



## Podstawy systemu Linux

07. Obejście prostych mechanizmów  
blokowania użytkowników



# Podstawy systemu Linux

## 07. Obejście prostych mechanizmów blokowania użytkowników

### Spis treści

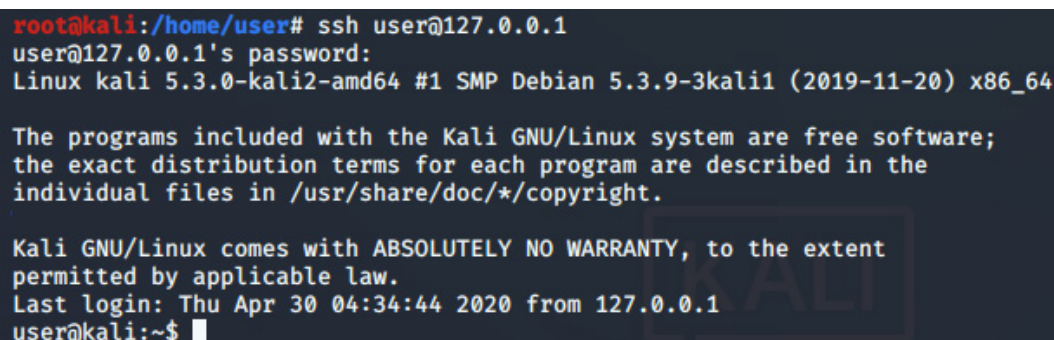
Dodatkowe opcje połączenia secure shell	4
Przesyłanie plików przez ssh	5
Kontrola dostępu	6
Zaawansowane opcje uprawnień	8
Wyszukiwanie plików po uprawnieniach	10
Obsługa plików w systemie plików	12
Spis komend	15

## Dodatkowe opcje połączenia secure shell

W systemach linuksowych zdalne połączenie odbywa się za pomocą **ssh**.

Przykładowo - jeżeli użytkownik chce się zalogować jako „user” na komputerze pod adresem 192.168.10.10, to użyłby polecenia:

```
ssh user@192.168.10.10
```



```
root@kali:/home/user# ssh user@127.0.0.1
user@127.0.0.1's password:
Linux kali 5.3.0-kali2-amd64 #1 SMP Debian 5.3.9-3kali1 (2019-11-20) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Apr 30 04:34:44 2020 from 127.0.0.1
user@kali:~$
```

Ryc. 1. Logowanie jako użytkownik „user” przez ssh.

Po zalogowaniu się na serwerze za pomocą ssh, serwer rozpocznie sesję użytkownika. W praktyce wygląda to tak, jakby zwykły użytkownik zalogował się na komputerze i została uruchomiona jego powłoka (np. bash).

Mówi się wtedy, że ssh działa w **trybie interaktywnym**, ponieważ użytkownik może za pomocą klienta wprowadzać dane, a serwer będzie na to odpowiednio reagował.

Możliwe jest jeszcze zdalne uruchomienie polecenia - w tym przypadku nie zostanie uruchomiona powłoka użytkownika. Warto zauważyć, że jeżeli powłoka (np. bash) ustawiała jakieś zmienne środowiskowe lub uruchamiała jakieś polecenia po starcie, to w tym przypadku te czynności nie zostaną wykonane.

Aby uruchomić polecenie przez ssh, należy podać to polecenie na końcu polecenia ssh.

```
ssh user@192.168.10.10 ls -l
```

Powyższe polecenie pozwoli na zalogowanie jako użytkownik user na maszynie 192.168.10.10, a następnie wykonane zostanie polecenie `ls -l`.

Połączenie z serwerem zostanie automatycznie zakończone po tym, jak wskazane polecenie skończy pracę.

Jeżeli uruchamiany program wymaga, aby użytkownik wprowadził dane, należy uruchomić go w trybie interaktywnym za pomocą opcji `-t`.



```
root@kali:/home/user# ssh user@127.0.0.1 ls -l
user@127.0.0.1's password:
total 60
drwxr-xr-x 2 user user 4096 Dec 23 09:31 Desktop
drwxr-xr-x 2 user user 4096 Feb  2 17:10 Documents
drwxr-xr-x 2 user user 4096 Apr  2 10:15 Downloads
-rw-r--r-- 1 user user   17 Apr 19 07:33 file
-rw-r--r-- 1 user user   17 Apr 19 07:40 file_content
drwxr-xr-x 2 user user 4096 Dec 23 09:31 Music
drwxr-xr-x 3 user user 4096 Feb  2 16:45 my_works
-rw----- 1 user user   17 Apr 19 07:33 nano.save
drwxr-xr-x 2 user user 4096 Dec 23 09:31 Pictures
drwxr-xr-x 2 user user 4096 Dec 23 09:31 Public
-rw-r--r-- 1 user user   57 Apr 19 07:49 rm_errors
-rwxr--r-- 1 user user   49 Apr 19 07:21 script.sh
drwxr-xr-x 2 user user 4096 Dec 23 09:31 Templates
----- 1 user user   15 Apr  2 10:06 test
-rw-r--r-- 1 user user    0 Dec 23 11:28 urls.txt
drwxr-xr-x 2 user user 4096 Dec 23 09:31 Videos
root@kali:/home/user#
```

Ryc. 2. Uruchamianie pojedynczego polecenia przez ssh.

```
ssh -t user@192.168.10.10 base64
```

Program ssh w większości przypadków rozpoznaje, że program działa w trybie interaktywnym i umożliwia wprowadzanie sterowania nim przez użytkownika. Czasami jednak może się zdarzyć, że tryb interaktywny nie zostanie rozpoznany - opcja `-t` wymusza użycie trybu interaktywnego.

```
root@kali:/home/user# ssh -t user@127.0.0.1 base64
user@127.0.0.1's password:
test
dGVzdAo=
Connection to 127.0.0.1 closed.
root@kali:/home/user#
```

Ryc. 3. Uruchomienie programu base64 przez ssh w trybie interaktywnym.

## Przesyłanie plików przez ssh

Protokół ssh pozwala też na przesyłanie plików - do tego służy polecenie **scp**. Aby przesłać pliki na serwer, należy użyć polecenia o poniższej składni:

```
scp nazwa_pliku użytkownik@adres_serwera:docelowa_lokalizacja
```

Drugi argument ma postać którą można wyjaśnić w następujący sposób:

- `użytkownik` - to nazwa użytkownika, którą zalogujemy się na serwer,
- `adres_serwera` - to adres serwera, na który zostanie przesłany plik,
- `docelowa_lokalizacja` - to miejsce na serwerze, w którym zostanie zapisany przesłany plik.

Przykładowo, aby wysłać plik `file.txt` na serwer `192.168.10.10` jako użytkownik „user” i zapisać go jako „./file.txt”, należy użyć poniższego polecenia

```
scp file.txt user@192.168.10.10:./file.txt
```

Warto zauważyć, że zarówno ssh, jak i scp domyślnie rozpoczyna pracę w katalogu domowym - zatem „.” w „./file.txt” odnosi się do katalogu domowego.

```
root@kali:~# cat file.txt
it is a test file
root@kali:~# scp file.txt user@127.0.0.1:./file.txt
user@127.0.0.1's password:
file.txt                                100% 18   19.5KB/s   00:00
root@kali:~# ssh user@127.0.0.1
user@127.0.0.1's password:
Linux kali 5.3.0-kali2-amd64 #1 SMP Debian 5.3.9-3kali1 (2019-11-20) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Apr 30 04:39:47 2020 from 127.0.0.1
user@kali:~$ cat file.txt
it is a test file
user@kali:~$
```

Ryc. 4. Wysyłanie plików na serwer za pomocą scp.

Aby pobrać plik z serwera, należy zamienić miejscami pierwszy i drugi argument:  
scp użytkownik@adres.serwera:plik\_na\_serwerze docelowa\_lokalizacja

Aby z powrotem pobrać plik „file.txt” i zapisać jako „new\_file.txt”, należy użyć poniższego polecenia:

```
scp user@192.168.10.10:./file.txt new_file.txt
```

## Kontrola dostępu

W systemach linuksowych częstą praktyką jest tworzenie użytkowników dla działających usług. Przykładowo, usługa PulseAudio, odpowiedzialna za obsługę dźwięku, zazwyczaj ma oddzielnego użytkownika pulse. Dzięki temu można dać tej usłudze dostęp do odpowiednich zasobów, które potrzebuje do poprawnego działania i żadnych innych. Gdyby taka usługa miała pełny dostęp (np. działała z uprawnieniami root), mogłaby niechcący wpłynąć na działanie innych usług lub mogłaby zostać wykorzystana do przejęcia serwera przez intruza.

Pomimo wielu dobrych praktyk, zdarzają się błędy - czasami wynikają po prostu z błędnej konfiguracji systemu. Niektórzy użytkownicy dedykowani usługom mogą mieć nadmiar uprawnień, co złośliwy użytkownik mógłby wykorzystać poprzez wykorzystanie uruchomionych programów.

Warto zauważyć, że w systemach linuksowych nazwa użytkownika nie jest jedynym identyfikatorem. Każda nazwa użytkownika i nazwa grupy ma swój identyfikator numeryczny. Identyfikator użytkownika określa się inaczej UID, a identyfikator grupy GID.

Można zobaczyć identyfikatory przypisane do użytkownika uruchamiając poniższe polecenie:

```
id nazwa_użytkownika
```

Wykonując `id root` można zauważyć, że ma on identyfikator 0 (rys. 5). Jest to celowe - w każdym systemie linuksowym identyfikator 0 oznacza użytkownika z pełnymi uprawnieniami.

```
user@kali:~$ id user
uid=1000(user) gid=1000(user) groups=1000(user)
user@kali:~$ id root
uid=0(root) gid=0(root) groups=0(root)
user@kali:~$
```

Ryc. 5. Sprawdzanie identyfikatora użytkowników „user” oraz „root”.

Identyfikator zalogowanego użytkownika jest też przechowywany w zmiennej środowiskowej „EUID”. Można go wyświetlić za pomocą poniższego polecenia

```
echo "$EUID"
```

Niektóre skrypty bash mogą sprawdzać, czy skrypt jest uruchomiony jako root - na przykład jeżeli to jest skrypt instalacyjny usługi albo programu. Zazwyczaj taki skrypt może wyglądać tak:

```
#!/bin/bash
if [[ "$EUID" -ne 0 ]]
then
    echo "This script needs to be run as root." 2>&1
    exit 1
fi
echo "You are root"
```

Za pomocą „if” można dodać warunek do kodu. Jeżeli to, co następuje po „if”, to prawda, to wykona się kod od „then” do „fi”.

W tym przypadku sprawdzane jest, czy „\$EUID” nie jest równe 0 (ne - *not equal*). Jeżeli to prawda (nie jest równe 0), to wyświetlany jest komunikat na standardowym wyjściu błędu, a następnie skrypt jest kończony za pomocą `exit`.

```
user@kali:~$ cat run_as_root.sh
#!/bin/bash
if [[ "$EUID" -ne 0 ]]
then
    echo "This script needs to be run as root." 2>&1
    exit 1
fi
echo "You are root"

user@kali:~$ ./run_as_root.sh
This script needs to be run as root.
user@kali:~$ sudo ./run_as_root.sh
[sudo] password for user:
You are root
user@kali:~$
```

Ryc. 6. Uruchomienie skryptu sprawdzającego użytkownika.

Jak widać na rysunku 6, `sudo` jest w stanie nadać poleceniu takie uprawnienia, jakie ma `root`. Dlatego należy być ostrożnym podczas używania `sudo`.

Niektóre programy, chociaż są uruchomione przez zwykłego użytkownika, mogą mieć takie uprawnienia jak `root`. Na skutek konfiguracji takie programy po prostu działają w taki sam sposób, jakby uruchomiło się je z `sudo`, jednak nie potrzebują go do uzyskania uprawnień. Należy unikać takiej konfiguracji systemu, gdyż jest ona niebezpieczna i użytkownik może nadużyć programu.

## Zaawansowane opcje uprawnień

Jak dotąd używane były trzy typy uprawnień - `rwX`. Te uprawnienia są w systemie reprezentowane liczbowo w systemie ósemkowym według poniższego schematu:

Liczba	Wyjaśnienie	Symbol
0	Brak uprawnień	---
1	Wykonywanie	--X
2	Zapisywanie	-w-
3	Zapisywanie + wykonywanie (2+1)	-wX
4	Odczyt	r--
5	Odczyt + wykonywanie (4+1)	r-X
6	Odczyt + zapis (4+2)	rw-
7	Odczyt + zapis + wykonywanie (4+2+1)	rwX

Uprawnienia dla każdego typu (użytkownik, grupa, inni) zapisuje się za pomocą trzech liczb. Przykładowo `rwXr-Xr--` można zapisać jako `754`.

Linux wprowadza także trzy dodatkowe uprawnienia:

- SUID,
- GUID,
- Sticky bit (albo restricted deletion flag - flaga ograniczonego usuwania).

**SUID** powoduje uruchomienie pliku z uprawnieniami **właściciela**, niezależnie od tego, jaki użytkownik uruchamia ten plik.

Przykładowo, właścicielem pliku „`script.sh`” jest *alice* i plik ma ustawione SUID. Obecnie zalogowanym użytkownikiem jest natomiast *bob*. W tym przypadku `./script.sh` zostanie uruchomione z uprawnieniami *alice*, pomimo że zalogowany jest *bob*.

**GUID** powoduje uruchomienie pliku z uprawnieniami grupy, niezależnie od tego, do jakiej grupy należy użytkownik, który uruchamia ten plik.



Przykładowo, plik „script.sh” należy do grupy *wheel* i ma ustawione GUID. Obecnie zalogowanym użytkownikiem jest *bob*, który należy do grupy *bob*. W tym przypadku `./script.sh` zostanie uruchomione z uprawnieniami grupy *wheel*, pomimo że zalogowany jest *bob* z grupy *bob*.

**Sticky bit** (*restricted deletion flag* - flaga ograniczonego usuwania) jest uprawnieniem, które uniemożliwia usunięcie bądź zmianę nazwy pliku przez osoby, które nie są właścicielem pliku. Flagę tę ustawia się na folderze i ma ona zastosowanie do plików wewnątrz niego. Tę flagę można ustawić na pojedynczych plikach, ale wtedy nie będzie miało to żadnego efektu.

Przykładowo, właścicielem pliku „script.sh” jest użytkownik *alice*, a plik ma uprawnienia `rw-rw-rw-` (każdy może odczytać i zapisać) i obecnie zalogowany jest *bob*. Plik „script.sh” znajduje się w folderze z ustawionym Sticky bit. W tym przypadku `rm script.sh` i `mv script.sh script_2.sh` nie może zostać wykonane, ponieważ *bob* nie jest właścicielem pliku. Użytkownik *bob* może natomiast do woli edytować ten plik.

Te trzy uprawnienia mogą w `chmod` są reprezentowane w następujący sposób:

- SUID - `u+s`,
- GUID - `g+s`,
- Sticky bit - `+t`.

Te uprawnienia mają również swoje liczbowe odpowiedniki:

Liczba	Wyjaśnienie	Symbol
0	Brak uprawnień	-----
1	Sticky bit	-----t
2	GUID	--s-----
3	GUID + Sticky bit (2+1)	--s-----t
4	SUID	-----s---
5	SUID + Sticky bit (4+1)	-----s--t
6	SUID + GUID (4+2)	--s--s---
7	SUID + GUID + Sticky bit (4+2+1)	--s--s--t

Przykładowo `rw-r-sr-t` można inaczej zapisać jako 5754. Pierwsza cyfra (5) oznacza dodatkowe uprawnienia i w tym przypadku oznacza SUID + Sticky bit.

Jeżeli plik nie ma dodatkowych uprawnień, to na początku wstawia się 0 lub się je pomija - np. `rw-r-xr--` można zapisać jako 0754 albo 754.

Warto zauważyć, że w przypadku, gdy plik nie ma uprawnień do wykonywania, ale ma ustawione SUID albo GUID, to zamiast „s” w uprawnieniach będzie „S” (duża litera). W przypadku sticky bit sytuacja jest analogiczna - jeżeli inni nie mają uprawnień do wykonywania, to zamiast „t” będzie „T”.

```
user@kali:~/permissions$ ls -la
total 8
drwxrwxrwt  2 user user 4096 May  1 05:51 .
drwxr-xr-x 18 user user 4096 May  1 05:35 ..
-rw-r-Sr--  1 user user   0 May  1 05:51 another_one
-rw-rw-rwx  1 user user   0 May  1 05:31 file
-rwsr-xr-x  1 user user   0 May  1 05:50 script.sh
user@kali:~/permissions$ chmod u-x script.sh
user@kali:~/permissions$ ls -la
total 8
drwxrwxrwt  2 user user 4096 May  1 05:51 .
drwxr-xr-x 18 user user 4096 May  1 05:35 ..
-rw-r-Sr--  1 user user   0 May  1 05:51 another_one
-rw-rw-rwx  1 user user   0 May  1 05:31 file
-rwSr-xr-x  1 user user   0 May  1 05:50 script.sh
user@kali:~/permissions$
```

Ryc. 7. Dodatkowe uprawnienia i uprawnienia do wykonywania.

## Wyszukiwanie plików po uprawnieniach

Program **find** pozwala na wyszukiwanie plików z danymi uprawnieniami - pozwala na to opcja **-perm**.

Opcja ta obsługuje dwa zapisy:

- liczbowy - np. 0667,
- literowy - np. u=rw, g=rw, o=rwx.

W przypadku wyszukiwania liczbowego wystarczy wpisać dokładne uprawnienia, które ma mieć szukany plik, np.

**find . -perm 0667.**

W tym przypadku **find** przeszuka obecny folder w poszukiwaniu wszystkich plików o uprawnieniach 667, czyli **rw-rw-rwx**.

```
user@kali:~/permissions$ find . -perm 0667
./file
user@kali:~/permissions$ ls -l file
-rw-rw-rwx 1 user user 0 May  1 05:31 file
user@kali:~/permissions$
```

Ryc. 8. Wyszukiwanie plików po uprawnieniach używając zapisu liczbowego.

Zamiast zapisu liczbowego można również zapis literowy, w tym przypadku należy wypisać uprawnienia, które ma mieć plik w postaci:

**<typ>=<uprawnienia>**

Pole <typ> może być jedną z poniższych wartości:

- u - użytkownik,
- g - grupa,
- o - inni,
- a - wszyscy.

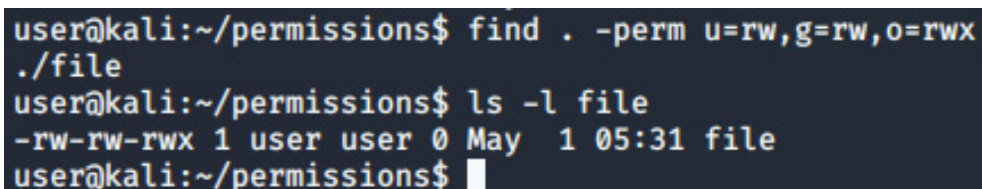
Natomiast <uprawnienia> oznaczają nadane uprawnienia danego typu (np. r, rw, rx itd.).

Aby podać uprawnienia innego typu, należy wypisać je po przecinku. Przykładowo zapis u=rwx, g=rwx, o=rwx oznacza pliki o uprawnieniach rwxr-xr-x.

Wyszukiwanie zapisem literowym odbywa się w następujący sposób:

```
find . -perm u=rw,g=rw,o=rwx
```

Powyższy zapis jest tożsamy z `find . -perm 0667`.



```
user@kali:~/permissions$ find . -perm u=rw,g=rw,o=rwx
./file
user@kali:~/permissions$ ls -l file
-rw-rw-rwx 1 user user 0 May 1 05:31 file
user@kali:~/permissions$
```

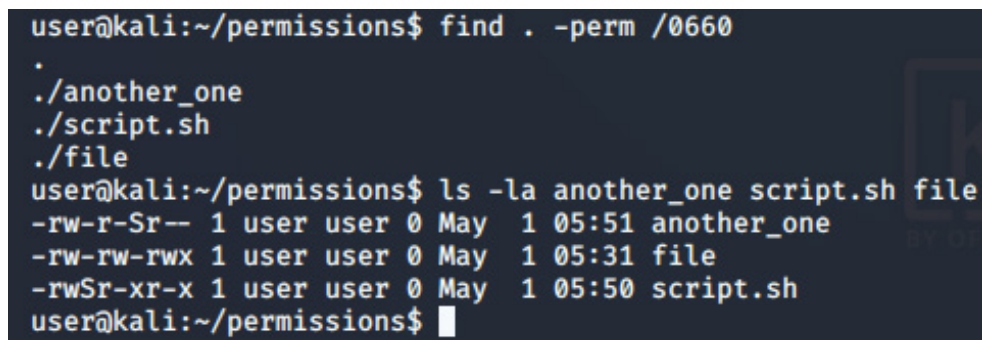
Ryc. 9. Wyszukiwanie plików po uprawnieniach używając zapisu literowego.

Możliwe jest też **wyszukiwanie maskowane**. W tym przypadku program `find` nie będzie szukał plików o konkretnych uprawnieniach (np. 0667), ale będzie szukał plików zawierających wskazane uprawnienie.

Aby to zrobić, należy przed uprawnieniami dodać „/”.

```
find . -perm /0660
```

W tym przypadku `find` w miejscu zer dopasuje dowolne uprawnienia. Zostaną odnalezione pliki zawierające uprawnienia `rw?rw????`, gdzie zamiast znaków zapytania będą dowolne inne uprawnienia.



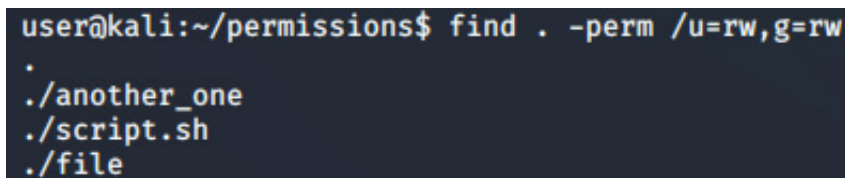
```
user@kali:~/permissions$ find . -perm /0660
.
./another_one
./script.sh
./file
user@kali:~/permissions$ ls -la another_one script.sh file
-rw-r-Sr-- 1 user user 0 May 1 05:51 another_one
-rw-rw-rwx 1 user user 0 May 1 05:31 file
-rwSr-xr-x 1 user user 0 May 1 05:50 script.sh
user@kali:~/permissions$
```

Ryc. 10. Wyszukiwanie plików po uprawnieniach z użyciem maskowanego zapisu liczbowego.

Również w przypadku zapisu literowego można stosować wyszukiwanie maskowane.

```
find . -perm /u=rw,g=rw
```

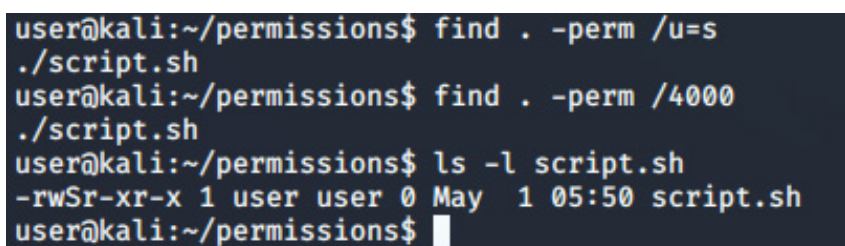
Powyższy przykład jest tożsamy z `find . -perm /0660`, to znaczy pozwoli na wyszukanie plików z uprawnieniami pasującymi do wzoru `rw?rw????`.



Ryc. 11. Wyszukiwanie plików po uprawnieniach z użyciem maskowanego zapisu literowego.

W taki sam sposób można wyszukiwać pliki z uprawnieniami specjalnymi. Żeby wyszukać pliki z uprawnieniami SUID, należy użyć zapisu

```
find . -perm /u=s  
lub  
find . -perm /4000
```



Ryc. 12. Wyszukiwanie plików ze specjalnymi uprawnieniami.

Podobnie można wyszukać pliki z uprawnieniami GUID:

```
find . -perm /g=s
```

A także ze sticky bit:

```
find . -perm /o=t
```

## Obsługa plików w systemie plików

W systemach linuxowych, w porównaniu do systemu Windows, możliwa jest edycja i usuwanie plików podczas gdy używa ich inny program.

W systemie Windows zdarza się, że jakiś program otworzy plik i tym samym założy na nim blokadę. Blokada ta powoduje, że żaden inny program nie może modyfikować tego pliku, a ponadto nie można go też usunąć. W takich sytuacjach pojawia się komunikat „nie można usunąć pliku, ponieważ jest on używany przez inny program”.

W systemach linuksowych nie ma tego problemu - pozwala on na edycję, przenoszenie i usuwanie plików podczas gdy jest on otwarty w innym programie. Taki usunięty plik nie jest widoczny po wpisaniu `ls`, ale to nie znaczy, że został usunięty fizycznie. System po prostu usuwa informację o tym, że plik w tym miejscu istnieje, a sam plik zostanie usunięty dopiero wtedy, gdy ostatni używający go program przestanie go używać.

Warto o tym pamiętać podczas usuwania poufnych danych - mogą one wciąż istnieć w pamięci. Zazwyczaj restart systemu pozwala na usunięcie takich „pozostałości”, ponieważ wyłączy on wszystkie programy.

Używane przez programy pliki można podejrzeć w prosty sposób. Aby to zrobić, należy najpierw przejść do folderu `/proc`. Jest to folder, który zawiera informacje o aktualnej konfiguracji systemu i sprzętu oraz informacje o uruchomionych procesach.

```
user@kali:/proc$ ls
1      1245  1326  144  2077  26   477  68   78   865  asound  fb      kpagecount  sched_debug  uptime
10     1252  1329  145  21    27   478  69   784  875  buddyinfo  filesystems  kpageflags  schedstat    version
11     1258  1348  146  2184  274  479  70   790  880  bus        fs         loadavg     self         vmallocinfo
1163   1259  1353  15   2186  275  482  71   798  888  cgroups   interrupts  locks       slabinfo     vmstat
1200   1265  1357  150  22    3    511  72   8    89   cmdline   iomem      meminfo     softirqs     zoneinfo
1217   1266  136   151  2235  320  536  737  836  892  consoles  ioports    misc        stat
1221   1269  137  1545  2236  326  548  74   837  898  cpuinfo   irq        modules     swaps
1225   1274  138   16   2237  327  549  75   839  9    crypto    kallsyms   mounts      sys
1226   1280  139   18   228   338  553  76   840  904  devices   kcore      mtrr        sysrq-trigger
1227   1285  1393  19   229   389  584  768  845  906  diskstats  keys       net          sysvipc
1228   1294  14    1906  23    4    6    77   850  921  dma        key-users  pagetypeinfo  thread-self
1229   13    140   2    24    474  662  773  852  949  driver     kmsg       partitions    timer_list
1244   1325  142   20   25    476  665  774  856  acpi  execdomains  kpagecgroup  pressure     tty
```

Ryc. 13. Przykładowa zawartość folderu `/proc`.

Jak widać po wyświetleniu zawartości folderu `/proc`, są w nim foldery, których nazwami są liczby. Te liczby to identyfikatory procesów (PID) uruchomionych programów.

Można na przykład spróbować wyświetlić zawartość jednego z folderów, np. 773. Znajdują się tam różne foldery diagnostyczne, które opisują różne cechy procesu.

```
user@kali:/proc$ ls 773
arch_status  cmdline      exe          loginuid    mountstats  oom_score_adj  sched      stack      timers
attr         comm         fd           map_files  net         pagemap        schedstat  stat       timerslack_ns
autogroup   coredump_filter  fdinfo      maps       ns          patch_state    sessionid  statm      uid_map
auxv        cpuset      gid_map     mem        numa_maps   personality    setgroups  status     wchan
cgroup      cwd         io          mountinfo  oom_adj     projid_map     smaps      syscall
clear_refs  environ     limits      mounts     oom_score   root           smaps_rollup  task
```

Ryc. 14. Zawartość folderu opisującego proces o identyfikatorze 773.

W przypadku systemu plików ważnym folderem jest **fd**, czyli *file descriptor* (deskryptor pliku). Zawiera on wszystkie strumienie używane przez proces, m.in. strumienie wejścia, wyjścia oraz otwarte pliki.

Jak widać na rysunku 15, nazwy strumieni są liczbami zaczynającymi się od 0. Strumień 0 to *stdin* (standardowe wejście), 1 to *stdout* (standardowe wyjście), a 2 to *stderr* (wyjście błędów). Każdy kolejny strumień powyżej 2 oznacza otwarty plik bądź dane przesyłane z innego miejsca.

W przypadku, gdy lista plików w folderze `fd` zostanie wyświetlona za pomocą `ls -l`, zostaną wyświetlone także miejsca docelowe tych strumieni. Na przykład strumień 14 to tak naprawdę plik `/proc/swaps`.



```

user@kali:/proc/773/fd$ ls -l
total 0
lr-x----- 1 user user 64 May 1 03:23 0 -> /dev/null
lrwx----- 1 user user 64 May 1 03:23 1 -> 'socket:[19597]'
lr-x----- 1 user user 64 May 1 03:23 10 -> /proc/773/mountinfo
lr-x----- 1 user user 64 May 1 03:23 11 -> anon_inode:inotify
lrwx----- 1 user user 64 May 1 03:23 12 -> 'socket:[19638]'
lr-x----- 1 user user 64 May 1 03:23 13 -> anon_inode:inotify
lr-x----- 1 user user 64 May 1 03:23 14 -> /proc/swaps
lrwx----- 1 user user 64 May 1 03:23 15 -> 'socket:[19625]'
lrwx----- 1 user user 64 May 1 03:23 16 -> 'socket:[19627]'
lrwx----- 1 user user 64 May 1 03:23 17 -> 'socket:[19629]'
lrwx----- 1 user user 64 May 1 03:23 18 -> 'socket:[19630]'
lrwx----- 1 user user 64 May 1 03:23 19 -> 'socket:[19631]'
lrwx----- 1 user user 64 May 1 03:23 2 -> 'socket:[19597]'
lrwx----- 1 user user 64 May 1 03:23 20 -> 'socket:[19633]'
lrwx----- 1 user user 64 May 1 03:23 21 -> 'anon_inode:[timerfd]'
lrwx----- 1 user user 64 May 1 03:23 26 -> 'socket:[19640]'
lrwx----- 1 user user 64 May 1 03:23 27 -> 'socket:[19643]'
lrwx----- 1 user user 64 May 1 03:23 28 -> 'socket:[19647]'
lrwx----- 1 user user 64 May 1 03:23 29 -> 'socket:[19649]'
lrwx----- 1 user user 64 May 1 03:23 3 -> 'socket:[19614]'
lrwx----- 1 user user 64 May 1 03:23 30 -> 'socket:[19651]'
lrwx----- 1 user user 64 May 1 03:23 31 -> 'socket:[19653]'
lrwx----- 1 user user 64 May 1 03:23 32 -> 'socket:[19269]'
lrwx----- 1 user user 64 May 1 03:23 4 -> 'anon_inode:[eventpoll]'
lrwx----- 1 user user 64 May 1 03:23 5 -> 'anon_inode:[signalfd]'
lr-x----- 1 user user 64 May 1 03:23 6 -> anon_inode:inotify
lr-x----- 1 user user 64 May 1 03:23 7 -> /sys/fs/cgroup/unified/user.slice/user-1000.slice/user@1000.service
lrwx----- 1 user user 64 May 1 03:23 8 -> 'anon_inode:[timerfd]'
lrwx----- 1 user user 64 May 1 03:23 9 -> 'anon_inode:[eventpoll]'
user@kali:/proc/773/fd$

```

Ryc. 15. Strumienie używane przez proces 773.

Aby zobaczyć zawartość strumienia, można po prostu użyć komendy **cat** (rys. 16). Zadziała to także w przypadku usuniętych plików - będą one nadal dostępne jako strumień w folderze **fd** do momentu wyłączenia aplikacji.

```

user@kali:/proc/773/fd$ cat 14
Filename                                     Type      Size      Used      Priority
/dev/sda5                                  partition 2095100    0         -2
user@kali:/proc/773/fd$

```

Ryc. 16. Wyświetlanie zawartości strumienia.

## Spis komend

**ssh** [opcje] <użytkownik>@<adres\_serwera> [komenda]

Służy do zdalnego połączenia z serwerem.

Opcje:

- p <port> łączy się pod wybrany port
- t wymusza użycie trybu interaktywnego

**scp** <lokalizacja> <cel>

Wgrywa na serwer bądź pobiera plik z serwera. Źródło lub cel może być lokalizacją na obecnym komputerze (np. ./plik.txt) bądź lokalizacją zdalną w formacie użytkownik@adres\_serwera:lokalizacja\_pliku (np. user@192.168.10.10:./plik.txt).

**id** <użytkownik>

Wyświetla identyfikator użytkownika.

**find** <lokalizacja> [opcje]

Wyszukuje pliki.

Opcje:

- |                      |  |
|----------------------|--|
| -perm <uprawnienia>  | wyszukuje pliki o uprawnieniach identycznych |
| jak wskazane         |  |
| -perm /<uprawnienia> | wyszukuje pliki o uprawnieniach zawierają-   |
| cych te wskazane     |  |