Odzyskiwanie danych z linuksowych systemów plików

Bartosz Przybylski

Kiedy, na przykład w wyniku włamania, zdarzy się nam utrata ważnych plików w Linuksie, nie musimy rozpaczać. Istnieje wiele metod odzyskania danych. Choć często jest to czasochłonne zajęcie, dobry zestaw narzędzi pozwoli na odzyskanie nawet całej zawartości uszkodzonego systemu plików.

Nasz serwer padł ofiarą włamywacza. Intruz był na tyle złośliwy, że pokasował nam z dysku sporo ważnych plików, w tym program nad którym pracowaliśmy parę miesięcy. Zanim zrobimy reinstalację systemu (aby nie pozostał w nim na pewno żaden złośliwy kod pozostawiony przez włamywacza) warto byłoby odzyskać dane. Aby to zrobić, musimy posłużyć się kilkoma narzędziami, które zawiera każda dystrybucja Linuksa.

Potrzebne narzędzia

Pierwszym niezbędnym elementem jest zbiór narzędzi do pracy na systemach plików *ext2* i *xt3* – mowa o pakiecie *e2fsprogs*. Dla nas najważniejszy będzie *debugfs*, który, jak nazwa wskazuje, służy do debugowania systemu plików. Standardowo (dla platformy x86) cały pakiet instalowany jest razem z systemem.

Następnym niezbędnym narzędziem jest *reiserfsck* będący częścią pakietu *reiserfsprogs*, służącego do edycji systemu plików *ReiserFS*. Ten pakiet również powinien być załączony do systemu. Z kolei program *dd* posłuży nam do odzyskania całej partycji z systemem plików *ReiserFS* i jako alternatywa do odzyskania danych z różnych typów systemów plików.

Przygotowanie partycji do odzyskania danych

Niezależnie od systemu plików, z którego będziemy odzyskiwać dane, musimy odmontować partycję, na której będziemy pracować. Aby mieć chociaż cząstkową pewność, że nasze dane nie zostały w żaden sposób naruszone powinniśmy ten krok wykonać jak najszybciej po usunięciu plików.

Aby odmontować partycję wystarczy `umount /dev/hdaX` (gdzie X to numer partycji, z której skasowano dane, w naszym przypadku nosi ona nu-

Z artykułu dowiesz się...

- jak odzyskiwać dane z systemów plików typu *ext2* i *ext3*,
- w jaki sposób uratować pliki z partycji *ReiserFS*.

Co powinieneś wiedzieć...

- powinieneś umieć posługiwać się linią poleceń w Linuksie,
- powinieneś znać podstawy budowy systemów plików.

Pojęcia związane z przestrzenią dyskową

I-węzły

I-węzeł (ang. *inode*) to struktura danych używana w linuksowych systemach plików do opisu pliku. W skład i-węzła wchodzi:

- typ pliku – plik zwykły, katalog lub plik urządzenia,
- identyfikator UID właściciela,
- wykaz bloków dyskowych i ich fragmentów tworzących plik.

I-węzeł możemy traktować jako swoisty identyfikator pliku na dysku, którym system posługuje się w celu odnalezienia żądanego pliku. Każdy plik na danej partycji ma przyporządkowany tylko jeden i-węzeł.

Blok dyskowy

Blok dyskowy to przechowująca informacje część przestrzeni na partycji. Rozmiar bloku definiowany jest przez użytkownika podczas podziału dysku na partycje. Może jednak zostać zmieniony przy użyciu programów modyfikujących dany system plików. W przeciwieństwie do i-węzłów, wiele bloków może należeć do jednego pliku.

Księgowanie

Księgowanie (ang. *journaling*, rejestrowanie zmian) jest jedną z metod przechowywania danych na dysku. Zasada jest prosta, ale nadzwyczaj skuteczna. Nieco uproszczony schemat działania widoczny jest na Rysunku 1.

Jak widać, *Plik1* po zmodyfikowaniu nie zmieni danych zawartych w swoim starym położeniu (w przeciwieństwie do systemów plików bez księgowania), lecz dane zostaną zapisane w nowym miejscu. Jest to duża zaleta – gdy dojdziemy do wniosku, że poprzednia wersja była lepsza, nawet po znacznej modyfikacji możemy odzyskać starą postać pliku.

mer 10). Jeżeli jednak podczas tej operacji otrzymamy komunikat:

```
# umount /dev/hda10
umount: /tmp: device is busy
```

oznacza to, że jakiś proces korzysta z tej partycji.

Z takiej sytuacji są dwa wyjścia. Jednym z nich jest zabicie procesu wykorzystującego daną partycję. Najpierw trzeba jednak sprawdzić, jakie procesy blokują partycję. Skorzystamy z programu *fuser*, służącego do identyfikacji użytkowników i procesów korzystających z określonych plików lub soketów:

```
# fuser -v -m  /dev/hda10
```

Opcja `-m /dev/hda10` nakaże programowi sprawdzić jakie usługi używają partycji *hda10*. Natomiast przełącznik `-v` (*verbose*) uczyni dane wyjściowe bardziej szczegółowymi, przez co zamiast samych numerów PID ujrzymy także zerowe argumenty programów. Jeśli stwierdzimy, że procesy są nam zbędne, wystarczy je zabić poleceniem:

```
fuser -k -v -m /dev/hda10
```

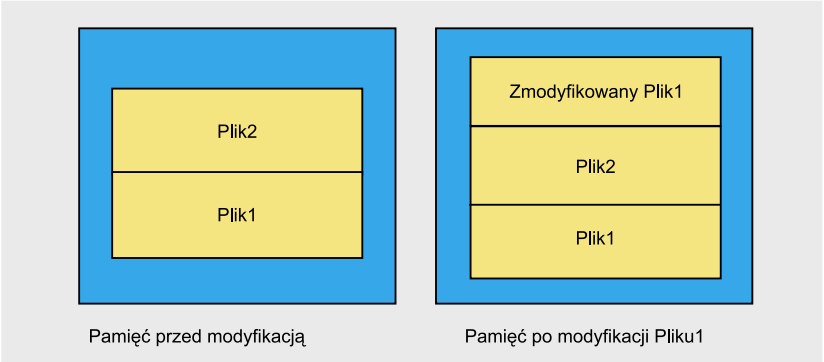
Jeśli natomiast wolimy normalnie zakończyć procesy, powinniśmy wykonać:

```
# fuser -TERM -v -m /dev/hda10
```

Drugim sposobem na odmontowanie systemu plików jest przełączenie go w tryb RO (*read only*). W ten sposób nasze pliki nie będą mogły zostać nadpisane. Aby wykonać tę czynność, wydajmy następujące polecenie:

```
# mount -o ro, remount /dev/hda10
```

Uwaga: polecenie nie zadziała, jeżeli partycja to *root directory*, czyli główny system plików. Jeżeli tak w istocie jest, musimy powiadomić o tym program *mount*, aby zmian nie zapisał do pliku */etc/mtab*. W tym celu dodajemy przełącznik `-n`.



Rysunek 1. *Schemat działania księgowania*

Odzyskiwanie danych w Ext2fs

Pierwszym rodzajem systemu plików, jakim się zajmiemy jest *ext2fs* (aby dowiedzieć się nieco więcej o tym i innych systemach plików, warto zajrzeć do Ramki *Linuksowe systemy plików*). Zaczniemy od odnalezienia skasowanych i-węzłów.

Szukanie skasowanych i-węzłów

Aby wykonać ten krok, użyjemy programu *debugfs* z pakietu *e2fsprogs*. Uruchommy aplikację otwierając żądaną partycję:

```
# debugfs /dev/hda10
```

Gdy ukaże się znak zachęty, powinniśmy wykonać polecenie *lsdel*, które pokaże nam wszystkie skasowane pliki od czasu stworzenia tej partycji (w przypadku systemów publicznych lista ta może mieć tysiące linii, jej stworzenie wymaga czasem trochę czasu). Teraz – wyłącznie na podstawie daty skasowania, UID użytkownika i rozmiaru – możemy

Listing 1. *Efekt polecenia lsdel programu debugfs*

```
debugfs: lsdel
Inode  Owner  Mode    Size    Blocks  Time deleted
(...)
   20  0      100644 41370  14/14   Tue Feb 15 19:13:25 2005
   24  0      100644 17104  5/5     Tue Feb 15 19:13:26 2005
352 deleted inodes found.
debugfs:
```

Listing 2. *Zrzucenie odzyskanych danych do pliku*

```
debugfs: dump <24> /home/aqu3l/recovered.000
debugfs: quit
# cat /home/aqu3l/recovered.000
(...)
```

wywnioskować, które pliki należały do nas i które chcemy odzyskać. Dobrym pomysłem jest spisanie lub wydrukowanie numerów i-węzłów.

Przyjrzyjmy się bliżej wynikowi polecenia *lsdel* (patrz Listing 1). Kolumny w wynikach polecenia *lsdel* przedstawiają kolejno:

- numer i-węzła (*inode*),
- właściciela (*owner*),
- opcje dostępu (*mode*),
- rozmiar w bajtach (*size*),
- liczbę zajmowanych bloków (*blocks*),
- czas skasowania (*time deleted*).

Jak widać, skasowane pliki mają numery i-węzłów równe 20 i 24. To właśnie te dane spróbujemy odzyskać.

Zrzucanie danych

Możemy teraz spróbować odzyskać i-węzeł 24 poprzez zrzucenie (ang. *dump*) danych do innego pliku. Jak widać na Listingu 1, zajmuje on 5 bloków. Jest to dość ważna informacja – ta metoda może nie zawsze skutkować przy plikach zajmujących więcej niż 12 bloków. Przykład takiego odzyskania znajduje się na Listingu 2.

W nawiasach ostrych podajemy nazwę pliku bądź numer i-węzła. Drugim parametrem jest nazwa pliku docelowego – należy ją podawać z pełną ścieżką dostępu, więc skrótowe `~/` nie poskutkuje.

Po wykonaniu polecenia wpisujemy *quit* i czytamy zawartość odzyskanego pliku. Często na końcu odzyskanego pliku mogą się pojawić różne znaki-śmieci; są to pozostałości po innych nadpisanych plikach. Można je usunąć przy użyciu dowolnego edytora tekstu. Metoda ta skutkuje tylko w przypadku plików tekstowych.

Pozostał nam do odzyskania plik z i-węzła 20 (patrz Listing 1). Zajmuje 14 bloków, a jak wspomnieliśmy, metoda zrzucenia danych z i-węzła liczącego więcej niż 12 bloków nie kończy się powodzeniem (patrz Ramka *Bloki i ich hierarchia w ext2fs*). Dlatego do odzyskania 20. i-węzła użyjemy programu *dd*.

Zanim odzyskamy plik, sprawdźmy podstawowe dane, czyli numery bloków i rozmiar bloku na partycji. Aby sprawdzić rozmiar bloku, użyjemy polecenia:

```
# dumpe2fs /dev/hda10 \
  | grep "Block size"
```

W odpowiedzi powinniśmy otrzymać:

```
dumpe2fs 1.35 (28-Feb-2004)
Block size:               4096
```

Linuksowe systemy plików

Ext2fs

System plików, którego głównym twórcą jest Theodore Ts'o. Nie posiada księgowania. Został zaprojektowany w taki sposób, aby możliwe było odzyskanie danych z partycji. Jest jednym z najpopularniejszych (właśnie ze względu na łatwość odzyskiwania) uniksowych systemów plików.

Ext3fs

Teoretycznie kolejna wersja *ext2*. Choć nie został zaprojektowany tak dobrze jak jego poprzednik, to oferuje możliwość księgowania. Ma także swoje wady – jedną z nich jest to, że projektanci nie przewidzieli w *ext3* możliwości odzyskania skasowanego pliku. Dzieje się tak, ponieważ system po oznaczeniu pliku jako usuniętego zwalnia także zajmowany przez niego i-węzeł, uniemożliwiając w ten sposób odczytanie usuniętych i-węzłów.

ReiserFS

System plików stworzony przez firmę NameSys, a dokładniej głównie przez Hansa Reisera, (stąd nazwa). Także udostępnia księgowanie; został zbudowany na algorytmie zbilansowanego drzewa (ang. *balanced tree*). Więcej informacji o specyficznej budowie *reiserfs* można znaleźć na stronie WWW twórców (patrz Ramka *W Sieci*).

Jfs

Jfs (*IBM's Journaled File System for Linux*) jest systemem plików napisanym przez IBM dla platformy Linux. Miał na celu usprawnienie komunikacji z produktami IBM. Opiera się na podobnej zasadzie księgowania co reszta stosujących go systemów. Oznacza to, że nowo zapisane dane wędrują na początek dysku, a informacje w bloku głównym zostają zaktualizowane.

Xfs

Extended filesystem zaprojektowany został z myślą o komputerach, które wymagają przechowywania dużej ilości plików w jednym katalogu i muszą mieć do nich błyskawiczny dostęp. Choć projektowany głównie z myślą o Iriksie, znalazł także zastosowanie w superkomputerach działających z systemem GNU/Linux. Ciekawostką jest fakt, że system potrafi przechowywać w jednym katalogu nawet 32 miliony plików.

Bloki i ich hierarchia w ext2fs

Bloki na dysku nie są jednym ciągiem przypisanym do pliku (i-węzła). W pewnych miejscach(zależnych od systemu plików, nie użytkownika) występują tzw. bloki pośredniczące, w trzech rodzajach:

- blok pośredni (ang. *indirect block*) – IND,
- blok podwójnie pośredni (ang. *double indirect block*) – DIND,
- blok potrójnie pośredni (ang. *triple indirect block*) – TIND.

Każdy kolejno numerowany blok jest zależny od tego numerowanego wyżej, ale też każdy kolejny może przechowywać większą ilość bloków:

- numery pierwszych 12 bloków przechowywanie są bezpośrednio w i-węźle (to one często nazywane są blokami pośrednimi),
- i-węzeł zawiera numer pośredniego bloku; blok pośredni zawiera numery kolejnych 256 bloków z danymi,
- i-węzeł zawiera numer podwójnie pośredniego bloku; blok podwójnie pośredni zawiera numery dodatkowych 256 bloków pośrednich,
- i-węzeł zawiera numer potrójnie pośredniego bloku; blok potrójnie pośredni zawiera numery dodatkowych 256 bloków podwójnie pośrednich.

Strukturę przedstawia Rysunek 2.

Właśnie ta ostatnia liczba (4096) jest rozmiarem bloku. Teraz, gdy mamy już rozmiar bloku, sprawdźmy bloki do odzyskania. Tę operację widzimy na Listingu 3 – zwróćmy uwagę, że blok 22027 jest blokiem pośrednim (IND).

Interesuje nas przedostatnia linia, w niej właśnie podane są bloki należące do danego i-węzła. Wykorzystajmy program *dd* do odzyskania bloków od 0 (od tej liczby zawsze rozpoczynamy liczenie bloków) do 11:

Listing 3. *Sprawdzenie bloków do odzyskania*

```
# debugfs /dev/hda10
debugfs: stat <20>
Inode: 20   Type: regular   Mode: 0644   Flags: 0x0   Generation: 14863
User: 0   Group: 0   Size: 41370
(...)
BLOCKS:
(0-11):22015-22026, (IND): 22027, (12):22028
TOTAL: 14
```

Listing 4. *Odzyskanie plików przez bezpośrednią modyfikację i-węzła*

```
# debugfs -w /dev/hda10
debugfs: mi <24>
       Mode            [0100644]
       User ID         [0]
       Group ID        [0]
(...)
        Deletion time  [1108684119]  0
       Link count      [0]  1
(...)
debugfs: quit
# e2fsck -f /dev/hda10
e2fsck 1.35 (28-Feb-2004)
(...)
Unattached inode 14
Connect to /lost+found<y>? yes
(...)
```

```
# dd bs=4k if=/dev/hda10 \
  skip=22015 count=12 \
  > ~/recovered.001
# dd bs=4k if=/dev/hda10 \
  skip=22028 count=1 \
  >> ~/recovered.001
```

Kilka słów wyjaśnienia:

- `bs` oznacza rozmiar bloku (podany w kilobajtach), który otrzymywaliśmy wcześniej,
- `if` oznacza plik wejściowy (ang. *input file*),
- `skip` nakazuje programowi przeskoczyć 22015 bloków o zadanym rozmiarze `bs`,
- `count` oznacza liczbę bloków do zebrania.

Blok 22027 jest podwójnie pośredni, więc ominęliśmy go i od razu zebraliśmy blok 22028.

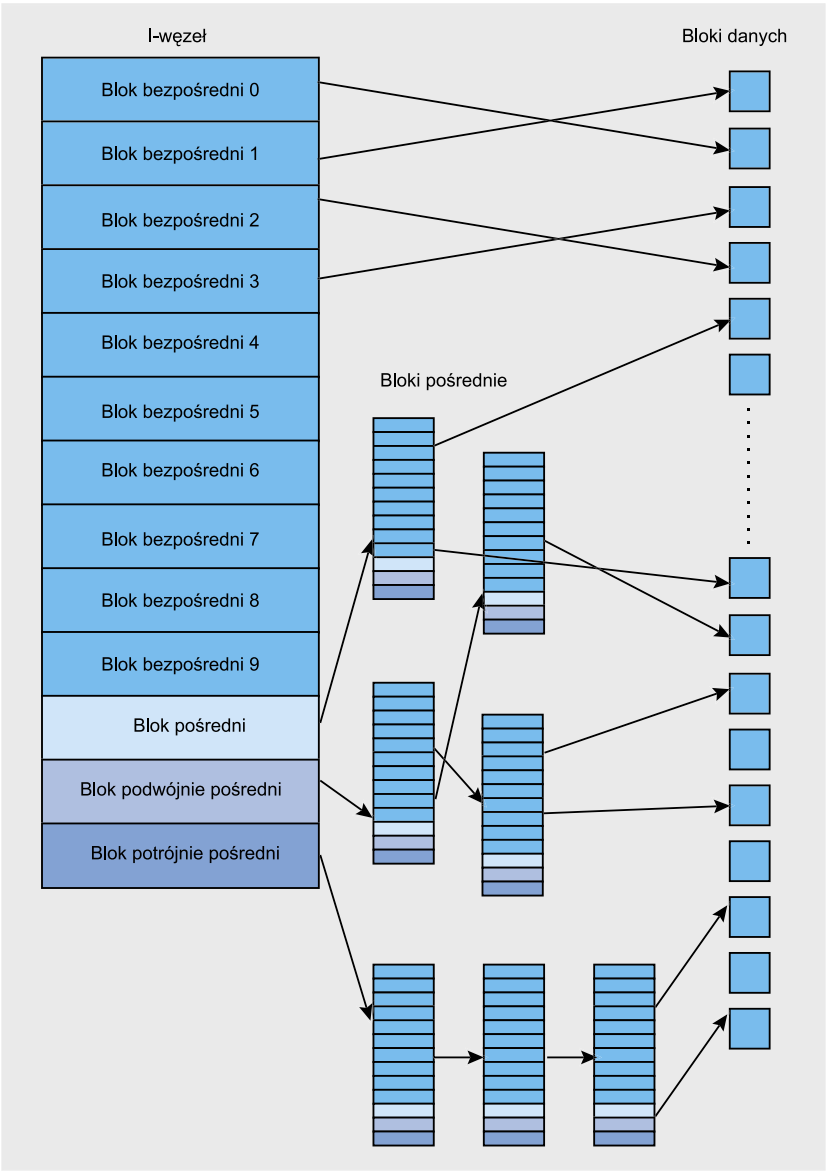
Modyfikacja i-węzłów

Teraz zajmiemy się innym sposobem odzyskiwania danych – bezpośrednią modyfikacją i-węzłów. Polega ona na takiej zmianie i-węzła, żeby system plików potraktował odpowiednie dane jako nigdy nie kasowane i przy najbliższym sprawdzeniu dysku przeniósł skasowany plik do folderu *lost+found* na danej partycji. Do modyfikacji także użyjemy programu *debugfs*, a przebieg całej operacji znajduje się na Listingu 4.

Jak widać, modyfikacji uległy tylko dwa wpisy: czas skasowania (*deletion time* – nie jest to jednak do końca prawda, bo system nie jest przecież w stanie określić daty usunięcia pliku) oraz liczba dowiązań do pliku (*link count*). Teraz, po zakończeniu pracy przez *debugfs*, wystarczy wykonać polecenie:

```
# e2fsck -f /dev/hda10
```

Program po napotkaniu zmodyfikowanego i-węzła uzna, że jest on bez nadzoru (ang. *unattached*) i zapyta, czy dane opisane w tym i-węźle dowiązać do folderu *lost+found*. Jeżeli zależy nam na pliku, to oczywiście wciskamy klawisz *y*. Jednak nie ma róży bez ognia – po zajrze-



Rysunek 2. *Struktura bloków w systemie plików ext2*

Listing 5. *Wyszukanie skasowanych i-węzłów w ext3fs*

```
# debugfs /dev/hda10
debugfs:  lsdel
 Inode      Owner        Mode       Size       Blocks          Time deleted
0 deleted inodes found.
debugfs: q
```

Listing 6. *Odzyskanie danych z partycji ext3 zamontowanej jako ext2*

```
debugfs: lsdel
Inode  Owner  Mode   Size  Blocks  Time deleted
(...)
   20  0      100644 41370 14/14   Tue Feb 14 19:20:25 2005
(...)
   24  0      100644 17104 5/5     Tue Feb 15 19:13:26 2005
352 deleted inodes found.
debugfs:
```

niu do folderu zobaczymy nie eleganckie nazwy plików, lecz wyłącznie numery odbudowanych i-węzłów (np. *24*). Należy więc przejrzeć plik i po treści rozpoznać jego oryginalną nazwę.

Ext3fs

Odzyskiwanie danych w tym systemie plików jest specyficzne, czasem nawet bardzo czasochłonne (patrz też Ramka *Linuksowe systemy plików*). Prawdę mówiąc, nie ma żadnego zatwierdzonego sposobu odzysku z tego typu partycji. Istnieją jednak nieoficjalne metody ratowania danych.

Czy to ext3 czy ext2?

Ext3 i *ext2* są bardzo podobnymi systemami plików (z wyjątkiem księgowania i sposobu kasowania plików) – wykorzystajmy więc ten fakt, aby odzyskać nasze dane. Spróbujemy użyć *debugfs*; proces ten przedstawiono na Listingu 5.

Spójrzmy na Listing 5. Nasze i-węzły zostały skasowane przez system plików. Wybrana przez nas droga z pozoru prowadzi donikąd.

Możemy jednak spróbować pewnej sztuczki – sprawić, by system operacyjny traktował system plików jako *ext2*. Rozwiązanie to dzieli się na trzy etapy:

- odmontowanie systemu plików,
- ponowne zamontowanie, ale tym razem jako *ext2*,
- odzyskanie plików.

Odmontujmy więc partycję:

```
# umount /dev/hda10
```

Następnie musimy ją ponownie zamontować jako *ext2*, dla większego bezpieczeństwa w trybie *read only*:

```
# mount -o ro -t ext2 \
  /dev/hda10 /tmp
```

Teraz spróbujmy pracować z *debugfs* w sposób, który przedstawiliśmy przy omawianiu systemu *ext2*. Wyszukiwanie usuniętych z partycji

ext3 i-węzłów przedstawiono na Listingu 6.

I-węzeł 20 ma nieprawdziwą datę skasowania. Dzieje się tak dlatego, że po zwolnieniu i-węzła przez *ext3* system *ext2* może mieć problemy z odczytaniem poprawnych danych o plikach.

Po dokładne analizie całej listy usuniętych plików możemy już zająć się odzyskaniem tych, których potrzebujemy. Metoda jest taka sama jak w przypadku *ext2*, jednak *ext3* może mieć problem po bezpośrednim zmodyfikowaniu i-węzła. W niektórych przypadkach może to nawet doprowadzić do uczynienia partycji nieczytelną dla systemu.

Żmudna praca się opłaca

Druga metoda na odzyskanie plików z *ext3* jest o wiele trudniejsza, umożliwia jednak odzyskanie znacznie większej liczby skasowanych plików tekstowych. Ta metoda także, niestety, ma poważną wadę – wymaga ręcznego przeglądania zawartości dysków, więc uratowanie plików binarnych jest bardzo trudne.

Dobrym pomysłem jest wcześniejsze wykonanie kopii zapasowej całego dysku. Zróbmy to poleceniem:

```
$ dd if=/dev/hda10 \
  >~/hda10.backup.img
```

Aby choć trochę ułatwić sobie pracę, można podzielić naszą partycję na mniejsze części. Jeżeli partycja ma pojemność 1 GB, to rozsądnie będzie ją podzielić na 10 części po 100 MB. Prosty skrypt przeznaczony do tego celu przedstawiono na Listingu 7 – dysk możemy podzielić poleceniem:

Listing 7. *dsksplitter.pl – prosty skrypt do dzielenia dysków*

```
#!/usr/bin/perl
if ($ARGV[3] eq "") {
  print "Usage:\ndsksplitter.pl <dsk_parts> <part_size in Kb>
        <partition_to_split> <target_dir>";
  }
else {
  $parts = $ARGV[0];
  $size = $ARGV[1];
  $partition = $ARGV[2];
  $tardir = $ARGV[3];
  for ($i = 1; $i <= $parts; $i++) {
    system "dd bs=1k if=$partition of=$tardir/dks.$i count=$size
            skip=$ix$size";
  }
}
```

```
$ dsksplitter.pl 10 1000000 \
  /dev/hda10 ~/dsk.split
```

Teraz skorzystajmy z systemowego polecenia *grep* w celu wyszukania interesujących nas ciągów znakowych (do tej czynności można oczywiście użyć polecenia *strings*):

```
$ grep -n -a -1 \
  "int main" ~/dsk.split/*
```

Przełącznik `-n` pokaże nam numer wiersza pliku, w którym znajduje się ciąg. Przełącznik `-a` nakazuje traktowanie plików binarnych jak tekstowych, natomiast `-1` wyświetli jeden wiersz przed i jeden wiersz po znalezionym ciągu. Oczywiście możemy zmienić ciąg `int main` na dowolny. Otrzymaliśmy wyniki:

```
~/dsk.split/dsk.1:40210:←
  #include <sys/socket.h>
~/dsk.split/dsk.1:40211:←
  int main (int argc, char *argv[])
~/dsk.split/dsk.1:40212:←
  { (...)
```

Ext3 zapisuje nowe pliki na początku dysku, możemy więc przypuszczać, że znaleziona przez nas linia jest tą, której poszukujemy. Spróbujmy więc jeszcze raz podzielić plik na mniejsze części i tam poszukać danych:

```
$ mkdir ~/dsk1.split
$ dsksplitter.pl 10 10000 \
  ~/dsk.split/dsk.1 ~/dsk1.split
```

Wykonajmy teraz polecenie *grep* na podzielonym pliku *dsk.1*:

```
$ grep -n -a -1 \
  "int main" ~/dsk1.split/*
```

Otrzymany wynik:

```
~/dsk1.split/dsk.3:143:←
  #include <sys/socket.h>
~/dsk1.split/dsk.3:144:←
  int main (int argc, char *argv[])
~/dsk1.split/dsk.3:145:←
  { (...)
```

Teraz mamy już plik z programem, który skasował włamywacz. Co prawda plik, w którym go odnaleźliśmy ma 10 MB, ale to i tak daleko lepsza perspektywa niż przeszukiwanie 1 GB danych. Jeżeli jednak ta dokładność nam nie wystarczy, możemy pokusić się o podzielenie pliku badanego fragmentu dysku na jeszcze mniejsze części. Gdy plik zo-

W Sieci

- *http://e2fsprogs.sourceforge.net* – strona pakietu *e2fsprogs*,
- *http://web.mit.edu/tytso/www/linux/ext2fs.html* – strona domowa *ext2fs*,
- *http://www.namesys.com* – strona domowa twórców *ReiserFS*,
- *http://oss.software.ibm.com/developerworks/opensource/jfs* – witryna systemu plików *jfs*,
- *http://oss.sgi.com/projects/xfs* – strona projektu *xfs*,
- *http://www.securiteam.com/tools/6R00T0K06S.html* – pakiet *unrm*.

stanie już odpowiednio zmniejszony, pozostaje nam uruchomić edytor tekstu i zająć się żmudnym usuwaniem zbędnych linii.

Technika ta jest czasochłonna, jednak skuteczna. Została przetestowana w kilku dystrybucjach Linuksa, choć nie możemy ręczyć, że ten sposób zadziała na wszystkich systemach linuksowych.

Odzyskiwanie w ReiserFS

Do odzyskania plików posłużymy się standardowymi linuksowymi programami. Zaczniemy od *dd* – użyjemy go do stworzenia obrazu partycji. Jest to niezbędne, ponieważ działania które będziemy wykonywać mogą wyrządzić nieodwracalne szkody. Wykonajmy zatem polecenie:

```
$ dd bs=4k if=/dev/hda10 \
  conv=noerror \
  > ~/recovery/hda10.img
```

gdzie `/dev/hda10` to partycja do odzyskania, a `bs` (*block size*) określiliśmy poleceniem:

```
$ echo "Yes" | reiserfstune \
  -f /dev/hda10 | grep "Blocksize"
```

Parametr `conv=noerror` spowoduje konwersję do pliku bez przekazywania błędów, co oznacza, że nawet jeśli program napotka błędy na dysku, to nadal przetworzy dane do pliku. Po wpisaniu polecenia będziemy zmuszeni odczekać odpowiedni czas, w zależności od rozmiaru partycji.

Teraz należy przenieść zawartość naszego obrazu partycji na urządzenie pętli zwrotnej *loop0*, wcześniej upewniając się, że jest wolne:

```
# losetup -d /dev/loop0
# losetup /dev/loop0 \
  /home/aqu3l/recovery/hda10.img
```

Następnie trzeba *odbudować drzewo* – cała partycja zostanie sprawdzona, zaś jakiekolwiek pozostałości po i-węzłach zostaną naprawione i przywrócone. Służy do tego komenda:

```
# reiserfsck -rebuild-tree -S \
  -l /home/aqu3l/recovery/log\
  /dev/loop0
```

Dodatkowy przełącznik `-S` sprawi, że sprawdzeniu ulegnie cały dysk, a nie tylko jego zajęta część. Przełącznik `-l` z parametrem `/home/user/recovery/log` spowoduje zapisanie logu do wskazanego katalogu. Teraz tworzymy katalog na naszą partycję i montujemy ją:

```
# mkdir /mnt/recover; \
  mount /dev/loop0 /mnt/recover
```

Odzyskane pliki mogą się znajdować w jednym z trzech miejsc. Jedno to oryginalny katalog pliku (traktujemy */mnt/recover/* jako *root directory*). Drugie to katalog *lost+found* w naszym czasowym katalogu głównym (*/mnt/recover*). Trzecie to po prostu główny katalog partycji.

Szukany plik prawie na pewno znajduje się w jednym z trzech wymienionych miejsc. Jeżeli go tam nie ma, mamy dwa wyjaśnienia tej sytuacji – albo był pierwszym plikiem na partycji i został zamazany, albo został omyłkowo przeniesiony do innego katalogu. W pierwszym przypadku możemy pożegnać się z naszymi danymi, natomiast w drugim możemy liczyć na odnalezienie go w innym miejscu, korzystając z narzędzia *find*:

```
find /mnt/recover \
  -name nasz_plik
```

Odzyskiwanie ostatnio zmodyfikowanego pliku

Skupimy się teraz na odzyskaniu tylko jednego, ostatnio zmodyfikowanego pliku. Metodę tę można także przekształcić na pliki dawne, jednak próba takiego odzyskania opiera się na żmudnych obliczeniach, dobrej znajomości własnego systemu plików oraz szczęściu.

Jak widać na Rysunku 1, w systemach plików z księgowaniem nowe pliki zapisywanie są na samym początku dysku. Teoretycznie nasz plik znajduje się zaraz za tzw. blokiem głównym (ang. *root block*), czyli blokiem dyskowym określającym miejsce, od którego zaczynają się dane.

Aby określić położenie naszego *root block* należy wydać polecenie:

```
# debugreiserfs /dev/hda10 \
  | grep "Root block"
```

W odpowiedzi powinniśmy otrzymać coś w rodzaju:

```
debugreiserfs 3.6.17 (2003 www.namesys)
Root block: 8221
```

Jak łatwo się domyślić, 8221 to numer naszego bloku głównego.

Musimy też przynajmniej w przybliżeniu określić rozmiar naszego pliku – powiedzmy, że miał on 10 kB, więc trzykrotność rozmiaru bloków powinna być wystarczająca. Gdy mamy już informacje o pliku, możemy wykonać polecenie:

```
# dd bs=4k if=/dev/hda10 \
  skip=8221 count=3 \
  > ~/recovered.003
```

Po odzyskaniu danych należy sprawdzić ich zgodność z szukanym plikiem:

```
# cat ~/recovered.003
```

Tak jak w przypadku *ext2fs* pod koniec pliku można napotkać różnego rodzaju śmieci – z łatwością je usuniemy.

Ułatwić sobie życie

Istnieją programy, które automatyzują przedstawione sposoby odzyskiwania danych. Najwięcej tego typu narzędzi działa z systemem plików *ext2*. Godny polecenia jest pakiet *unrm* oraz napisana przez Oliviera Diedricha biblioteka *e2undel*, działająca z pakietem *e2fsprogs*. Oczywiście nie powinniśmy się zawsze spodziewać, że odzyskamy 100 procent skasowanych plików (choć czasem jest to możliwe) – jeżeli uda nam się uratować około 80 procent dużego pliku, będziemy mogli uznać to za sukces. ■