Atrybuty rozszerzone: chattr, lsattr

◫◫ Poziom: średni

◫◫ Dotyczy: RHEL/Rocky 9–10 (ext4), różnice dla Debian/Ubuntu i SLES zaznaczone

▯ Czas: ~14 min

`chattr` i `lsattr` sterują flagami zapisanymi w samym systemie plików, nie w metadanych `rwx`. Dzięki temu plik może być „chroniony przed zmianą” nawet dla właściciela i dla roota — co bywa ratunkiem, a bywa też sztuczką atakującego.

◫◫ Skróty użyte na tej stronie (6)

Skrót	Co znaczy
ACL	Access Control List: lista, kto ma jakie prawa do pliku lub zasobu
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SELinux	Security-Enhanced Linux: dodatkowa warstwa uprawnień w jądrze RHEL
SLES	SUSE Linux Enterprise Server: płatna dystrybucja Linuksa firmy SUSE, konkurent Rocky/RHEL
SSD	Solid State Drive: dysk półprzewodnikowy
XFS	system plików stworzony przez Silicon Graphics, domyślny w Rocky Linux i RHEL

Fiszka

◫◫ TYLKO ODCZYT

Wyświetlenie atrybutów pliku: lsattr

```
lsattr /etc/ssh/sshd_config
```

Pokazujemy bieżące flagi ustawione na pliku, w stałej kolejności liter, niezależnie od tego, które z nich są aktywne.

Przykład:

```
----i---------e----- /etc/ssh/sshd_config
```

Aktywna flaga to litera na swoim miejscu; myślnik oznacza brak danej flagi. Tutaj aktywne jest tylko `i` (immutable); litera `e` z kolei jest widoczna niemal zawsze na ext4 i nie jest tym, co ustawiliśmy ręcznie — patrz *Jak to działa*.

lsattr = list attributes: pokaż dodatkowe atrybuty pliku w systemie plików, na przykład niezmienność

TYLKO ODCZYT

Wyświetlenie atrybutów samego katalogu: lsattr -d

```
lsattr -d /opt/NazwaKatalogu/
```

Bez `-d` `lsattr` na katalogu pokazałby atrybuty wszystkich plików **wewnątrz** niego. Z `-d` dostajemy wiersz dotyczący samego katalogu jako obiektu.

Przykład:

```
----i---------e----- /opt/produkcja/
```

lsattr = list attributes: pokaż dodatkowe atrybuty pliku w systemie plików, na przykład niezmienność

TYLKO ODCZYT

Rekurencyjne wyświetlenie atrybutów: lsattr -R

```
lsattr -R /opt/NazwaKatalogu/
```

Schodzimy do wszystkich podkatalogów i wypisujemy atrybuty każdego napotkanego pliku. Używamy tego, żeby sprawdzić, czy ochrona nałożona na katalog faktycznie objęła całą jego zawartość, czy tylko sam katalog.

Przykład:

```
----i---------e----- /opt/produkcja/
----i---------e----- /opt/produkcja/config.yml
------------e----- /opt/produkcja/cache.tmp
```

lsattr = list attributes: pokaż dodatkowe atrybuty pliku w systemie plików, na przykład niezmienność

TYLKO ODCZYT

Wyświetlenie atrybutów wraz z plikami ukrytymi: lsattr -a

```
lsattr -a /home/nazwa_usera/
```

Domyślnie `lsattr` pomija pliki, których nazwa zaczyna się od kropki — dokładnie tak samo jak `ls` bez `-a`. Flaga `-a` dopisuje je do wyniku.

Przykład:

```
----i---------e----- /home/adm_user/.bashrc
-------------e----- /home/adm_user/notatki.txt
```

lsattr = list attributes: pokaż dodatkowe atrybuty pliku w systemie plików, na przykład niezmienność

TYLKO ODCZYT

Ustawienie atrybutu niezmienności (immutable): chattr +i

```
sudo chattr +i /etc/ssh/sshd_config
```

Plik nie może być zmodyfikowany, usunięty, przeniesiony ani nadpisany — **nawet przez roota**, dopóki ktoś świadomie nie zdejmie tego atrybutu. Powodzenie tej komendy nie wypisuje niczego na ekranie.

TYLKO ODCZYT

Usunięcie atrybutu niezmienności: chattr -i

```
sudo chattr -i /etc/ssh/sshd_config
```

Odwracamy `+i`. Bez tego kroku każda kolejna próba edycji pliku — nawet przez `vim` z uprawnieniami roota — zakończy się błędem zapisu.

TYLKO ODCZYT

Ustawienie atrybutu tylko-do-dopisywania (append only): chattr +a

```
sudo chattr +a /var/log/NazwaUslugi.log
```

Do pliku można wyłącznie dopisywać na końcu (`>>`); modyfikacja lub skasowanie istniejącej treści jest zablokowane, tak samo dla roota jak dla właściciela. Naturalny wybór dla plików logów, których integralność ma znaczenie audytowe.

□□ TYLKO ODCZYT

Wyłączenie aktualizacji czasu dostępu: chattr +A

```
sudo chattr +A /srv/cache/NazwaPliku
```

Jądro normalnie aktualizuje znacznik `atime` przy każdym odczycie pliku, co przy plikach czytanych tysiące razy na sekundę (cache, bazy danych) generuje niepotrzebny ruch zapisu na dysku. `+A` wyłącza tę aktualizację dla pojedynczego pliku — odpowiednik opcji montowania `noatime`, ale bez przemontowywania całego systemu plików.

□□ TYLKO ODCZYT

Włączenie kompresji transparentnej: chattr +c

```
sudo chattr +c NazwaPliku
```

Jądro kompresuje blok danych przed zapisem na dysk i dekompresuje przy odczycie — działa niewidocznie dla aplikacji, o ile system plików to wspiera (na standardowym ext4 z RHEL/Rocky ta funkcja zwykle nie jest wkompilowana; działa pewniej na Btrfs).

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

□□ TYLKO ODCZYT

Pominięcie pliku przy dump: chattr +d

```
sudo chattr +d /var/cache/NazwaUslugi/tmp.cache
```

Narzędzie `dump` (klasyczny mechanizm kopii zapasowych na poziomie bloków, dziś rzadziej spotykany niż `tar`/`rsync`) pomija plik z tym atrybutem. Przydatne dla danych tymczasowych, których odtwarzanie z backupu nie ma sensu.

TYLKO ODCZYT

Włączenie bezpiecznego usuwania: chattr +s

```
sudo chattr +s /root/NazwaPliku
```

Przy usunięciu pliku jądro powinno nadpisać jego bloki zerami zamiast po prostu zwolnić je do ponownego użycia — utrudnia to odzyskanie danych narzędziami do undelete. W praktyce na współczesnych systemach plików z dziennikiem (ext4) i na dyskach SSD z TRIM-em ta gwarancja jest zawodna, więc traktujemy `+s` jako dodatkową warstwę, nie jako pewność.

SSD = Solid State Drive: dysk półprzewodnikowy

TYLKO ODCZYT

Włączenie zachowania danych do odzyskania: chattr +u

```
sudo chattr +u /home/nazwa_usera/NazwaPliku
```

Odwrotność `+s`: system stara się zachować zawartość usuniętego pliku, żeby dało się ją odtworzyć narzędziem odzyskującym. Wymaga wsparcia ze strony systemu plików i narzędzia odzyskującego — na standardowym ext4 bez dodatkowych narzędzi efekt bywa ograniczony.

TYLKO ODCZYT

Włączenie synchronicznych aktualizacji: chattr +S

```
sudo chattr +S /etc/NazwaPliku
```

Każda zmiana pliku jest natychmiast zapisywana fizycznie na dysk, zamiast czekać w buforze jądra do okresowego `sync`. Zwiększa odporność na utratę danych przy nagłym zaniku zasilania, kosztem wydajności zapisu — podobny efekt do montowania z opcją `sync`, ale dla jednego pliku.

TYLKO ODCZYT

Rekurencyjna zmiana atrybutu: chattr -R

```
sudo chattr -R +i /etc/NazwaKatalogu/
```

Nakładamy (lub zdejmujemy, z `-i`) atrybut na katalog i na wszystko, co się w nim znajduje, włącznie z podkatalogami. Bez `-R` atrybut trafiłby tylko na sam katalog jako obiekt — pliki wewnątrz zostałyby edytowalne.

```
sudo chattr -R -i /etc/NazwaKatalogu/
```

Zdjęcie ochrony z całego drzewa katalogów przed planowaną aktualizacją.

◫◫ TYLKO ODCZYT

Odczyt rozszerzonych atrybutów użytkownika: getfattr -d

```
getfattr -d /srv/dane/NazwaPliku
```

To zupełnie inny mechanizm niż `chattr` — pary klucz-wartość przypisane do pliku w przestrzeni nazw `user`, `security`, `system` albo `trusted`. `-d` (dump) wypisuje wartości atrybutów, nie tylko ich nazwy.

Przykład:

```
# file: srv/dane/plik.csv
user.zrodlo="import-2026-09"
user.zatwierdzil="adm_user"
```

Bez `-d` `getfattr` pokazałby same nazwy atrybutów, bez ich zawartości.

◫◫ ZMIENIA SYSTEM

Ustawienie atrybutu użytkownika: setfattr -n -v

```
setfattr -n user.zrodlo -v "import-2026-09" /srv/dane/NazwaPliku
```

`-n` podaje pełną nazwę atrybutu (musi zaczynać się od przestrzeni nazw, tu `user.`), `-v` podaje wartość. Przestrzeń `user` jest dostępna zwykłemu właścicielowi pliku bez uprawnień roota — pozostałe przestrzenie (`security`, `system`, `trusted`) są zarezerwowane dla mechanizmów jądra (SELinux, ACL, capabilities) i wymagają uprawnień administracyjnych.

◫◫ **SELinux** = Security-Enhanced Linux: dodatkowa warstwa uprawnień w jądrze RHEL · **ACL** = Access Control List: lista, kto ma jakie prawa do pliku lub zasobu

TYLKO ODCZYT

Wszystkie flagi

Atrybut	Nazwa	Co robi
`i`	immutable	blokuje modyfikację, usunięcie, zmianę nazwy i dowiązania — nawet dla roota
`a`	append only	pozwala wyłącznie dopisywać na końcu pliku
`A`	no atime update	nie aktualizuje czasu ostatniego dostępu
`c`	compressed	transparentna kompresja na poziomie jądra (zależna od systemu plików)
`d`	no dump	pomijany przez narzędzie `dump` przy tworzeniu kopii
`s`	secure deletion	próba nadpisania zerami bloków przy usuwaniu (zawodna na SSD/ext4 z dziennikiem)
`u`	undeletable	próba zachowania danych do odzyskania po usunięciu
`S`	synchronous updates	każdy zapis trafia od razu na dysk, bez buforowania
`e`	extents	plik używa nowoczesnej mapy ekstentów zamiast klasycznych bloków — ustawiane automatycznie przez ext4, nie ręcznie

Uważaj

- `chattr +i` blokuje **roota** tak samo jak zwykłego użytkownika — to nie jest zabezpieczenie omijalne przez `sudo`.
- Atrybuty rozszerzone (`chattr`) i atrybuty użytkownika (`getfattr` / `setfattr`) to dwa niezależne mechanizmy — usunięcie jednego nie rusza drugiego.
- Przy kopiowaniu między różnymi systemami plików (np. `cp` na FAT albo transfer przez sieć bez `-X` / `--xattrs`) atrybuty zwykle giną bez ostrzeżenia.

SSD = Solid State Drive: dysk półprzewodnikowy

Z życia wzięte

Z ŻYCIA WZIĘTE

O 3:20 automat wdrożeniowy na app01 próbuje nadpisać plik konfiguracyjny podczas nocnego wdrożenia i dostaje `Operation not permitted`, mimo że działa jako root. Rano Tomek sprawdza `lsattr` i widzi flagę `i` założoną tydzień wcześniej po poprzednim incydencie, o której cały zespół zdążył zapomnieć. Zdejmuje atrybut na chwilę, wdrożenie przechodzi, i zakłada go z powrotem zaraz po zapisaniu zmian.

Po co to istnieje

Pojęcie: atrybut systemu plików. Standardowe uprawnienia `rwx` odpowiadają na pytanie „kto może czytać, pisać i wykonywać ten plik” — to jest lista kontroli dostępu przypisana do użytkownika, grupy i reszty świata. Atrybut systemu plików to zupełnie inna warstwa: flaga zapisana bezpośrednio w metadanych pliku na dysku, która **wyłącza pewne operacje dla wszystkich**, niezależnie od tego, kim jest wywołujący. Prawa `rwx` mówią „kto może”, atrybut mówi „czy w ogóle wolno, komukolwiek”. `root` może zawsze zmienić uprawnienia `rwx` na dowolnym pliku (bo `root` z definicji omija kontrolę dostępu), ale atrybutu `i` nie omija — musi go najpierw świadomie zdjąć.

Pojęcie: niezmienność. Niezmienność (immutability) to stan pliku, w którym żadna operacja modyfikująca — zapis, usunięcie, zmiana nazwy, utworzenie twardego dowiązania — nie jest dozwolona, dopóki ten stan nie zostanie jawnie zniesiony. To pojęcie z ogólnej informatyki (dane niezmienne pojawiają się też w językach programowania i w bazach danych), a `chattr +i` jest jego konkretną implementacją na poziomie systemu plików Linuksa.

Te mechanizmy istnieją, bo same uprawnienia `rwx` nie chronią przed jedną konkretną sytuacją: pomyłką administratora zalogowanego jako `root`. Skrypt uruchomiony przez pomyłkę z `sudo`, literówka w `rm -rf`, automat wdrożeniowy, który nadpisuje plik konfiguracyjny bez pytania — wszystko to normalnie przechodzi bez przeszkód, bo `root` ma prawo zrobić wszystko. `chattr +i` na krytycznym pliku konfiguracyjnym stawia dodatkową przeszkodę dokładnie w tym miejscu, gdzie `rwx` jest bezradne.

Analogia

Wyobraź sobie **plombę na liczniku gazu albo prądu**.

Sam licznik ma zamek, obudowę, śrubki — to jest jego „`rwx`”: kto ma klucz i uprawnienia, może otworzyć obudowę i majstrować przy mechanizmie. Plomba to coś innego. To ołowiana lub plastikowa nakładka z unikalnym numerem, założona przez inspektora, która nie chroni przed otwarciem obudowy w sensie mechanicznym — śrubokręt wciąż da radę odkręcić śrubki. Plomba sprawia coś innego: jakakolwiek próba otwarcia zostawia widoczny, nieusuwalny ślad. Nikt, łącznie z właścicielem domu, nie może „legalnie” zdjąć plomby bez wezwania inspektora, który ją zdejmie i założy nową.

`chattr +i` działa dokładnie tak. Nie zabrania fizycznie zapisu na dysku (to samo jądro wciąż ma dostęp do bloków), ale każda próba modyfikacji przez normalne wywołania systemowe zostaje odrzucona na poziomie jądra, zanim dotrze do samego zapisu — łącznie z próbami roota. Żeby legalnie „zdjąć plombę”, trzeba wykonać osobny, świadomy krok: `chattr -i`. I tu pułapka: dokładnie z tego samego powodu atakujący, który już zdobył dostęp roota, może założyć `+i` na swoim backdoorze albo na zmodyfikowanym pliku konfiguracyjnym — administrator próbujący go usunąć czy przywrócić oryginał dostanie odmowę i, jeśli nie zna `chattr`, straci czas, szukając problemu w złym miejscu (uprawnieniach, SELinuksie, ACL), zamiast sprawdzić `lsattr`.

Jak to działa

Rozwiń: jak to działa pod spodem

Flagi `chattr` są zapisywane w samym i-węźle (inode) pliku — w polu, które istnieje niezależnie od standardowych bitów uprawnień `rwx` i od właściciela. Jądro sprawdza te flagi na najwcześniejszym możliwym etapie obsługi wywołania systemowego (`write()`, `unlink()`, `rename()`) — zanim w ogóle dojdzie do sprawdzenia, czy wywołujący ma prawo `w`. To dlatego atrybut wygrywa z uprawnieniami i z tożsamością roota: kontrola uprawnień w ogóle nie zostaje osiągnięta, jeśli flaga blokuje operację wcześniej.

Nie każdy system plików obsługuje każdy atrybut. Mechanizm `chattr`/`lsattr` pochodzi z rodziny ext2/ext3/ext4 i jest tam zaimplementowany najpełniej; XFS (domyślny na RHEL/Rocky) rozumie tylko część atrybutów — praktycznie `i` i `a` działają, reszta (`c`, `s`, `u`) jest ignorowana albo nieobsługiwana. Btrfs ma własną, bogatszą obsługę kompresji przez `chattr +c`. Zawsze warto sprawdzić `lsattr` po ustawieniu atrybutu, żeby upewnić się, że system plików faktycznie go przyjął, a nie po cichu zignorował.

Litera `e` widoczna niemal na każdym pliku ext4 w wyniku `lsattr` to atrybut ekstentów (extents) — nowoczesny sposób mapowania bloków pliku na dysku, ustawiany automatycznie przez system plików od ext4 wzwyż. Nie jest to nic, co ustawiasz ręcznie, i nie da się go zdjąć przez `chattr -e` bez przebudowy systemu plików — pojawia się w wyniku `lsattr`, żeby pokazać

pełny, aktualny zestaw flag pliku, nie tylko te „administracyjne”.

`getfattr`/`setfattr` to osobny mechanizm — rozszerzone atrybuty (extended attributes, xattr) zapisane jako dowolne pary klucz-wartość, pogrupowane w przestrzenie nazw: `user` (dostępna zwykłemu właścicielowi), `security` (używana przez SELinux do przechowywania kontekstu bezpieczeństwa pliku), `system` (używana m.in. przez ACL-e) i `trusted` (widoczna tylko dla procesów z uprawnieniami `CAP_SYS_ADMIN`). `chattr` i `getfattr`/`setfattr` są od siebie niezależne: plik może mieć ustawiony `chattr +i` i jednocześnie zero atrybutów `user.*`, albo odwrotnie.

Krok po kroku

1. Ochrona krytycznego pliku konfiguracyjnego.

```
sudo lsattr /etc/ssh/sshd_config
```

```
-------------e----- /etc/ssh/sshd_config
```

Na start żadnych atrybutów administracyjnych.

```
sudo chattr +i /etc/ssh/sshd_config
sudo lsattr /etc/ssh/sshd_config
```

```
----i---------e----- /etc/ssh/sshd_config
```

Litera `i` na swoim miejscu potwierdza, że atrybut się przyjął.

2. Próba edycji — i dlaczego to nie jest błąd uprawnień.

```
sudo vim /etc/ssh/sshd_config
```

`vim` otworzy plik do odczytu bez problemu, ale przy próbie zapisu (`:wq`) dostaniesz komunikat o błędzie zapisu na poziomie systemu — mimo że `sudo` daje pełne prawa `rwx`. To jest moment, w którym administrator bez wiedzy o `chattr` zaczyna podejrzewać SELinux albo złe uprawnienia pliku.

3. Zdjęcie ochrony przed planowaną zmianą.

```
sudo chattr -i /etc/ssh/sshd_config
sudo vim /etc/ssh/sshd_config
sudo chattr +i /etc/ssh/sshd_config
```

Dobra praktyka: zdejmujesz atrybut tuż przed edycją i zakładasz go z powrotem od razu po zapisaniu zmian, żeby okno bez ochrony było jak najkrótsze.

4. Log tylko-do-dopisywania.

```
sudo chattr +a /var/log/aplikacja.log
echo "$(date) test wpisu" | sudo tee -a /var/log/aplikacja.log
```

```
pon. 21 wrz 2026 10:14:02 CEST test wpisu
```

Dopisanie działa normalnie. Próba `sudo truncate -s 0 /var/log/aplikacja.log` albo edycja w środku pliku zostanie odrzucona — plik może tylko rosnąć.

5. Ochrona całego katalogu produkcyjnego.

```
sudo chattr -R +i /opt/produkcja/
sudo lsattr -R /opt/produkcja/ | head -3
```

```
----i---------e----- /opt/produkcja/
----i---------e----- /opt/produkcja/config.yml
----i---------e----- /opt/produkcja/app.jar
```

Wszystkie pliki w drzewie dostały atrybut, nie tylko sam katalog.

Częste błędy

`rm` na pliku z `+i` — komunikat, który myli z problemem uprawnień.

```
$ sudo rm /etc/ssh/sshd_config
rm: nie można usunąć '/etc/ssh/sshd_config': Operation not permitted
```

Ten komunikat wygląda identycznie jak przy złych uprawnieniach albo złym właścicielu — a `sudo` przecież powinno dawać pełne prawa. Administrator traci czas na sprawdzanie `ls -l`, SELinuksa, ACL-i, zanim w ogóle pomyśli o `lsattr`. Odruch powinien być odwrotny: **każdy `Operation not permitted` przy operacji wykonywanej jako root to sygnał, żeby najpierw sprawdzić `lsattr`**, zanim zacznie się szukać gdzie indziej.

```
lsattr /etc/ssh/sshd_config
sudo chattr -i /etc/ssh/sshd_config
```

Edycja pliku `+a` w środku, nie na końcu.

```
$ sudo sed -i 's/error/warning/' /var/log/aplikacja.log
sed: nie można zmienić nazwy /var/log/sedXXXXXX na /var/log/aplikacja.log: Operation not
permitted
```

`sed -i` działa przez utworzenie nowego pliku tymczasowego i podmianę nazwy — a atrybut `a` zabrania nawet tej podmiany, bo formalnie usuwa oryginalny plik. Jedyna dozwolona operacja to dopisywanie na końcu (`>>`, `tee -a`).

Atrybut wykorzystany przez atakującego do utrudnienia sprzątania.

Po włamaniu, jeśli atakujący zdobył chwilowo uprawnienia roota, jednym z możliwych kroków jest ustawienie `chattr +i` na podrzuconym backdoorze albo na zmodyfikowanym pliku (np. `~/.ssh/authorized_keys`, `/etc/crontab`). Administrator próbujący posprzątać po incydencie widzi `Operation not permitted` przy próbie usunięcia albo nadpisania złośliwego pliku i — jeśli nie sprawdzi `lsattr` — może błędnie uznać, że problem został już usunięty (bo np. zamknął sesję atakującego), podczas gdy sam plik nadal tam jest, chroniony przed czyszczeniem. Dlatego procedura reagowania na incydent powinna zawierać `lsattr -R` na podejrzanych katalogach jako jeden z pierwszych kroków.

Utrata atrybutów przy kopiowaniu lub transferze.

```
$ cp /etc/ssh/sshd_config /mnt/backup_fat32/sshd_config.bak
$ lsattr /mnt/backup_fat32/sshd_config.bak
lsattr: Operacja niewspierana przez system podczas odczytywania flag
/mnt/backup_fat32/sshd_config.bak
```

System plików FAT32 (typowy dla nośników wymiennych) w ogóle nie zna pojęcia rozszerzonych atrybutów ext4 — kopia trafia tam „goła”. To samo dotyczy standardowego `rsync` bez `-X`/`--xattrs` między dwoma katalogami na różnych partycjach oraz większości narzędzi do archiwizacji bez specjalnych flag. Jeśli atrybuty mają znaczenie operacyjne (np. `+a` na logach audytowych), trzeba to udokumentować i pilnować przy migracjach — sam plik „wygląda tak samo”, ale ochrona zniknęła bez żadnego ostrzeżenia.

Zapomnienie `-R` przy ochronie katalogu.

```
$ sudo chattr +i /opt/produkcja/
$ echo "test" > /opt/produkcja/nowy_plik.txt
$ lsattr /opt/produkcja/nowy_plik.txt
----------e----- /opt/produkcja/nowy_plik.txt
```

Bez `-R` atrybut trafił wyłącznie na sam katalog jako obiekt (co i tak zablokuje tworzenie/usuwanie plików bezpośrednio w nim, bo to modyfikacja katalogu), ale pliki, które już w nim leżały przed nałożeniem atrybutu, zostają nietknięte i w pełni edytowalne. Administrator, który sprawdził tylko `lsattr -d`

, może błędnie założyć, że cała zawartość jest chroniona.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie, w katalogu `/tmp`. Zmiana atrybutów wymaga `sudo` niezależnie od tego, czyją własnością jest plik — o tym właśnie jest ta strona.

Ćwiczenie 1 — niezmienność i komunikat, który myli

```
cd /tmp
echo "dane produkcyjne" > chroniony.txt
sudo chattr +i chroniony.txt
lsattr chroniony.txt
rm chroniony.txt
```

Jaki dokładnie komunikat zobaczysz i dlaczego pierwszy odruch „zła komenda `rm`, spróbuję z `sudo`” nic nie zmieni?

Odpowiedź

```
rm: nie można usunąć 'chroniony.txt': Operation not permitted
```

`sudo rm chroniony.txt` zwróci dokładnie to samo — atrybut `i` blokuje operację niezależnie od tego, kim jest wywołujący, więc podniesienie uprawnień nic nie daje. Jedyna droga to zdjęcie atrybutu:

```
sudo chattr -i chroniony.txt
rm chroniony.txt
```

Ćwiczenie 2 — plik tylko-do-dopisywania

```
cd /tmp
echo "wpis 1" > log_testowy.txt
```

```
sudo chattr +a log_testowy.txt
echo "wpis 2" >> log_testowy.txt
cat log_testowy.txt
sed -i 's/wpis 1/ZMIENIONY/' log_testowy.txt
```

Które polecenie się powiedzie, a które nie, i dlaczego `sed -i` zawodzi mimo że próbuje tylko podmienić fragment tekstu, a nie skasować cały plik?

Odpowiedź

`echo ... >>` się powiedzie — dopisywanie na końcu jest jedyną dozwoloną operacją przy `+a`. `sed -i` zawodzi komunikatem `Operation not permitted`, bo mimo że logicznie „tylko zmienia fragment", mechanicznie działa przez utworzenie nowego pliku tymczasowego i zastąpienie nim oryginału (`rename()`) — a to jest operacja usunięcia oryginalnego pliku, zabroniona przez atrybut `a` tak samo jak zwykłe `rm`. Sprzątanie:

```
sudo chattr -a log_testowy.txt
rm log_testowy.txt
```

Pytania kontrolne

1. Dlaczego `chattr +i` na pliku nie jest tym samym co `chmod 000` na tym pliku?

Odpowiedź

`chmod 000` odbiera wszystkie prawa `rwx` zwykłym użytkownikom, ale `root` i tak może odczytać, zapisać i usunąć plik — kontrola dostępu w Linuksie z definicji nie dotyczy roota. `chattr +i` blokuje operację na poziomie jądra, zanim dojdzie w ogóle do sprawdzenia uprawnień, więc obowiązuje również roota. To dwa niezależne mechanizmy: jeden kontroluje „kto", drugi kontroluje „czy w ogóle".

2. Masz katalog z atrybutem `+i` nałożonym bez `-R`. Czy możesz dodać nowy plik do tego katalogu?

Odpowiedź

Nie — dodanie pliku to modyfikacja zawartości katalogu (zapis nowego wpisu w strukturze katalogu), a atrybut `i` na samym katalogu blokuje właśnie takie modyfikacje, niezależnie od tego, czy objął też pliki wewnątrz. Zablokowane jest tworzenie, usuwanie i zmiana nazwy

wewnątrz takiego katalogu; pliki, które już tam były, zachowują swój własny, niezależny stan atrybutu.

3. Po incydencie bezpieczeństwa podejrzewasz, że atakujący zostawił plik, którego nie możesz usunąć. Jaka jedna komenda diagnostyczna sprawdzi to najszybciej?

Odpowiedź

`lsattr -R` na podejrzanym katalogu (albo `lsattr` na konkretnym pliku) — jeśli atrybut `i` albo `a` jest aktywny, `rm` będzie zwracać `Operation not permitted` mimo pełnych uprawnień roota, a to jest dokładnie objaw, który `lsattr` natychmiast wyjaśnia.

Powiązane: Uprawnienia rwx: chmod, chown, umask · ACL: getfacl, setfacl · SELinux: konteksty i tryby · sudo i /etc/sudoers: konfiguracja i audyt

Źródła: `man 1 chattr` · `man 1 lsattr` · `man 1 getfattr` · `man 1 setfattr` · `man 5 xattr` · stare BookStack: Atrybuty plików i katalogów

---

Revision #16
Created 2026-09-17 21:42:53 UTC by Tomasz Konieczny
Updated 2026-10-03 19:08:33 UTC by Tomasz Konieczny