umask, setuid, setgid, sticky bit

Poziom: średni **Dotyczy: wszystkie dystrybucje** **Czas: ~16 min**

`chmod`, `chown` i `chgrp` z poprzedniej strony ustawiają uprawnienia jawnie, po fakcie. Ta strona pokazuje trzy mechanizmy, które działają inaczej: `umask` decyduje, jakie uprawnienia dostanie plik, zanim jeszcze powstanie, a setuid, setgid i sticky bit to trzy dodatkowe bity, które zmieniają znaczenie zwykłego `rwx` w konkretnych, wąskich sytuacjach.

Skróty użyte na tej stronie (6)

Skrót	Co znaczy
CIFS	Common Internet File System: starsza nazwa protokołu współdzielenia plików Windows, dziś częściej nazywanego SMB, pojawia się w wynikach poleceń FreeIPA dotyczących zaufania z Active Directory
GID	Group Identifier: numer grupy systemowej, drugi obok UID element decydujący o prawach dostępu do pliku
IT	Information Technology: dział albo praca związana z komputerami i infrastrukturą firmy
NFS	Network File System: udostępnianie katalogów po sieci w świecie Linuksa
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
UID	User ID: numer użytkownika w Linuksie

Fiszka

TYLKO ODCZYT

Odczyt bieżącej maski: umask

```
umask
```

Wypisujemy maskę w postaci ósemkowej — liczbę, którą system odejmuje od maksymalnych domyślnych uprawnień przy tworzeniu nowego pliku lub katalogu.

Przykład:

```
0022
```

umask = user mask: domyślna maska ograniczająca uprawnienia nowo tworzonych plików i katalogów

TYLKO ODCZYT

Odczyt maski w postaci symbolicznej: umask -S

```
umask -S
```

To samo, czytelniej — bez przeliczania cyfr w głowie.

Przykład:

```
u=rwx,g=rx,o=rx
```

umask = user mask: domyślna maska ograniczająca uprawnienia nowo tworzonych plików i katalogów

TYLKO ODCZYT

Ustawienie maski standardowej: umask 022

```
umask 022
```

Ustawiamy maskę na bieżącą powłokę. Nowe pliki dostaną `644` (666 z wyłączeniem bitów wskazanych przez `022`), nowe katalogi `755` (777 z tym samym wyłączeniem). Efekt jest tymczasowy — znika po zakończeniu sesji, jeśli nie zostanie zapisany w pliku startowym.

umask = user mask: domyślna maska ograniczająca uprawnienia nowo tworzonych plików i katalogów

TYLKO ODCZYT

Ustawienie maski restrykcyjnej: umask 077

```
umask 077
```

Nowe pliki: `600`, nowe katalogi: `700` — dostęp wyłącznie dla właściciela. Typowe ustawienie dla kont usługowych, które tworzą pliki z danymi wrażliwymi: kluczami, tokenami, zrzutami baz danych.

umask = user mask: domyślna maska ograniczająca uprawnienia nowo tworzonych plików i katalogów

TYLKO ODCZYT

Ustawienie trwałe w `/etc/profile`

```
grep -n umask /etc/profile
```

`umask` wpisany do komendy działa tylko w bieżącej sesji. Wpis w `/etc/profile` ustawia go globalnie, dla każdej powłoki logowania w systemie — miejsce, w którym większość dystrybucji już ma domyślny wpis `umask 022`.

Przykład:

```
27:umask 022
```

grep = global regular expression print: szukaj wzorca w tekście

ZMIENIA SYSTEM

Ustawienie trwałe w `~/.bashrc`

```
echo 'umask 077' >> ~/.bashrc
```

To samo, ale dla jednego konkretnego użytkownika, niezależnie od globalnego ustawienia w `/etc/profile`. Kolejność wczytywania plików startowych ma znaczenie — plik wczytany później nadpisuje wcześniejszy, więc wpis w `~/.bashrc` użytkownika wygrywa z globalnym `/etc/profile`.

ZMIENIA SYSTEM

Ustawienie setuid na pliku: chmod u+s

```
chmod u+s /usr/bin/passwd
```

Ustawiamy bit setuid. Plik uruchomiony przez kogokolwiek zaczyna działać z efektywnym UID-em **właściciela pliku**, a nie użytkownika, który go uruchomił. W `ls -l` widać to jako `s` w miejscu, gdzie zwykle jest `x` właściciela.

Przykład:

```
-rwsr-xr-x. 1 root root 35040 kwi 12 2025 /usr/bin/passwd
```

chmod = change mode: zmień prawa dostępu

ZMIENIA SYSTEM

Ustawienie setgid na pliku: chmod g+s

```
chmod g+s /usr/bin/wall
```

Analogicznie do setuid, ale proces dziedziczy GID właściciela pliku zamiast GID-u uruchamiającego. Rzadziej spotykane niż setuid, bo mniej programów wymaga podniesienia uprawnień grupowych zamiast pełnych praw roota.

GID = Group Identifier: numer grupy systemowej, drugi obok UID element decydujący o prawach dostępu do pliku · **chmod** = change mode: zmień prawa dostępu

ZMIENIA SYSTEM

Ustawienie setgid na katalogu (dziedziczenie grupy): chmod g+s

```
chmod g+s /srv/NazwaKatalogu
```

Na katalogu setgid oznacza coś zupełnie innego niż na pliku: każdy nowy plik i podkatalog utworzony wewnątrz dziedziczy **grupę katalogu nadrzędnego**, a nie grupę główną osoby, która go tworzy. To fundament katalogów współdzielonych przez zespół — bez tego bitu każdy członek zespołu tworzyłby pliki ze swoją własną grupą główną, i reszta zespołu traciłaby do nich dostęp grupowy.

Przykład:

```
drwxrws---. 2 adm_user deweloperzy 4096 wrz 17 10:00 /srv/wspolne
```

chmod = change mode: zmień prawa dostępu

ZMIENIA SYSTEM

Ustawienie sticky bit: chmod +t

```
chmod +t /srv/NazwaKatalogu
```

W katalogu ze stickybitem tylko właściciel danego pliku (albo właściciel katalogu, albo root) może ten plik usunąć lub zmienić mu nazwę — nawet jeśli inni mają prawo zapisu do całego katalogu. W `ls -l` widać to jako `t` na końcu uprawnień pozostałych.

Przykład:

```
drwxrwxrwt. 10 root root 4096 wrz 17 10:00 /tmp
```

chmod = change mode: zmień prawa dostępu

ZMIENIA SYSTEM

Audyt plików z setuid: find -perm -4000

```
find / -xdev -perm -4000 -type f 2>/dev/null
```

Wypisujemy listę wszystkich plików z bitem setuid w systemie — standardowy krok audytu bezpieczeństwa. `-xdev` ogranicza wyszukiwanie do jednego systemu plików, żeby nie wleźć na sieciowe montowania; `2>/dev/null` tłumi zalew `Permission denied` z katalogów niedostępnych dla bieżącego użytkownika.

Przykład:

```
/usr/bin/su
/usr/bin/sudo
/usr/bin/passwd
/usr/bin/mount
/usr/bin/umount
```

find = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

ZMIENIA SYSTEM

Audyt plików i katalogów z setgid: find -perm -2000

```
find / -xdev -perm -2000 2>/dev/null
```

To samo dla setgid — obejmuje zarówno rzadkie pliki wykonywalne z tym bitem, jak i (znacznie częściej) katalogi współdzielone, np. `/var/mail` czy foldery projektowe zakładane ręcznie przez administratorów.

Przykład:

```
/var/mail
/srv/wspolne
/usr/bin/wall
```

find = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

TYLKO ODCZYT

Wszystkie komendy

Komenda	Co robi
`umask` / `umask -S`	odczyt bieżącej maski, ósemkowo / symbolicznie
`umask 022`, `umask 077`	ustawienie maski na bieżącą sesję
wpis w `/etc/profile`	trwała maska dla całego systemu
wpis w `~/.bashrc`	trwała maska dla jednego użytkownika, nadpisuje globalną
`chmod u+s`	setuid na pliku — uruchomienie na prawach właściciela
`chmod g+s` (plik)	setgid na pliku — uruchomienie na prawach grupy właściciela
`chmod g+s` (katalog)	dziedziczenie grupy przez nowe pliki wewnątrz
`chmod +t`	sticky bit — kasować może tylko właściciel pliku
`find / -perm -4000`	audyt plików z setuid
`find / -perm -2000`	audyt plików/katalogów z setgid

Uważaj

- `umask` wpisany w terminalu obowiązuje tylko do końca sesji — bez wpisu w pliku startowym znika po wylogowaniu.
- Setuid na skrypcie powłoki (nie na binarce) jest przez jądro **ignorowany** od dekad — patrz *Częste błędy*.
- `chmod 777 /tmp` kasuje przy okazji sticky bit, jeśli robimy to bez zastanowienia — patrz *Częste błędy*.

Z życia wzięte

Z ŻYCIA WZIĘTE

O 13:00 pani Halina prosi o zmianę swojego hasła do systemu księgowego, bo stare wygasło. Tomek pokazuje jej zwykłe `passwd`, które mimo że uruchamia je zwykły użytkownik, potrafi zapisać nowy hash do `/etc/shadow`, plik dostępny normalnie wyłącznie dla roota. Dzięki bitowi setuid proces na chwilę działa jako root, ale sam program pilnuje, żeby Halina zmieniła wyłącznie własne hasło, nie cudze.

Po co to istnieje

Pojęcie: maska uprawnień. Umask to wartość, która **nie ustawia** żadnych uprawnień wprost — odejmuje bity z maksymalnych domyślnych uprawnień, jakie system nadałby nowo tworzonemu obiektowi. Dla plików ten punkt startowy to `666` (`rw-rw-rw-`), dla katalogów `777` (`rwxrwxrwx`) — same programy tworzące plik decydują o tym maksimum przy wywołaniu systemowym `open()`, umask tylko przycina wynik. To dlatego świeżo utworzony plik tekstowy nigdy nie ma bitu `x`, nawet przy `umask 000`: żaden zwykły program nie prosi o `x` przy tworzeniu pliku, więc nie ma co przycinać. Katalogi startują z `x` w maksimum, bo bez tego bitu nie dałoby się do nich w ogóle wejść.

Pojęcie: bit specjalny. Poza dziewięcioma bitami `rwx` dla trzech odbiorców, każdy plik ma jeszcze trzy dodatkowe bity, które modyfikują znaczenie zwykłych uprawnień w konkretnych sytuacjach: setuid, setgid i sticky bit. Nazywa się je bitami specjalnymi, bo nie mieszczą się w prostym modelu „kto może czytać/pisać/wykonywać” — zamiast tego zmieniają, **jako kto** proces się uruchamia albo **kto** może usunąć plik z katalogu. Wszystkie trzy razem zajmują jedną dodatkową cyfrę ósemkową przed standardowymi trzema — stąd zapis `chmod 4755` zamiast `chmod 755` dla pliku z setuid.

Analogia

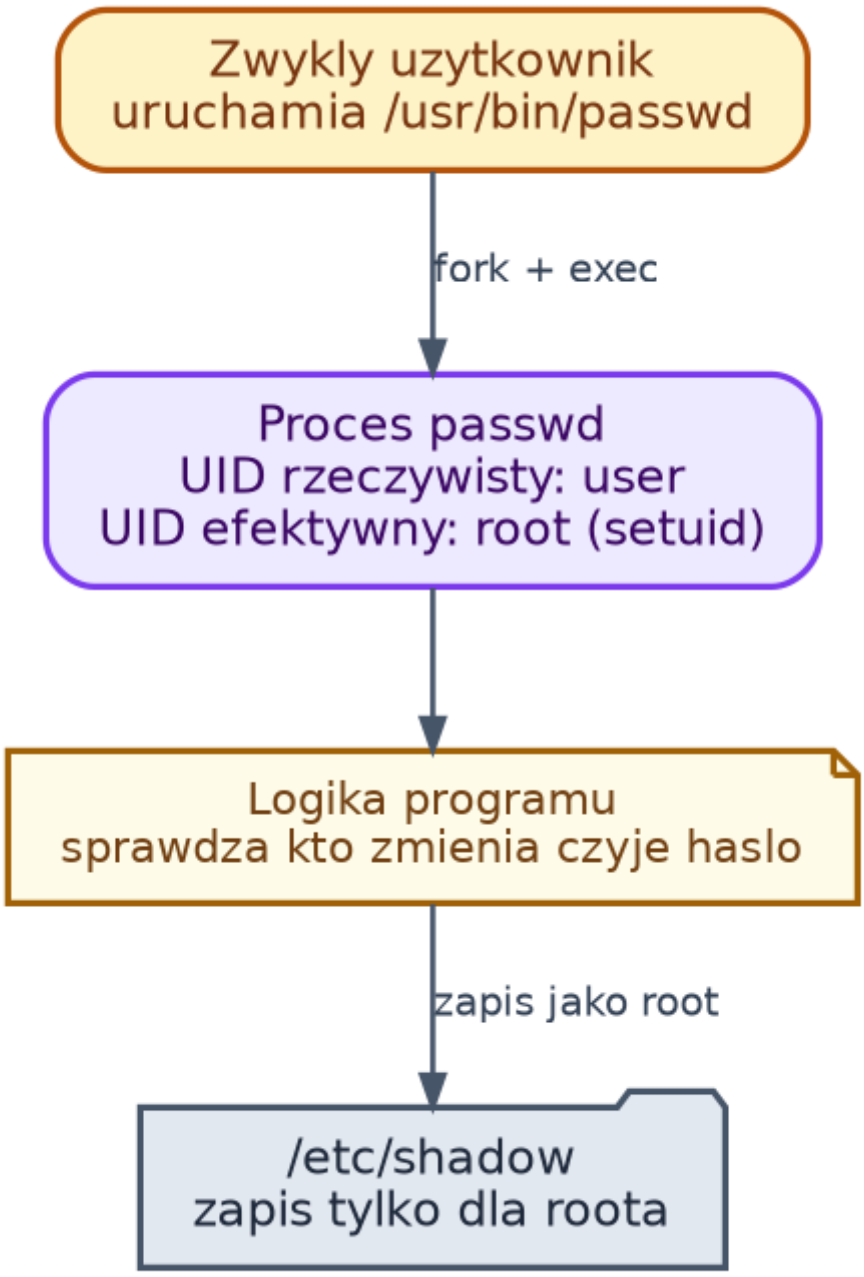
`umask` działa jak **domyślne ustawienie ksero w firmowej drukarni**: dział IT skonfigurował maszynę tak, żeby każdy nowy wydruk domyślnie miał znak wodny „POUFNE” i nie trafiał do wspólnej tacki, tylko do zamkniętej skrytki. Nikt nie ustawia tego ręcznie przy każdym wydruku — konfiguracja działa w tle, zanim ktokolwiek wciśnie „drukuj”.

Bity specjalne to inny rodzaj mechanizmu — **przepustka tymczasowa**, a nie stałe ustawienie. Setuid na `/usr/bin/passwd` jest jak przepustka do archiwum kadr, którą dostaje każdy, kto korzysta z konkretnego automatu do zmiany hasła — sam automat ma dostęp do zamkniętej szafy z aktami, użytkownik automatu nie dostaje własnego klucza do tej szafy, tylko na chwilę „pożycza” uprawnienia maszyny, żeby wykonać jedną, ściśle określoną operację. Sticky bit na `/tmp` to z kolei recepcjonista pilnujący wspólnej szafki na bagaż: każdy gość może włożyć tam swoją walizkę, ale zabrać może tylko własną — mimo że szafka technicznie jest otwarta dla wszystkich.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: efektywny UID.** Każdy proces ma dwa (a właściwie trzy, licząc `saved UID`) identyfikatory użytkownika: rzeczywisty UID (kto naprawdę uruchomił proces) i efektywny UID (na czyich prawach jądro sprawdza dostęp do zasobów w danej chwili). W normalnej sytuacji oba są identyczne. Setuid na pliku wykonywalnym zmienia to przy starcie procesu: jądro ustawia efektywny UID na właściciela pliku, zachowując rzeczywisty UID uruchamiającego — proces „na chwilę” działa z cudzymi uprawnieniami, dokładnie na czas swojego działania, i tylko w zakresie tego, co sam program robi z podniesionym UID-em.



Setuid podnosi efektywny UID procesu do właściciela pliku na czas jego działania, a logika programu sama decyduje, co wolno zrobić z tą podniesioną władzą (kliknięcie otwiera pełny rozmiar)

Dlaczego `/usr/bin/passwd` ma setuid. Zmiana hasła wymaga zapisu do `/etc/shadow` — pliku czytelnego i zapisywalnego wyłącznie przez roota (patrz poprzednia strona rozdziału). Zwykły użytkownik nie ma i nie powinien mieć bezpośredniego prawa zapisu do tego pliku — gdyby miał, mógłby edytować cudze hasła, nie tylko własne. Zamiast tego `passwd` jest plikiem należącym do roota, z bitem setuid: gdy uruchamia go zwykły użytkownik, proces przez chwilę

działa jako root, ale kod programu wewnątrz sam pilnuje, żeby pozwolić zmienić wyłącznie hasło właściciela procesu rzeczywistego (chyba że to root, który może zmienić każde). Setuid nie omija kontroli dostępu — przenosi ją z poziomu systemu plików do logiki samego programu, który musi być napisany na tyle ostrożnie, żeby tej podniesionej władzy nie nadużyć. To dlatego liczba plików z setuid w dobrze utrzymanym systemie jest celowo mała — każdy taki plik to potencjalna furtka, jeśli ma błąd.

Po co `/tmp` ma sticky bit. Katalog `/tmp` musi być zapisywalny przez każdego — inaczej żaden program nie mógłby w nim tworzyć plików tymczasowych. Ale prawo zapisu do katalogu w klasycznym modelu uprawnień oznacza też prawo **usuwania i zmiany nazwy** dowolnego pliku wewnątrz, niezależnie od tego, kto jest jego właścicielem — bo operacja usunięcia pliku to w rzeczywistości modyfikacja zawartości katalogu (usunięcie wpisu), nie modyfikacja samego pliku. Bez sticky bitu każdy użytkownik mógłby usunąć albo podmienić plik tymczasowy innego użytkownika w `/tmp` — a niektóre programy ufają, że plik, który same tam zostawiły chwilę wcześniej, wciąż jest ich. Sticky bit dodaje wyjątek do tej reguły: w katalogu z ustawionym `+t` usunięcie lub zmiana nazwy pliku wymaga bycia jego właścicielem (albo właścicielem katalogu, albo rootem) — prawo zapisu do katalogu samo w sobie już nie wystarcza.

Krok po kroku

1. Sprawdź bieżącą maskę i jej efekt.

```
umask -S
touch test_plik
mkdir test_katalog
ls -ld test_plik test_katalog
```

```
u=rwx,g=rx,o=rx
-rw-r--r--  1 adm_user adm_user    0 wrz 17 10:12 test_plik
drwxr-xr-x  2 adm_user adm_user 4096 wrz 17 10:12 test_katalog
```

2. Zaostrz maskę i porównaj efekt na nowych obiektach.

```
umask 077
touch test_plik2
mkdir test_katalog2
ls -ld test_plik2 test_katalog2
```

```
-rw-------  1 adm_user adm_user    0 wrz 17 10:13 test_plik2
drwx------  2 adm_user adm_user 4096 wrz 17 10:13 test_katalog2
```

Zwróć uwagę: `test_plik` sprzed zmiany maski **nie zmienił** swoich uprawnień — `umask` działa wyłącznie w momencie tworzenia obiektu, nigdy retroaktywnie.

3. Zbuduj katalog projektowy z dziedziczeniem grupy.

```
groupadd projekt_zespol
mkdir /srv/projekt
chgrp projekt_zespol /srv/projekt
chmod 2770 /srv/projekt
```

Cyfra `2` przed `770` to setgid w zapisie ósemkowym (`4` = setuid, `2` = setgid, `1` = sticky — sumują się tak samo jak `rwx`).

4. Zweryfikuj, że nowe pliki dziedziczą grupę katalogu.

```
sudo -u inny_czlonek_zespolu touch /srv/projekt/plik_kolegi.txt
ls -l /srv/projekt/plik_kolegi.txt
```

```
-rw-rw-r-- 1 inny_czlonek_zespolu projekt_zespol 0 wrz 17 10:15 plik_kolegi.txt
```

Plik ma grupę `projekt_zespol`, chociaż grupą główną `inny_czlonek_zespolu` może być coś zupełnie innego.

Częste błędy

Setuid na skrypcie powłoki, licząc na to samo działanie co na binarce.

```
chmod u+s /usr/local/bin/backup.sh
ls -l /usr/local/bin/backup.sh
```

```
-rwsr-xr-x 1 root root 128 wrz 17 09:00 /usr/local/bin/backup.sh
```

`ls -l` pokazuje `s`, ale uruchomienie skryptu przez zwykłego użytkownika i tak działa na jego prawach, nie na prawach roota. Jądro Linuksa (i większość Uniksów) **ignoruje** bit setuid na plikach interpretowanych — skryptach zaczynających się od `#!` — od końca lat 80., z powodu wyścigu (race condition) między odczytem linii `#!` a faktycznym uruchomieniem interpretera, który dawał się wykorzystać do przejęcia uprawnień. Setuid działa wyłącznie na skompilowanych plikach binarnych. Naprawa realnego problemu: opakować logikę w mały program binarny albo użyć `sudo`

z precyzyjną regułą w `/etc/sudoers` zamiast setuid.

Ustawienie `umask` tylko w bieżącej sesji, bez zapisania go na trwałe.

```
umask 077
# ... praca, tworzenie plików z 600 ...
exit
# nowa sesja SSH
umask
```

```
0022
```

Maska wraca do wartości domyślnej po zamknięciu sesji, bo `umask 077` wpisane w terminalu nie zostało zapisane w żadnym pliku startowym. Kolejne pliki tworzone w nowej sesji dostają znowu `644`/`755`, mimo że „wczoraj było dobrze”. Naprawa: wpis w `~/.bashrc` (dla jednego użytkownika) albo `/etc/profile` (dla całego systemu).

Setgid na katalogu nie obejmuje plików utworzonych przed jego ustawieniem.

```
chmod g+s /srv/projekt
ls -l /srv/projekt/stary_plik.txt
```

```
-rw-r--r-- 1 adm_user adm_user 812 wrz 10 08:00 stary_plik.txt
```

Grupa pliku to nadal `adm_user`, nie `projekt_zespol` — setgid działa tylko na nowo tworzone obiekty, od momentu ustawienia bitu. Pliki, które już istniały, trzeba dociągnąć ręcznie:

```
chgrp -R projekt_zespol /srv/projekt
```

`chmod 777 /tmp` kasuje sticky bit razem z „naprawą” uprawnień.

```
chmod 777 /tmp
ls -ld /tmp
```

```
drwxrwxrwx 10 root root 4096 wrz 17 10:20 /tmp
```

Zero `t` na końcu — sticky bit zniknął, bo `chmod` w trybie numerycznym bez czwartej cyfry ustawia bity specjalne na zero. Od tej chwili każdy użytkownik może usunąć lub podmienić plik tymczasowy każdego innego użytkownika w `/tmp` — realna furtka bezpieczeństwa, nie kosmetyka. Naprawa: `chmod 1777 /tmp`, nigdy `777` na katalogach współdzielonych.

Uruchomienie audytu setuid bez `-xdev` na systemie z montowaniami sieciowymi.

```
find / -perm -4000 -type f
```

Polecenie może wisieć bardzo długo albo w ogóle się zawiesić, jeśli w drzewie katalogów jest zamontowany zasób sieciowy (NFS, CIFS), który akurat jest niedostępny — `find` czeka na odpowiedź serwera zamiast pominąć punkt montowania. Naprawa: zawsze `-xdev`, żeby ograniczyć przeszukiwanie do lokalnego systemu plików, i `2>/dev/null`, żeby nie zalać ekranu komunikatami `Permission denied` z katalogów, do których bieżący użytkownik nie ma dostępu.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie, w katalogu `/tmp`. Krok z `sudo -u` w Ćwiczeniu 2 wymaga konta z prawem `sudo` — pomiń go, jeśli go nie masz, i przeczytaj samą odpowiedź.

Ćwiczenie 1 — umask nie działa retroaktywnie

```
cd /tmp
mkdir umask_test && cd umask_test
umask 022
touch przed.txt
umask 077
touch po.txt
ls -l przed.txt po.txt
```

Oba pliki powstały w tym samym katalogu, w tej samej sesji. Dlaczego mają różne uprawnienia?

Odpowiedź

```
-rw-r--r-- 1 adm_user adm_user 0 wrz 17 10:30 przed.txt
-rw------- 1 adm_user adm_user 0 wrz 17 10:30 po.txt
```

`umask` jest odczytywany w momencie tworzenia pliku, nie działa jak reguła obowiązująca wstecz. Zmiana maski po fakcie nie dotyka plików, które już powstały — analogicznie do `chmod`, który też trzeba wywołać jawnie na istniejących plikach.

Ćwiczenie 2 — audyt setuid we własnym systemie

```
cd /tmp
find /usr/bin /usr/sbin -xdev -perm -4000 -type f 2>/dev/null | tee setuid_audyt.txt
wc -l setuid_audyt.txt
```

Ile plików z setuid znalazłeś? Sprawdź dla dwóch–trzech z nich, dlaczego akurat one potrzebują tego bitu (podpowiedź: `man` dla każdego z nich zwykle tłumaczy, do jakiego zasobu wymagającego uprawnień roota program musi mieć dostęp).

Odpowiedź

Typowa lista na systemie RHEL/Rocky obejmuje `passwd`, `su`, `sudo`, `mount`, `umount`, `crontab`, `ping` (na starszych jądrach — nowsze używają zamiast tego capabilities). Każdy z nich potrzebuje dostępu do zasobu zastrzeżonego dla roota: `passwd` do `/etc/shadow`, `mount` do wywołań systemowych zarządzania montowaniem, `ping` (tam gdzie nadal ma setuid) do gniazd surowych (raw sockets). Krótsza lista niż się spodziewano to dobry znak — każdy dodatkowy plik z setuid ponad te standardowe warto zweryfikować, skąd się wziął.

Pytania kontrolne

1. Ustawiłeś `umask 027`. Jakie uprawnienia dostanie nowo utworzony katalog?

Odpowiedź

`750` — `rwxr-x---`. Punkt startowy dla katalogu to `777`; maska `027` odejmuje `w` i `x` grupie (zostaje `r-x`... w tym wypadku odejmuje `-w-` z pozycji grupy i całość `rwx` z pozycji pozostałych) — dokładnie: `777` minus `027` bit po bicie daje `750`.

2. Dlaczego setuid na `/usr/bin/passwd` nie pozwala zwykłemu użytkownikowi zmienić hasła roota?

Odpowiedź

Setuid podnosi wyłącznie efektywny UID procesu do właściciela pliku — sam program `passwd` wewnątrz swojego kodu sprawdza rzeczywisty UID uruchamiającego i pozwala mu zmienić tylko

własne hasło (chyba że to root, który może zmieniać dowolne). Setuid daje dostęp do zasobu (`/etc/shadow`), ale logikę autoryzacji — kto może zmienić czyje hasło — realizuje sam program, nie sam bit.

3. Katalog `/srv/wspolne` ma ustawiony sticky bit, ale nie setgid. Dwóch użytkowników z różnymi grupami głównymi tworzy w nim pliki. Co się stanie z grupami tych plików?

Odpowiedź

Każdy plik dostanie grupę główną swojego twórcy — sticky bit reguluje wyłącznie prawo usuwania/zmiany nazwy plików w katalogu, nie ma żadnego wpływu na dziedziczenie grupy. Do tego, żeby nowe pliki dziedziczyły grupę katalogu, potrzebny jest osobny bit — setgid (`g+s`) na katalogu.

4. `find / -perm -2000` bez `-type f` zwraca zarówno pliki, jak i katalogi. Dlaczego to akurat przy setgid ma sens, a przy setuid (`-perm -4000`) audyt zwykle ogranicza się do plików?

Odpowiedź

Setgid na katalogu ma realne, powszechnie wykorzystywane zastosowanie — dziedziczenie grupy przez nowe pliki, więc katalogi z tym bitem są normalnym, oczekiwanym elementem systemu (współdzielone foldery projektowe). Setuid na katalogu nie ma analogicznego, sensownego znaczenia w standardowym Linuksie — jego obecność na katalogu byłaby nietypowa i raczej sygnałem błędnej konfiguracji niż zamierzonego mechanizmu, dlatego audyt setuid zwykle zawęża się do plików wykonywalnych.

Powiązane: chmod, chown, chgrp — uprawnienia klasyczne · Użytkownicy, grupy i pliki, w których system ich trzyma · ACL — uprawnienia ponad model właściciel-grupa-inni

Źródła: `man 1 chmod` · `man 2 setuid` · `man 5 login.defs` · `man 1 find`

Revision #16
Created 2026-09-17 21:42:50 UTC by Tomasz Konieczny
Updated 2026-10-03 19:08:29 UTC by Tomasz Konieczny