un **challenge**: un șir aleator pe care clientul trebuie să îl semneze.
4. Clientul îl criptează cu cheia sa privată. Rezultatul se numește **authenticator**.
5. Serverul verifică folosind cheia publică preconfigurată. Dacă rezultatul coincide cu șirul trimis, clientul deține cheia privată - deci este cine pretinde.

Observați ce nu s-a întâmplat niciodată în cei cinci pași: **cheia privată nu a fost transmisă**. Serverul nu o cunoaște și nu o poate afla. Un server compromis nu vă compromite identitatea pe alte servere - spre deosebire de o parolă.

De ce doar challenge-ul este criptat asimetric. Criptarea asimetrică este cu ordine de mărime mai lentă decât cea simetrică. A cripta astfel tot traficul ar fi inacceptabil. Cheia privată este folosită strict pentru operația de autentificare; restul sesiunii merge pe cheia simetrică obținută prin Diffie-Hellman.

Același principiu - **asimetric pentru schimbul de chei, simetric pentru date** - stă la baza TLS și a oricărei conexiuni HTTPS pe care o deschideți.

Integritatea: codul MAC

Un **MAC** (Message Authentication Code) este, în esență, un hash calculat **cu o cheie**. Emițătorul îl calculează peste mesaj și îl trimite alături; receptorul îl recalculează cu aceeași cheie și compară. Dacă valorile diferă, mesajul a fost modificat pe drum.

Cheia este esențială. Un hash simplu nu ar ajuta: atacatorul ar modifica mesajul și ar recalcula hash-ul. Fără cheie, nu poate produce un MAC valid pentru mesajul modificat.

Ce se vede și ce nu, într-o captură SSH. Un Wireshark pornit pe o sesiune SSH arată adresele IP, porturile, momentele și dimensiunile pachetelor - deci *că* există o conversație, cu cine și cât de intensă este. Se numesc **meta-date**, și ele scapă criptării.

Nu arată conținutul, nu arată parola, nu arată comenzile executate și nu permite modificarea traficului fără ca ambele capete să observe imediat.

8.10 Greșeli frecvente

Greșeli frecvente.

„Regula mea iptables nu are niciun efect”

Aproape sigur e pe lanțul greșit: INPUT în loc de FORWARD, sau invers.

Întrebați cine e destinatarul din antetul IP. Mașina însăși → INPUT. Altceva → FORWARD.

„Am pus politica DROP și m-am deconectat”

Politica s-a aplicat înaintea regulii care permitea SSH-ul dumneavoastră.

Întâi toate regulile ACCEPT, apoi -P DROP . Pe mașini la distanță, testați cu o a doua sesiune deschisă.

„Firewall-ul a dispărut după repornire”

Regulile trăiesc în memoria nucleului și nu se salvează singure.

iptables-save sau serviciul netfilter-persistent .

„Am folosit wildcard-ul Cisco în iptables”

-s 10.0.0.0/0.0.0.255 nu înseamnă ce credeți. iptables folosește masca normală sau CIDR.

-s 10.0.0.0/24 . Verificați cu iptables -L -n , care afișează prefixul rezultat.

„Am blocat ICMP complet, ca să fiu în siguranță”

Ați blocat și mesajele fragmentation needed , esențiale pentru descoperirea MTU-ului. Rezultatul: conexiuni care se deschid și apoi îngheață inexplicabil.

Blocați doar echo-request dinspre exterior; lăsați ICMP-ul de eroare să treacă.

„Diffie-Hellman mă protejează de tot”

Vă protejează de cineva care ascultă. Nu de cineva care se așază în mijloc.

Verificați amprenta cheii la prima conectare. Este singurul moment în care contează.

„Am confundat DROP cu REJECT”

REJECT confirmă atacatorului că mașina există.

DROP spre exterior, REJECT spre interior - ca aplicațiile proprii să nu aștepte degeaba.

8.11 Rezumat și glosar

Ce trebuie să rămână.

- Nivelul 4 adaugă **portul**; perechea IP + port se numește **socket**.
- TCP e sigur și cu conexiune; UDP e rapid și fără garanții. Alegerea depinde de aplicație.
- Strângerea de mână are **trei** segmente; închiderea are **patru**, pentru că fiecare sens se închide separat.
- Un **SYN flood** umple coada de conexiuni pe jumătate deschise; apărarea sunt SYN cookies și limitarea ratei.
- În iptables, **lanțul** spune *unde*, **tabela** spune *ce fel de decizie*.
- **INPUT** = pentru mașină, **FORWARD** = prin mașină, **OUTPUT** = de la mașină.
- Politica corectă este DROP, iar regulile ACCEPT se pun **înaintea** ei.
- --state ESTABLISHED, RELATED rezolvă tot traficul de întoarcere într-o linie.
- **Autentificare, confidențialitate, integritate** sunt trei probleme distincte, cu trei mecanisme distincte.
- Diffie-Hellman rezolvă schimbul de chei, **nu** identitatea celui alt capăt.

Exerciții

1. O regula iptables trebuie sa blocheze traficul destinat chiar masinii Linux. Ce lant folositi?
 - a. FORWARD
 - b. INPUT
 - c. OUTPUT
 - d. POSTROUTING
2. De ce deschiderea unei conexiuni TCP are trei segmente, iar inchiderea patru?
 - a. Pentru ca inchiderea e mai lenta
 - b. Pentru ca la deschidere SYN-ul si ACK-ul serverului incap in acelasi segment, iar la inchidere intre ACK si FIN mai poate curge trafic
 - c. Pentru ca inchiderea foloseste UDP
 - d. Este o conventie arbitrara
3. Ce exploateaza un atac SYN flood?
 - a. O eroare de programare din serverul web
 - b. Faptul ca serverul alocă resurse la pasul 2 al strangerii de mana, inainte sa primeasca ACK-ul final
 - c. Lipsa criptarii
 - d. Dimensiunea mica a antetului TCP
4. Ce garanteaza schimbul Diffie-Hellman?
 - a. Ca serverul este cine pretinde ca este
 - b. Ca cele doua capete obtin o cheie comuna fara ca aceasta sa fie transmisa pe canal
 - c. Ca mesajul nu a fost modificat
 - d. Ca parola este suficient de lunga
5. Ce diferentiaza un firewall stateful de unul stateless?
 - a. Viteza de procesare
 - b. Faptul ca tine minte sesiunile stabilite si poate recunoaste traficul de intoarcere
 - c. Numarul de interfete
 - d. Faptul ca poate cripta traficul
6. In ce ordine se dau comenzile, cand configurati un firewall prin SSH?
 - a. Intai politica DROP, apoi regulile ACCEPT
 - b. Intai regulile ACCEPT, apoi politica DROP
 - c. Nu conteaza ordinea
 - d. Intai se salveaza, apoi se configureaza
7. Ce face tinta LOG in iptables?
 - a. Arunca pachetul si il inregistreaza
 - b. Inregistreaza pachetul si continua evaluarea lantului
 - c. Opreste conexiunea si trimite un RST
 - d. Salveaza regulile pe disc
8. De ce se prefera DROP in loc de REJECT pe interfata dinspre Internet?
 - a. Este mai rapid de procesat
 - b. Pentru ca REJECT confirma atacatorului ca masina exista, iar DROP il obliga sa astepte expirarea temporizatorului
 - c. Pentru ca REJECT nu functioneaza pe TCP
 - d. Pentru ca DROP inregistreaza automat in jurnal

9. De ce nu se criptează asimetric întreaga sesiune SSH?
- Pentru că nu este sigură
 - Pentru că este cu ordine de mărime mai lentă decât criptarea simetrică
 - Pentru că nu funcționează pe fișiere mari
 - Pentru că ar necesita două chei publice

Note bibliografice

Cursul următor folosește exact aceleași unelte - ACL-uri pe Cisco, iptables pe Linux - dar pentru altceva decât filtrare: **traducerea adreselor** și construirea de **tuneluri**. Veți vedea acolo tabela `nat` pe care am amânat-o azi, și veți înțelege de ce o rețea privată poate naviga pe Internet cu o singură adresă publică.

Laboratorul 7 aplică practic partea de securitate: SSH pe echipamente, plus apărarea împotriva atacurilor de nivel 2 din cursul 12.

- RFC 9293 - Transmission Control Protocol (versiunea consolidată a lui RFC 793)
- RFC 4987 - TCP SYN Flooding Attacks and Common Mitigations
- RFC 4251–4254 - arhitectura SSH
- Manualele `iptables(8)` și `iptables-extensions(8)`
- William Stallings, *Cryptography and Network Security* - capitolele despre Diffie-Hellman și MAC

Extras din volumul „Rețele Locale - Note de curs și lucrări de laborator”. Varianta online a acestui capitol conține figurile în formă interactivă.