Warstwa pod firewalld: nftables i iptables

◫◫ Poziom: średni

◫◫ Dotyczy: RHEL/Rocky 9–10 (nftables jako backend), różnice Debian/Ubuntu i SLES zaznaczone

▯ Czas: ~22 min

`firewalld` to tylko fasada. Pod spodem system filtruje pakiety przez netfilter, a od RHEL 8 językiem tego filtrowania jest nftables — `iptables` jest już tylko tłumaczem na starą składnię. Ta strona pokazuje warstwę, do której `firewalld` faktycznie pisze, i to, co potrafi pójść źle, kiedy administrator sięga po nią ręcznie.

◫◫ Skróty użyte na tej stronie (11)

Skrót	Co znaczy
DNAT	Destination Network Address Translation: podmiana adresu docelowego pakietu, używana do przekierowania ruchu z zewnątrz do serwera wewnątrz sieci
DNS	Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
IPv4	Internet Protocol wersja 4: adresy zapisane czterema liczbami z kropkami, na przykład 10.20.10.77
IPv6	Internet Protocol wersja 6: nowszy, dłuższy zapis adresów, w liczbach szesnastkowych oddzielonych dwukropkami
NAT	Network Address Translation: podmiana adresów IP na granicy sieci
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SLES	SUSE Linux Enterprise Server: płatna dystrybucja Linuksa firmy SUSE, konkurent Rocky/RHEL
SNAT	Source NAT: podmiana adresu nadawcy w ruchu wychodzącym
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
TCP	Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia

Fiszka

□□ TYLKO ODCZYT

Wypisanie całego aktualnego zestawu reguł: nft list ruleset

```
sudo nft list ruleset
```

Wypisuje wszystko: tabele, łańcuchy i reguły, w formacie, który da się zapisać do pliku i odtworzyć. To pierwsza komenda, którą uruchamiamy, gdy nie wiadomo, co naprawdę filtruje pakiety — w tym `firewalld`, który swoje reguły trzyma właśnie tutaj.

Przykład:

```
table inet firewalld {
□chain filter_IN_public {
□□icmp type { echo-request } accept
□□tcp dport 22 accept
□□tcp dport 443 accept
□}
}
table inet filter {
□chain input {
□□type filter hook input priority filter; policy accept;
□}
}
```

□□ **nft** = nftables: nowa zapora jądra Linuksa

□□ TYLKO ODCZYT

Wypisanie samych tabel, bez reguł: nft list tables

```
sudo nft list tables
```

Skrócony widok — same nazwy tabel wraz z rodziną adresową. Przydaje się, żeby zorientować się, ile warstw filtrowania w ogóle działa w systemie, zanim zejdziemy do szczegółów.

Przykład:

```
table inet firewalld
table ip filter
```

```
table ip6 filter
```

nft = nftables: nowa zapora jądra Linuksa

TYLKO ODCZYT

Utworzenie nowej tabeli: nft add table inet filter

```
sudo nft add table inet filter
```

Tworzy pustą tabelę o nazwie `filter` w rodzinie `inet` (obsługuje IPv4 i IPv6 jednym zestawem reguł — nowość względem starego, rozdzielonego `iptables`/`ip6tables`). Sama tabela nic nie filtruje — to tylko pojemnik na łańcuchy, patrz *Jak to działa*.

IPv4 = Internet Protocol wersja 4: adresy zapisane czterema liczbami z kropkami, na przykład 10.20.10.77 · **IPv6** = Internet Protocol wersja 6: nowszy, dłuższy zapis adresów, w liczbach szesnastkowych oddzielonych dwukropkami · **nft** = nftables: nowa zapora jądra Linuksa

TYLKO ODCZYT

Dodanie łańcucha podłączonego do haka: nft add chain

```
sudo nft add chain inet filter input { type filter hook input priority 0 \; policy accept \; }
```

Dodaje do tabeli `filter` łańcuch `input`, podczepiony pod hak `input` z priorytetem 0 i domyślną polityką `accept`. Średnik w środkowej klamrze trzeba escapować (`\;`) w powłoce, albo cały zapis wsadzić do pliku `.nft` — bez tego powłoka przerywa komendę na pierwszym średniku.

nft = nftables: nowa zapora jądra Linuksa

TYLKO ODCZYT

Dodanie pojedynczej reguły w łańcuchu: nft add rule

```
sudo nft add rule inet filter input tcp dport 22 accept
```

Dopisuje regułę na koniec łańcucha `input`: pakiety TCP na port 22 są przyjmowane. Kolejność dopisywania ma znaczenie — patrz zasada pierwszego trafienia w *Jak to działa*.

□□ **TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia ·**nft** = nftables: nowa zapora jądra Linuksa

□□ TYLKO ODCZYT

Wczytanie całej konfiguracji z pliku: nft -f plik.nft

```
sudo nft -f /etc/nftables/custom.nft
```

Ładuje kompletny zestaw tabel, łańcuchów i reguł z pliku tekstowego. To jedyny rozsądny sposób pracy z `nft` na produkcji — reguły wpisywane pojedynczo w terminalu nie przetrwają restartu i trudno je wersjonować w gicie.

□□ **nft** = nftables: nowa zapora jądra Linuksa

□□ TYLKO ODCZYT

Wyczyszczenie całego zestawu reguł: nft flush ruleset

```
sudo nft flush ruleset
```

Usuwa **wszystkie** tabele, łańcuchy i reguły, włącznie z tymi, które napisał `firewalld`. Po tej komendzie system przepuszcza cały ruch bez żadnego filtrowania — dobre do testów na maszynie bez dostępu z sieci, fatalne na produkcyjnym serwerze wystawionym do internetu.

□□ **nft** = nftables: nowa zapora jądra Linuksa

□□ TYLKO ODCZYT

Wypisanie reguł klasycznym widokiem: iptables -L -n -v

```
sudo iptables -L -n -v
```

`-L` wypisuje reguły, `-n` pokazuje adresy i porty numerycznie (bez rozwiązywania nazw — szybciej i bez zależności od DNS), `-v` dodaje liczniki pakietów/bajtów i nazwę interfejsu. Na RHEL 8+ ta komenda nie czyta osobnej, starej tabeli jądra — tłumaczy na bieżąco reguły trzymane w nftables (patrz *Jak to działa*).

Przykład:

```
Chain INPUT (policy ACCEPT 128K packets, 42M bytes)
 pkts bytes target     prot opt in     out     source               destination
 118K   38M ACCEPT     tcp  --  ens192 *       0.0.0.0/0
0.0.0.0/0            tcp dpt:22
```

DNS = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP · **RHEL** = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **iptables** = starsza zapora sieciowa jądra Linuksa, sterowana regułami w tabelach i łańcuchach

TYLKO ODCZYT

Wypisanie reguł w formacie do skopiowania: iptables -S

```
sudo iptables -S
```

Ten sam zestaw reguł co `-L`, ale w formacie identycznym ze składnią, którą wpisujemy przy dodawaniu (`-A INPUT -p tcp --dport 22 -j ACCEPT`). Wygodne do skopiowania reguły i odtworzenia jej na innym serwerze albo w skrypcie.

Przykład:

```
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
```

iptables = starsza zapora sieciowa jądra Linuksa, sterowana regułami w tabelach i łańcuchach

TYLKO ODCZYT

Wypisanie reguł translacji adresów: iptables -t nat -L -n

```
sudo iptables -t nat -L -n
```

Domyślnie `iptables` pokazuje tabelę `filter`. Flaga `-t nat` przełącza na tabelę odpowiedzialną za NAT — tu szukamy reguł DNAT/SNAT/MASQUERADE, temat kolejnej strony rozdziału.

Przykład:

```
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       tcp  --  0.0.0.0/0            203.0.113.10         tcp dpt:25
to:10.20.10.42:25

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  tcp  --  10.20.10.0/24        0.0.0.0/0            tcp dpt:25
```

□□ **NAT** = Network Address Translation: podmiana adresów IP na granicy sieci · **DNAT** = Destination Network Address Translation: podmiana adresu docelowego pakietu, używana do przekierowania ruchu z zewnątrz do serwera wewnątrz sieci · **iptables** = starsza zapora sieciowa jądra Linuksa, sterowana regułami w tabelach i łańcuchach

□□ TYLKO ODCZYT

Dopisanie reguły na koniec łańcucha: iptables -A INPUT -p tcp --dport NumerPortu -j ACCEPT

```
sudo iptables -A INPUT -p tcp --dport NumerPortu -j ACCEPT
```

`-A` (append) dokleja regułę na **koniec** łańcucha `INPUT`. Jeśli wcześniejsza reguła w tym łańcuchu już odrzuca ten ruch, ta reguła nigdy nie zostanie sprawdzona — patrz zasada pierwszego trafienia w *Jak to działa*.

□□ **iptables** = starsza zapora sieciowa jądra Linuksa, sterowana regułami w tabelach i łańcuchach

□□ TYLKO ODCZYT

Usunięcie konkretnej reguły: iptables -D

```
sudo iptables -D INPUT -p tcp --dport NumerPortu -j ACCEPT
```

Usuwa regułę, która dokładnie odpowiada podanemu opisowi — składnia musi się zgadzać znak w znak z tym, co pokazuje `-S`. Wygodniejsza alternatywa przy wielu regułach: numer z `--line-numbers`.

Przykład:

```
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
```

```
1    ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:22
2    ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:443
```

iptables = starsza zapora sieciowa jądra Linuksa, sterowana regułami w tabelach i łańcuchach

ZMIENIA SYSTEM

Zrzut reguł do formatu tekstowego: iptables-save

```
sudo iptables-save > /etc/sysconfig/iptables
```

Zapisuje cały aktualny zestaw reguł (wszystkie tabele) w formacie do wczytania przez `iptables-restore`. Bez tego kroku reguły dodane ręcznie przez `-A` znikają przy najbliższym restarcie — żyją tylko w pamięci jądra.

ZMIENIA SYSTEM

Wczytanie reguł z pliku: iptables-restore

```
sudo iptables-restore < /etc/sysconfig/iptables
```

Odwrotność `iptables-save`: wgrywa cały zestaw reguł z pliku, zastępując to, co jest aktualnie załadowane. Na systemach z `firewalld` ta para komend zwykle nie jest potrzebna — `firewalld` ma własny mechanizm trwałości (`--permanent` + `--reload`).

TYLKO ODCZYT

Podejrzenie tłumaczenia reguły na nft: iptables-translate

```
iptables-translate -A INPUT -p tcp --dport NumerPortu -j ACCEPT
```

Nie wykonuje reguły — tylko pokazuje, jak dokładnie zostałaby zapisana w składni `nft`. Dobry sposób nauki nowej składni: bierzemy starą regułę, którą rozumiemy, i patrzymy na jej odpowiednik.

Przykład:

```
nft add rule ip filter INPUT tcp dport 22 counter accept
```

◫◫ TYLKO ODCZYT

Odczyt przełącznika przekazywania pakietów: sysctl net.ipv4.ip_forward

```
sysctl net.ipv4.ip_forward
```

Sam odczyt, bez zmiany. Wynik `0` znaczy, że jądro nie przekazuje pakietów między interfejsami — nawet najlepiej napisane reguły NAT nic nie zdziałają, jeśli ten przełącznik jest wyłączony. Pełne omówienie i trwałe włączenie: następna strona rozdziału.

Przykład:

```
net.ipv4.ip_forward = 0
```

◫◫ **NAT** = Network Address Translation: podmiana adresów IP na granicy sieci · **sysctl** = odczytaj albo zmień parametr jądra Linuksa w czasie działania systemu, tu limit vm.max_map_count wymagany przez OpenSearch

◫◫ TYLKO ODCZYT

Wszystkie flagi i komendy

Komenda / flaga	Co robi
`nft list ruleset`	pełny zrzut tabel/łańcuchów/reguł
`nft list tables`	tylko nazwy tabel
`nft add table`	nowa, pusta tabela
`nft add chain ... hook ...`	nowy łańcuch podczepiony pod hak netfiltera
`nft add rule`	dopisanie reguły na koniec łańcucha
`nft -f plik.nft`	wczytanie konfiguracji z pliku
`nft flush ruleset`	usunięcie wszystkiego
`iptables -L -n -v`	reguły z licznikami, adresy numerycznie
`iptables -S`	reguły w formacie do skopiowania
`iptables -t nat -L -n`	reguły tabeli NAT
`iptables -A ... -j ACCEPT`	dopisanie reguły na koniec łańcucha
`iptables -D`	usunięcie konkretnej reguły
`iptables-save` / `-restore`	zrzut / wczytanie reguł z pliku

Komenda / flaga	Co robi
`iptables-translate`	podgląd tłumaczenia reguły na `nft`
`sysctl net.ipv4.ip_forward`	odczyt przełącznika przekazywania pakietów

Uważaj

- `nft flush ruleset` usuwa też reguły `firewalld` — na serwerze produkcyjnym to odcięcie się od SSH, jeśli łączysz się przez sieć.
- Ręczne `iptables -A` na systemie z `firewalld` przetrwa tylko do najbliższego `firewall-cmd --reload`, które nadpisze tabelę nftables z własnej, trwałej konfiguracji.
- `iptables -D` wymaga podania reguły **identycznej** co do znaku ze składnią z `-S` — łatwiej usuwać po numerze z `--line-numbers`.

SSH = Secure Shell: szyfrowane zdalne logowanie do serwera

Z życia wzięte

Z ŻYCIA WZIĘTE

Wtorek. Stara reguła iptables na ipa01 blokuje ruch, mimo że administrator, który ją pisał, dawno odszedł z firmy. Tomek sprawdza nftables, żeby zobaczyć, przez które łańcuchy faktycznie przechodzi pakiet, zanim zdecyduje, którą regułę bezpiecznie usunąć.

Po co to istnieje

Pojęcie: filtrowanie pakietów w jądrze. Każdy pakiet sieciowy, zanim trafi do aplikacji albo zostanie przekazany dalej, przechodzi przez jądro systemu. Jądro Linuksa ma wbudowany mechanizm, który w konkretnych, ustalonych miejscach tej drogi pyta: czy ten pakiet ma przejść, zostać odrzucony, czy poddany jakiejś przemianie? Odpowiedź nie jest zaszyta w kodzie jądra — administrator konfiguruje ją z zewnątrz, w postaci reguł. Ta konfigurowalna warstwa nazywa się netfilter, a `nftables` i `iptables` to dwa różne języki, w których można te reguły zapisać.

`firewalld` jest wygodną fasadą nad tym mechanizmem — strefy, usługi, `--add-port`. Ale sam nie filtruje ani jednego pakietu. Kiedy wydajesz `firewall-cmd --add-port=443/tcp --permanent &&`

`firewall-cmd --reload`, `firewalld` tłumaczy to na zestaw tabel i reguł nftables i wgrywa je do jądra. Ta strona jest o tej warstwie: co tam faktycznie się dzieje, jak to podglądnąć i czym różni się `nftables` od starszego `iptables`.

Sięgasz po tę wiedzę wtedy, gdy `firewall-cmd --list-all` mówi, że port jest otwarty, a połączenie mimo to nie przechodzi — bo w grę wchodzi reguła NAT, inna strefa, albo ręcznie dopisana reguła `iptables`, o której `firewalld` nie wie.

Analogia

Wyobraź sobie **lotniskową kontrolę bezpieczeństwa z trzema niezależnymi punktami**: przy wejściu do terminala, przy wejściu na konkretne gate'y i przy wyjściu na płytę.

Każdy punkt kontroli to **hak** — miejsce, w którym ktoś *na pewno* zatrzyma Cię i sprawdzi dokumenty. W każdym punkcie stoi konkretna **procedura** (łańcuch reguł) wykonywana po kolei, od pierwszej instrukcji do ostatniej. Strażnik zatrzymuje się na **pierwszej instrukcji, która daje jednoznaczną odpowiedź** — jeśli trzecia instrukcja mówi „zatrzymaj tę osobę", nikogo nie interesuje, że dziesiąta mówiła „wpuszczaj wszystkich z tego kraju".

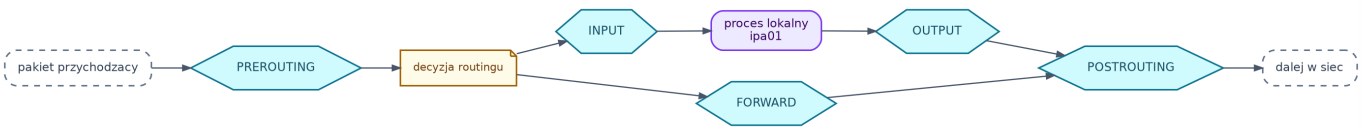
`firewalld` to kierownik zmiany, który wydaje polecenia strażnikom w ludzkim języku: „wpuszczaj ruch na gate 22, blokuj resztę". Strażnicy i tak wykonują te same, surowe procedury (nftables) — kierownik tylko je za nich zapisuje. Jeśli ktoś inny — administrator korzystający z `iptables` albo z `nft` bezpośrednio — dopisze własną instrukcję do procedury strażnika, a potem kierownik wróci i **na nowo przepisze całą procedurę** (odpowiednik `firewall-cmd --reload`), dopisana ręcznie instrukcja przepada. Nikt jej nie usunął złośliwie — kierownik po prostu nie wiedział, że tam była.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: tabela, łańcuch, reguła.** `nftables` organizuje konfigurację w trzech poziomach zagnieżdżenia. **Tabela** to najwyższy pojemnik, przypisany do rodziny adresów (`ip`, `ip6`, `inet` — czyli oba naraz, `arp`, `bridge`). W tabeli tworzysz **łańcuchy** — nazwane listy reguł, z których część jest podczepiona pod konkretny punkt przepływu pakietu, a część istnieje tylko jako cel do przeskoku z innego łańcucha (`jump`). W łańcuchu leżą **reguły** — pojedyncze warunki plus akcja, sprawdzane po kolei od pierwszej do ostatniej. `iptables` ma identyczną trójpoziomową strukturę — tabele takie jak `filter` czy `nat` z gotowymi łańcuchami `INPUT` / `OUTPUT` / `FORWARD` — tylko starszą,

mniej elastyczną składnię i osobne binarki dla IPv4 (`iptables`) i IPv6 (`ip6tables`).



Pakiet przechodzi przez kolejne hooki netfilter, zanim trafi do procesu lokalnego albo dalej w sieć. (kliknięcie otwiera pełny rozmiar)

Pojęcie: haki netfiltera. Hak to ustalone miejsce w drodze pakietu przez jądro, do którego można podczepić łańcuch reguł. Pięć podstawowych haków to: `prerouting` (zanim jądro zdecyduje, czy pakiet jest dla tej maszyny, czy ma być przekazany dalej), `input` (pakiet zaadresowany do tej maszyny), `forward` (pakiet przekazywany przez tę maszynę do innego hosta), `output` (pakiet wygenerowany przez tę maszynę) i `postrouting` (zaraz przed wysłaniem na kartę sieciową). Nazwy `PREROUTING` / `INPUT` / `FORWARD` / `OUTPUT` / `POSTROUTING` w `iptables` to te same pięć haków, tylko pisane wielkimi literami z przyzwyczajenia starszej składni. Reguły NAT (temat kolejnej strony) zawsze siedzą w `prerouting` albo `postrouting`, bo tylko tam ma sens zmiana adresu — przed albo po podjęciu decyzji o trasowaniu.

Zasada pierwszego trafienia. Jądro sprawdza reguły w łańcuchu w kolejności, w jakiej zostały dodane, i **zatrzymuje się na pierwszej, która definitywnie decyduje o pakiecie** (`accept`, `drop`, `reject` — w terminologii `iptables`: `ACCEPT`, `DROP`, `REJECT`). Reguła numer dziesięć nigdy nie zostanie sprawdzona, jeśli reguła numer trzy już odrzuciła pakiet. To jest powód, dla którego kolejność dodawania reguł (`-A` na koniec, `-I` na początek) ma bezpośredni wpływ na to, co faktycznie się dzieje — dwie identyczne reguły w innej kolejności dają inny efekt końcowy.

Pojęcie: translacja adresów (NAT). NAT (Network Address Translation) to podmiana adresu IP (i czasem portu) w pakiecie w trakcie jego przechodzenia przez router czy firewall — tak, żeby dla jednej strony połączenia wyglądało to inaczej niż dla drugiej. Typowy powód: maszyny w sieci prywatnej (`10.20.10.0/24`) mają adresy, które nie istnieją w internecie, więc router podmienia ich adres źródłowy na swój publiczny adres, zanim pakiet wyjdzie na zewnątrz. Pełny podział na DNAT, SNAT i maskaradę oraz konkretne reguły — następna strona rozdziału, *Przekierowanie ruchu między interfejsami i portami*.

`iptables-nft`**: nakładka, nie osobny mechanizm.** Od RHEL 8 (i od Debiana 10/Buster) binarka `/usr/sbin/iptables` **nie** rozmawia z osobnym, starym modułem jądra. To jest `iptables-nft` — program, który przyjmuje starą składnię `iptables`, tłumaczy ją na reguły nftables i zapisuje je do tych samych tabel, które widzi `nft list ruleset`. Efekt: `iptables -L` i `nft list ruleset` pokazują **ten sam stan systemu**, tylko w dwóch różnych językach — i jednocześnie pułapka opisana w *Częste błędy*: mieszanie obu składni w tym samym skrypcie administracyjnym prowadzi do reguł, które jedna z komend pokazuje inaczej niż druga, choć obie czytają to samo jądro.

Krok po kroku

1. Zobacz, co naprawdę filtruje ruch, zanim zaczniesz cokolwiek zmieniać — z dwóch stron naraz.

```
sudo nft list ruleset | head -30
sudo iptables -L -n -v | head -8
```

```
Chain INPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
   12   720 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0
ctstate RELATED,ESTABLISHED
```

Szukaj tabeli `firewalld` w pierwszym wyniku — jeśli usługa działa, prawie cała konfiguracja będzie tam. Ten sam stan jądra widziany przez `iptables` wygląda inaczej — grupuje reguły `firewalld` mniej czytelnie niż natywne `nft list`.

2. Dodaj tymczasową regułę testową przez `nft` i sprawdź, czy `iptables` w ogóle ją widzi.

```
sudo nft add table inet test
sudo nft add chain inet test input { type filter hook input priority 0 \; }
sudo nft add rule inet test input tcp dport 12345 accept
sudo iptables -L -n | grep 12345 || echo "brak — iptables-nft czyta tylko tabele o
standardowych nazwach"
sudo nft delete table inet test
```

Reguła istnieje i działa, ale `iptables -L` jej nie pokaże — `iptables-nft` czyta tylko tabele o standardowych nazwach (`filter`, `nat`, `mangle`, `raw`, `security`), nie każdą tabelę, jaką stworzysz w `nft`. Ostatnia linia sprząta po teście.

3. Sprawdź przekazywanie pakietów, zanim zabierzesz się za NAT.

```
sysctl net.ipv4.ip_forward
```

```
net.ipv4.ip_forward = 0
```

Zero — jeśli następny krok to konfiguracja przekazywania między interfejsami, ten przełącznik trzeba włączyć najpierw. Szczegóły: następna strona.

Częste błędy

Reguła dodana ręcznie znika po `firewall-cmd --reload`.

```
sudo iptables -A INPUT -p tcp --dport 8443 -j ACCEPT
sudo firewall-cmd --reload
sudo iptables -S | grep 8443
```

Drugie polecenie nic nie wypisuje. `firewall-cmd --reload` każe `firewalld` odtworzyć całą konfigurację nftables **od zera**, na podstawie własnych, trwałych ustawień (`--permanent`). Reguła dopisana bezpośrednio do jądra przez `iptables -A`, bez wiedzy `firewalld`, nie jest częścią tej konfiguracji i przepada. Naprawa: albo trwałe otwarcie portu przez `firewall-cmd --add-port=8443/tcp --permanent`, albo — jeśli `firewalld` faktycznie nie jest potrzebny — jego całkowite wyłączenie i przejście na własne pliki `.nft` wczytywane przy starcie.

Mieszanie `-I` (insert) z `-A` (append) i zdziwienie kolejnością.

```
sudo iptables -A INPUT -j DROP
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
sudo iptables -L INPUT -n --line-numbers
```

```
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    DROP       all  --  0.0.0.0/0            0.0.0.0/0
2    ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:22
```

Reguła 2 nigdy nie zostanie sprawdzona — reguła 1 odrzuca już wszystko, wcześniej. SSH przestaje działać, mimo że reguła zezwalająca na port 22 fizycznie jest w łańcuchu. Naprawa: `iptables -I INPUT 1 -p tcp --dport 22 -j ACCEPT`, żeby wstawić regułę zezwalającą **przed** ogólnym `DROP`, albo usunięcie i ponowne dodanie w poprawnej kolejności.

`nft add chain` bez klamry z hakiem — łańcuch nic nie robi.

```
sudo nft add table inet filter
sudo nft add chain inet filter input
sudo nft add rule inet filter input tcp dport 22 accept
```

Wszystko wykonuje się bez błędu, ale port 22 i tak jest zablokowany, jeśli coś innego blokuje ruch. Przyczyna: łańcuch stworzony bez `{ type filter hook input priority 0; }` to łańcuch **regularny**, niepodłączony do żadnego haka jądra — istnieje, ale jądro nigdy go nie wywoła. Do niego trzeba by skoczyć jawnie (`jump`) z innego, podłączonego łańcucha. Naprawa: podać typ i hak przy tworzeniu, jak w Fiszce.

Próba usunięcia reguły `iptables -D` ze złym formatem i komunikat o braku reguły.

```
sudo iptables -D INPUT -p tcp --dport 22 -j ACCEPT -m comment --comment "ssh"
```

```
iptables: Bad rule (does a matching rule exist in that chain?)
```

Reguła w łańcuchu ma inny zestaw modułów (`-m`) niż ta, którą próbujesz usunąć — `iptables -D` wymaga dopasowania **każdego** parametru, nie tylko efektu końcowego. Naprawa: `iptables -L INPUT --line-numbers -n`, znalezienie numeru reguły wzrokiem, i `iptables -D INPUT <numer>`.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz lokalnie, na własnej maszynie — żadne polecenie nie dotyka Twojej realnej konfiguracji sieci ani produkcyjnego firewalla, jeśli sprzątniesz po sobie, jak pokazano w odpowiedziach.

Ćwiczenie 1 — własna tabela i reguła bez ryzyka dla systemu

```
sudo nft add table inet cwiczenie
sudo nft add chain inet cwiczenie input { type filter hook input priority 0 \; policy accept \; }
sudo nft add rule inet cwiczenie input tcp dport 9999 counter drop
sudo nft list table inet cwiczenie
```

Sprawdź licznik pakietów po komendzie `nc -zv 127.0.0.1 9999`. Jak zmienia się wynik `nft list table inet cwiczenie` po kilku takich próbach?

Odpowiedź

Licznik w regule `counter drop` rośnie o jeden pakiet (i o odpowiednią liczbę bajtów) przy każdej próbie połączenia — nawet jeśli port jest tylko odrzucany, a nie obsługiwany przez żadną usługę. To dowód, że reguła w tabeli `inet cwiczenie` faktycznie przechwytuje ruch przed jakąkolwiek aplikacją. Po ćwiczeniu:

```
sudo nft delete table inet cwiczenie
```

Ćwiczenie 2 — kolejność reguł na żywym przykładzie

```
sudo nft add table inet porzadek
sudo nft add chain inet porzadek input { type filter hook input priority 0 \; policy accept \;
}
sudo nft add rule inet porzadek input tcp dport 7000 drop
sudo nft add rule inet porzadek input tcp dport 7000 accept
sudo nft list table inet porzadek
```

Uruchom w jednym terminalu `nc -l 127.0.0.1 7000`, a w drugim `nc -zv 127.0.0.1 7000`. Połączenie przejdzie czy zostanie odrzucone — i dlaczego, mimo że druga reguła mówi `accept`?

Odpowiedź

Połączenie zostanie odrzucone. Reguła `drop` jest pierwsza w łańcuchu i to ona decyduje o pakiecie — reguła `accept` nigdy nie zostaje sprawdzona, bo jądro zatrzymuje się na pierwszym dopasowaniu. Zamiana kolejności zmienia wynik na przeciwny. Sprzątanie:

```
sudo nft delete table inet porzadek
```

Pytania kontrolne

1. Dlaczego `iptables -L` i `nft list ruleset` na RHEL 9 pokazują ten sam stan systemu, mimo że to dwie różne komendy?

Odpowiedź

Bo od RHEL 8 `/usr/sbin/iptables` to `iptables-nft` — nakładka, która tłumaczy starą składnię na reguły nftables i zapisuje je do tych samych tabel jądra, które czyta `nft`. Nie ma dwóch niezależnych mechanizmów filtrowania działających równolegle; jest jeden mechanizm (netfilter/nftables) i dwa języki do jego opisywania.

2. Administrator dodał `iptables -A INPUT -p tcp --dport 3000 -j ACCEPT`, ale nie wywołał żadnej komendy zapisu. Co się stanie z tą regułą po restarcie serwera?

Odpowiedź

Zniknie. Reguły dodane samym `-A` żyją tylko w pamięci jądra, w bieżącej sesji nftables — nie są automatycznie zapisywane na dysk. Trzeba je jawnie zapisać (`iptables-save > plik` i wczytać przy starcie) albo, na systemie z `firewalld`, dodać przez `firewall-cmd --add-port ... --permanent`, co `firewalld` sam utrwala.

3. Masz dwie reguły w łańcuchu `INPUT`: pierwsza blokuje cały ruch z podsieci `10.20.20.0/24`, druga zezwala na SSH z hosta `10.20.20.20`. Ruch z `10.20.20.20` na port 22 przejdzie?

Odpowiedź

Nie przejdzie. `10.20.20.20` należy do `10.20.20.0/24`, więc pierwsza reguła dopasuje się i zablokuje pakiet, zanim jądro dojdzie do drugiej reguły — zasada pierwszego trafienia. Żeby wyjątek zadziałał, reguła zezwalająca musi być **przed** ogólną blokadą, nie za nią.

4. Czym różni się łańcuch podłączony pod hak od łańcucha „regularnego" w nftables?

Odpowiedź

Łańcuch podłączony pod hak (`{ type filter hook input priority 0; }`) jest automatycznie wywoływany przez jądro w konkretnym punkcie przepływu pakietu. Łańcuch regularny, stworzony bez tej klamry, jest tylko nazwaną listą reguł, do której trzeba doskoczyć jawnie (`jump`) z innego łańcucha — jądro samo z siebie nigdy go nie sprawdzi.

Powiązane: Przekierowanie ruchu między interfejsami i portami · firewalld: strefy, porty, usługi, rich rules · DNS z perspektywy administratora: dig, host, resolvectl **Źródła:** `man 8 nft` · `man 8 iptables` · `man 8 iptables-translate` · `man 5 nftables.conf`

Revision #16
Created 2026-09-17 21:43:20 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:03 UTC by Tomasz Konieczny