sudo i /etc/sudoers: konfiguracja i audyt

◫ Poziom: średni

◫ Dotyczy: RHEL/Rocky 9–10 (grupa wheel), różnice dla Debian/Ubuntu (grupa sudo) zaznaczone

▯ Czas: ~18 min

`sudo` pozwala wykonać pojedyncze polecenie z uprawnieniami innego użytkownika — najczęściej roota — bez logowania się na jego konto. Reguły, kto i co może uruchomić, siedzą w jednym pliku, `/etc/sudoers`, który psuje się wyjątkowo boleśnie, jeśli edytujesz go niewłaściwym narzędziem.

◫ Skróty użyte na tej stronie (8)

Skrót	Co znaczy
CSV	w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami
GRUB	Grand Unified Bootloader: program, który jako pierwszy startuje po włączeniu komputera i podaje jądru, co ma uruchomić
LDAP	Lightweight Directory Access Protocol: protokół do odpytywania katalogu użytkowników (AD, FreeIPA)
PAM	Pluggable Authentication Modules: wymienne moduły logowania w Linuksie, decydują na przykład o sile wymaganego hasła
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
UID	User ID: numer użytkownika w Linuksie
WWW	World Wide Web: strony i aplikacje otwierane w przeglądarce po HTTP albo HTTPS

Fiszka

◫ TYLKO ODCZYT

Sprawdzenie własnych uprawnień sudo: sudo -l

```
sudo -l
```

Wypisujemy wszystkie reguły `sudoers` dotyczące bieżącego użytkownika na tym hoście: jakie polecenia, jako kto i z jakimi ograniczeniami. Pierwsze wywołanie w sesji poprosi o nasze własne hasło.

Przykład:

```
Dopasowane domyślne wpisy dla adm_user na tym hoście:
    !visiblepw, always_set_home, match_group_by_gid, env_reset,
    env_keep+="COLORS DISPLAY HOSTNAME HISTSIZE KDEDIR LS_COLORS",
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

Użytkownik adm_user może uruchamiać na srv01 następujące polecenia:
    (ALL) ALL
```

TYLKO ODCZYT

Sprawdzenie uprawnień innego użytkownika: sudo -l -U

```
sudo -l -U user1
```

To samo pytanie, ale o innego użytkownika niż my — wymaga, żebyśmy sami mieli uprawnienia sudo (zwykle pełne, `(ALL) ALL`), bo sprawdzamy cudze uprawnienia. Kluczowe narzędzie audytowe: sprawdzamy, co ktoś może zrobić, zanim mu na to pozwolimy albo zanim to komuś odbierzemy.

Przykład:

```
Dopasowane domyślne wpisy dla user1 na tym hoście:
    !visiblepw, always_set_home, match_group_by_gid, env_reset,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

Użytkownik user1 może uruchamiać na srv01 następujące polecenia:
    (ALL) NOPASSWD: /usr/bin/systemctl restart httpd, /usr/bin/systemctl status httpd
```

`user1` ma dostęp punktowy — tylko dwie komendy, bez hasła — a nie pełnego roota.

TYLKO ODCZYT

Uruchomienie komendy jako konkretny użytkownik: sudo -u

```
sudo -u postgres psql -c '\l'
```

Nie każde `sudo` musi kończyć się na roocie. `-u` pozwala wskazać dowolnego użytkownika docelowego, o ile reguły `sudoers` na to zezwalają — typowy wzorzec przy pracy z kontem serwisowym bazy danych.

TYLKO ODCZYT

Otwarcie pełnej sesji logowania jako root: sudo -i

```
sudo -i
```

Otwieramy powłokę logowania (login shell) na koncie roota — z jego `$HOME`, jego plikami startowymi (`.bash_profile`) i jego środowiskiem, tak jakbyśmy zalogowali się na `root` bezpośrednio. Domyślny cel to `root`; `sudo -i -u user1` robi to samo dla innego konta.

Przykład:

```
[root@srv01 ~]#
```

TYLKO ODCZYT

Otwarcie powłoki z uprawnieniami roota bez pełnego logowania: sudo -s

```
sudo -s
```

Podobne do `-i`, ale **nie** ładuje plików startowych roota — dziedziczy nasze bieżące środowisko i katalog roboczy, tylko z podniesionymi uprawnieniami. Różnica bywa źródłem zaskoczeń: `$HOME` po `sudo -s` może dalej wskazywać na nasz katalog domowy, nie na `/root`.

TYLKO ODCZYT

Unieważnienie cache'u haseł: sudo -k

```
sudo -k
```

Kasujemy zapamiętane poświadczenie (patrz *Jak to działa*) — następne wywołanie `sudo`, nawet za sekundę, ponownie poprosi o hasło. Nic nie wypisuje na ekranie przy powodzeniu.

□□ TYLKO ODCZYT

Odświeżenie cache'u bez uruchamiania komendy: sudo -v

```
sudo -v
```

Przedłużamy ważność zapamiętanego poświadczenia o kolejny okres (domyślnie kilkanaście minut), bez wykonywania żadnej właściwej komendy. Wygodne na początku długiej sesji administracyjnej w skrypcie, żeby uniknąć pytania o hasło w środku automatyzacji.

□□ TYLKO ODCZYT

Bezpieczna edycja sudoers: visudo

```
sudo visudo
```

Otwieramy `/etc/sudoers` w edytorze (domyślnie `vi`, konfigurowalne przez zmienną `EDITOR`), ale `visudo` **blokuje plik na czas edycji** i, co najważniejsze, **sprawdza poprawność składni przed zapisaniem zmian na docelowy plik**. Jeśli składnia jest błędna, `visudo` odmawia zapisu i pyta, czy poprawić błąd, zapisać mimo to, czy porzucić zmiany — dając nam szansę na naprawę, zanim cokolwiek zepsujemy.

□□ **visudo** = Linux: bezpiecznie edytuje pliki sudoers i sprawdza ich składnię, flaga -cf sprawdza plik bez edycji

□□ TYLKO ODCZYT

Weryfikacja składni sudoers: visudo -c

```
sudo visudo -c
```

Sprawdzamy bieżący `/etc/sudoers` i wszystkie pliki w `/etc/sudoers.d/` pod kątem błędów składniowych, bez otwierania edytora. Uruchamiamy to po każdej zmianie wykonanej poza `visudo` (np. przez system zarządzania konfiguracją) albo zanim zgłosimy problem dalej.

Przykład:

```
/etc/sudoers: parsed OK
/etc/sudoers.d/appteam: parsed OK
```

visudo = Linux: bezpiecznie edytuje pliki sudoers i sprawdza ich składnię, flaga -cf sprawdza plik bez edycji

TYLKO ODCZYT

Edycja pliku fragmentu reguł: visudo -f

```
sudo visudo -f /etc/sudoers.d/appteam
```

Ta sama walidacja składni co przy głównym pliku, ale dla osobnego pliku w katalogu `sudoers.d`, dołączanego automatycznie przez dyrektywę `#includedir /etc/sudoers.d` w głównym `/etc/sudoers`. Dobra praktyka: reguły specyficzne dla jednego zespołu czy aplikacji trzymamy w osobnym pliku, nie w monolitycznym `sudoers`, żeby zmiana jednego zespołu nie ryzykowała literówki w regułach całej reszty.

visudo = Linux: bezpiecznie edytuje pliki sudoers i sprawdza ich składnię, flaga -cf sprawdza plik bez edycji

TYLKO ODCZYT

Reguła oparta na grupie: %grupa ALL=(ALL) ALL

```
%wheel  ALL=(ALL)       ALL
```

Czytane od lewej: każdy członek grupy `wheel` (`%` oznacza grupę, nie użytkownika), na dowolnym hoście (pierwsze `ALL`), może uruchamiać komendy jako dowolny użytkownik docelowy (`(ALL)`), dowolną komendę (ostatnie `ALL`). To jest dokładnie ten wiersz, który w praktyce daje „pełne sudo” każdemu, kogo dodamy do grupy `wheel` przez `usermod -aG wheel user1`. Na Debianie/Ubuntu odpowiednikiem jest grupa `sudo`, a nie `wheel` — reguła w `/etc/sudoers.d/README` tam wygląda tak samo, tylko z inną nazwą grupy.

ZMIENIA SYSTEM

Pominięcie pytania o hasło: NOPASSWD

```
%appteam ALL=(ALL) NOPASSWD: /usr/bin/systemctl restart httpd, /usr/bin/systemctl status httpd
```

Bez tego słowa kluczowego każde wywołanie `sudo` z nowej sesji terminala pyta o hasło (z zastrzeżeniem cache'u poświadczeń, patrz niżej). `NOPASSWD` wyłącza to pytanie dla wskazanych komend — praktyczne dla automatyzacji i skryptów uruchamianych bez interakcji, ryzykowne, jeśli lista komend jest zbyt szeroka.

□□ **NOPASSWD** = wpis w sudoers pozwalający wykonać wskazane polecenie przez sudo bez podawania hasła

□□ **ZMIENIA SYSTEM**

Nazwanie grupy komend aliasem: Cmnd_Alias

```
Cmnd_Alias APACHE_CTL = /usr/bin/systemctl restart httpd, /usr/bin/systemctl status httpd, /usr/bin/systemctl reload httpd
%appteam ALL=(ALL) NOPASSWD: APACHE_CTL
```

Zamiast wypisywać tę samą listę komend w wielu regułach, definiujemy ją raz pod nazwą (konwencja: same wielkie litery) i odwołujemy się do niej w regułach. Analogicznie istnieją `User_Alias` (grupy użytkowników) i `Host_Alias` (grupy hostów) — przydatne, gdy jeden plik `sudoers` obsługuje wiele serwerów albo zespołów.

□□ **TYLKO ODCZYT**

Ustawienie długości ważności cache'u: Defaults timestamp_timeout

```
Defaults timestamp_timeout=15
```

Zmieniamy domyślny czas (w minutach), przez jaki `sudo` pamięta, że hasło zostało już podane w tej sesji terminala, zanim spyta ponownie. Wartość `0` wymusza pytanie o hasło przy **każdym** wywołaniu; wartość ujemna oznacza „nie wygasaj nigdy w tej sesji” — rzadko uzasadniony wybór na serwerze produkcyjnym.

□□ **TYLKO ODCZYT**

Ustawienie osobnego logu dla sudo: Defaults logfile

```
Defaults logfile="/var/log/sudo.log"
```

Poza standardowym logowaniem do dziennika systemowego (`journald` / `/var/log/secure`), `sudo` może dodatkowo pisać do wskazanego pliku tekstowego — wygodne, gdy chcemy mieć jeden, łatwy do `grep`owania plik z historią wszystkich wywołań `sudo` na hoście, niezależny od rotacji reszty logów systemowych.

□□ TYLKO ODCZYT

Sprawdzenie, kto należy do grupy z sudo: getent group wheel

```
getent group wheel
```

Odpytujemy bazę grup przez NSS (Name Service Switch) — działa identycznie niezależnie od tego, czy grupa pochodzi z lokalnego `/etc/group`, czy z LDAP/FreeIPA/AD. Szybszy i pewniejszy sposób sprawdzenia członków grupy niż ręczne przeglądanie plików.

Przykład:

```
wheel:x:10:adm_user,user1
```

□□ **LDAP** = Lightweight Directory Access Protocol: protokół do odpytywania katalogu użytkowników (AD, FreeIPA) · **getent** = get entries: odpytaj bazę systemową (użytkowników, grup, hostów) niezależnie od tego, czy to plik lokalny czy AD

□□ ZMIENIA SYSTEM

Audyt: skrypt sprawdzający wszystkich lokalnych użytkowników pod kątem sudo

```
#!/bin/bash
# Pobieramy nazwy użytkowników z /etc/passwd i zapisujemy do pliku csv
awk -F: '{print $1}' /etc/passwd > users.csv

# Sprawdzamy dla każdego, czy ma uprawnienia sudo
while IFS=, read -r username
do
    sudo -l -U "$username" 2>/dev/null | grep -q "(ALL)" \
        && echo "$username,sudo" >> sudo_users.csv \
        || echo "$username,brak_sudo" >> sudo_users.csv
done < users.csv
```

Iterujemy po wszystkich kontach systemowych, uruchamiamy dla każdego `sudo -l -U`, i klasyfikujemy wynik po obecności napisu `(ALL)` w regułach. Prosty, ale ma dwie pułapki: łapie też konta systemowe bez powłoki logowania (`bin`, `daemon` — warto filtrować po UID ≥ 1000) i przeoczy użytkownika, którego reguła daje dostęp punktowy bez słowa `ALL` (np. samo `NOPASSWD: APACHE_CTL` z poprzedniego przykładu) — taki wpis trafi do kolumny `brak_sudo`, mimo że jakieś uprawnienia jednak ma.

Przykład:

```
$ cat sudo_users.csv
root,sudo
adm_user,sudo
user1,brak_sudo
postgres,brak_sudo
```

Uważaj

- Reguła `NOPASSWD: ALL` na całej grupie oznacza, że każdy jej członek może również **zmienić sam plik sudoers** — to jest równoważne pełnemu, trwałemu dostępowi roota, nie tylko punktowemu wyjątkowi.
- `sudo -k` czyści tylko cache czasu — jeśli reguła ma `NOPASSWD`, kolejne wywołanie i tak nie zapyta o hasło.
- Plik w `/etc/sudoers.d/` musi mieć uprawnienia `0440` i właściciela `root:root` — `visudo -f` ustawia je automatycznie, ręczne `cp` czy edytor tego nie robią.

UID = User ID: numer użytkownika w Linuksie · **NOPASSWD** = wpis w sudoers pozwalający wykonać wskazane polecenie przez sudo bez podawania hasła · **awk** = od nazwisk autorów Aho, Weinberger, Kernighan: język do danych w kolumnach · **IFS** = Internal Field Separator: zmienna powłoki mówiąca, po jakim znaku dzielimy tekst na słowa, domyślnie spacja, tabulator i nowa linia

Z życia wzięte

Z ŻYCIA WZIĘTE

O 14:00 Ola prosi o nadanie nowemu administratorowi dostępu sudo na app01. Tomek dopisuje go do grupy wheel przez `usermod -aG wheel`, ale kolega w tej samej, już otwartej sesji SSH nadal dostaje `Sorry, user may not run sudo on app01`. Tomek tłumaczy, że członkostwo w grupach czyta się przy logowaniu, więc dopiero nowa sesja SSH widzi zmianę.

Po co to istnieje

> **Pojęcie: eskalacja uprawnień.** Eskalacja uprawnień to każda kontrolowana zmiana z niższego poziomu dostępu na wyższy — najczęściej z konta zwykłego użytkownika na roota. `sudo` jest legalnym, audytowalnym mechanizmem eskalacji: w przeciwieństwie do dzielenia się hasłem roota albo logowania bezpośrednio na to konto, każde użycie `sudo` zostaje zapisane w logu razem z tożsamością osoby, która je wywołała, i dokładną komendą. To samo słowo, „eskalacja uprawnień", opisuje też technikę ataku — różnica jest w tym, czy proces jest kontrolowany i widoczny (jak przy `sudo`), czy stanowi obejście zabezpieczeń wbrew twórcy systemu (np. przez lukę w oprogramowaniu).

`sudo` powstało jako odpowiedź na prosty problem: administratorzy w organizacji potrzebują różnych, ograniczonych fragmentów uprawnień roota — jeden ma restartować tylko usługę WWW, drugi ma zarządzać kopiami zapasowymi, trzeci ma pełny dostęp — a dzielenie się hasłem `root` między wszystkimi albo daje wszystkim wszystko, albo wymaga osobnego konta na każdą kombinację uprawnień. `/etc/sudoers` centralizuje te reguły w jednym miejscu i pozwala je różnicować bez mnożenia kont systemowych.

> **Pojęcie: plik sudoers i jego składnia.** `/etc/sudoers` (razem z fragmentami w `/etc/sudoers.d/`) to plik tekstowy opisujący, kto może uruchomić co, jako kto, i na jakich hostach. Pojedyncza reguła ma postać `kto hosty=(jako_kto) co`: `adm_user ALL=(ALL) ALL` znaczy „użytkownik `adm_user`, na dowolnym hoście, może uruchomić dowolną komendę jako dowolny użytkownik". `%wheel` zamiast nazwy użytkownika oznacza regułę dla całej grupy. Ta sama składnia obsługuje wyjątki (`NOPASSWD`), ograniczenia do konkretnych komend i nazwane grupy komend przez `Cmnd_Alias` — to właśnie ta gramatyka jest tak wrażliwa na literówki, że dostała dedykowany, blokujący edytor (`visudo`) zamiast zwykłego `vim /etc/sudoers`.

> **Pojęcie: cache poświadczeń.** Żeby nie pytać o hasło przy każdej pojedynczej komendzie w ciągu jednej sesji roboczej, `sudo` zapamiętuje fakt udanego uwierzytelnienia na określony czas (`timestamp_timeout`, domyślnie 5 lub 15 minut w zależności od dystrybucji), przypisany do konkretnego terminala i użytkownika. To jest dokładnie ten mechanizm, który sprawia, że drugie i trzecie `sudo` z rzędu w tej samej sesji nie proszą o hasło — a `sudo -k` go czyści, `sudo -v` go odświeża.

Analogia

Wyobraź sobie **księgę przepustek prowadzoną przez ochronę w dużym biurowcu**.

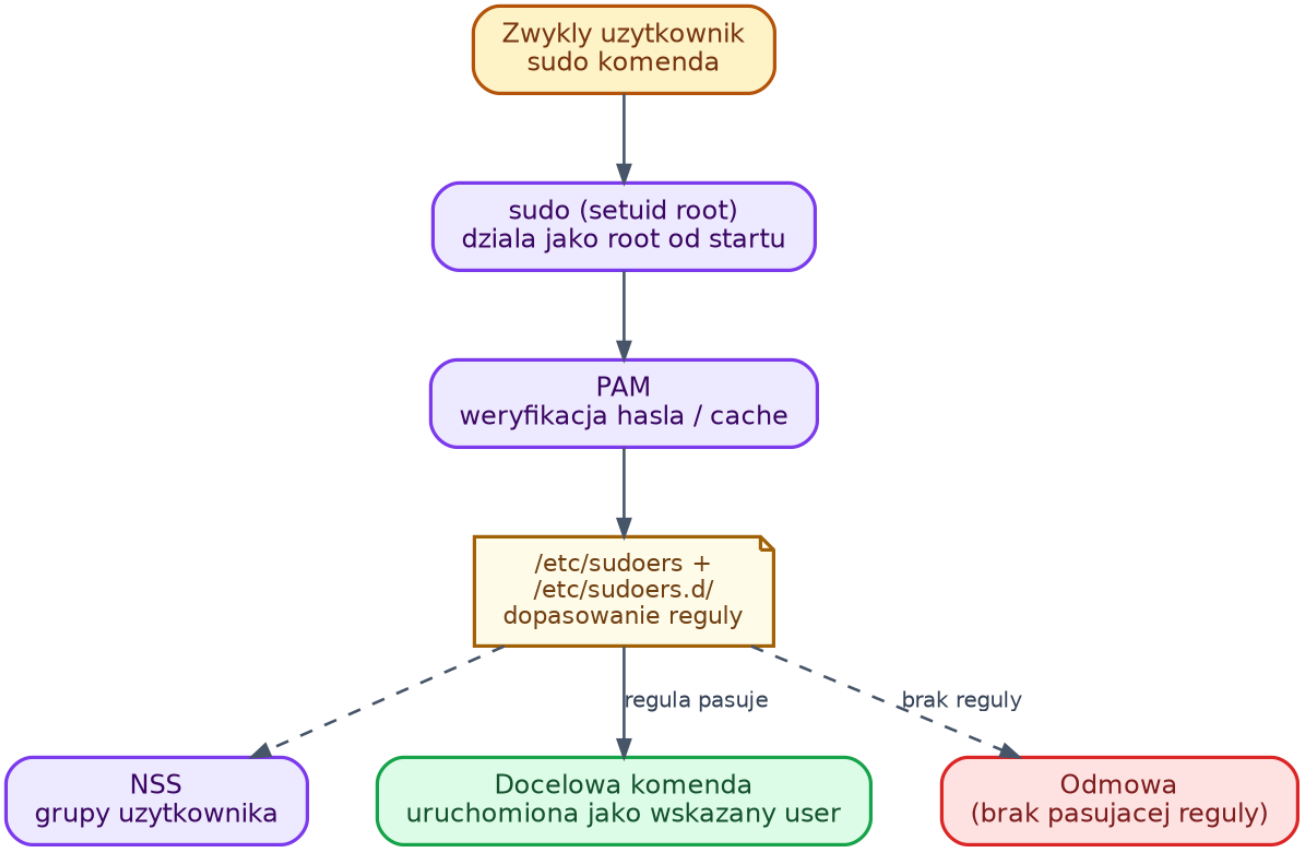
Każdy pracownik dostaje przepustkę, która wymienia dokładnie, do których drzwi i pięter ma dostęp — księgowa ma dostęp do piętra księgowości, ale nie do serwerowni; administrator ma dostęp wszędzie. Ochrona nie prosi o podpis przy każdych drzwiach — jeśli pokazałeś przepustkę przy wejściu do budynku niedawno, przez najbliższe kilkanaście minut kolejne drzwi otwierają się bez ponownej weryfikacji. To jest dokładnie mechanizm cache'u poświadczeń: uwierzytelniasz się raz, a `sudo` pamięta ten fakt przez chwilę.

Ale cała ta księga to jeden, fizyczny dokument — nie osobna kartka na osobę. Jeśli ktoś, kto go ostatnio aktualizował, zostawił niejednoznaczny wpis (brakującą kropkę, nieczytelny numer piętra), ochrona przy **każdych** drzwiach w budynku, dla **każdego** pracownika, zaczyna odmawiać wpuszczenia — łącznie z dyrektorem, którego przepustka jest bez zarzutu. System nie zgaduje, o co chodziło; jeśli nie rozumie zapisu, blokuje wszystko, dopóki ktoś uprawniony do prowadzenia księgi (jedyny, zaufany urzędnik — odpowiednik `visudo`) nie poprawi wpisu i nie potwierdzi, że reszta księgi nadal ma sens. Dlatego tej księgi nigdy nie edytuje się przypadkowym długopisem wziętym z biurka — tylko przez tego jednego urzędnika, który sprawdza każdą zmianę, zanim ją zatwierdzi.

Jak to działa

Rozwiń: jak to działa pod spodem

Binarka `/usr/bin/sudo` ma ustawiony bit `setuid root` — dlatego, mimo że uruchamia ją zwykły użytkownik, proces działa z realnymi uprawnieniami roota od samego startu i sam decyduje, czy w ogóle pozwoli cokolwiek zrobić. Decyzję podejmuje w kilku krokach: najpierw stos PAM (Pluggable Authentication Modules) weryfikuje Twoje hasło (albo odczytuje ważny cache poświadczeń i pomija to pytanie), potem `sudo` parsuje `/etc/sudoers` razem z zawartością `/etc/sudoers.d/` (dołączaną przez `#includedir`), dopasowuje Twojego użytkownika i grupy (odczytane przez NSS — stąd `getent group wheel` działa tak samo dla grup lokalnych, jak i domenowych) do reguł, i na końcu sprawdza, czy konkretna komenda pasuje do dozwolonej listy. Dopiero po przejściu wszystkich tych kroków `sudo` faktycznie uruchamia docelowy program z docelowym UID.



sudo działa jako root od startu dzięki setuid, uwierzytelnia przez PAM, dopasowuje regułę w sudoers wspartą o grupy z NSS, i dopiero wtedy uruchamia komendę albo odmawia (kliknięcie otwiera pełny rozmiar)

`visudo` blokuje plik `/etc/sudoers` na czas edycji (żeby dwie osoby nie edytowały go równocześnie), pracuje na kopii tymczasowej, i po zapisaniu uruchamia wewnętrzny parser `sudo` na tej kopii — dopiero jeśli parser potwierdzi, że składnia jest poprawna, kopia zastępuje oryginalny plik. Zwykły edytor tego kroku nie wykonuje: zapisze dosłownie to, co wpiszesz, nawet jeśli zawiera błąd składniowy, który zablokuje `sudo` dla wszystkich od następnego wywołania.

`sudo -i` a `sudo su -` dają na pierwszy rzut oka ten sam efekt — interaktywną powłokę roota z jego środowiskiem — ale różnią się mechanizmem. `sudo -i` to natywna funkcja `sudo`: sam program ładuje plik startowy roota, ustawia jego `$HOME` i przechodzi do jego katalogu domowego, bez uruchamiania żadnego dodatkowego procesu pośredniczącego. `sudo su -` to dwa programy z rzędu: `sudo` uruchamia `su`, a dopiero `su` (już jako root, więc bez pytania o dodatkowe hasło) przełącza się na docelowe konto. Efekt końcowy bywa nieodróżnialny dla użytkownika, ale ścieżka audytowa jest inna — log `sudo` zapisze samo wywołanie `sudo su -`, a nie żadną z komend uruchomionych już wewnątrz powłoki roota. To ograniczenie dotyczy zresztą obu wariantów: po wejściu do interaktywnej powłoki roota (`sudo -i` czy `sudo su -`) kolejne komendy nie są już pojedynczo logowane przez `sudo` — masz zwykłą sesję roota, z jej

własnym, znacznie uboższym logowaniem. Dlatego dobra praktyka administracyjna to unikanie stałej pracy z interaktywnej powłoki roota na rzecz pojedynczych `sudo komenda` — każde wywołanie zostaje osobno zapisane z pełnym kontekstem.

Logi `sudo` trafiają domyślnie do dziennika systemowego:

```
journalctl _COMM=sudo --since today
```

```
wrz 17 10:02:14 srv01 sudo[3021]: adm_user : TTY=pts/0 ; PWD=/home/adm_user ; USER=root ;
COMMAND=/usr/bin/systemctl restart httpd
```

Na RHEL/Rocky ten sam wpis trafia też do `/var/log/secure`; na Debianie/Ubuntu odpowiednikiem jest `/var/log/auth.log`. Wpis `Defaults logfile` z Fiszki dokłada dodatkowo osobny, dedykowany plik tekstowy.

Krok po kroku

1. Sprawdzenie, kto ma sudo, i dodanie kolejnego administratora.

```
getent group wheel
sudo usermod -aG wheel user2
id user2
```

```
wheel:x:10:adm_user
uid=1002(user2) gid=1002(user2) groups=1002(user2)
```

Grupa `wheel` jeszcze się nie pojawiła w `id user2` — członkostwo jest odczytywane przy logowaniu, więc `user2` musi się wylogować i zalogować ponownie (albo `newgrp wheel`), zanim `sudo -l -U user2` pokaże nowe uprawnienia.

2. Dostęp punktowy zamiast pełnego roota, z weryfikacją.

```
sudo visudo -f /etc/sudoers.d/appteam
```

```
Cmnd_Alias APACHE_CTL = /usr/bin/systemctl restart httpd, /usr/bin/systemctl status httpd
%appteam ALL=(ALL) NOPASSWD: APACHE_CTL
```

`visudo` sam sprawdza składnię i nadaje plikowi uprawnienia `0440`.

```
sudo -l -U user3
```

```
Użytkownik user3 może uruchamiać na srv01 następujące polecenia:
    (ALL) NOPASSWD: /usr/bin/systemctl restart httpd, /usr/bin/systemctl status httpd
```

3. Zaostrzenie cache'u, log i audyt zbiorczy.

```
sudo visudo   # Defaults timestamp_timeout=5
              # Defaults logfile="/var/log/sudo.log"
sudo bash audyt_sudo.sh
journalctl _COMM=sudo --since "-1 hour"
```

Częste błędy

Edycja `/etc/sudoers` zwykłym edytorem — literówka blokuje sudo dla wszystkich.

```
>>> /etc/sudoers: syntax error near line 28 <<<
sudo: parse error in /etc/sudoers near line 28
sudo: no valid sudoers sources found, quitting
```

Ten komunikat pojawia się przy **każdym** kolejnym wywołaniu `sudo`, dla **każdego** użytkownika na hoście, dopóki błąd nie zostanie naprawiony — a naprawić go można tylko z uprawnieniami roota, których nikt już nie potrafi zdobyć przez `sudo`. Ratunek: jeśli masz gdziekolwiek już otwartą, aktywną sesję z prawami roota (dlatego nigdy nie zamykaj jedynej takiej sesji przed potwierdzeniem, że zmiana działa), napraw plik przez `visudo` z tamtego okna natychmiast. Jeśli nie masz żadnej otwartej sesji roota, jedyna droga to fizyczny albo konsolowy dostęp do maszyny: restart do trybu ratunkowego (`rescue.target` / single-user) i edycja pliku bezpośrednio jako root bootowany z płyty ratunkowej albo z linii poleceń GRUB. Zapobieganie: nigdy nie edytuj `/etc/sudoers` inaczej niż przez `visudo`, i zawsze trzymaj przynajmniej jedną otwartą sesję administracyjną, dopóki nie zweryfikujesz zmiany osobnym, nowym połączeniem.

`sudo -u user komenda` nie przełącza środowiska — skrypt nie widzi swoich plików.

```
$ sudo -u appuser /opt/app/run.sh
/opt/app/run.sh: line 4: /home/adm_user/.config/app/settings.env: No such file or directory
```

`sudo -u` zmienia tożsamość procesu (UID/GID), ale bez `-i` zmienne środowiskowe — w tym `$HOME` — pozostają Twoje, nie docelowego użytkownika. Skrypt szukający pliku względem `$HOME` trafia we własny katalog administratora, nie w katalog `appuser`. Naprawa: `sudo -iu appuser /opt/app/run.sh` (pełne środowisko logowania) albo `sudo -Hu appuser /opt/app/run.sh` (samo przestawienie `$HOME`, bez pełnego logowania).

Dodanie do grupy `wheel` nie działa „od razu” w już otwartej sesji.

```
$ sudo -l
Sorry, user user2 may not run sudo on srv01.
```

Członkostwo w grupach jest odczytywane przy tworzeniu sesji logowania (token grup przypisany do procesu powłoki), nie sprawdzane na bieżąco. `usermod -aG wheel user2` zmienia `/etc/group`, ale powłoka, w której `user2` już siedzi, wciąż działa ze starym zestawem grup. Naprawa: wylogowanie i ponowne zalogowanie, `newgrp wheel` w bieżącej powłoce, albo `su - user2` z sesji, która już widzi nowe członkostwo.

Plik w `/etc/sudoers.d/` zignorowany przez zbyt otwarte uprawnienia.

```
$ sudo -l
sudo: /etc/sudoers.d/appteam is world writable
sudo: ignoring /etc/sudoers.d/appteam
```

Jeśli plik trafił tam przez zwykłe `cp` albo system zarządzania konfiguracją bez ustawienia trybu `0440`, `sudo` odmawia jego wczytania — z powodów bezpieczeństwa (plik zapisywalny przez wszystkich mógłby posłużyć do przemycenia własnej reguły). Naprawa: `sudo chmod 0440 /etc/sudoers.d/appteam && sudo chown root:root /etc/sudoers.d/appteam`, albo od razu edycja przez `visudo -f`, które ustawia poprawne uprawnienia automatycznie.

`sudo -k` mylone ze cofnięciem `NOPASSWD`.

Administrator, który chce natychmiast wymusić ponowne uwierzytelnienie użytkownika z regułą `NOPASSWD: ALL`, uruchamia `sudo -k` w jego sesji i jest zaskoczony, że kolejne `sudo` mimo to nie pyta o hasło. `-k` czyści wyłącznie cache czasowy poświadczeń — jeśli reguła w `sudoers` w ogóle nie wymaga hasła, nie ma czego czyścić. Jedyny sposób na odebranie dostępu bez hasła to zmiana samej reguły w `sudoers` (usunięcie `NOPASSWD` albo całego wpisu) przez `visudo`.

□□ Ćwiczenia

> Ćwiczenia 2 i 3 wymagają `sudo` i modyfikują `/etc/sudoers.d/` — wykonaj je na maszynie testowej, nie na produkcyjnej. Wyniki pośrednie (pliki CSV) zapisujesz w `/tmp`.

Ćwiczenie 1 — czytanie własnych uprawnień

```
sudo -l
```

Znajdź w wyniku wartość `secure_path`. Dlaczego `sudo` w ogóle narzuca własną ścieżkę wyszukiwania programów, zamiast używać Twojej zmiennej `$PATH`?

Odpowiedź

`secure_path` chroni przed atakiem polegającym na podstawieniu złośliwego programu o typowej nazwie (np. własnego `ls` albo `sudo`) wcześniej w `$PATH` użytkownika niż systemowe katalogi. Gdyby `sudo` honorowało `$PATH` wywołującego, komenda uruchomiona z podniesionymi uprawnieniami mogłaby w rzeczywistości wykonać podstawiony plik zamiast systemowego binarium. `secure_path` wymusza sztywną, znaną listę katalogów, niezależną od tego, co użytkownik ma ustawione we własnym środowisku.

Ćwiczenie 2 — reguła punktowa i test `NOPASSWD` (wymaga sudo)

```
cd /tmp
sudo visudo -f /etc/sudoers.d/test-cwiczenie
```

W edytorze wpisz (podstaw swoją nazwę użytkownika):

```
twoj_user ALL=(ALL) NOPASSWD: /usr/bin/whoami
```

Zapisz, wyjdź, a następnie:

```
sudo -k
sudo whoami
sudo -l -U twoj_user
```

Dlaczego `sudo whoami` nie zapytało o hasło mimo `sudo -k` chwilę wcześniej? Posprzątaj po sobie:

```
sudo rm /etc/sudoers.d/test-cwiczenie
```

Odpowiedź

`sudo -k` unieważnia tylko zapamiętany fakt wcześniejszego podania hasła (cache poświadczeń) — nie ma wpływu na to, czy konkretna reguła w sudoers w ogóle wymaga hasła. Reguła z

`NOPASSWD` jawnie mówi „nie pytaj o hasło dla tej komendy”, więc `-k` nie ma tu nic do wyczyszczenia. `sudo -l -U twoj_user` potwierdzi regułę: `(ALL) NOPASSWD: /usr/bin/whoami`.

Ćwiczenie 3 — audyt lokalnych kont (wymaga sudo)

```
cd /tmp
sudo bash -c '
awk -F: "\$3 >= 1000 {print \$1}" /etc/passwd > users.csv
while IFS=, read -r username; do
    sudo -l -U "$username" 2>/dev/null | grep -q "(ALL)" \
        && echo "$username,sudo" >> sudo_users.csv \
        || echo "$username,brak_sudo" >> sudo_users.csv
done < users.csv
'
cat /tmp/sudo_users.csv
```

Ta wersja filtruje `/etc/passwd` po `UID >= 1000`. Jaką kategorię kont ten filtr celowo pomija, i dlaczego to bywa uzasadnione w audycie?

Odpowiedź

Filtr pomija konta systemowe (UID poniżej 1000: `root` ma UID 0 i jest wyjątkiem wartym osobnego sprawdzenia, ale `bin`, `daemon`, `sshd`, `postgres` i inne konta usług zwykle mają UID w zakresie systemowym). W audycie „kto z ludzi ma sudo” te konta zwykle nie są interesujące — chodzi o zidentyfikowanie osób, a nie usług, które i tak z definicji nie logują się interaktywnie. Pełny audyt bezpieczeństwa powinien mimo to osobno zweryfikować `root` i konta usług, bo część z nich bywa błędnie skonfigurowana z regułami `sudo`.

Pytania kontrolne

1. Dlaczego `/etc/sudoers` nigdy nie edytuje się przez `vim /etc/sudoers`, tylko przez `visudo`?

Odpowiedź

`visudo` blokuje plik na czas edycji, pracuje na kopii i przed zapisaniem uruchamia parser sprawdzający składnię — błąd zatrzymuje zapis i daje szansę na poprawkę. Zwykły edytor zapisze plik z błędem tak, jak jest, a błąd składniowy w `sudoers` blokuje działanie `sudo` dla **wszystkich** użytkowników na hoście od następnego wywołania, nie tylko dla osoby edytującej.

2. Czym różni się `sudo -i` od `sudo su -`, skoro oba dają interaktywną powłokę roota?

Odpowiedź

`sudo -i` to wbudowana funkcja `sudo` — jeden proces, który sam ładuje środowisko i pliki startowe roota. `sudo su -` uruchamia dwa programy z rzędu: `sudo` startuje `su`, a `su` (mając już uprawnienia roota nadane przez `sudo`) przełącza tożsamość dalej. Efekt końcowy dla użytkownika jest niemal identyczny, ale ścieżka audytowa i log `sudo` różnią się — zapisany zostaje fakt uruchomienia `su`, a nie bezpośrednie przejście na roota. W obu przypadkach komendy wykonane już wewnątrz uzyskanej powłoki nie są dalej logowane pojedynczo przez `sudo`.

3. Co dokładnie robi `sudo -k`, a co `sudo -v`, i dlaczego to operacje przeciwne?

Odpowiedź

`sudo -k` unieważnia zapamiętane poświadczenie natychmiast — następne `sudo` zapyta o hasło (chyba że reguła ma `NOPASSWD`). `sudo -v` robi coś przeciwnego: odświeża i przedłuża ważność cache'u bez uruchamiania żadnej komendy, więc kolejne wywołania `sudo` przez najbliższy okres nie zapytają o hasło. Jedno kończy okres łaski, drugie go przedłuża.

4. Świeżo dodałeś użytkownika do grupy `wheel` przez `usermod -aG wheel user2`. Dlaczego `sudo -l -U user2` sprawdzone od razu z tej samej sesji nadal pokazuje brak uprawnień?

Odpowiedź

`sudo -l -U user2` sprawdza aktualną konfigurację `sudoers` i grup systemowych dla `user2`, więc **powinno** pokazać nowe uprawnienia niemal od razu, bo `getent`/NSS odczytuje `/etc/group` na bieżąco. Problem członkostwa „nieaktywnego od razu” dotyczy sesji **samego** `user2` — jeśli `user2` ma już otwartą powłokę sprzed zmiany, ta konkretna powłoka wciąż działa ze starym zestawem grup przypisanym przy logowaniu, dopóki się nie wyloguje i zaloguje ponownie (albo nie użyje `newgrp wheel`).

Powiązane: Atrybuty rozszerzone: chattr, lsattr · Hasła i wygasanie kont: chage, passwd, faillock · SSH: klucze, ssh-agent, config, hartowanie · Zarządzanie użytkownikami i grupami: useradd,

usermod, groups

Źródła: `man 8 sudo` · `man 5 sudoers` · `man 8 visudo` · stare BookStack: Sprawdzenie czy user może korzystać z sudo · OneNote „Ogolne" (Sprawdzenie czy user może korzystać z sudo; Audyt lokalnych userów z uprawnieniem sudo - skryptem)

Revision #17
Created 2026-09-17 21:42:55 UTC by Tomasz Konieczny
Updated 2026-10-03 19:08:35 UTC by Tomasz Konieczny