Wszystko o prawach dostępu

Marcin Ulikowski

Instalując system Linux jesteśmy proszeni o dokonanie podziału dysku twardego na partycje. Większość nowoczesnych dystrybucji oferuje nawet w pełni automatyczne ich przydzielanie. Ilość partycji wpływa na poziom bezpieczeństwa systemu. Dlatego warto zrezygnować z automatyzacji tego procesu i poświęcić trochę więcej czasu na własnoręczne przystosowanie dysku twardego dla instalowanego systemu.

Partycje to obszary na dysku twardym zarezerwowane dla określonych systemów plików. Linux cechuje się dużą elastycznością w tworzeniu i zarządzaniu wieloma partycjami. Poprzez podział naszego dysku na odpowiednią ilość partycji zyskujemy możliwość zastosowania bardziej rygorystycznej kontroli nad poszczególnymi systemami plików oraz nad sposobami ich montowania (ang. *mount*) w systemie. Pliki systemowe oddzielone są od plików użytkowników. Ponadto już na starcie ułatwiamy sobie tworzenie kopii zapasowych i administrację całego systemu.

W przypadku instalacji systemu na pojedynczej partycji jego administracja jest nieco utrudniona. Bardzo kłopotliwe będzie na przykład wykonywanie aktualizacji czy tworzenie kopii zapasowych. Takie rozwiązanie zwiększa szansę na wykorzystanie przez intruza błędów programów z ustawionym bitem SUID. Ponadto, jeśli cały system znajduje się tylko na jednej partycji, nawet niewielkie uszkodzenie pliku może doprowadzić do pojawienia się większych kłopotów (uszkodzenie jednego katalogu może mieć wpływ na inne katalogi). Może to spowodować konieczność powtórnej instalacji systemu.

Aby uniknąć problemów, należy dla każdego systemu plików utworzyć oddzielną partycję. Wraz z oddzielaniem poszczególnych obszarów systemu zyskujemy większą stabilność, zwiększony poziom bezpieczeństwa i kontrolę nad sposobem montowania poszczególnych partycji. Tworzenie wielu partycji ma jeszcze kilka zalet. Zapobiega to m.in. przypadkowemu unieruchomieniu systemu oraz chroni główny katalog przed przepełnieniem. Na przykład katalog */var* przechowuje informacje o zdarzeniach w systemie. Rozrastając się nie zablokuje całego systemu plików jak w przypadku pojedynczej partycji.

O autorze

Autor zajmuje się bezpieczeństwem sieci i systemów komputerowych. Jest studentem drugiego roku informatyki.

Kontakt z autorem: *elceef@itsec.pl*

Dla przykładu podzielmy dysk twardy na sześć partycji. Podziału na partycje możemy dokonać programem `fdisk` (występuje w każdej dystrybucji) lub jego bardziej przyjaznym odpowiednikiem - `cfdisk`. W architekturze opartej na procesorach Intel ilość partycji podstawowych jest ograniczona do czterech. Więcej partycji tworzymy wykorzystując jedną partycję rozszerzoną i kilka partycji logicznych. Podczas podziału dysku ważne jest, aby dwie pierwsze partycje (w naszym przypadku partycja główna i partycja pamięci wirtualnej) posiadały status podstawowych (ang. *primary*), a reszta logicznych (ang. *logical*) utworzonych na partycji rozszerzonej (ang. *extended*).

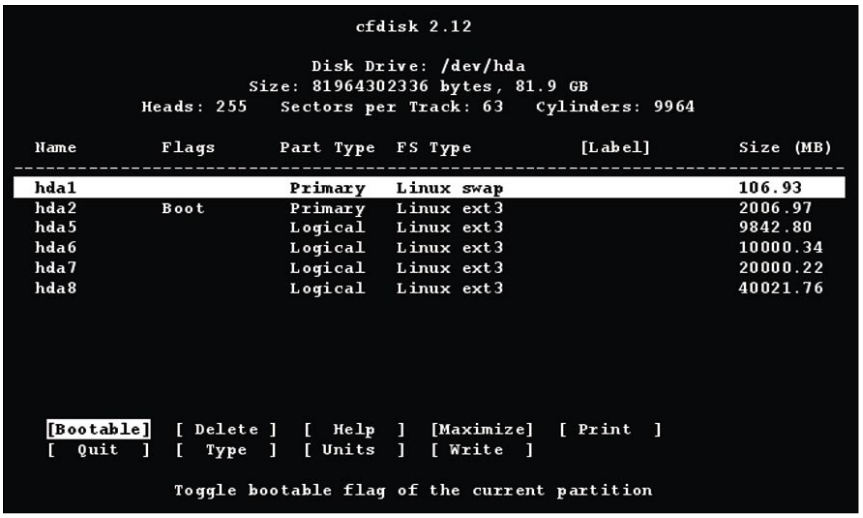
Najtrudniejszym aspektem podziału dysku (szczególnie mniejszego) na partycje jest określenie poziomu jego obciążenia. Tworząc wiele partycji ograniczamy poszczególnym systemom plików możliwość rozrastania, chociaż czasami o to nam chodzi. Lepiej jednak dobrze przemyśleć swoje decyzje, aby nie okazało się, że pewnemu systemowi plików przydzieliliśmy zbyt mało miejsca.

Plik fstab

Dzięki podziałowi dysku twardego możemy określić opcje montowania poszczególnych systemów plików w naszym systemie, co daje nam większe pole manewru w zwiększaniu poziomu bezpieczeństwa. Odpowiada za to plik */etc/fstab*, z którego podczas startu systemu odczytywane są wszystkie dostępne systemy plików i montowane według podanych w nim reguł. Każdy wiersz w tym pliku odpowiada jednemu systemowi plików. Listing 1 zawiera przykładowy plik */etc/fstab*. Zawiera on absolutnie minimalny podział pod względem ilości partycji. Nic nie stoi na przeszkodzie, aby było ich znacznie więcej. Ilość zależy głównie od usług, jakimi chcemy dysponować. Na przykład kolejną partycją może być partycja na pamięć cache serwera proxy, lub także na zasoby serwera FTP.

Każdy wiersz tego pliku składa się z sześciu pól:

- nazwa urządzenia blokowego (np. */dev/hda1*), albo system plików, który ma być montowany;
- plik lokalizujący system plików, czyli punkt montowania, np. */home;*
- typ systemu plików, w naszym przykładzie są to `ext3` oraz `swap`;
- opcje montowania, w których określamy zakres dostępu do danego systemu plików, zawartość tego pola ma dla nas największe znaczenie z punktu widzenia bezpieczeństwa systemu;
- parametr tworzenia kopii zapasowych danego systemu plików;
- parametr kolejności sprawdzania integralności systemu, jest to priorytet, z jakim program `fsck` sprawdza spójność systemu. Partycje o takich samych numerach są sprawdzane jednocześnie, natomiast z wartością 0 omijane.



Rysunek 1. Tworzenie partycji programem cfdisk

Jak możemy przeczytać na Listingu 1, niektóre z partycji są niepotrzebnie montowane z przyznaniem zbyt dużych przywilejów, wyznaczonych flagą `defaults`. Przywileje możemy ograniczyć takimi flagami jak `rw` (możliwy odczyt i zapis) oraz `ro`, która powoduje zamontowanie plików w trybie tylko do odczytu, powstrzymując wszystkie modyfikacje informacji dotyczących plików. Flaga `ro` jest bardzo przydatna w przypadku systemów używanych jako katalog plików, które mają pozostać niezmienione podczas pracy systemu. Będziemy stosować kilka flag jednocześnie, np: `defaults,ro`. Zapis ten oznacza, że system przyjmuje opcję domyślną (flaga `defaults`), ale ograniczoną wpisem `ro` (tylko do odczytu). Pozostałe flagi wchodzące w skład `defaults` zostaną aktywne (oczywiście poza flagą `rw`, która została zastąpiona wpisem `ro`).

Flaga `nosuid` zapobiega uwzględnianiu bitów set-UID oraz set-GID w przypadku dowolnego pliku wykonywalnego. Powinna być stosowana dla partycji użytkowników i partycji plików tymczasowych. Przykładowo po dodaniu tej flagi dla partycji plików tymczasowych, stary exploit wykorzystujący błąd gry Abuse (dostarczanej z dystrybucją Red Hat 2.1 - jeden z plików gry posiadał ustawiony bit SUID) nie zadziała.

Flaga `noexec` zapobiega wykonywaniu plików wykonywalnych w danym systemie plików. Jest ona szczególnie przydatna w przypadku tych systemów plików, w których nie powinny być umieszczone żadne pliki wykonywalne. Należy pamiętać, iż flaga `noexec` nie ogranicza skryptów powłoki (powłoka */bin/sh* znajduje się na partycji, która nie jest ograniczona tą flagą), które będą wykonywane na takich partycjach. Jeśli chcemy być bardziej rygorystyczni możemy flagę `noexec` dodać do partycji */home*, przez co użytkownicy będą mogli korzystać tylko z programów oferowanych przez nasz system.

Łamanie zabezpieczeń

Dodane przez nasz ograniczenia dla poszczególnych partycji mogą zostać ominięte przez sprytnego użytkownika, jeśli w naszym systemie polecenie */bin/mount* posiada ustawiony bit SUID oraz jądro zostało skompilowane z opcją *Loopback device support*. Podane warunki spełnia większość dostępnych dystrybucji, więc szanse są spore. Sprytny użytkownik może utworzyć swój własny system plików (niezbyt duży, np. wielkości 1MB) w swoim katalogu domowym i następnie zamontować bez restrykcyjnych flag. Aby to zrobić wystarczą trzy polecenia:

```
$ dd if=/dev/zero of=ext2.fs
count=2048
$ mkfs -t ext2 ext2.fs
$ mount ext2.fs ~/mnt -o
loop,rw,suid,exec
```

Po ich wykonaniu użytkownik w punkcie *~/mnt* będzie posiadał własny system plików, w którym może wykonywać swoje własne

programy (flaga `exec`). Jądro skompilowane z opcją *Loopback device support* umożliwia nam tworzenie w zwykłym pliku (w naszym przypadku plik ma nazwę *ext2.fs*) dowolnego systemu plików (np. ext2), który potem można zamontować w systemie. Jeśli jądro nie ma wsparcia dla urządzeń *loopback* można do tego samego celu wykorzystać np. protokół zdalnego udostępniania plików NFS (*Network File System*).

Główną przyczyną, dla której użytkownik miał możliwość utworzenia własnego systemu plików był bit SUID ustawiony dla polecenia */bin/mount*. Dlatego zaleca się usunięcie bitu set-ID z tego polecenia (nie zapominając także o */bin/umount*). Dzięki temu przy próbie zamontowania systemu plików zwykły użytkownik otrzyma komunikat:

```
mount: only root can do that.
```

Do pliku */etc/fstab* obok takich flag jak `noexec` i `nosuid` należy także dodać jeszcze jedną przydatną flagę o nazwie `nodev`. Powoduje ona ignorowanie obecnych w systemie plików urządzeń specjalnych, dzięki czemu jest bardzo pożyteczna. Warto także odznaczyć opcję *Loopback device support* podczas konfiguracji jądra, jeśli nie jest przez nas wykorzystywana.

Prawa dostępu do plików

Główną zaletą Linuksa jest rozbudowany mechanizm kontroli dostępu do plików i katalogów. Każdemu użytkownikowi można przydzielić osobne prawa. Jeden z nich może dany plik czytać i zapisywać do niego dane, inny może tylko odczytać, a jeszcze inny może mieć odebrane wszelkie prawa dostępu.

Każdy użytkownik posiada swój własny katalog domowy i tylko w jego obrębie może tworzyć pliki, katalogi oraz dokonywać innych zmian. Reszta systemu powinna pozostawać "tylko do odczytu" (z pewnymi wyjątkami, np. katalog */tmp*), a do niektórych zasobów użytkownik nie powinien mieć żadnego prawa dostępu. Jest to bardzo dobre rozwiązanie uniemożliwiające modyfikację krytycznych elementów systemu przez osoby niepowołane. Dzięki takiemu podejściu użytkownicy są też oddzieleni od siebie nawzajem.

Zanim przejdziemy do ustalania praw dostępu, należy pamiętać, aby zbytnio nie przesadzić z ich odbieraniem (np. prawa odczytu pliku */etc/passwd*) czy nadawaniem. Najlepiej przed dokonaniem zmian, zrobić kopię dotychczasowych praw dostępu w osobnym pliku. Bardzo pomocne będzie także stworzenie konta testowego o uprawnieniach zwykłego użytkownika, które umożliwi nam sprawdzenie swoich założeń co do nałożonych restrykcji.

Główną zasadą podczas modyfikacji praw jest pełna świadomość, jaką funkcję pełni ograniczany plik lub katalog oraz jakie konsekwencje wynikną po ingerencji w jego prawa dostępu. Najlepszym rozwiązaniem jest sprawdzenie działania programu przed oraz po zmianie.

Powinniśmy także zwrócić szczególną uwagę na pliki zawierające ustawiony bit SUID. Zaraz po instalacji Linuksa stanowczo zbyt wiele programów posiada ten bit. Dużo starych eksploitów wykorzystywało ten fakt. Wyszukaniem wszystkich plików tego rodzaju zajmie się polecenie:

```
$ find / -perm +4000
```

Zawartość otrzymanej listy jest zależna od programów, jakie wybraliśmy podczas instalacji systemu. Dla niektórych programów ustawiony bit SUID jest niezbędny do poprawnego działania, dlatego przed usunięciem atrybutu `+s` musimy się upewnić, do czego służy dany program. Pomocą będzie nam służyć odpowiednia stro-

Listing 1. Przykładowy plik /etc/fstab

```
/dev/hda1      swap      swap      defaults      0     0
/dev/hda2      /         ext3      defaults      1     1
/dev/hda5      /home     ext3      defaults      1     1
/dev/hda6      /tmp      ext3      defaults      1     1
/dev/hda7      /var      ext3      defaults      1     1
/dev/hda8      /usr      ext3      defaults      1     1
```

Listing 2. Restrykcyjny plik /etc/fstab

```
/dev/hda1      swap      swap      defaults      0     0
/dev/hda2      /         ext3      defaults      1     1
/dev/hda5      /home     ext3      defaults,nosuid     1     1
/dev/hda6      /tmp      ext3      defaults,nosuid,noexec     1     1
/dev/hda7      /var      ext3      defaults,nosuid,noexec     1     1
/dev/hda8      /usr      ext3      defaults      1     1
```

Listing 3. ulimit -a

```
core file size          (blocks, -c) unlimited
data seg size           (kbytes, -d) unlimited
file size               (blocks, -f) unlimited
max locked memory       (kbytes, -l) unlimited
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 1022
virtual memory          (kbytes, -v) unlimited
```

Programy SUID i SGID

Plik wykonywalny z ustawionym bitem SUID ma specjalną właściwość: niezależnie od tego, kto go uruchomi, wykonywany jest z przywilejami jego właściciela. Podobnie jest z bitem SGID – ale wtedy to grupa, do której należy program przekazuje swoje uprawnienia. Na przykład, jeśli właścicielem pliku SUID jest użytkownik root, w trakcie pracy program będzie posiadał wszystkie uprawnienia tego użytkownika. Będzie miał możliwość odczytu i zapisu do każdego miejsca w systemie. Wykorzystanie błędu w takim programie może zagrozić całemu systemowi.

na podręcznika systemowego (*man nazwa*).

W przypadku katalogu */tmp* należy stosować tzw. bit lepkości (ang. *sticky bit*). Jego ustawienie powoduje, że pliki znajdujące się w danym katalogu mogą być usunięte tylko przez ich właścicieli lub przez właściciela katalogu. Dzięki temu można tworzyć katalogi, w których wszyscy mogą zapisywać pliki, ale nie mogą ich sobie nawzajem kasować, co w przypadku katalogu plików tymczasowych jest wskazane. Bit lepkości dodajemy poleceniem:

```
$ chmod +t /tmp
```

Innymi przykładami "wrażliwych" katalogów są: */var/tmp*, */var/lock* i */var/spool/mail*.

Prawa dostępu do urządzeń

W systemach typu Unix wszystko jest plikiem. Także urządzenia sprzętowe mają przydzielone odpowiednie nazwy plików, które znajdują się w katalogu */dev* i jego podkatalogach.

W przypadku bezpiecznego systemu zwyczajni użytkownicy nie powinni korzystać z bezpośredniego dostępu do urządzeń dyskowych (*/dev/hd** oraz */dev/sd**), ponieważ spowodowałoby to pominięcie wszystkich zabezpieczeń systemu plików. W żadnym wypadku nie mogą mieć dostępu do większości urządzeń `tty`, gdyż w przeciwnym wypadku pozwalałoby to na przechwytywanie klawiszy naciskanych przez poszczególne osoby. Kolejne urządzenia, do których dostęp może mieć wyłącznie administrator to urządzenie pamięci jądra (*/dev/kmem*) i pamięci fizycznej (*/dev/mem*). Dostęp do tych plików przez osoby niepowołane może być bardzo niebezpieczny w skutkach. Oba urządzenia powinny należeć do użytkownika `root`, a ich prawa dostępu należy ustawić na `600`.

Niektóre urządzenia muszą mieć ustawione pełne prawa dostępu (odczyt i zapis). Do tej grupy należą z pewnością */dev/zero* i */dev/null*. Natomiast prawo tylko do odczytu na pewno potrzebne jest urządzeniom generatorów losowych danych, czyli */dev/random* i */dev/urandom*.

Restrykcje

Należy z rozwagą dobierać flagi montowania partycji. Czasami narzucenie zbyt restrykcyjnych reguł dostępu może stać irytujące dla administratora. Przykładowo, jeśli ktoś montowałby partycję z oprogramowaniem w trybie *tylko do odczytu*, będzie utrudniał sobie próby jego aktualizacji.

Ograniczenia powłoki

Powłoka Bash posiada wbudowane polecenie *ulimit*. Za jego pomocą można ustawić m.in. maksymalny rozmiar plików, pamięci wirtualnej, zużycie czasu procesora, największe rozmiary plików zrzutu pamięci (ang. *core file*), nieprzekraczalny limit otwartych na raz deskryptorów plików, maksymalną liczbę procesów, jakie dany użytkownik może rozpocząć oraz maksymalny rozmiar zajmowanej pamięci. Takie ograniczenia są bardzo ważne i mogą w zasadniczy sposób ułatwić administrację. Na przykład, gdy użytkownik uruchamia dużą liczbę procesów, czyni system mniej wydajnym i utrudnia korzystanie z niego przez innych użytkowników. Wydanie polecenia `ulimit -a` spowoduje wyświetlenie listy ograniczeń użytkownika, podobnej do tej widocznej na Listingu 3.

Widać wyraźnie, że większość ograniczeń ma wartość `unlimited`. Dlatego pierwszą rzeczą, jaką powinniśmy zrobić jest ograniczenie rozmiaru pliku zrzutu pamięci do zera, wydając polecenie `ulimit -c 0`. Następnie ograniczymy maksymalny rozmiar tworzonego pliku, na przykład do 1MB:

```
$ ulimit -f 1024
$ cat /dev/urandom > plik.txt
File size limit exceeded
```

W ten sposób możemy zapobiec tworzeniu ogromnych plików przez proces. Polecenie to niestety nie uniemożliwi utworzeniu wielu plików przez użytkownika (służy do tego mechanizm o nazwie *quota*).

Kolejny parametr *ulimit* umożliwia ograniczyć maksymalną liczbę procesów potomnych uruchamianych przez jednego użytkownika. Na Listingu 3 ograniczenie ma wartość `1022`, czyli stanowczo za dużo. Zmniejszymy ją dziesięciokrotnie:

```
$ ulimit -u 100
$ :(){ :|:&};:
-bash: fork: Resource temporarily
unavailable
```

Przedstawiony wyżej skrypt powłoki, tzw. *fork-bomba*, w bardzo krótkim czasie powoduje zużycie dostępnych zasobów, co może doprowadzić do zawieszenia systemu. Skrypt ten jest w rzeczywistości zwykłą funkcją rekurencyjną. Lepiej nie wywoływać jej w powłoce systemowej, jeśli nie zostały ustalone odpowiednie ograniczenia. W powyższym przykładzie ograniczenie ilości procesów uchroniło zasoby systemu.

Aby limity obowiązywały po każdym logowaniu użytkownika do systemu, musimy je zapisać w pliku konfiguracyjnym powłoki - */etc/profile*. Jeśli nowe ograniczenia mają dotyczyć tylko zwykłych użytkowników, wystarczy umieścić polecenia w instrukcji warunkowej, która sprawdza czy numer ID użytkownika jest większy od zera:

```
if [ `id -u` -gt 0 ]; then
  ulimit -f 1024 -u 100
fi
```

Powyższy sposób możemy wykorzystać do stosowania ograniczeń dla wybranych użytkowników lub grup użytkowników.

Mechanizm *ulimit* uniemożliwia ponowne zwiększenie własnych limitów przez zwykłego użytkownika (może tylko zmniejszyć). Ponadto każdy nowo uruchomiony program dziedziczy ustawienia dla powłoki.

```
$ ulimit -u unlimited
-bash: ulimit: cannot modify limit:
Operation not permitted
```

Użytkownicy przyjmują ograniczenia bez entuzjazmu i trudno się temu dziwić. Jeśli jednak nałożone przez nas limity nie będą zbyt rygorystyczne to na pewno nie zauważą żadnych zmian.

Na koniec

Instalując i konfigurując system warto kierować się zasadą, że jeśli coś jest zbędne i niepotrzebne to należy to usunąć lub chociaż wyłączyć. W pewnym stopniu możemy wykluczyć taką sytuację poprzez uniknięcie standardowej instalacji systemu. Dlatego zawsze, gdy jest to możliwe, powinniśmy uruchamiać szczegółową procedurę instalacji i w ten sposób zrezygnować z dołączania niepotrzebnych pakietów.

Bezpieczeństwo jest procesem rozpoczynającym się już podczas instalacji systemu. Tworzymy wtedy fundament naszego systemu. To od naszych decyzji zależy czy ów fundament będzie solidny i czy wytrzyma próbę czasu.