Diagnostyka: ping, traceroute, mtr, tcpdump, curl

Poziom: średni **Dotyczy: RHEL/Rocky 9–10, różnice Debian/Ubuntu i SLES zaznaczone**

Czas: ~22 min

„Nie działa” to nie jest punkt startu diagnozy sieciowej — to punkt końcowy sześciu możliwych przyczyn. Ta strona porządkuje narzędzia w kolejność, w jakiej naprawdę warto je odpalać: od łącza fizycznego, przez adres i trasę, po port, aplikację i na końcu certyfikat TLS.

Skróty użyte na tej stronie (15)

Skrót	Co znaczy
B2B	business to business: sprzedaż i systemy skierowane do innych firm, nie do klientów indywidualnych
DNS	Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP
HTTP	Hypertext Transfer Protocol: protokół stron WWW
HTTPS	HTTP zabezpieczony szyfrowaniem TLS
ICMP	Internet Control Message Protocol: protokół diagnostyczny sieci, którym działa między innymi ping
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
MTU	Maximum Transmission Unit: największy pakiet, jaki łącze przepuści bez dzielenia, zwykle 1500 bajtów w Ethernecie, a przy PPPoE 1492
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SLES	SUSE Linux Enterprise Server: płatna dystrybucja Linuksa firmy SUSE, konkurent Rocky/RHEL
SNI	Server Name Indication: pole w pierwszym komunikacie TLS, w którym klient podaje nazwę serwera, z którym chce rozmawiać, co pozwala HAProxy wybrać backend bez rozszyfrowywania ruchu
TCP	Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia

Skrót	Co znaczy
TLS	Transport Layer Security: szyfrowanie połączeń, np. w HTTPS
TTL	Time To Live: czas życia rekordu lub danych
UDP	User Datagram Protocol: szybki protokół sieciowy bez potwierdzeń
VPN	Virtual Private Network: szyfrowany tunel do sieci firmowej

Fiszka

TYLKO ODCZYT

Sprawdzenie, czy host odpowiada na ICMP: ping -c 4

```
ping -c 4 NazwaSerwera
```

`-c 4` ogranicza liczbę pakietów do czterech i kończy polecenie samo — bez tej flagi `ping` pinguje w nieskończoność, aż przerwiemy `Ctrl+C`. To pierwszy, najtańszy test: czy pakiet w ogóle dociera i wraca.

Przykład:

```
64 bytes from app01.zielona.example (10.20.10.41): icmp_seq=1 ttl=64 time=0.312 ms
64 bytes from app01.zielona.example (10.20.10.41): icmp_seq=2 ttl=64 time=0.298 ms

--- app01.zielona.example ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3054ms
```

ICMP = Internet Control Message Protocol: protokół diagnostyczny sieci, którym działa między innymi ping · **ping** = Packet InterNet Groper: sprawdź, czy host odpowiada w sieci i jak długo trwa odpowiedź

TYLKO ODCZYT

Sprawdzenie MTU bez fragmentacji: ping -M do -s 1472

```
ping -M do -s 1472 -c 2 AdresIP
```

`-M do` zabrania jądru fragmentować pakiet, `-s 1472` ustawia rozmiar danych tak, żeby cały pakiet z nagłówkami wyszedł dokładnie na 1500 bajtów — standardowe MTU Ethernetu. Jeśli to przechodzi, a większy rozmiar nie, mamy namierzone gdzieś po drodze urządzenie z mniejszym MTU (klasyczny problem tuneli VPN).

Przykład:

```
ping: local error: message too long, mtu=1400
```

MTU = Maximum Transmission Unit: największy pakiet, jaki łącze przepuści bez dzielenia, zwykle 1500 bajtów w Ethernecie, a przy PPPoE 1492 · **VPN** = Virtual Private Network: szyfrowany tunel do sieci firmowej · **ping** = Packet InterNet Groper: sprawdź, czy host odpowiada w sieci i jak długo trwa odpowiedź

TYLKO ODCZYT

Wymuszenie konkretnego interfejsu źródłowego: ping -I

```
ping -I NazwaInterfejsu AdresIP -c 2
```

Na maszynie z kilkoma kartami sieciowymi `ping` domyślnie wybiera interfejs wg tabeli routingu — `-I` wymusza konkretny, przydatne przy diagnozowaniu, czy problem dotyczy jednej konkretnej karty.

ping = Packet InterNet Groper: sprawdź, czy host odpowiada w sieci i jak długo trwa odpowiedź

TYLKO ODCZYT

Sprawdzenie trasy pakietu, przystanek po przystanku: traceroute

```
traceroute NazwaSerwera
```

Pokazuje każdy router (hop) po drodze do celu, mierząc czas odpowiedzi na każdym z nich. Sięgamy po to, gdy `ping` do celu nie odpowiada i chcemy wiedzieć, **gdzie** dokładnie po drodze pakiet ginie, zamiast tylko wiedzieć, że gdzieś ginie.

Przykład:

```
1  fw01.zielona.example (10.20.20.1)  0.412 ms
2  10.20.10.2 (10.20.10.2)  1.203 ms
3  * * *
4  app01.zielona.example (10.20.10.41)  3.812 ms
```

TYLKO ODCZYT

Sprawdzenie trasy przez konkretny port TCP: traceroute -T -p 443

```
sudo traceroute -T -p NumerPortu NazwaSerwera
```

Domyślnie `traceroute` wysyła pakiety UDP albo ICMP, które firewalle często blokują inaczej niż ruch TCP. `-T` przełącza na TCP, `-p 443` wybiera port — trasa zmierzona w ten sposób odzwierciedla to, co faktycznie napotka prawdziwe połączenie HTTPS.

TCP = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia · **UDP** = User Datagram Protocol: szybki protokół sieciowy bez potwierdzeń · **ICMP** = Internet Control Message Protocol: protokół diagnostyczny sieci, którym działa między innymi ping · **HTTPS** = HTTP zabezpieczony szyfrowaniem TLS

TYLKO ODCZYT

Połączenie traceroute i ping w jeden raport: mtr -r -c 10

```
mtr -r -c 10 NazwaSerwera
```

`mtr` łączy `ping` i `traceroute` w jedno: wysyła serię pakietów do każdego hopa po drodze i pokazuje procent utraconych pakietów oraz czasy dla **każdego** z nich, nie tylko dla celu końcowego. `-r` przełącza z trybu interaktywnego na jednorazowy raport (do logów, do wklejenia w zgłoszenie), `-c 10` ustala liczbę cykli.

Przykład:

```
Host                    Loss%   Snt   Last   Avg  Best  Wrst StDev
1. fw01.zielona.example  0.0%    10    0.4   0.5   0.3   0.9   0.1
2. 10.20.10.2            0.0%    10    1.2   1.4   1.1   2.0   0.3
3. app01.zielona.example 0.0%    10    3.8   3.9   3.6   4.5   0.2
```

Utrata pakietów rosnąca na konkretnym hopie i utrzymująca się na wszystkich kolejnych wskazuje winowajcę — utrata tylko na jednym hopie, która **nie** propaguje się dalej, jest zwykle nieszkodliwa (router ten po prostu ogranicza priorytet odpowiedzi ICMP na siebie).

ICMP = Internet Control Message Protocol: protokół diagnostyczny sieci, którym działa między innymi ping · **mtr** = mtr: Linux, łączy ping i traceroute i na bieżąco pokazuje straty i opóźnienia na każdym przeskoku do wskazanego adresu

TYLKO ODCZYT

Sprawdzenie trasy bez uprawnień roota: tracepath

```
tracepath NazwaSerwera
```

Robi w przybliżeniu to samo co `traceroute`, ale nie wymaga uprawnień administratora ani specjalnych capability — dobre rozwiązanie awaryjne na koncie bez sudo.

□□ TYLKO ODCZYT

Sprawdzenie, czy port jest otwarty, bez wysyłania danych: nc -zv host port

```
nc -zv NazwaSerwera NumerPortu
```

`-z` (zero-I/O) każe `nc` tylko sprawdzić, czy da się nawiązać połączenie, bez wysyłania i odbierania żadnych danych, `-v` włącza komunikat o wyniku. Szybszy i czystszy test portu niż otwieranie pełnej sesji aplikacyjnej.

Przykład:

```
Connection to app01.zielona.example 443 port [tcp/https] succeeded!
```

□□ TYLKO ODCZYT

Nasłuchiwanie na porcie testowym: nc -l -p 8080

```
nc -l -p NumerPortu
```

Uruchamia `nc` jako prostego nasłuchującego — przydatne jako cel testowy do sprawdzenia, czy inny host w ogóle dotrze na dany port, zanim postawisz tam prawdziwą usługę. Na niektórych dystrybucjach (`nc` z pakietu `nmap-ncat` na RHEL) flaga to `-l` bez `-p`, port podaje się jako argument końcowy — sprawdź `nc -h` na konkretnym systemie.

□□ **RHEL** = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa

□□ TYLKO ODCZYT

Sprawdzenie portu bez żadnego dodatkowego narzędzia: timeout bash /dev/tcp

```
timeout 3 bash -c '</dev/tcp/NazwaSerwera/NumerPortu' && echo OTWARTY || echo ZAMKNIETY
```

Bash ma wbudowaną obsługę gniazd TCP przez pseudo-plik `/dev/tcp/host/port` — otwarcie go do odczytu/zapisu jest równoznaczne z próbą połączenia. Działa wszędzie, gdzie jest `bash`, nawet gdy `nc`, `telnet` czy `nmap` nie są zainstalowane. `timeout 3` chroni przed wisieniem w nieskończoność, gdy port jest filtrowany, a nie tylko zamknięty.

Przykład:

```
OTWARTY
```

TCP = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia · **timeout** = uruchom polecenie i przerwij je, jeśli nie skończy się w zadanym czasie

TYLKO ODCZYT

Sprawdzenie pełnego przebiegu połączenia HTTP: curl -v

```
curl -v https://NazwaSerwera/
```

Pokazuje każdy krok: rozwiązanie nazwy, połączenie TCP, uzgadnianie TLS z certyfikatem serwera, wysłane nagłówki żądania i otrzymane nagłówki odpowiedzi. Pierwsze narzędzie, po które sięgamy, gdy port jest otwarty (`nc -zv` to potwierdził), ale aplikacja i tak się nie łączy poprawnie.

Przykład:

```
* Connected to app01.zielona.example (10.20.10.41) port 443
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
> GET / HTTP/1.1
< HTTP/1.1 200 OK
```

HTTP = Hypertext Transfer Protocol: protokół stron WWW · **TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia · **TLS** = Transport Layer Security: szyfrowanie połączeń, np. w HTTPS · **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

TYLKO ODCZYT

Wypisanie samych nagłówków odpowiedzi: curl -I

```
curl -I https://NazwaSerwera/
```

Wysyła żądanie `HEAD` zamiast `GET` (serwer nie odsyła treści) i pokazuje tylko nagłówki odpowiedzi. Szybki sposób sprawdzenia kodu odpowiedzi i typu serwera bez ściągania całej strony.

Przykład:

```
HTTP/1.1 200 OK
Server: nginx/1.24.0
Content-Type: text/html
```

GET = Metoda HTTP, którą prosimy serwer o odczyt zasobu bez zmieniania czegokolwiek, np. o wartość wilgotności z czujnika · **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

TYLKO ODCZYT

Pomiar czasu odpowiedzi: curl -w '%{time_total}'

```
curl -o /dev/null -s -w '%{time_total}\n' https://NazwaSerwera/
```

`-w` pozwala wypisać dowolne zmierzone wartości zamiast (albo obok) treści odpowiedzi — tu: całkowity czas żądania w sekundach. `-o /dev/null` odrzuca treść, `-s` wyłącza pasek postępu — zostaje sama liczba, gotowa do wrzucenia do skryptu monitorującego.

Przykład:

```
0.087432
```

curl = client URL: pobieranie i wysyłanie przez HTTP z terminala

TYLKO ODCZYT

Wymuszenie adresu bez zmiany DNS: curl --resolve

```
curl --resolve NazwaSerwera:NumerPortu:AdresIP https://NazwaSerwera/
```

Podmienia rozwiązywanie nazwy tylko dla tego jednego wywołania `curl`, bez dotykania `/etc/hosts` czy DNS. Nieocenione przy sprawdzaniu nowego serwera pod starą nazwą domenową — np. przed przełączeniem produkcji na nowy adres.

DNS = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP ·**curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

TYLKO ODCZYT

Pominięcie weryfikacji certyfikatu, tylko do diagnozy: curl -k

```
curl -k -v https://AdresIP/
```

`-k` (`--insecure`) wyłącza sprawdzanie ważności certyfikatu TLS — pozwala sprawdzić, czy usługa HTTPS w ogóle odpowiada, gdy certyfikat jest samopodpisany, wygasły albo wystawiony na inną nazwę. Tej flagi **nie używamy** nigdzie poza jednorazową diagnozą — w skrypcie czy monitoringu produkcyjnym `-k` cicho wyłącza całą ochronę przed atakiem pośrednika (MITM), a przyszłe „to działa" nic już nie mówi o tym, czy certyfikat jest poprawny.

TLS = Transport Layer Security: szyfrowanie połączeń, np. w HTTPS · **HTTPS** = HTTP zabezpieczony szyfrowaniem TLS · **curl** = client URL: pobieranie i wysyłanie przez HTTP z terminala

TYLKO ODCZYT

Sprawdzenie pełnego szczegółu uzgadniania TLS: openssl s_client -connect host:port -servername

```
openssl s_client -connect NazwaSerwera:NumerPortu -servername NazwaSerwera </dev/null
```

Pokazuje cały łańcuch certyfikatów, użyty protokół i szyfr, oraz dokładny powód odrzucenia, jeśli uzgadnianie TLS się nie powiedzie — dużo więcej szczegółu niż `curl -v`. `-servername` jest konieczne przy serwerach obsługujących wiele domen na jednym adresie IP (SNI) — bez niego serwer może zwrócić certyfikat dla zupełnie innej nazwy.

Przykład:

```
Certificate chain
 0 s:CN = app01.zielona.example
   i:CN = Zielona Internal CA
Verify return code: 0 (ok)
```

TLS = Transport Layer Security: szyfrowanie połączeń, np. w HTTPS · **IP** = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **SNI** = Server Name Indication: pole w pierwszym komunikacie TLS, w którym klient podaje nazwę serwera, z którym chce rozmawiać, co pozwala HAProxy wybrać backend bez rozszyfrowywania ruchu · **openssl** = zestaw narzędzi do certyfikatów i szyfrowania

TYLKO ODCZYT

Podejrzenie ruchu na żywo: tcpdump -i any -nn port 443

```
sudo tcpdump -i any -nn port NumerPortu
```

`-i any` nasłuchuje na wszystkich interfejsach naraz, `-nn` wyłącza rozwiązywanie nazw hostów i portów (szybciej, i bez zależności od tego samego DNS, który akurat diagnozujemy). Ostateczny dowód: pakiety albo idą, albo nie idą — koniec zgadywania na podstawie konfiguracji.

Przykład:

```
12:41:02.881233 IP 10.20.20.20.51422 > 10.20.10.41.443: Flags [S]
12:41:02.881501 IP 10.20.10.41.443 > 10.20.20.20.51422: Flags [S.]
```

DNS = Domain Name System: książka telefoniczna internetu, zamienia nazwy na adresy IP · **tcpdump** = podsłuch ruchu sieciowego na interfejsie

TYLKO ODCZYT

Zapisanie przechwytu do pliku zamiast ekranu: tcpdump -w plik.pcap

```
sudo tcpdump -i NazwaInterfejsu -w /tmp/przechwyt.pcap port NumerPortu -c 500
```

Na produkcji nie wypisujemy przechwyconego ruchu na ekran — zapisujemy go do pliku, z limitem liczby pakietów (`-c 500`) albo czasu, i analizujemy później albo przenosimy do Wiresharka. Pisanie na ekran w czasie rzeczywistym na obciążonym łączu gubi pakiety i zapycha terminal bezużytecznie.

tcpdump = podsłuch ruchu sieciowego na interfejsie

TYLKO ODCZYT

Odczytanie zapisanego przechwytu: tcpdump -r

```
tcpdump -r /tmp/przechwyt.pcap -nn 'tcp[tcpflags] & tcp-syn != 0'
```

Czyta plik zapisany wcześniej przez `-w`, tutaj z dodatkowym filtrem pokazującym tylko pakiety z flagą SYN — czyli same próby nawiązania połączenia, bez reszty sesji.

tcpdump = podsłuch ruchu sieciowego na interfejsie

◫◫ TYLKO ODCZYT

Wypisanie połączeń TCP z nazwą procesu: ss -tp

```
sudo ss -tp
```

Pokazuje aktywne połączenia TCP (`-t`) razem z procesem, który je trzyma (`-p`) — różni się od `ss -tunlp` (Fiszka poprzedniej strony) tym, że pokazuje **nawiązane połączenia**, nie tylko gniazda nasłuchujące. Dobre pytanie „z kim ta aplikacja aktualnie rozmawia".

Przykład:

```
State   Recv-Q Send-Q  Local Address:Port   Peer Address:Port  Process
ESTAB   0      0          10.20.10.41:443 198.51.100.44:51022
users:(("nginx",pid=1234,fd=9))
```

◫◫ **TCP** = Transmission Control Protocol: protokół sieciowy z potwierdzaniem dostarczenia ·**ss** = socket statistics: kto nasłuchuje i jakie są połączenia

◫◫ TYLKO ODCZYT

Wszystkie flagi i komendy

Warstwa diagnozy	Komenda
Łącze / adres	`ping -c`, `ping -I`
MTU	`ping -M do -s`
Trasa	`traceroute`, `traceroute -T -p`, `mtr -r -c`, `tracepath`
Port	`nc -zv`, `nc -l`, `timeout ... /dev/tcp/...`, `ss -tp`
Aplikacja / HTTP	`curl -v`, `curl -I`, `curl -w`, `curl --resolve`
TLS	`curl -k`, `openssl s_client -connect ... -servername`
Przechwytywanie	`tcpdump -i`, `tcpdump -w`, `tcpdump -r`

Uważaj

- Brak odpowiedzi na `ping` **nie znaczy**, że host nie działa — mnóstwo firewalli blokuje wyłącznie ICMP, a wszystkie porty aplikacyjne są otwarte.
- `curl -k` wolno używać tylko do jednorazowej diagnozy — nigdy w kodzie ani w monitoringu produkcyjnym.

- `tcpdump` bez `-w` i bez filtra na zajętym łączu potrafi zalać terminal tysiącami linii na sekundę — zawsze dopisz filtr portu/hosta.
- Starsze materiały i skrypty sprawdzają nasłuchujące porty przez `netstat -tunlp` albo `lsof -nP -iTCP -sTCP:LISTEN` — działają nadal, jeśli pakiet `net-tools` / `lsof` jest zainstalowany, ale `ss -tunlp` (poprzednia strona rozdziału) jest szybszy i domyślnie obecny na RHEL 8+.

ICMP = Internet Control Message Protocol: protokół diagnostyczny sieci, którym działa między innymi ping · **RHEL** = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa

Z życia wzięte

Z ŻYCIA WZIĘTE

Piątek, 16:00. Sklep B2B na my01 przestaje odpowiadać dla klientów zewnętrznych, a wewnątrz biura działa bez zarzutu. Tomek przechodzi po kolei: ping do my01 działa, traceroute pokazuje trasę bez zmian, ale port 443 z zewnątrz jest zamknięty, co szybko prowadzi do reguły na fw01 zmienionej tego samego dnia.

Po co to istnieje

Pojęcie: ICMP i dlaczego `ping` bywa blokowany. ICMP (Internet Control Message Protocol) to protokół do komunikatów kontrolnych i diagnostycznych w sieci IP — `ping` wysyła pakiet ICMP typu *echo request* i czeka na *echo reply*. ICMP nie jest jednak protokołem transportowym takim jak TCP czy UDP — nie niesie żadnego ruchu aplikacyjnego, więc administratorzy firewalli często blokują go selektywnie, uznając, że odpowiadanie na `ping` z zewnątrz to tylko ułatwienie rozpoznania sieci atakującemu, bez żadnej korzyści biznesowej. Efekt: host bez odpowiedzi na `ping` bywa całkowicie sprawny i osiągalny na każdym porcie aplikacyjnym — brak echa ICMP mówi tylko o polityce firewalla wobec ICMP, nic więcej.

Ta strona zbiera narzędzia diagnostyczne w jedną, uporządkowaną **procedurę**, a nie w przypadkową listę komend do wypróbowania po kolei, aż coś zadziała. Idea jest prosta: sieć to stos warstw, a awaria na wyższej warstwie często wygląda identycznie jak awaria na niższej, dopóki nie sprawdzisz ich po kolei, od dołu.

Analogia

Wyobraź sobie **reklamację niedziałającej dostawy paczki**, rozpatrywaną krok po kroku, a nie na pierwszy rzut oka.

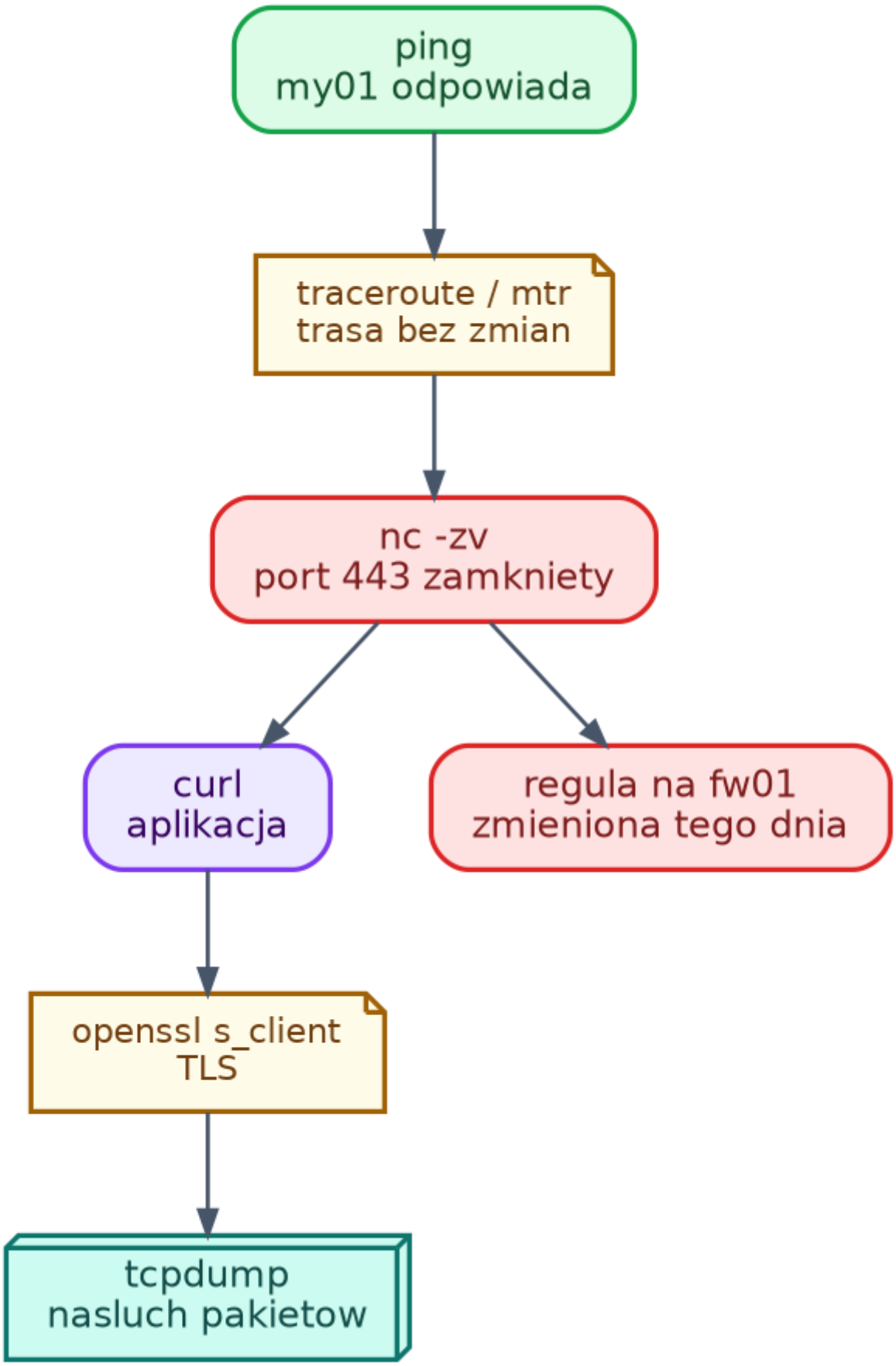
Najpierw sprawdzasz, czy kurier w ogóle wyjechał z magazynu (łącze fizyczne, `ping`). Potem, czy adres na paczce jest poprawny (adres IP, trasa routingu). Potem, czy paczka dotarła pod właściwe drzwi, a nie do sąsiada trzy numery dalej (port aplikacyjny, `nc -zv`). Dopiero gdy paczka fizycznie leży pod drzwiami, sprawdzasz, czy odbiorca w ogóle ją otworzył i czy zawartość jest tym, czego się spodziewał (odpowiedź aplikacji, `curl -v`). A na samym końcu, jeśli przesyłka wymagała podpisu potwierdzającego tożsamość obu stron, sprawdzasz, czy ten podpis w ogóle się zgadza (certyfikat TLS, `openssl s_client`).

Reklamacja rozpatrywana od końca — „paczki nie ma, więc na pewno magazyn spalił się do cna” — traci czas na katastroficzne teorie, zanim ktokolwiek sprawdzi, czy kurier w ogóle miał otwartą bramę wyjazdową. Dokładnie to samo dzieje się, gdy ktoś od razu otwiera Wireshark, zanim sprawdzi, czy port jest w ogóle otwarty.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: trasa pakietu i TTL.** Każdy pakiet IP niesie pole TTL (Time To Live) — licznik, który każdy router po drodze zmniejsza o jeden, a gdy licznik spadnie do zera, pakiet jest porzucany i router odsyła do nadawcy komunikat ICMP *Time Exceeded*. `traceroute` (i `mtr`) wykorzystują to mechanicznie: wysyłają pierwszy pakiet z TTL=1 (ginie na pierwszym routerze, który odsyła komunikat i tym samym zdradza swój adres), potem z TTL=2 (ginie na drugim), i tak dalej, aż pakiet dotrze do celu. Każdy „przystanek” w wyniku `traceroute` to jeden router, który akurat porzucił pakiet z odpowiednim TTL i się przy tym ujawnił.


ping
my01 odpowiada
traceroute / mtr
trasa bez zmian
nc -zv
port 443 zamkniety
curl
aplikacja
regula na fw01
zmieniona tego dnia
openssl s_client
TLS
tcpdump
nasluch pakietow

Diagnostyka sieci idzie warstwa po warstwie, od pinga po podsłuch pakietów, aż znajdzie winną regułę. (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: przechwytywanie pakietów.** Przechwytywanie pakietów (packet capture) to podgląd surowego ruchu sieciowego przechodzącego przez interfejs, zanim zostanie on zinterpretowany przez którąkolwiek warstwę aplikacyjną. Karta sieciowa w trybie nasłuchu (albo, we współczesnym Linuksie, mechanizm `AF_PACKET`/BPF w jądrze) kopiuje każdy pakiet do narzędzia takiego jak `tcpdump`, niezależnie od tego, czy jest on adresowany do tej maszyny, do przekazania dalej, czy odrzucany przez firewall. To jest ostateczne źródło prawdy w diagnostyce sieciowej: konfiguracja może kłamać (reguła zapisana, ale nieaktywna po błędnym `--reload`), ale pakiet na przewodzie albo istnieje, albo nie.

> **Pojęcie: uzgadnianie TLS.** Zanim jakiekolwiek dane HTTPS popłyną między klientem a serwerem, obie strony przechodzą przez uzgadnianie TLS (TLS handshake): klient i serwer negocjują wersję protokołu i zestaw szyfrów, serwer przedstawia swój certyfikat (z nazwą, dla której został wystawiony, i podpisem urzędu certyfikacji), a klient go weryfikuje, zanim zgodzi się na dalszą wymianę. Awaria na tym etapie wygląda z zewnątrz identycznie jak zwykłe „strona się nie ładuje”, ale przyczyna jest zupełnie inna niż zablokowany port — port jest otwarty, TCP się łączy, problem leży w samym certyfikacie albo w negocjacji protokołu. `curl -v` i `openssl s_client` to dwa narzędzia rozkładające ten etap na czynniki pierwsze, drugie ze znacznie większą ilością szczegółu.

Procedura diagnozowania od najniższej warstwy. Skuteczna diagnoza sieciowa idzie w ustalonej kolejności, a nie od najbardziej efektownego narzędzia: **łącze** (czy interfejs w ogóle ma sygnał i adres — `ip link`, `ip addr`) → **adres i trasa** (czy pakiet w ogóle znajdzie drogę do celu — `ping`, `traceroute`, `mtr`) → **port** (czy usługa na docelowym hoście w ogóle nasłuchuje na oczekiwanym porcie — `nc -zv`, `ss -tp`) → **aplikacja** (czy usługa na tym porcie odpowiada sensownie na żądanie w swoim protokole — `curl -v`) → **TLS** (czy warstwa szyfrowania i certyfikatu jest poprawna — `openssl s_client`). Pominięcie kroku i przeskoczenie od razu do `tcpdump` albo do analizy logów aplikacji jest najczęstszą przyczyną przedłużającej się diagnozy — administrator analizuje logi aplikacji przez pół godziny, zanim odkryje, że port był zwyczajnie zamknięty przez firewall, co `nc -zv` pokazałoby w sekundę.

Krok po kroku

1. Łącze i adres — czy interfejs w ogóle działa.

```
ip -brief link show
ip -brief addr show
```

```
ens192           UP             10.20.20.20/24
```

2. Trasa — czy pakiet dociera do celu.

```
ping -c 4 app01.zielona.example
```

Brak odpowiedzi tutaj **nie kończy** diagnozy — przechodzisz dalej, pamiętając, że to może być tylko blokada ICMP.

3. Port — czy usługa nasłuchuje.

```
nc -zv app01.zielona.example 443
```

```
Connection to app01.zielona.example 443 port [tcp/https] succeeded!
```

Port otwarty — problem nie leży w firewallu ani w tym, czy proces w ogóle działa.

4. Aplikacja — czy odpowiada sensownie.

```
curl -v https://app01.zielona.example/ 2>&1 | tail -15
```

```
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
< HTTP/1.1 200 OK
```

5. Potwierdzenie na poziomie pakietów, gdy coś nadal nie zgadza się z opisem z kroków 1–4.

```
sudo tcpdump -i any -nn host app01.zielona.example and port 443 -c 20
```

Jeśli wszystkie cztery poprzednie kroki wyglądają poprawnie, a problem nadal występuje z poziomu konkretnej aplikacji klienckiej — różnica leży w samej aplikacji (błędna konfiguracja, inny certyfikat zaufania), nie w sieci.

Częste błędy

„Host nie odpowiada na ping, więc jest padnięty” — a usługa działa bez zarzutu.

```
ping -c 2 app01.zielona.example
```

```
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
```

```
nc -zv app01.zielona.example 443
```

```
Connection to app01.zielona.example 443 port [tcp/https] succeeded!
```

Firewall na serwerze albo po drodze blokuje wyłącznie ICMP (typowa, celowa konfiguracja), a port 443 jest w pełni osiągalny. Wniosek „host nie działa” na podstawie samego `ping` jest błędny — trzeba sprawdzić konkretny port, którego faktycznie dotyczy zgłoszenie.

`tcpdump` bez filtra zapycha terminal na produkcyjnym łączu.

```
sudo tcpdump -i ens192
```

```
^C
1284729 packets captured
1284801 packets received by filter
```

Ponad milion pakietów w kilkanaście sekund na obciążonym interfejsie — terminal nie nadąża wypisywać, część pakietów jest gubiona przez sam system przechwytywania, a znalezienie tej jednej istotnej sesji w takim zalewie jest praktycznie niemożliwe. Naprawa: zawsze filtr (`port`, `host`) i zapis do pliku (`-w`) zamiast wypisywania na ekran, jak w Fiszce.

`curl -k` zostawione w skrypcie monitorującym „tymczasowo”, na stałe.

```
curl -k -s -o /dev/null -w '%{http_code}' https://api.wewnetrzne.zielona.example/health
```

Monitoring raportuje „wszystko działa”, mimo że certyfikat serwera wygasł trzy tygodnie temu — `-k` cicho pomija tę informację za każdym razem. Naprawa: usunąć `-k` z każdego skryptu, który ma działać dłużej niż jedno, świadome sprawdzenie na miejscu; jeśli certyfikat jest wewnętrzny, dodać jego urząd certyfikacji do zaufanych (`curl --cacert`), zamiast wyłączać weryfikację w ogóle.

Test portu przez otwarcie pełnej sesji aplikacyjnej zamiast prostego sprawdzenia.

```
telnet app01.zielona.example 443
```

```
-bash: telnet: command not found
```

`telnet` bywa nieobecny domyślnie na współczesnych minimalnych instalacjach (RHEL 9/10 go nie instaluje) — pakiet trzeba by doinstalować tylko po to, żeby sprawdzić, czy port jest otwarty. Naprawa: `nc -zv host port` albo, gdy nawet `nc` nie jest dostępne, wbudowana w powłokę sztuczka `timeout 3 bash -c '</dev/tcp/host/port'` z Fiszki — działa bez instalowania czegokolwiek.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz lokalnie, na loopbacku, bez potrzeby dostępu do internetu poza jednym oznaczonym poleceniem.

Ćwiczenie 1 — cała procedura na własnym serwerze testowym

```
# Terminal 1: prosty serwer HTTP na loopbacku
python3 -m http.server 8080 --bind 127.0.0.1 &

# Terminal 2: przejdź przez procedurę krok po kroku
ping -c 2 127.0.0.1
nc -zv 127.0.0.1 8080
curl -I http://127.0.0.1:8080/
sudo tcpdump -i lo -nn port 8080 -c 6 &
curl -s http://127.0.0.1:8080/ -o /dev/null
```

Który krok jako pierwszy dałby jednoznaczny sygnał, gdyby serwer w terminalu 1 w ogóle nie wystartował?

Odpowiedź

Krok `nc -zv 127.0.0.1 8080` — `ping` do `127.0.0.1` zawsze odpowie (to loopback, nie zależy od żadnej usługi), więc nic nie wykaże. Dopiero test portu pokaże `Connection refused`, jednoznacznie wskazując brak nasłuchującej usługi, zanim jeszcze spróbujesz czegokolwiek na poziomie HTTP.

Ćwiczenie 2 — test portu bez `nc`, samym bashem

```
nc -l -p 9000 &
timeout 2 bash -c '</dev/tcp/127.0.0.1/9000' && echo OTWARTY || echo ZAMKNIETY
kill %1 2>/dev/null
timeout 2 bash -c '</dev/tcp/127.0.0.1/9000' && echo OTWARTY || echo ZAMKNIETY
```

Dlaczego wynik drugiego testu różni się od pierwszego, mimo że polecenie testujące jest identyczne?

Odpowiedź

Pierwszy test trafia na `nc -l -p 9000`, wciąż działający w tle — port jest otwarty, więc `/dev/tcp/127.0.0.1/9000` łączy się bez problemu. Po `kill %1` proces `nc` kończy działanie i przestaje nasłuchiwać, więc drugi test dostaje odmowę połączenia (`ZAMKNIETY`) — dowód, że `/dev/tcp/...` faktycznie testuje żywy stan portu w danej chwili, nie zapamiętany wynik.

Pytania kontrolne

1. `ping` do serwera nie zwraca odpowiedzi. Czy to wystarczający dowód, że serwer jest niedostępny?

Odpowiedź

Nie. `ping` używa ICMP, a wiele firewalli blokuje ICMP celowo, zostawiając wszystkie porty aplikacyjne w pełni osiągalne. Brak odpowiedzi na `ping` mówi wyłącznie o polityce wobec ICMP — właściwym następnym krokiem jest sprawdzenie konkretnego portu przez `nc -zv` albo `curl`.

2. Dlaczego `traceroute -T -p 443` bywa dokładniejszy niż zwykłe `traceroute` przy diagnozowaniu problemów z HTTPS?

Odpowiedź

Bo zwykły `traceroute` domyślnie wysyła pakiety UDP (albo ICMP), które firewalle po drodze traktują często inaczej niż ruch TCP na porcie 443. `-T -p 443` wysyła pakiety TCP na ten konkretny port, więc zmierzona trasa i ewentualne blokady odzwierciedlają dokładnie to, co

napotka prawdziwe połączenie HTTPS — a nie ruch zupełnie innego rodzaju.

3. Dlaczego na produkcyjnym, obciążonym serwerze `tcpdump` powinien pisać do pliku z filtrem, a nie wypisywać wszystko na ekran?

Odpowiedź

Bo bez filtra `tcpdump` przechwytuje każdy pakiet na interfejsie — na obciążonym łączu to setki tysięcy linii na minutę, które zapychają terminal, utrudniają znalezienie istotnej sesji i mogą prowadzić do gubienia pakietów przez sam mechanizm przechwytywania. Zapis do pliku (`-w`) z filtrem na konkretny port lub host pozwala przeanalizować dokładnie to, co potrzebne, bez obciążania systemu w czasie rzeczywistym.

4. Kiedy — jeśli w ogóle — wolno zostawić `curl -k` w stałym skrypcie monitorującym?

Odpowiedź

Praktycznie nigdy. `-k` wyłącza całą weryfikację certyfikatu TLS, więc monitoring przestaje wykrywać wygasłe, niepoprawne albo podmienione certyfikaty — realny, poważny problem bezpieczeństwa zacznie wyglądać jak „wszystko działa”. Właściwe rozwiązanie dla certyfikatów wewnętrznych to dodanie ich urzędu certyfikacji do zaufanych przez `curl --cacert`, nie wyłączanie weryfikacji w ogóle.

Powiązane: Przekierowanie ruchu między interfejsami i portami · DNS z perspektywy administratora: dig, host, resolvectl · Warstwa pod firewalld: nftables i iptables **Źródła:** `man 8 ping` · `man 8 traceroute` · `man 8 mtr` · `man 1 nc` · `man 1 curl` · `man 1 openssl` (sekcja `s_client`) · `man 8 tcpdump` · notatki wewnętrzne: sprawdzanie nasłuchujących portów (`netstat`/`lsof` jako historyczne odpowiedniki `ss`)

Revision #16
Created 2026-09-17 21:43:25 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:10 UTC by Tomasz Konieczny