Pliki, które nie są tekstem: file, strings, hexdump, xxd

Poziom: średnio **Dotyczy: wszystkie dystrybucje** **Czas: ~12 min**

Jak sprawdzić, czym naprawdę jest plik, i jak zajrzeć do środka czegoś, co nie jest tekstem, nie rozwalając przy tym terminala.

Skróty użyte na tej stronie (6)

Skrót	Co znaczy
ANSI	American National Standards Institute: tu chodzi o kody sterujące kolorem i pozycją kursora w terminalu
ASCII	American Standard Code for Information Interchange: podstawowy zestaw znaków (litery, cyfry) używany w komputerach
BOM	Byte Order Mark: znacznik na początku pliku tekstowego UTF-8, który dopisuje Excel
CSV	w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami
PNG	Portable Network Graphics: format obrazów rastrowych bez straty jakości
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa

Fiszka

TYLKO ODCZYT

Rozpoznanie typu pliku: file

```
file zagadka.dat
```

Czytamy pierwsze bajty pliku i porównujemy je z bazą sygnatur, niezależnie od rozszerzenia w nazwie. To pierwszy odruch przy nieznanym pliku.

Przykład:

```
zagadka.dat: gzip compressed data, from Unix, original size modulo 2^32 5242880
```

file = rozpoznaj rzeczywisty typ pliku po jego zawartości, niezależnie od rozszerzenia

TYLKO ODCZYT

Pominięcie nazwy pliku w wyniku: file -b

```
file -b zagadka.dat
```

Flaga `-b` (*brief*) pomija dwukropek i nazwę pliku na początku wyniku — wygodne w skryptach, gdzie liczy się sam opis typu, a nie to, do jakiego pliku należy.

Przykład:

```
gzip compressed data, from Unix, original size modulo 2^32 5242880
```

file = rozpoznaj rzeczywisty typ pliku po jego zawartości, niezależnie od rozszerzenia

TYLKO ODCZYT

Sprawdzenie typu MIME i kodowania: file -i

```
file -i /tmp/config.ini
```

Zamiast opisu słownego dostajemy typ MIME i, dla plików tekstowych, wykryte kodowanie znaków (`charset=`). To pierwsze miejsce, w którym widać, czy plik jest w UTF-8, czy w czymś innym.

Przykład:

```
/tmp/config.ini: text/plain; charset=utf-8
```

file = rozpoznaj rzeczywisty typ pliku po jego zawartości, niezależnie od rozszerzenia

TYLKO ODCZYT

Wypisanie czytelnych ciągów znaków w binarce: strings

```
strings program | less
```

Wypisujemy każdy ciąg co najmniej czterech kolejnych znaków drukowalnych, otoczony bajtami niedrukowalnymi. Domyślnie skanujemy tylko sekcje z danymi zainicjalizowanymi w pliku wykonywalnym; `-a` wymusza skan całego pliku.

□□ TYLKO ODCZYT

Ustawienie progu długości ciągu: strings -n 12

```
strings -n 12 program | grep -i version
```

Podnosimy minimalną długość wypisywanego ciągu z domyślnych czterech do dwunastu znaków — odcinamy masę przypadkowego szumu (fragmenty nazw funkcji, pojedyncze słowa) i zostawiamy dłuższe, sensowne napisy jak numery wersji czy ścieżki.

Przykład:

```
Server build 4.11.2 (2026-03-08)
```

□□ TYLKO ODCZYT

Wypisanie zrzutu szesnastkowego: xxd

```
xxd -l 64 plik
```

Wypisujemy bajty pliku w postaci szesnastkowej, z kolumną ASCII obok. `-l 64` ogranicza wynik do pierwszych 64 bajtów — w zupełności wystarcza do rozpoznania nagłówka.

Przykład:

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 306d 0000 0000 0000  ..>.....0m......
```

□□ **ASCII** = American Standard Code for Information Interchange: podstawowy zestaw znaków (litery, cyfry) używany w komputerach · **xxd** = pokaż zawartość pliku w postaci szesnastkowej, do podglądu binariów

□□ ZMIENIA SYSTEM

Powrót z heksa do binarki: xxd -r

```
xxd plik > plik.hex
xxd -r plik.hex > plik.nowy
```

Odwracamy operację `xxd`: z tekstowego dumpu odtwarzamy oryginalne bajty. Jedyne z narzędzi w tej grupie, które potrafi wrócić — pozwala wyedytować plik binarny w zwykłym edytorze tekstu i złożyć go z powrotem.

xxd = pokaż zawartość pliku w postaci szesnastkowej, do podglądu binariów

TYLKO ODCZYT

Wypisanie zrzutu heksadecymalnego z util-linux: hexdump -C

```
hexdump -C plik | head -n 5
```

To samo zadanie co `xxd`, inny format i inne pochodzenie (pakiet `util-linux` zamiast Vima). Kolumna ASCII jest otoczona pionowymi kreskami zamiast stać osobno po prawej.

Przykład:

```
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  03 00 3e 00 01 00 00 00  30 6d 00 00 00 00 00 00  |..>.....0m......|
```

ASCII = American Standard Code for Information Interchange: podstawowy zestaw znaków (litery, cyfry) używany w komputerach

TYLKO ODCZYT

Wypisanie bajtów jako znaków i escape'ów: od -c

```
od -c plik | head
```

Pokazujemy każdy bajt jako znak drukowalny albo jako notację escape (`\r`, `\t`, `\0`, `\n`) — czytelniejsze niż heks, gdy szukamy konkretnie znaków sterujących.

Przykład:

```
0000000    [   m   a   i   n   ]  \r  \n   p   o   r   t   =   8   0   8
0000020    0  \r  \n
```

TYLKO ODCZYT

Wypisanie heksa bez offsetu (coreutils): od -An -tx1

```
od -An -tx1 -N 16 plik
```

`-tx1` wypisuje pojedyncze bajty szesnastkowo, `-An` gasi kolumnę offsetu, `-N 16` ogranicza do pierwszych szesnastu bajtów. `od` należy do `coreutils`, więc jest dostępne zawsze — dobra alternatywa, gdy `xxd` i `hexdump` nie są zainstalowane.

Przykład:

```
 89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52
```

TYLKO ODCZYT

Sprawdzenie pierwszego różniącego się bajtu: cmp

```
cmp a.bin b.bin
```

Porównujemy dwa pliki bajt po bajcie. Gdy są identyczne, `cmp` milczy; przy różnicy podaje numer pierwszego różniącego się bajtu i linii.

Przykład:

```
a.bin b.bin differ: byte 4097, line 1
```

cmp = Linux: porównaj dwa pliki bajt po bajcie, u nas pierwsze megabajty dysku z zerami

TYLKO ODCZYT

Ratowanie rozsypanego terminala: reset

```
reset
```

Reinicjalizujemy terminal od zera: czyścimy ekran i przywracamy domyślne ustawienia trybu znaków, kodowania i kolorów. Wpisujemy go „na ślepo", gdy `cat` na pliku binarnym zamieniło terminal w śmietnik.

TYLKO ODCZYT

Przywrócenie ustawień linii terminala: stty sane

```
stty sane
```

Łagodniejsza wersja `reset` — przywraca tylko ustawienia trybu terminala (echo, tłumaczenie `CR` na `LF`), bez czyszczenia ekranu. Jeśli Enter nie działa, zatwierdzamy komendę przez `Ctrl+J` zamiast Entera.

TYLKO ODCZYT

Wszystkie komendy

Komenda	Co daje	Kiedy
`file`	typ pliku z magic number	przed otwarciem nieznanego pliku
`file -i`	typ MIME i kodowanie	UTF-8 czy ISO-8859-2?
`strings -n N`	ciągi min. N znaków	wersja, ścieżka albo URL w binarce
`xxd` / `hexdump -C`	dump heks + ASCII	nagłówek, BOM, CRLF
`xxd -r`	z heksa z powrotem do binarki	zmiana jednego bajtu
`od -c`	bajty jako znaki i escape'y	chcesz zobaczyć `\r`, `\t`, `\0`
`cmp`	pierwszy różniący się bajt	pliki „powinny" być identyczne

Uważaj

- `xxd` nie zawsze jest zainstalowane: na RHEL/Rocky siedzi w `vim-common`, na Debianie od 12 w osobnym `xxd`. `od` jest zawsze, bo należy do `coreutils`.
- `file` czyta tylko początek pliku — uszkodzony nagłówek daje werdykt `data`.
- Nie rób `cat` na czymkolwiek, czego nie sprawdziłeś wcześniej przez `file`.

URL = Uniform Resource Locator: adres strony lub zasobu · **BOM** = Byte Order Mark: znacznik na początku pliku tekstowego UTF-8, który dopisuje Excel · **RHEL** = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa

Z życia wzięte

Z ŻYCIA WZIĘTE

> Środa, 11:00. Magazyn w Swarzędzu dostaje od dostawcy plik `raport.csv` na `fs01`, a Excel odmawia otwarcia i sypie krzaczkami. Tomek sprawdza `file raport.csv` i widzi, że to w rzeczywistości archiwum ZIP z rozszerzeniem zmienionym ręcznie przez dostawcę. Rozpakowuje je i znajduje prawdziwy plik CSV w środku, zanim Marek zdąży się zdenerwować.

Po co to istnieje

W Windowsie rozszerzenie jest częścią kontraktu: `.txt` otwiera Notatnik, `.exe` uruchamia system. W Linuksie rozszerzenie to zwykłe znaki w nazwie — nic nie sprawdza, czy `raport.pdf` jest PDF-em, a plik wykonywalny może nazywać się `dane` i działać.

> **Pojęcie: liczba magiczna (magic number).** Kilka pierwszych bajtów pliku, charakterystycznych dla jego formatu i niezależnych od nazwy czy rozszerzenia — PNG zaczyna się od `89 50 4e 47`, ELF od `7f 45 4c 46`, gzip od `1f 8b`. Linux nie traktuje rozszerzenia jako kontraktu: jądro i biblioteki systemowe, które muszą wiedzieć, co otwierają, sprawdzają rzeczywistą zawartość pliku, a nie trzy znaki po kropce w nazwie. Dlatego plik nazwany `raport.txt` może być archiwum ZIP i normalnie się rozpakuje, a zmiana nazwy pliku wykonywalnego na `dane.jpg` nie przeszkodzi mu się uruchomić.

Przekłada się to na konkretne sytuacje. Przychodzi `eksport.csv`, który okazuje się skompresowanym archiwum. Backup nazwany `.tar.gz` nie chce się rozpakować, bo jest zwykłym `.tar`. Plik konfiguracyjny wygląda poprawnie w `vim`, ale usługa go odrzuca — bo edytor dokleił na początku trzy bajty BOM-a.

Te narzędzia odpowiadają na trzy pytania: **czym to jest** (`file`), **co w środku jest czytelne dla człowieka** (`strings`) i **co dokładnie stoi w bajcie numer N** (`xxd`, `hexdump`, `od`).

Analogia

To jest **prześwietlenie bagażu na lotnisku**.

Walizka ma naklejkę „ubrania". Naklejka nic nie kosztuje i nikt jej nie weryfikuje — dokładnie tak działa rozszerzenie pliku. Operator skanera naklejek nie czyta: puszcza walizkę przez aparat i patrzy na kształty. Cylinder z gwintem to butla, prostokąt z gęstym wnętrzem to laptop, wąski charakterystyczny obrys to nóż. Rozpoznaje zawartość po sylwetkach, nie po deklaracji właściciela.

`file` robi to samo. Czyta pierwsze bajty pliku i porównuje je ze zbiorem znanych sygnatur. To są te sylwetki na ekranie skanera.

Analogia ma praktyczne ograniczenie. Skaner widzi tylko to, co zmieści się w polu widzenia aparatu; jeśli coś leży głęboko pod ubraniami i nie ma wyraźnego kształtu, operator nie wie, co to jest. `file` czyta pierwsze kilkaset bajtów — plik z uszkodzonym albo zaszyfrowanym początkiem dostanie werdykt `data`, czyli „nie wiem, coś tam jest". To nie awaria narzędzia, tylko granica metody. Operator skanera myli się zresztą podobnie: suszarka i wiertarka mają zbliżony obrys.

Jak to działa

Rozwiń: jak to działa pod spodem

`file` wykonuje trzy testy w tej kolejności i zatrzymuje się na pierwszym, który da wynik:

1. **Testy systemu plików** — katalog, urządzenie, dowiązanie, gniazdo. To `stat()`, nie czytanie zawartości.
2. **Testy magic** — porównanie bajtów z bazą sygnatur.
3. **Testy językowe** — jeśli plik wygląda na tekst, `file` zgaduje kodowanie znaków i rozpoznaje skrypty po linii `#!`.

Baza sygnatur to zwykły tekst, który można obejrzeć i rozszerzyć: źródła leżą w `/usr/share/misc/magic` i `/etc/magic` (miejsce na własne reguły), a `file` używa skompilowanej wersji `/usr/share/misc/magic.mgc`, bo parsowanie tekstu przy każdym uruchomieniu byłoby wolne. Ścieżki potwierdzi `file --version`:

```
file-5.45
magic file from /etc/magic:/usr/share/misc/magic
```

Warto znać kilka sygnatur na pamięć, bo rozpoznasz je w dumpie bez sięgania po `file`:

Format	Pierwsze bajty	W ASCII
ELF (binarka, `.so`)	`7f 45 4c 46`	`.ELF`
PNG	`89 50 4e 47 0d 0a 1a 0a`	`.PNG....`
ZIP, `.docx`, `.jar`	`50 4b 03 04`	`PK..`
gzip	`1f 8b 08`	—
PDF	`25 50 44 46`	`%PDF`
BOM UTF-8	`ef bb bf`	—

> **Pojęcie: BOM (Byte Order Mark).** Trzy bajty (`ef bb bf` w UTF-8) czasem doklejane na samym początku pliku tekstowego przez edytory pochodzące ze świata Windows, żeby zaznaczyć kodowanie. Dla człowieka są niewidoczne — plik wygląda normalnie w każdym edytorze — ale dla parsera to trzy dodatkowe bajty przed właściwą treścią. Plik `.ini`, który zaczyna się od BOM-u, ma sekcję `[main]` poprzedzoną śmieciem, więc parser jej nie rozpoznaje i zgłasza brakującą sekcję, choć w edytorze wszystko wygląda w porządku.

`strings` działa inaczej: skanuje plik bajt po bajcie i wypisuje każdy ciąg co najmniej czterech kolejnych znaków drukowalnych zakończony bajtem niedrukowalnym. Domyślnie przegląda tylko sekcje z danymi zainicjalizowanymi; `-a` wymusza skan całego pliku.

`xxd`, `hexdump -C` i `od` robią to samo: wypisują bajty szesnastkowo plus kolumnę ASCII. Różnią się formatem i pochodzeniem — `xxd` przyszedł z Vimem, `hexdump` z `util-linux`, `od` należy do `coreutils` i przez to jest dostępne zawsze. `xxd` ma jedną cechę, której pozostałe nie mają: `-r` odwraca operację, więc da się wyedytować dump w edytorze i złożyć plik z powrotem.

Krok po kroku

1. Sprawdź kilka plików naraz, zanim zdecydujesz, co z nimi zrobić.

```
file /usr/bin/ls /etc/hosts /var/log/messages-20260916.gz raport.pdf
```

```
/usr/bin/ls:                    ELF 64-bit LSB pie executable, x86-64, dynamically linked
/etc/hosts:                     ASCII text
/var/log/messages-20260916.gz:  gzip compressed data, from Unix, original size modulo 2^32
1204331
raport.pdf:                     PDF document, version 1.4
```

Cztery różne pliki, jedna komenda. `file` z wieloma argumentami wypisuje każdy wynik w osobnej linii, z nazwą pliku dopasowaną do najdłuższej z podanych.

2. Rozpoznaj format po samym nagłówku, bez `file`.

```
xxd -l 16 /usr/bin/ls
xxd -l 16 /usr/share/pixmaps/przyklad.png
xxd -l 16 /var/log/messages-20260916.gz
```

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000000: 8950 4e47 0d0a 1a0a 0000 000d 4948 4452  .PNG........IHDR
00000000: 1f8b 0800 0000 0000 0203 8dd4 c96e db30  .............n.0
```

Trzy nagłówki, trzy sygnatury z tabeli wyżej: `7f 45 4c 46` (ELF), `89 50 4e 47` (PNG), `1f 8b 08` (gzip). Po tych czterech–ośmiu bajtach rozpoznasz format szybciej, niż `file` zdąży się uruchomić.

3. Zobacz typ MIME i kodowanie.

```
file -i /tmp/config.ini
```

```
/tmp/config.ini: text/plain; charset=utf-8
```

> **Pojęcie: kodowanie znaków.** Sposób, w jaki bajty na dysku przekładają się na znaki na ekranie. W ASCII jeden bajt to jeden znak i zakres jest wąski — same litery bez ogonków; w UTF-8 znak może zajmować od jednego do czterech bajtów, dzięki czemu mieszczą się w nim polskie „ą" czy chińskie znaki, ale liczba bajtów przestaje być równa liczbie znaków. Plik zapisany w jednym kodowaniu, a odczytany przy założeniu innego, pokazuje krzaki zamiast liter — to nie uszkodzenie danych, tylko niezgodność interpretacji.

`charset` jest tu ważniejszy od typu. Jeśli zobaczysz `charset=iso-8859-2` tam, gdzie spodziewasz się UTF-8, polskie znaki są zapisane w innym kodowaniu i wymagają `iconv -f iso-8859-2 -t utf-8`.

4. Wyciągnij czytelne ciągi z pliku wykonywalnego.

Program nie ma flagi `--version`, a musisz ustalić wersję:

```
strings -n 12 /opt/aplikacja/bin/serwer | grep -iE 'version|build|[0-9]+\.[0-9]+'
```

```
Server build 4.11.2 (2026-03-08)
OpenSSL 3.0.7 1 Nov 2022
/opt/aplikacja/etc/serwer.conf
```

Przy okazji wyszła wbudowana na sztywno ścieżka do konfiguracji — najszybsza metoda ustalenia, którego pliku program naprawdę szuka, gdy dokumentacja mówi co innego niż rzeczywistość.

5. Obejrzyj nagłówek pliku i ustal bitowość.

```
xxd -l 48 /usr/bin/ls
```

```
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 3e00 0100 0000 306d 0000 0000 0000  ..>.....0m......
```

`hexdump -C` pokazuje to samo, tylko bajt po bajcie i z kolumną ASCII w pionowych kreskach (`|.ELF............|`). Bajt piąty (`02`) mówi, że to binarka 64-bitowa — `01` oznaczałoby 32 bity. Bajt szósty (`01`) to little-endian.

6. Zdiagnozuj plik konfiguracyjny, który „wygląda dobrze".

```
xxd /tmp/bom.ini
```

```
00000000: efbb bf5b 6d61 696e 5d0d 0a70 6f72 743d  ...[main]..port=
00000010: 3830 3830 0d0a                           8080..
```

Dwie usterki naraz. `ef bb bf` na początku to BOM UTF-8 — parser czyta pierwszą sekcję jako `<BOM>[main]` i zgłasza, że sekcji `[main]` nie ma. `0d 0a` na końcu każdej linii to CRLF, więc wartość portu to naprawdę `8080\r`. Naprawa obu:

```
sed -i '1s/^\xef\xbb\xbf//; s/\r$//' /tmp/bom.ini
```

7. Porównaj dwa pliki binarne.

`cmp a.bin b.bin` wypisuje pierwszą różnicę (`differ: char 4097, line 1`), `cmp -l` wszystkie. Po kontekst sięgnij po `diff <(xxd a.bin) <(xxd b.bin) | head` — `<(...)` to podstawienie procesu: powłoka uruchamia `xxd` i podstawia jego wyjście jako plik tymczasowy. Działa w Bashu i Zsh, nie w `dash`, czyli w `/bin/sh` na Debianie.

Częste błędy

`cat` na binarce i terminal przestaje odpowiadać.

```
cat /usr/bin/ls
```

Na ekranie sypią się śmieci, potem litery zamieniają się w znaki graficzne, Enter przestaje przechodzić do nowej linii, a to, co piszesz, nie pojawia się. Powód: w danych binarnych trafiły się sekwencje sterujące ANSI — jedna przełączyła terminal na alternatywny zestaw znaków, inna wyłączyła echo.

Wpisz na ślepo `reset` i naciśnij Enter. Jeśli Enter nic nie robi (terminal jest w trybie surowym i nie tłumaczy `CR` na `LF`), zatwierdź przez `Ctrl+J`, który wysyła gołe `LF`; alternatywą jest `stty sane`, też zatwierdzone `Ctrl+J`. Zapobieganie jest prostsze: `file plik` przed `cat plik`, albo od razu `less plik` — `less` wykrywa dane binarne i pyta `"plik" may be a binary file. See it anyway?`.

`xxd: command not found` na świeżo zainstalowanym serwerze.

```
-bash: xxd: command not found
```

Minimalna instalacja RHEL/Rocky nie zawiera Vima, a `xxd` przychodzi z nim w pakiecie `vim-common`. Na Debianie/Ubuntu od wersji 12 jest to osobny pakiet `xxd`, wcześniej też `vim-common`. Zamiast instalować cokolwiek na produkcji, użyj `od` z `coreutils`:

```
od -An -tx1z -N 32 plik
```

`-tx1` wypisuje pojedyncze bajty szesnastkowo, `-z` dokłada kolumnę ASCII, `-An` gasi offset. `hexdump -C` też bywa niedostępne — na Debianie 12 przeniesiono je do pakietu `bsdextrautils`.

Wniosek z `strings`, że plik „zawiera wirusa".

`strings` na dowolnej binarce wypluje setki ciągów: nazwy funkcji systemowych, komunikaty błędów, ścieżki, fragmenty URL-i. Znalezienie `/bin/sh`, `wget` albo `password` nie oznacza niczego — to nazwy funkcji z bibliotek i szablony komunikatów. `strings` mówi, jakie napisy są w pliku, a nie co program robi; do tego drugiego służą `strace`, `ltrace` i analiza ruchu sieciowego.

Interpretowanie dumpu jako liczby bez patrzenia na kolejność bajtów.

W dumpie ELF-a widzisz `3e 00` i czytasz `0x3e00`. Błąd: x86-64 jest little-endian, więc to wartość `0x003e` = 62, identyfikator maszyny `EM_X86_64`. Grupowanie `xxd` po dwa bajty (`3e00`) wzmacnia złudzenie; `xxd -g 1` je wyłącza.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie, w katalogu `/tmp`. Nic nie wymaga uprawnień administratora.

Ćwiczenie 1 — rozpoznaj plik po sygnaturze, bez `file`

```
cd /tmp
echo "test" | gzip -c > proba.bin; cp /usr/bin/ls kopia.bin
xxd -l 8 proba.bin; xxd -l 8 kopia.bin
```

`1f 8b 08` to gzip, `7f 45 4c 46` (`.ELF`) to plik wykonywalny. Bez `xxd`: `od -An -tx1 -N 8 proba.bin`.

Ćwiczenie 2 — znajdź BOM i CRLF

```
cd /tmp
printf '\xef\xbb\xbf[main]\r\nport=8080\r\n' > psuje.ini
cat psuje.ini; file psuje.ini; xxd psuje.ini
```

`cat` nie pokazuje niczego podejrzanego. Napraw plik i sprawdź wynikiem `file`.

Odpowiedź

`file` mówi `Unicode text, UTF-8 (with BOM) text, with CRLF line terminators` — obie usterki naraz. W dumpie widać `efbb bf` na offsecie 0 i `0d 0a` przed każdym końcem linii. Po `sed -i '1s/^\xef\xbb\xbf//; s/\r$//' /tmp/psuje.ini` narzędzie powinno powiedzieć `ASCII text`.

Ćwiczenie 3 — zmień jeden bajt i wróć do binarki

```
cd /tmp
printf 'HASLO=tajne\n' > dane.bin
xxd dane.bin > dane.hex
sed -i 's/7461 6a6e/7a61 6a6e/' dane.hex
xxd -r dane.hex > dane2.bin
cat dane2.bin; cmp dane.bin dane2.bin
```

Odpowiedź

`cat` pokaże `HASLO=zajne`, a `cmp` zgłosi `differ: char 7, line 1`: bajt `74` (`t`) zamienił się w `7a` (`z`). To cały mechanizm ręcznej edycji binariów — `xxd` → edytor → `xxd -r`. Przy prawdziwych plikach wolno podmieniać bajty, nie wolno dodawać ani usuwać: zmiana długości dumpu psuje wynik.

Pytania kontrolne

1. `file raport.pdf` odpowiada `Zip archive data`. Plik jest uszkodzony?

Odpowiedź

Niekoniecznie. `file` czyta bajty, nie nazwę — ten plik naprawdę zaczyna się od `50 4b 03 04`, czyli sygnatury ZIP. Najpewniej ktoś zmienił rozszerzenie pliku `.docx`, `.xlsx` albo `.odt`, bo wszystkie te formaty to archiwa ZIP. Potwierdzi to `unzip -l raport.pdf`.

2. Dlaczego `file` może odpowiedzieć `data` na pliku, który działa poprawnie?

Odpowiedź

Bo `file` rozpoznaje tylko formaty ze swojej bazy i czyta wyłącznie początek pliku. Format własny producenta, plik zaszyfrowany albo obraz dysku, w którym system plików zaczyna się za tablicą partycji, nie pasują do niczego na offsecie 0. `data` znaczy „nie rozpoznaję", nie „uszkodzony". Przy urządzeniach i obrazach użyj `file -s`.

3. W dumpie widzisz `3e 00` w nagłówku ELF. Dlaczego to nie jest wartość 0x3e00?

Odpowiedź

Bo x86-64 jest little-endian: najmniej znaczący bajt leży pierwszy. Te dwa bajty to `0x003e` = 62, czyli `EM_X86_64` w polu `e_machine`. Domyślne grupowanie `xxd` po dwa bajty (`3e00`) zachęca do złego odczytu — przy ręcznym czytaniu struktur używaj `xxd -g 1`.

4. Po `cat` na pliku binarnym terminal nie wyświetla tego, co piszesz, a Enter nic nie robi. Co wpisujesz i czym zatwierdzasz?

Odpowiedź

`stty sane` zatwierdzone przez `Ctrl+J`. Terminal jest w trybie surowym i nie tłumaczy `CR` (który wysyła Enter) na `LF`, a `Ctrl+J` wysyła `LF` bezpośrednio. Alternatywnie `reset`, też zatwierdzone `Ctrl+J`: robi więcej, bo reinicjalizuje terminal i czyści ekran, podczas gdy `stty sane` przywraca tylko ustawienia linii.

Powiązane: cat: wypisywanie i sklejanie plików · Drobnica: wc, nl, tac, rev, split, truncate · Archiwa i logi rotowane: zcat, zless, zgrep, journalctl

Źródła: `man 1 file` · `man 5 magic` · `man 1 strings` · `man 1 xxd` · `man 1 od` · `man 1 cmp`

Revision #19
Created 2026-09-17 11:56:35 UTC by Tomasz Konieczny
Updated 2026-10-03 19:07:45 UTC by Tomasz Konieczny