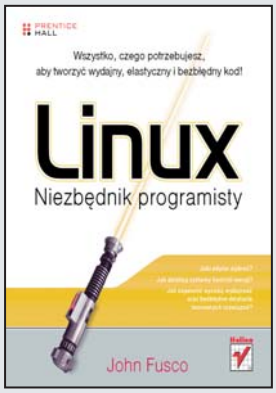
Linux. Niezbędnik programisty

Autor: John Fusco
ISBN: 978-83-246-1485-1
Tytuł oryginału: The Linux Programmer's Toolbox
Format: 200x230, stron: 736



Wszystko, czego potrzebujesz,
aby tworzyć wydajny, elastyczny i bezbłędny kod!

- Jaki edytor wybrać?
- Jak działają systemy kontroli wersji?
- Jak zapewnić wysoką wydajność oraz bezbłędne działanie tworzonych rozwiązań?

Programista, jak każdy fachowiec, posiada perfekcyjnie dobrany zestaw narzędzi, pozwalający mu na szybkie, wygodne, elastyczne i – co najważniejsze – optymalne rozwiązywanie postawionych problemów. Wybranie odpowiednich narzędzi i skomponowanie ich zbioru zabiera często wiele dni, miesięcy, a nawet i lat. A przecież my, programiści, nie mamy aż tyle czasu! Koniecznie trzeba znaleźć jakiś szybszy sposób!

Najlepszą odpowiedzią na ten problem jest właśnie niniejsza książka. Dzięki niej opanujesz sposoby pobierania i instalacji różnych narzędzi, a nowo nabyta wiedza na temat sposobów zarządzania pakietami w różnych dystrybucjach na pewno nie pójdzie na marne. W kolejnych rozdziałach poznasz przebieg procesu kompilacji oraz dowiesz się, jak interpretować poszczególne błędy i ostrzeżenia. John Fusco omawia tu także edytory Vim oraz Emacs – ale nie wskazuje, który z nich jest lepszy! Dzięki książce „Linux. Niezbędnik projektanta" wykorzystanie systemów kontroli wersji nie będzie stanowiło dla Ciebie najmniejszego problemu. Na kolejnych stronach szczegółowo omawiane są zasady funkcjonowania jądra systemu oraz sposób działania procesów i komunikacji między nimi. Lektura kolejnych rozdziałów dostarczy Ci niezbędnych informacji na temat zapewniania wysokiej wydajności tworzonych rozwiązań oraz metod diagnozowania problemów z oprogramowaniem.

- Zdobywanie i instalacja oprogramowania open source
- Proces kompilacji kodu źródłowego
- Zarządzanie pakietami w różnych dystrybucjach
- Interpretacja komunikatów o błędach i ostrzeżeń
- Edytory plików tekstowych
- Wykorzystanie systemów kontroli wersji
- Wykorzystanie procesów
- Komunikacja między procesami
- Diagnozowanie problemów z komunikacją pomiędzy procesami
- Zwiększanie wydajności tworzonych rozwiązań
- Wykrywanie problemów w napisanym oprogramowaniu

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Rozdział 1

Pobieranie i instalacja narzędzi oferowanych w trybie open source

1.1. Wprowadzenie

W tym rozdziale omówię różne formy dystrybuowania darmowego oprogramowania, techniki jego stosowania i miejsca, gdzie należy go szukać. Szczegółowo opiszę pliki archiwalne i pliki pakietów, a także najbardziej popularne narzędzia utworzone z myślą o operowaniu na tych plikach.

Stosowanie oprogramowania nieznanych autorów bywa ryzykowne. W związku z tym omówię rozmaite zagadnienia związane z bezpieczeństwem, które warto mieć na uwadze, i techniki, które mogą nas chronić. Wprowadzę zagadnienia uwierzytelniania i zaufania, po czym spróbuję je odnieść do problematyki bezpieczeństwa. Z myślą o sytuacjach, w których uwierzytelnianie jest niemożliwe, zaproponuję techniki przeglądania pakietów i archiwów.

I wreszcie wprowadzę kilka narzędzi umożliwiających efektywne zarządzanie pakietami i dystrybucjami opartymi na pakietach oraz sposoby ich uzyskiwania.

1.2. Czym jest tryb open source?

Nazwa **open source** jest marketingowym określeniem darmowego oprogramowania tworzonego zgodnie z założeniami projektu Open Source Initiative (OSI)[1]. Organizację OSI założono z myślą o promowaniu darmowego oprogramowania, a faktycznym źródłem tej idei był projekt GNU Richarda Stallmana. Jednym z celów tej organizacji jest zwalczanie negatywnych stereotypów związanych z darmowym oprogramowaniem i propagowanie darmowego dzielenia się kodem źródłowym aplikacji.

Początkowo wiele organizacji biznesowych obawiało się stosowania oprogramowania oferowanego w trybie open source. Nie ma wątpliwości, że spory udział w tym stanie rzeczy miały działy marketingu wielkich przedsiębiorstw informatycznych. Jak mówi powiedzenie: „Otrzymujesz to, za co płacisz". Niektórzy obawiali się, że nowe licencje (np. GNU Public License) będą się rozprzestrzeniały jak wirus, który sprawi, że kod tworzony z wykorzystaniem darmowego oprogramowania także będzie musiał być upubliczniony.

Zdecydowana większość tych obaw została na szczęście rozwiana. Wiele ogromnych przedsiębiorstw korzysta obecnie z darmowego oprogramowania i promuje ideę otwartego dostępu do kodu źródłowego we własnych projektach. Co więcej, część koncernów korzysta wyłącznie z oprogramowania open source. Dżin ostatecznie został uwolniony ze swojej lampy.

1.3. Co idea otwartego dostępu do kodu źródłowego oznacza dla nas?

Dla większości z nas idea oprogramowania open source oznacza w istocie dostępność darmowych narzędzi wysokiej jakości. Okazuje się, niestety, że oprócz wspomnianego oprogramowania wysokiej jakości istnieje mnóstwo gorszych produktów, co jest nieuniknione. Dobre koncepcje projektowe są stale rozwijane, koncepcje chybione z czasem zanikają. W tej sytuacji wybór oprogramowania open source przypomina trochę wybór owoców — wskazanie naprawdę dojrzałych wymaga sporego doświadczenia.

Właściwie prowadzony proces wyboru musi uwzględniać wiele poziomów. Na poziomie kodu źródłowego dobieramy funkcje i konstrukcje (w tym łatki usuwające usterki), aby dysponować tylko najlepszymi rozwiązaniami. Jako konsumenci wybieramy i pobieramy z internetu te produkty, które w naszej ocenie zwiększają szanse powodzenia realizowanych projektów. Trudno oczekiwać, by ktoś tworzył kod, który nie będzie wykorzystywany w żadnym projekcie. Mniej pobieranego oprogramowania oznacza mniej pracy dla programistów. Więcej pobrań zawsze oznacza więcej pracy dla programistów, co z kolei przekłada się na szerszy wybór kodu i — tym

[1] Patrz witryna internetowa *http://www.opensource.org/*.

samym — lepszy kod. Zdarza się, że wybór właściwego projektu przypomina rozgrywkę hazardową, w której o sile naszych kart decyduje wyłącznie ilość poświęcanego czasu i wkładanego wysiłku. Sytuacje, w których będziemy żałować dokonanych wcześniej wyborów, są nieuniknione — warto więc pamiętać, że podobne zdarzenia są nieodłączną częścią tego procesu.

Dla części użytkowników niepełna wiedza o pobieranym oprogramowaniu bywa źródłem pozytywnych emocji. Traktują poznawanie nowych narzędzi jak rozpakowywanie prezentów urodzinowych. Inni użytkownicy uważają, że tego rodzaju eksperymenty są wyjątkowo kłopotliwe i czasochłonne. Okazuje się jednak, że także osoby zainteresowane wygodą związaną z gotowym oprogramowaniem, które wystarczy zainstalować i uruchomić, znajdą w internecie coś dla siebie — problem w tym, że wybór naprawdę dojrzałych projektów jest nieco mniejszy. Internet oferuje wiele zasobów, które ułatwiają wybór pożądanych projektów.

1.3.1. Odnajdywanie właściwych narzędzi

Pierwszym miejscem, od którego należy zacząć poszukiwania (zanim przystąpimy do przeglądania zasobów dostępnych w internecie), są płyty CD lub DVD naszej dystrybucji systemu Linux. Jeśli zainstalowaliśmy system znajdujący się na wielu nośnikach, najprawdopodobniej pominięto w tym procesie mnóstwo użytecznych narzędzi. Większość dystrybucji zawiera na płytach CD lub DVD dużo więcej oprogramowania, niż instaluje się zgodnie z ustawieniami domyślnymi. Z reguły użytkownik instalujący system operacyjny jest pytany o docelowe przeznaczenie danego komputera. Wskutek takiego wyboru na dysku jest instalowany określony podzbiór pakietów, który według twórców programu instalacyjnego najlepiej pasuje do profilu stacji roboczej czy serwera.

Zbiór zainstalowanych pakietów zawsze możemy rozszerzyć ręcznie, lokalizując interesujące nas narzędzia na płytach CD lub DVD. Wadą proponowanego podejścia jest brak jednego schematu organizowania pakietów i — tym samym — konieczność dysponowania dość precyzyjną wiedzą o poszukiwanych rozwiązaniach. Część dystrybucji oferuje jednak graficzne interfejsy organizujące pakiety w ramach kategorii, które znacznie ułatwiają dobór instalowanego oprogramowania.

Użytkownicy, którzy nie do końca wiedzą, czego szukają, powinni skorzystać z materiałów dostępnych w internecie. Istnieje wiele witryn internetowych, stworzonych wyłącznie z myślą o użytkownikach poszukujących oprogramowania open source. Jedną z takich witryn jest *http://freshmeat.net/*. Można tam znaleźć oprogramowanie uporządkowane według kategorii, co znacznie ułatwia przeszukiwanie oferowanych zasobów. W czasie prac nad tą książką wpisałem w wyszukiwarce Freshmeat wyrażenie *word processor* i znalazłem 71 dostępnych projektów. Aż trudno sobie wyobrazić wybór spomiędzy 71 edytorów tekstu!

Witryna Freshmeat umożliwia filtrowanie wyników przeszukiwania, co dodatkowo pozwala zawęzić zbiór dostępnych narzędzi. Początkowe wyniki obejmują oprogramowanie dla różnych systemów operacyjnych (w tym innych niż Linux), a także projekty na różnych etapach realizacji. W tej sytuacji podjąłem decyzję o ograniczeniu przeszukiwania do projektów tworzonych z myślą o systemie Linux, projektów dojrzałych i takich, które są oferowane zgodnie z licencją Open Source, zaakceptowaną przez organizację OSI. (Witryna Freshmeat domyślnie odnajduje także oprogramowanie komercyjne). W ten sposób ograniczyłem liczbę projektów do 12 — wybór spośród takiej liczby edytorów jest dużo prostszy. Bliższa analiza wyników ujawniła jednak, że część projektów nie do końca odpowiada moim oczekiwaniom i wynika z dość szerokiej interpretacji wyrażenia *word processor*. Po zastosowaniu kilku dodatkowych filtrów byłem w stanie zidentyfikować kilka znanych mi wcześniej, sprawdzonych projektów wysokiej jakości, np. edytor AbiWord, oraz kilka projektów, o których do tej pory nie słyszałem. Wyniki nie obejmowały jednak tak ważnych projektów jak pakiet OpenOffice, w którym pisałem tę książkę. Okazało się, że wspomniany pakiet nie znalazł się w wynikach przeszukiwania, ponieważ należy do kategorii *Office/Business — Office Suites*, a nie *Word Processors*. Moje doświadczenia pokazały, że jeśli nie możemy odnaleźć interesującego nas projektu, nie powinniśmy rezygnować, tylko starać się szukać do skutku.

1.3.2. Formaty dystrybucji oprogramowania

Kiedy już uda nam się odnaleźć potrzebne oprogramowanie, najprawdopodobniej staniemy przed kolejnym ważnym wyborem. Dojrzałe projekty z reguły oferują pakiety w postaci gotowej do zainstalowania, zwykle w co najmniej jednym formacie. Mniej dojrzałe projekty często mają postać kodu źródłowego lub plików binarnych, zawartych w pliku archiwalnym. Okazuje się, że sam format bywa cenną wskazówką sugerującą, z jakim pakietem mamy do czynienia. Pobieranie pliku pakietu dojrzałego oprogramowania można porównać z kupowaniem nowego samochodu — ponieważ nie wiemy, jak to działa, ograniczamy się do przekręcenia kluczyka w stacyjce. Pobieranie archiwum z plikami źródłowymi lub binarnymi bardziej przypomina kupowanie używanego samochodu — jeśli niewiele wiemy o samochodach, tak naprawdę nie wiemy, co kupujemy.

Jeśli jakiś projekt obejmuje pakiet gotowy do natychmiastowej instalacji, zwykle jest to sygnał o dojrzałości tego projektu. Co więcej, istnienie takiego pakietu sugeruje, że cykl wydań danego projektu jest dość stabilny. Gdyby nowe wydania tego projektu pojawiały się co tydzień, najprawdopodobniej nikt nie zawracałby sobie głowy tworzeniem pakietów. Dysponując plikiem pakietu oprogramowania i odrobiną szczęścia najprawdopodobniej będziemy w stanie od razu zainstalować i uruchomić nowe narzędzie. Okazuje się jednak, że podobnie jak w przypadku nowych samochodów, szybko możemy się przekonać, iż dokonany wybór nie był najlepszy.

Alternatywą dla pliku pakietu jest plik archiwum (plik archiwalny). W projektach tworzonych z myślą o systemie operacyjnym Linux archiwa z reguły mają postać skompresowanych plików z rozszerzeniem *.tar*. **Plik archiwalny** (ang. *archive file*) jest kolekcją plików upakowanych w jednym pliku za pomocą narzędzia archiwizującego, czyli np. polecenia `tar`. Tego rodzaju pliki z reguły są dodatkowo kompresowane za pomocą programu `gzip`, co pozwala oszczędzić cenną przestrzeń na dysku — tak przygotowane pliki określa się mianem **plików tar**.

Pliki tar są preferowanym formatem dystrybucji kodu źródłowego projektów. Ich tworzenie i stosowanie jest bardzo proste, a obsługa programu `tar` nie stanowi problemu dla żadnego programisty. Dużo rzadziej mamy do dyspozycji pliki tar obejmujące binarne pliki wykonywalne. Należy tego rozwiązania unikać, chyba że dobrze wiemy, co tak naprawdę robimy. Pliki tar powinny być przekazywane osobom, które dysponują pewną wiedzą o programowaniu i administracji systemami.

1.4. Wprowadzenie do tematyki plików archiwalnych

Podczas pobierania i instalacji oprogramowania open source w pewnym momencie zetkniemy się z plikiem archiwalnym w tej czy innej formie. Mianem **pliku archiwalnego** określamy każdy plik zawierający kolekcję innych plików. Użytkownicy systemu operacyjnego Windows zapewne znają dominujące na tej platformie narzędzie archiwizujące PKZip. Jego odpowiedniki pracujące w systemach Linux działają podobnie, z tą różnicą, że nie oferują możliwość kompresji. Narzędzia archiwizujące systemu Linux koncentrują się na samej archiwizacji i pozostawiają kompresję innym narzędziom (zwykle `gzip` lub `bzip2`). Na tym polega filozofia platformy Unix.

Ponieważ pracujemy w systemie Linux, mamy oczywiście szerszy wybór narzędzi archiwizujących. Jako konsumenci oprogramowania open source nie powinniśmy się jednak wykazywać przesadną wybrednością. Mimo że w naszej pracy będziemy mieli do czynienia przede wszystkim z plikami tar, warto przynajmniej wiedzieć o istnieniu innych narzędzi.

Narzędziom archiwizującym stawiamy większe wymagania niż tylko te związane ze składowaniem nazw i danych plików. Oprócz ścieżek i danych archiwum musi obejmować metadane właściwe dla poszczególnych plików. Przez określenie **metadane** (ang. *metadata*) rozumiemy właściciela pliku, grupę i inne atrybuty (w tym uprawnienia odczytu, zapisu i wykonywania). Narzędzie archiwizujące rejestruje wszystkie te informacje, co oznacza, że stwarza możliwość usunięcia pliku z systemu plików i jego późniejsze przywrócenie z pliku archiwalnego bez utraty jakichkolwiek informacji. Jeśli umieścimy w archiwum jakiś plik wykonywalny, po czym usuniemy go z systemu plików, odtworzenie tego pliku z archiwum umożliwi nam jego ponowne uruchomienie. W systemie Windows samo rozszerzenie pliku określa, czy mamy do czynienia

z plikiem wykonywalnym (tak jest np. w przypadku plików *.exe*); w systemie Linux o możliwości uruchamiania plików decydują ich metadane, dlatego konieczne jest składowanie tych informacji przez narzędzie archiwizujące.

Najbardziej popularne narzędzia archiwizujące, stosowane w systemie Linux, wymieniono i krótko opisano w tabeli 1.1. Zdecydowanie najpopularniejszym formatem archiwizowania plików jest *tar*. Nazwa tego narzędzia i formatu jest skrótem od angielskiego określenia *tape archive*, czyli nazwy stosowanych kiedyś systemów archiwizowania danych na taśmach magnetycznych. Narzędzie `tar` jest obecnie wykorzystywane przede wszystkim w roli uniwersalnego programu do archiwizacji kolekcji plików w ramach pojedynczych plików wynikowych. Mniej popularną alternatywą dla tego narzędzia jest `cpio`, czyli program realizujący te same zadania, ale stosujący zupełnie odmienną składnię poleceń. Istnieje też narzędzie `pax`, które jest zgodne ze standardem POSIX i które potrafi prawidłowo interpretować pliki narzędzia `tar`, pliki archiwalne programu `cpio` i oczywiście pliki we własnym formacie. Co prawda, nigdy nie spotkałem się z dystrybucją w formacie narzędzia `pax`, warto jednak o nim wspomnieć dla kompletności prowadzonych rozważań.

TABELA 1.1. Najpopularniejsze narzędzia archiwizujące

Narzędzie	Uwagi
`tar`	Najbardziej popularne narzędzie
`cpio`	Narzędzie stosowane wewnętrznie przez format RPM, niewykorzystywane nigdzie indziej
`ar`	Narzędzie wykorzystywane wewnętrznie przez mechanizm pakujący Debiana; poza tym systemem stosuje się je wyłącznie dla bibliotek budowanego oprogramowania. Narzędzie `ar` nie składuje w archiwach informacji o ścieżkach

Warto poświęcić trochę uwagi narzędziu `ar`, które jest wykorzystywane przede wszystkim do tworzenia bibliotek z kodem obiektów stosowanych w procesie wytwarzania oprogramowania, ale także do tworzenia plików pakietów na potrzeby dystrybucji systemu Debian.

W internecie można bez trudu odnaleźć zarówno narzędzia przetwarzające pliki *.zip*, tworzone za pomocą aplikacji PKZip, jak i mniej znane narzędzia generujące skompresowane archiwa, np. `lha`. Warto jednak pamiętać, że podane formaty niemal nigdy nie są wykorzystywane do dystrybuowania programów open source dla systemu Linux. Oznacza to, że widząc archiwum z rozszerzeniem *.zip*, możemy być niemal pewni, że jest przeznaczone dla systemu operacyjnego firmy Microsoft.

W większości przypadków musimy dysponować dwiema informacjami na temat poszczególnych formatów: jak sprawdzić zawartość archiwum i jak wyodrębnić pliki z tego archiwum.

W przeciwieństwie do narzędzi archiwizujących, stosowanych w systemach operacyjnych Windows, które oferują rozmaite, często niebezpieczne funkcje dodatkowe, programy archiwizujące przeznaczone dla systemów Linux koncentrują się wyłącznie na podstawowych funkcjach. W tej sytuacji przeglądanie i rozpakowywanie plików archiwalnych jest dość bezpieczne (szczególnie jeśli nie dysponujemy uprawnieniami administratora). Mimo to zawsze warto przyjrzeć się zawartości archiwum przed przystąpieniem do wypakowywania plików, aby przypadkiem nie nadpisać tak samo nazwanych plików już istniejących.

1.4.1. Identyfikacja plików archiwalnych

Archiwa pobierane z internetu najczęściej są kompresowane z myślą o oszczędzaniu przepustowości łączy. Istnieje szereg konwencji nazewniczych, stosowanych dla plików skompresowanych (część spośród tych konwencji przedstawiono w tabeli 1.2).

TABELA 1.2. Konwencje nazewnictwa plików archiwalnych

Rozszerzenia	Typ
.tar	Nieskompresowany plik archiwalny narzędzia `tar`
.tar.gz, *.tgz*	Plik archiwalny narzędzia `tar`, skompresowany za pomocą narzędzia `gzip`
.tar.bz2	Plik archiwalny narzędzia `tar`, skompresowany za pomocą narzędzia `bzip2`
.tar.Z, *.taz*	Plik archiwalny narzędzia `tar`, skompresowany za pomocą polecenia `compress` systemu Unix
.ar, *.a*	Plik archiwalny narzędzia `ar`, stosowany przede wszystkim podczas tworzenia oprogramowania
.cpio	Nieskompresowany plik archiwalny narzędzia `cpio`

W razie wątpliwości warto pamiętać o przydatnym poleceniu `file`. Wspomniane narzędzie wprost doskonale nadaje się do identyfikacji plików, których nazwy niewiele nam mówią o faktycznej zawartości. Z tego rodzaju sytuacjami mamy do czynienia np. wtedy, gdy nasza przeglądarka internetowa lub inne narzędzie tak modyfikuje nazwy plików, że ich rozpoznanie staje się kłopotliwe. Przypuśćmy, że dysponujemy skompresowanym archiwum narzędzia `tar`, nazwanym *foo.x*. Nazwa ta nie informuje o zawartości tego pliku. W tej sytuacji można skorzystać z następującego polecenia:

```
$ file foo.x
foo.x: gzip compressed data, from UNIX, max compression
```

Teraz możemy być pewni, że nasz plik skompresowano za pomocą narzędzia `gzip`, nadal jednak nie wiemy, czy mamy do czynienia z plikiem wygenerowanym przez narzędzie `tar`. Możemy spróbować rozpakować ten plik za pomocą narzędzia `gzip` i ponownie użyć polecenia `file`. Lepszym rozwiązaniem będzie jednak zastosowanie opcji polecenia `-z`:

```
$ file -z foo.x
foo.x: tar archive (gzip compressed data, from UNIX, max compression)
```

Tera dokładnie wiemy, czym dysponujemy.

Z reguły użytkownicy kierują się intuicyjnymi konwencjami nazewnictwa plików, a same nazwy w zdecydowanej większości przypadków są podstawowym źródłem wiedzy o rodzajach plików archiwalnych i sposobie ich opracowania.

1.4.2. Przeglądanie zawartości plików archiwalnych

Pliki archiwalne śledzą informacje o zawieranych plikach w formie swoistego spisu treści, który można uzyskać (co jest dość wygodne), stosując flagę `-t`. Jest ona obsługiwana przez wszystkie wspomniane do tej pory narzędzia archiwizujące. Poniżej przedstawiono sposób użycia tej flagi na przykładzie instalacji `cron` systemu Debian:

```
$ tar -tzvf data.tar.gz
drwxr-xr-x root/root          0 2001-10-01 07:53:19 ./
drwxr-xr-x root/root          0 2001-10-01 07:53:15 ./usr/
drwxr-xr-x root/root          0 2001-10-01 07:53:18 ./usr/bin/
-rwsr-xr-x root/root      22460 2001-10-01 07:53:18 ./usr/bin/crontab
drwxr-xr-x root/root          0 2001-10-01 07:53:18 ./usr/sbin/
-rwxr-xr-x root/root      25116 2001-10-01 07:53:18 ./usr/sbin/cron
```

W powyższym przykładzie dodatkowo użyto opcji `-v`, wymuszającej dołączanie dodatkowych informacji podobnych do tych znanych z *długich* list polecenia `ls`. Dane wyjściowe obejmują uprawnienia właściwe dla poszczególnych plików (w pierwszej kolumnie) i informacje o własności (w drugiej kolumnie). Kolejna, trzecia kolumna zawiera rozmiary plików (wyrażone w bajtach); warto zwrócić uwagę na katalogi, których rozmiar wynosi `0`. Analizując zawartość pliku archiwalnego, należy zwracać szczególną uwagę na własność i uprawnienia właściwe dla poszczególnych plików.

Podstawowe polecenia umożliwiające wyświetlanie zawartości plików archiwalnych w różnych formatach przedstawiono w tabeli 1.3. Dla wszystkich trzech formatów uzyskujemy niemal identyczne dane wynikowe.

TABELA 1.3. Polecenia przeglądania plików archiwalnych

Format	Polecenie	Uwagi
Archiwum narzędzia `tar`	`tar –tvf` *`nazwa_pliku`*	
Archiwum narzędzia `tar`, skompresowane za pomocą narzędzia `gzip`	`tar -tzvf` *`nazwa_pliku`*	
Archiwum narzędzia `tar`, skompresowane za pomocą narzędzia `bzip2`	`tar -tjvf` *`nazwa_pliku`*	
Archiwum narzędzia `cpio`	`cpio –tv <` *`nazwa_pliku`*	Narzędzie `cpio` wykorzystuje `stdin` i `stdout` w roli strumieni binarnych

Czytanie symbolicznej reprezentacji uprawnień dostępu do pliku jest dość proste — wystarczy się przyzwyczaić do tego specyficznego formatu. Warto opanować popularne zabiegi mające na celu reprezentowanie dodatkowych informacji (oprócz standardowych uprawnień odczytu, zapisu i wykonywania).

W pierwszej kolejności zajmiemy się samym łańcuchem uprawnień. Składa się on z 10 znaków. Pierwszy znak określa rodzaj plików, pozostałe trzy trójznakowe grupy opisują odpowiednio: uprawnienia właściciela pliku, uprawnienia członków grupy i uprawnienia wszystkich pozostałych użytkowników.

Typ pliku jest reprezentowany przez pojedynczy znak. Dopuszczalne wartości tego znaku wraz ze znaczeniami przedstawiono w tabeli 1.4.

Kolejne dziewięć znaków można podzielić na trzy grupy po trzy bity. Poszczególne bity reprezentują odpowiednio: uprawnienia odczytu (`r`), zapisu (`w`) i wykonywania (`x`) pliku. Znak `-` na którejś z tych pozycji oznacza, że dane uprawnienia nie zostały ustawione. Przykładowo znak `-` na pozycji `w` oznacza, że dany plik nie jest dostępny do zapisu. Kilka prostych przykładów przedstawiono w tabeli 1.5.

Przy okazji analizy uprawnień warto jeszcze wspomnieć o bitach `setuid`, `setgid` i `sticky`. Wspomnianych bitów do tej pory nie uwzględnialiśmy, ponieważ ich wpływ ogranicza się do zachowania plików wykonywalnych.

Jeśli bit `setuid` jest ustawiony, kod danego pliku będzie wykonywany z wykorzystaniem identyfikatora właściciela tego pliku w roli faktycznego identyfikatora użytkownika. Oznacza to, że nasz program może realizować wszelkie zadania, do których jest uprawniony właściciel odpowiedniego pliku. Jeśli dany plik należy do administratora i jeśli jest ustawiony bit `setuid` tego pliku,

TABELA 1.4. Typy plików na listach zawartości archiwów

Kod	Znaczenie	Uwagi
`-`	Standardowy plik	Do tej kategorii zaliczamy pliki tekstowe, pliki z danymi, pliki wykonywalne itp.
`d`	Katalog	
`c`	Urządzenie znakowe	Plik specjalny, wykorzystywany w procesie komunikacji ze sterownikiem **urządzenia znakowego** (ang. *character device*). Tego rodzaju pliki tradycyjnie mogą być składowane tylko w katalogu */dev* i zwykle nie są umieszczane w plikach archiwalnych
`b`	Urządzenie blokowe	Plik specjalny, wykorzystywany w procesie komunikacji ze sterownikiem **urządzenia blokowego** (ang. *block device*). Tego rodzaju pliki tradycyjnie mogą być składowane tylko w katalogu */dev* i zwykle nie są umieszczane w plikach archiwalnych
`l`	Dowiązanie symboliczne	Nazwa pliku wskazująca na inny plik. Wskazywany plik może być składowany w innym systemie plików lub nawet może w ogóle nie istnieć

TABELA 1.5. Przykłady bitów uprawnień do pliku

Uprawnienia	
`rwx`	Dany plik może być odczytywany, zapisywany i wykonywany
`rw-`	Dany plik może być odczytywany i zapisywany, ale nie może być wykonywany
`r-x`	Dany plik może być odczytywany i wykonywany, ale nie może być zapisywany
`--x`	Dany plik może być wykonywany, ale nie może być odczytywany ani zapisywany

zawarty w nim kod może modyfikować i usuwać dowolne pliki w ramach systemu (niezależnie od tego, który użytkownik uruchomił ten program). Brzmi dziwnie, prawda? W przeszłości programy z ustawionym bitem `setuid` były przedmiotem ataków.

Za to samo zadanie odpowiada bit `setgid`, z tą różnicą, że kod zawarty w tak oznaczonym pliku jest wykonywany z przywilejami grupy, do której ten plik należy. Standardowo pliki wykonywalne są wykonywane z przywilejami właściwymi dla grupy, do której należy użytkownik uruchamiający odpowiednie programy. Jeśli bit `setgid` jest ustawiony, program działa tak, jakby został uruchomiony przez użytkownika należącego do tej samej grupy co jego właściciel.

Plik z ustawionym bitem `setuid` lub `setgid` można rozpoznać, zerkając na bit `x` łańcucha uprawnień. Litera `x` na tej pozycji standardowo oznacza, że mamy do czynienia z plikiem wykonywalnym, a znak `-` oznacza, że nie jest to plik wykonywalny.

Bity `setuid` i `setgid` dodają jeszcze dwie możliwe wartości tego znaku. Mała litera `s` (zamiast litery `x`) w bicie uprawnień właściciela oznacza, że dany plik może być wykonywany przez swojego właściciela i że ustawiono jego bit `setuid`. Także wielka litera `S` oznacza, że ustawiono bit `setuid`, ale właściciel danego pliku nie dysponuje prawem jego wykonywania. Wygląda to dość dziwacznie, ale taka konstrukcja jest dopuszczalna i bywa niebezpieczna. Plik może należeć np. do administratora, który nie może tego pliku wykonać. Linux daje administratorowi prawo uruchamiania każdego pliku, pod warunkiem że **ktokolwiek** dysponuje takimi uprawnieniami. Oznacza to, że nawet jeśli bit wykonywalności dla administratora nie zostanie ustawiony, dopóki bieżący użytkownik ma takie prawo, kod zawarty w danym pliku zostanie wykonany z przywilejami administratora.

Bit `setgid` , podobnie jak bit `setuid`, można zmienić, modyfikując pozycję `x` w ramach części łańcucha uprawnień, reprezentującej prawa grupy. Mała litera `s` oznacza, że dla danego pliku ustawiono bit `setgid` i że członkowie danej grupy dysponują prawem jego wykonywania. Wielka litera `S` oznacza, że dla danego pliku ustawiono bit `setgid`, ale członkowie danej grupy nie mogą tego pliku wykonywać.

W danych wynikowych, wygenerowanych dla pakietu `cron` (przedstawionych we wcześniejszej części tego rozdziału), widać, ze program `crontab` ma ustawiony bit `setuid` i że należy do administratora. Dodatkowe przykłady przedstawiono w tabeli 1.6.

Bit `sticky` należy traktować jak relikt przeszłości. W założeniu miał gwarantować szybkie ładowanie wybranych programów wykonywalnych przez utrzymywanie ich w stronach kodowych na dysku wymiany. W systemie Linux bit `sticky` jest stosowany wyłącznie dla katalogów i ma zupełnie inne znaczenie. Kiedy nadajemy innym użytkownikom uprawnienia zapisu i wykonywania plików składowanych w naszym katalogu, mogą oni swobodnie tworzyć pliki w tym katalogu i usuwać je. Jednym z przywilejów, których z różnych względów możemy nie chcieć nadawać, jest możliwość usuwania plików należących do innych użytkowników. Jeśli jakiś użytkownik dysponuje prawem zapisu w ramach jakiegoś katalogu, zwykle może usuwać z tego katalogu dowolne pliki (nie tylko te, które do niego należą). Możemy ten przywilej odebrać, ustawiając bit `sticky` dla interesującego nas katalogu. Z tak oznaczonego katalogu użytkownicy mogą usuwać tylko swoje pliki. Jak zwykle w tego rodzaju sytuacjach właściciel katalogu i administrator zachowuje prawo usuwania wszystkich plików. W większości systemów bit `sticky` jest ustawiany dla katalogu */tmp*.

Katalog z ustawionym bitem `sticky` jest oznaczany literą `t` lub `T` w miejscu reprezentującym uprawnienia wykonywania, nadane pozostałym użytkownikom. Przykłady takich rozwiązań przedstawiono poniżej:

`-rwxrwxrwt` Dany katalog może być odczytywany i zapisywany przez wszystkich użytkowników; dla tego katalogu ustawiono bit `sticky`

TABELA 1.6. Wybrane przykłady uprawnień wraz ze znaczeniami

Łańcuch uprawnień	Uprawnienia wykonywania	Efektywny identyfikator użytkownika	Efektywny identyfikator grupy
`-rwxr-xr-x`	Dany plik może być wykonywany przez wszystkich użytkowników	Użytkownik bieżący	Użytkownik bieżący
`-rw-r-xr-x`	Dany plik może być wykonywany przez wszystkich członków odpowiedniej grupy z wyjątkiem jego właściciela	Użytkownik bieżący	Użytkownik bieżący
`-rwsr-xr-x`	Dany plik może być wykonywany przez wszystkich użytkowników	Właściciel pliku	Użytkownik bieżący
`-rwSr-xr-x`	Dany plik może być wykonywany przez wszystkich z wyjątkiem jego właściciela	Właściciel pliku	Użytkownik bieżący
`-rwxr-sr-x`	Dany plik może być wykonywany przez wszystkich użytkowników	Użytkownik bieżący	Właściciel grupy
`-rwsr-sr-x`	Dany plik może być wykonywany przez wszystkich użytkowników	Właściciel pliku	Właściciel grupy
`-rwsr-Sr-x`	Dany plik może być wykonywany przez wszystkich użytkowników włącznie z jego właścicielem, ale z wyjątkiem członków danej grupy	Właściciel pliku	Właściciel grupy

`-rwxrwx–T` Zawartość danego katalogu może być odczytywana i zapisywana tylko przez jego właściciela i członków grupy; dla tego katalogu ustawiono bit `sticky`

1.4.3. Rozpakowywanie plików z pliku archiwalnego

Skoro wiemy już, jak przeglądać zawartość pliku archiwalnego, warto się bliżej przyjrzeć problemowi wyodrębniania plików zawartych w archiwach. Podstawowe polecenia przedstawiono w tabeli 1.7.

Chociaż rozpakowywanie plików z archiwów jest operacją dość bezpieczną, warto zwracać uwagę na ścieżki do katalogów, aby wyeliminować ryzyko nadpisania danych w naszym systemie. Szczególną ostrożność należy zachowywać, korzystając z narzędzia `cpio`, które może składować w plikach archiwalnych ścieżki bezwzględne (począwszy od katalogu głównego). Oznacza to, że jeśli plik archiwalny narzędzia `cpio` zawiera pliki w katalogu */etc*, jego rozpakowanie może dopro-

TABELA 1.7. Polecenia rozpakowywania plików z pliku archiwalnego

Format	Polecenie	Uwagi
Archiwum narzędzia `tar`	`tar -xf nazwa_pliku`	To polecenie domyślnie wypakowuje pliki do bieżącego katalogu
Archiwum narzędzia `tar`, skompresowane za pomocą narzędzia `gzip`	`tar -xzf nazwa_pliku`	
Archiwum narzędzia `tar`, skompresowane za pomocą narzędzia `bzip2`	`tar -xjf nazwa_pliku`	
Archiwum narzędzia `cpio`	`cpio -i -d < nazwa_pliku`	Musimy pamiętać o ścieżkach bezwzględnych
Archiwum narzędzia `ar`	`ar x nazwa_pliku`	Pliki nie obejmują informacji o ścieżkach

wadzić do niezamierzonego nadpisania plików już składowanych w tym katalogu. Wyobraźmy sobie archiwum narzędzia `cpio`, zawierające m.in. kopię pliku */etc/hosts*. Jeśli spróbujemy rozpakować pliki z tego archiwum, narzędzie `cpio` podejmie próbę nadpisania naszej kopii pliku */etc/hosts*. Łatwo to sprawdzić, wykonując następujące polecenie:

```
cpio -t < foo.cpio
/etc/hosts
```

Początkowy znak ukośnika (/) jest wskazówką sugerującą, że nasze archiwum przywróci **tę konkretną** kopię pliku */etc/hosts*. Oznacza to, że jeśli spróbujemy rozpakować pliki wyłącznie celem ich sprawdzenia, najprawdopodobniej będziemy chcieli uniknąć nadpisania naszych kopii tego samego pliku. W tej sytuacji należałoby użyć opcji GNU `--no-absolute-filenames`, aby plik *hosts* został wypakowany do ścieżki:

./etc/hosts

Na szczęście z archiwami narzędzia `cpio` będziemy mieli do czynienia wyłącznie w ramach plików pakietów RPM, a menedżer pakietów RPM zawsze stosuje ścieżki względne wobec bieżącego katalogu, aby wbrew naszej woli nie nadpisać plików systemowych.

Warto pamiętać, że także wersja narzędzia `tar`, oferowana w ramach niektórych wersji systemu operacyjnego Unix, umożliwia stosowanie ścieżek bezwzględnych. Wersja GNU narzędzia `tar`, stosowana w systemach Linux, automatycznie usuwa początkowy znak / ze ścieżek do rozpakowywanych plików. Oznacza to, że nawet jeśli będziemy dysponowali archiwum narzędzia

`tar`, wygenerowanym przez jedną ze wspomnianych wersji systemu Unix, narzędzie `tar` w wersji GNU wyeliminuje ryzyko nadpisania plików przez usunięcie ewentualnych znaków / z początku ścieżek do spakowanych plików.

1.5. Poznajmy wykorzystywany menedżer pakietów

Menedżery pakietów to skomplikowane narzędzia odpowiedzialne za instalowanie i konserwację oprogramowania w naszym systemie. Menedżery pakietów ułatwiają nam zarządzanie instalowanym oprogramowaniem i rozmieszczeniem plików. Za ich pomocą możemy też śledzić zależności pomiędzy pakietami, aby mieć pewność, że instalowane oprogramowanie jest zgodne z oprogramowaniem już zainstalowanym. Gdybyśmy np. chcieli zainstalować pakiet KDE w komputerze z zainstalowanym pakietem GNOME, menedżer pakietów powinien zasygnalizować brak wymaganych bibliotek czasu wykonywania. Taki mechanizm jest oczywiście nieporównanie bardziej wygodny od samodzielnego instalowania pakietów tylko po to, by stwierdzić, że ich uruchomienie z jakiegoś powodu jest niemożliwe.

Jednym z najcenniejszych elementów funkcjonalności, oferowanych przez menedżer pakietów, jest możliwość odinstalowywania oprogramowania. Za pomocą odpowiednich mechanizmów możemy instalować wybrane fragmenty oprogramowania, sprawdzać ich działanie i — jeśli nie przypadną nam do gustu — odinstalowywać niepotrzebne rozwiązania. Po odinstalowaniu pakietu system wraca do konfiguracji sprzed jego instalacji. Właśnie odinstalowywanie oprogramowania jest jednym ze sposobów jego aktualizacji. Wystarczy usunąć starą wersję i zainstalować nową. Większość menedżerów pakietów oferuje specjalne polecenie **aktualizacji**, które umożliwia nam realizację tego procesu w jednym kroku.

Menedżer pakietów tworzy scentralizowaną bazę danych, odpowiedzialną za śledzenie zainstalowanych aplikacji. Jest ona także cennym źródłem informacji o stanie naszego systemu. Możemy np. wyświetlić listę wszystkich aktualnie zainstalowanych aplikacji lub sprawdzić, czy interesujący nas program był stosowany od czasu instalacji. W niektórych sytuacjach samo przeglądanie tej bazy danych bywa ciekawym doświadczeniem — użytkownik odkrywa oprogramowanie, o którego istnieniu w ogóle nie wiedział.

Do najczęściej stosowanych formatów pakietów należy RPM (RPM Package Manager[2]) oraz Debian Package. Wybrane przykłady tych i kilku innych formatów przedstawiono w tabeli 1.8. Jak łatwo się domyślić, format RPM jest wykorzystywany przez dystrybucje Red Hat i Fedora, okazuje się jednak, że jest wykorzystywany także przez dystrybucję Suse i wiele innych. Format

[2] Nazywany wcześniej Red Hat Package Manager.

TABELA 1.8. Kilka popularnych dystrybucji systemu Linux wraz z wykorzystywanymi w nich formatami pakietów

Dystrybucja	Format pakietów
Red Hat	RPM
Fedora	RPM
Debian	Deb
Knoppix	Deb
Ubuntu	Deb
Gentoo	portage
Xandros	Deb
Mandriva (dawniej Mandrake)	RPM
MEPIS	Deb
Slackware	pkgtool

Debiana jest wykorzystywany nie tylko w ramach swojej macierzystej dystrybucji, ale też przez wiele innych (m.in. Knoppix czy Ubuntu). Warto też zwrócić uwagę na pozostałe menedżery pakietów: wykorzystywanego w dystrybucji Slackware menedżera `pkgtool` oraz stosowanego w dystrybucji Gentoo menedżera `portage`.

Decyzja o wyborze właściwego menedżera pakietu nie należy do nas (chyba że chcemy opracować własną dystrybucję systemu operacyjnego). Każda dystrybucja systemu Linux oferuje pojedyncze narzędzie odpowiedzialne za zarządzanie zainstalowanym oprogramowaniem. Stosowanie dwóch menedżerów pakietów w pojedynczym systemie nie miałoby najmniejszego sensu. Jeśli menedżer pakietów, oferowany w ramach dystrybucji, w której pracujemy, nie przypadł nam do gustu, wybór innej dystrybucji będzie lepszym rozwiązaniem niż próba wdrożenia innego menedżera.

Kiedy już zidentyfikujemy format, który powinniśmy pobrać, najprawdopodobniej staniemy przed jeszcze jednym istotnym wyborem. Ponieważ koncentrujemy się na oprogramowaniu typu open source, musimy zakładać możliwość pobrania z internetu kodu źródłowego.

1.5.1. Wybór pomiędzy kodem źródłowym a wersją binarną

Jeśli korzystamy z systemu operacyjnego Linux na platformie z procesorem 32-bitowym, zgodnym ze standardem Intela, najprawdopodobniej będziemy mieli możliwość pobierania oprogramowania w formie gotowych, skompilowanych plików binarnych (tzw. binariów). Binaria najczęściej są

dostępne w formie pakietów, rzadziej w formie archiwów narzędzia `tar`. Jeśli zdecydujemy się pobrać i zainstalować oprogramowanie w formie skompilowanych wcześniej binariów, w ogóle nie będziemy musieli się zajmować kodem źródłowym (chyba że będziemy tego chcieli).

Jeśli korzystamy z systemu Linux na innej platformie niż procesor zgodny ze standardem Intela, jedynym rozwiązaniem jest pobranie kodu źródłowego i jego samodzielna kompilacja. W niektórych przypadkach możemy sami podjąć decyzję o takiej kompilacji mimo dostępności odpowiednich plików binarnych. Twórcy oprogramowania celowo generują binaria tylko dla architektur zapewniających jak największą zgodność, aby trafić do możliwie szerokiej grupy odbiorców. Jeśli dysponujemy najnowszym, najszybszym procesorem CPU, możemy uznać za stosowne ponowne skompilowanie pobranego pakietu z myślą o optymalizacji pod kątem tej konkretnej architektury (zamiast korzystać z wersji zoptymalizowanej dla starszych, wolniejszych architektur).

Użytkownicy popularnej architektury Intela mogą znaleźć w internecie niezliczone, binarne pliki wykonywalne, skompilowane dla architektury i386. Oznaczenie i386 odwołuje się do procesora 80386, czyli swoistego najmniejszego wspólnego mianownika 32-bitowych architektur Intela. Obecnie oznaczenie pakietu etykietą i386 najczęściej odwołuje się do architektury Pentium lub nowszej. Wiele pakietów oznacza się bardziej precyzyjną etykietą i586, która wprost wskazuje na procesor Pentium. Tak czy inaczej, kod skompilowany i zoptymalizowany z myślą o procesorze Pentium wcale nie musi działać najefektywniej na procesorach Pentium 4 czy Xeon.

To, czy wydajność oprogramowania rzeczywiście wzrośnie po jego skompilowaniu dla nowszych procesorów, zależy od poszczególnych aplikacji. Nigdy nie mamy gwarancji, że każda aplikacja w widoczny sposób przyspieszy działanie wskutek kompilacji z myślą o docelowej platformie.

W tabeli 1.9 wymieniono i krótko opisano najczęściej spotykane etykiety architektur, stosowane w nazwach pakietów RPM. Mimo że przedstawione oznaczenia w wielu przypadkach są identyczne jak etykiety stosowane przez kompilator GNU, nie należy ich z sobą utożsamiać. Etykiety pakietów często są swobodnie wybierane przez twórców oprogramowania i jako takie nie muszą odpowiadać faktycznemu sposobowi kompilacji. Przykładowo: zdecydowana większość współczesnych pakietów oznaczanych etykietą i386 w praktyce jest kompilowana dla platformy Pentium lub Pentium II. Ponieważ mało kto uruchamia dziś system Linux na procesorach 80386, nikt nie protestuje przeciwko takiemu stanowi rzeczy.

Kompilowanie kodu źródłowego nie zawsze jest trudne. W przypadku stosunkowo prostych projektów, np. narzędzi tekstowych, taka kompilacja może być dość prosta. W przypadku projektów bardziej złożonych, np. przeglądarek internetowych lub edytorów tekstu, kompilacja może się okazać wyjątkowo kłopotliwa. Ogólnie rzecz biorąc, im większy projekt, tym więcej

TABELA 1.9. Przegląd architektur

Oznaczenie	Opis
`i386`	Najbardziej popularna architektura, z jaką będziemy mieli do czynienia (w kompilatorze `gcc` etykieta i386 odnosi się wyłącznie do procesorów 80386). Widząc tak oznaczony pakiet, należy przyjmować, że zawarte w nim oprogramowanie będzie wymagało procesora nie starszego niż Pentium I
`i486`	Wyjątkowo rzadko stosowana etykieta. Najczęściej założenie o zgodności tak oznaczonego pakietu z architekturą 80486 (lub zgodną) jest w pełni bezpieczne
`i586`	Coraz bardziej popularna etykieta. Kompilator GNU stosuje etykietę i586 do opisywania procesorów z serii Pentium I. Oznacza to, że od tak oznaczonych pakietów należy oczekiwać zgodności z dowolnym procesorem klasy Pentium lub nowszym
`i686`	Kompilator GNU wykorzystuje etykietę i686 do opisywania procesorów z serii Pentium Pro, które stanowiły podstawę dla procesorów Pentium II i nowszych. Widząc tak oznaczony pakiet, należy przyjmować, że zawarte w nim oprogramowanie będzie wymagało procesora Pentium II lub nowszego
`ix86`	Etykieta ix86 nie jest stosowana zbyt często, ale założenie o bezpiecznym działaniu na komputerach Pentium lub nowszych z reguły okazuje się zasadne
`x86_64`	Reprezentuje procesory AMD Opteron oraz Intel Pentium 4 z rozszerzeniami EM64T. Te nowoczesne architektury oferują możliwość przetwarzania 32- i 64-bitowego. Tak oznaczony kod jest kompilowany z myślą o pracy w trybie 64-bitowym, co oznacza brak jego zgodności z procesorami 32-bitowymi — taki kod nie będzie działał ani na procesorach Opteron, ani na procesorach z rozszerzeniem EM64T, jeśli spróbujemy go uruchomić w systemie z 32-bitowym jądrem systemu Linux
`IA64`	Etykieta IA64 odwołuje się wprost do 64-bitowych procesorów Itanium. Itanium jest jedyną w swoim rodzaju architekturą koncernów Intel i Hewlett-Packard, stosowaną wyłącznie w bardzo drogich stacjach roboczych i superkomputerach
`ppc`	Procesory PowerPC G2, G3 i G4, instalowane w niektórych komputerach Apple Macintosh i Apple iMac
`ppc64`	Procesor PowerPC G5, stosowany w komputerach Apple iMac
`sparc`	Procesor SPARC, stosowany w stacjach roboczych firmy Sun
`sparc64`	64-bitowy procesor SPARC, stosowany w stacjach roboczych firmy Sun
`mipseb`	Procesor MIPS, stosowany przede wszystkim w stacjach roboczych firmy SGI

bibliotek pomocniczych będziemy potrzebować do jego kompilacji. Na przykład wielkie projekty z graficznym interfejsem użytkownika (GUI) z reguły korzystają z wielu różnych bibliotek programowych, które rzadko są zainstalowane w naszym systemie. Poszukiwanie właściwych wersji wszystkich tych pakietów bywa czasochłonne, a w skrajnych przypadkach okazuje się wręcz niewykonalne. Problem kompilowania kodu źródłowego projektów omówimy bardziej szczegółowo w rozdziale 2. Poszukując oprogramowania, powinniśmy się kierować dostępnością odpowiednich plików binarnych.

1.5.2. Praca z pakietami

Twórcy wielu nowych dystrybucji systemu Linux starają się w możliwie dużym stopniu ułatwiać pracę użytkownikowi, aby mógł korzystać z systemu Linux nawet bez świadomości funkcjonowania w tle menedżera pakietów. Mimo to warto opanować działanie tego rodzaju narzędzi, jeśli tylko planujemy wychylić głowę poza miejsca przygotowane dla nas przez autorów dystrybucji. Znajomość otoczenia odpowiedzialnego za zarządzanie pakietami bywa bardzo przydatne, kiedy coś idzie nie po naszej myśli. Podstawowe funkcje, których możemy oczekiwać od menedżerów pakietów, obejmują:

- instalację nowego oprogramowania w systemie,
- usuwanie (odinstalowywanie) oprogramowania z systemu,
- weryfikację zainstalowanych plików pod kątem prawidłowej instalacji lub ewentualnych uszkodzeń,
- aktualizację zainstalowanych wersji oprogramowania,
- analizę zainstalowanego oprogramowania (np. identyfikację pakietów, w ramach których zainstalowano poszczególne pliki),
- analizę zawartości pakietów przed instalacją.

1.6. Kilka słów o bezpieczeństwie w kontekście pakietów

Niemal każdy użytkownik komputera ma jakieś doświadczenia z tzw. **złośliwym oprogramowaniem** (ang. *malware*, *malicious software*). Użytkownicy systemu Linux często otrzymują dziwaczne wiadomości poczty elektronicznej od swoich przyjaciół pracujących w systemie Windows — wiele takich wiadomości jest wynikiem najnowszych wirusów rozprzestrzeniających się pomiędzy komputerami z systemem Windows za pośrednictwem internetu.

Złośliwe oprogramowanie to coś więcej niż tylko wirusy rozsyłane za pośrednictwem poczty elektronicznej — zaliczamy do tej kategorii wszystkie programy, które są instalowane i uruchamiane w systemie bez naszej zgody i wiedzy. Są to wirusy, programy szpiegujące (ang. *spyware*) i każde destrukcyjne oprogramowanie, które w ten czy inny sposób uzyskuje dostęp do naszego systemu. Obrońcy systemu Windows argumentują, że platforma firmy Microsoft tak często staje się celem ataków twórców złośliwego oprogramowania tylko dlatego, że jest stosowana na dużo większej liczbie komputerów niż system Linux. Chociaż z samym uzasadnieniem trudno się spierać, należy pamiętać, że system Windows jest też dużo wdzięczniejszym obiektem ataków. Piętą achillesową systemów Windows 98, Windows ME i wersji domowych systemu Windows XP jest możliwość uzyskiwania dostępu do wszystkich plików przez wszystkich użytkowników i wprowadzania zmian o zasięgu systemowym. Oznacza to, że niedoświadczony użytkownik systemu Windows, klikając załącznik do wiadomości, może przekształcić swój komputer w potwora przeprowadzającego ataki blokowania usług (ang. *Denial of Service — DoS*) lub usuwającego zawartość dysku C.

Motywy kierujące autorami złośliwego oprogramowania bywają bardzo różne — zdarza się, że takie oprogramowanie jest narzędziem w działaniach przestępczości zorganizowanej, osób żądnych zemsty lub zwykłych wandali. Nigdy nie możemy zakładać, że akurat nasz komputer nie stanie się celem ataku.

Wielu użytkowników systemu Linux sądzi, że ich komputery są odporne na działanie złośliwego oprogramowania, co nie jest prawdą. Plik JBellz w formacie MP3 był np. koniem trojańskim, wykorzystującym lukę w zabezpieczeniach programu `mpg123` (odtwarzacza plików MP3, oferowanego w trybie open source dla systemów Linux). Kiedy użytkownik próbował odtworzyć ten plik za pomocą innego programu, wszystko wskazywało na to, że ma do czynienia z plikiem uszkodzonym bez możliwości odtworzenia. Plik JBellz był więc dobrze zaprojektowanym programem, który opracowano z myślą o konkretnej luce w programie `mpg123`. Program `mpg123` zawierał błąd przepełnienia bufora, umożliwiający odpowiednio spreparowanemu plikowi MP3 wykonywanie dowolnego skryptu. W przypadku pliku JBellz wspomniany skrypt usuwał zawartość katalogu domowego użytkownika.

Chociaż jest mało prawdopodobne, by nasze komputery pracujące pod kontrolą systemu Linux rozpowszechniały wirusy na taką skalę jak komputery z systemem Microsoft Windows, musimy pamiętać o potencjalnych lukach w zabezpieczeniach. Przewidywanie zdarzeń podobnych do pojawienia się konia trojańskiego JBellz jest niemożliwe, jedyne zatem, co nam pozostaje, to uwzględnianie z należytą uwagą wszelkich ostrzeżeń. W przypadku wspomnianego pliku uszkodzenia obejmowały swoim zasięgiem dane pojedynczego użytkownika — system nie był więc zagrożony przynajmniej do czasu odtworzenia tego pliku przez administratora.

Można by wskazać jeszcze wiele przykładów luk dla złośliwego oprogramowania, pozostawionych w kodzie źródłowym popularnych pakietów. Jednym z nich był kod źródłowy pakietu OpenSSH[3]. Luka w kodzie OpenSSH polegała na pozostawieniu furtki nadającej intruzowi takie same uprawnienia, jakimi dysponowała osoba, która skompilowała ten kod. Oznacza to, że jeśli pobraliśmy tak zmodyfikowany kod pakietu OpenSSH, skompilowaliśmy go i zainstalowaliśmy, musimy się liczyć z ryzykiem wykorzystania tej luki przez osobę niepowołaną.

Linux nie oferuje żadnego odpowiednika programu antywirusowego, który można by wykorzystać do skanowania programów pod kątem zainfekowania — w środowisku Linux musimy bazować na zaufaniu i uwierzytelnianiu. Istnieją dwa bardzo różne podejścia do problemu wirusów: proaktywne i reaktywne. W systemach Windows bez wątpienia obowiązuje zasada reaktywności, co wcale nie oznacza, że czynnik zaufania traci na znaczeniu. Kiedy pobieramy z internetu program systemu Windows, musimy przyjmować, że nasze definicje wirusów są aktualne i że nie jesteśmy pierwszymi użytkownikami, którzy ucierpią z powodu pojawienia się zupełnie nowego wirusa. Zupełnie inna zasada obowiązuje w systemach Linux, których użytkownicy powinni przestrzegać reguły „ufaj, ale sprawdzaj" — jej stosowanie w największym skrócie polega na uwierzytelnianiu oprogramowania i korzystaniu z aplikacji zaufanych producentów. (Problemem uwierzytelniania zajmiemy się w kolejnym podrozdziale).

Jednym z elementów decydujących o większym bezpieczeństwie systemów Linux jest brak możliwości instalowania oprogramowania przez użytkowników pozbawionych odpowiednich uprawnień. Użytkownik instalujący oprogramowanie musi dysponować przywilejami administratora, musi się zatem albo zalogować jako *root*, albo użyć programu `sudo` (lub podobnego). Okazuje się, że wspomniane wymaganie jest też źródłem słabości systemu, ponieważ większość programów wymaga wykonywania skryptów zarówno w czasie ich instalacji, jak i w czasie ich usuwania. Niezależnie od poziomu naszej świadomości w tej kwestii, uruchamiając skrypty pobrane w ramach pakietu, obdarzamy jego twórcę sporym zaufaniem (w nadziei, że uruchamiany skrypt nie naruszy zasad bezpieczeństwa). W tej sytuacji uwierzytelnianie autora pakietu jest kluczowym krokiem w kierunku bezpieczeństwa systemu, w którym pracujemy.

1.6.1. Potrzeba uwierzytelniania

O bezpieczeństwie systemu nie decyduje tylko to, co uruchamiamy, ale też to, kiedy to robimy. Użytkownik systemu Linux może tworzyć i próbować uruchamiać dowolne rodzaje złośliwego oprogramowania, bez uprawnień administratora nie będzie mógł jednak przejąć kontroli nad całym komputerem. Warto więc pamiętać, że wszystkie formaty pakietów mogą obejmować skrypty

[3] Patrz strona internetowa *http://www.cert.org/advisories/CA-2002-24.html*.

wykonywane w czasie ich instalacji i (lub) usuwania. Zwykle są to skrypty powłoki Bourne'a, wykonywane z uprawnieniami administratora i — tym samym — mogące wykonywać dosłownie wszystkie operacje. Właśnie te skrypty stanowią potencjalną kryjówkę dla złośliwego oprogramowania koni trojańskich, stąd konieczność każdorazowego uwierzytelniania oprogramowania jeszcze przed instalacją (nie przed uruchomieniem).

W tej sytuacji powinniśmy podchodzić niechętnie do narzędzi wymagających uprawnień administratora (jedynym wyjątkiem od tej reguły jest sam menedżer pakietów). Baza danych o pakietach jest sercem typowej dystrybucji i jako taka może być dostępna wyłącznie dla administratora. Pakiety można uwierzytelniać na wiele różnych sposobów. Narzędzie `rpm` oferuje np. wbudowane funkcje uwierzytelniające. Użytkownicy innych dystrybucji, np. Debiana, muszą uwierzytelniać oprogramowanie w ramach odrębnego kroku.

1.6.2. Podstawowe uwierzytelnianie pakietów

Uwierzytelnianie pakietów w najprostszej formie wymaga użycia tzw. funkcji mieszającej (ang. *hashing function*). Idea stosowania tej funkcji przypomina trochę mechanizm sum kontrolnych, które mają na celu unikatową identyfikację danych w formie sumy wszystkich składających się na nie bajtów. Suma kontrolna wszystkich bajtów danego pliku nie wystarczy jednak do zagwarantowania bezpieczeństwa. Nie dość, że identyczna suma kontrolna może cechować wiele różnych zbiorów danych, to jeszcze manipulowanie danymi celem osiągnięcia właściwej sumy nie stanowi żadnego problemu. W tej sytuacji proste sumy kontrolne nigdy nie są stosowane do uwierzytelniania pakietów z uwagi na łatwość podrabiania podpisów w tej formie.

Wartość generowaną przez funkcję mieszającą określa się mianem **skrótu** (ang. *hash*). Może on, podobnie jak suma kontrolna, reprezentować zbiór danych dowolnej wielkości w formie pojedynczej wartości stałej długości. Inaczej niż w przypadku sumy kontrolnej, wynik działania funkcji mieszającej jest nieprzewidywalny, co znacznie utrudnia modyfikowanie danych z myślą o osiągnięciu określonego kodu. Większość algorytmów mieszających wykorzystuje wielkie klucze (np. 128-bitowe), co sprawia, że prawdopodobieństwo wygenerowania identycznych kluczy dla dwóch różnych zbiorów danych jest bliskie zeru. Jeśli pobierzemy plik z nieznanego źródła i jeśli dysponujemy kodem ze źródła zaufanego, możemy być pewni dwóch rzeczy:

- prawdopodobieństwo uzyskania zmodyfikowanego pliku z identycznym kodem funkcji mieszającej jest wyjątkowo niskie;
- szanse na zmodyfikowanie oprogramowania przez wprowadzenie do niego złośliwego kodu z zachowaniem tego samego kodu są znikome.

Jednym z najbardziej popularnych narzędzi do tworzenia tego rodzaju kodów jest program `md5sum`, zbudowany z wykorzystaniem algorytmu MD5[4] i generujący kody 128-bitowe. Program `md5sum` nie tylko generuje kody funkcji mieszającej, ale też może je weryfikować. Możemy generować kody, wskazując nazwy interesujących nas plików w wierszu poleceń. W odpowiedzi program `md5sum` wyświetli w kolejnych wierszach kody wynikowe dla poszczególnych plików:

```
$ md5sum foo.tar bar.tar
af8e7b3117b93df1ef2ad8336976574f *foo.tar
2b1999f965e4abba2811d4e99e879f04 *bar.tar
```

Gotowe kody możemy wykorzystać w roli danych wejściowych programu `md5sum` celem weryfikacji pobranych plików:

```
$ md5sum foo.tar bar.tar > md5.sums
$ md5sum --check md5.sums
foo.tar: OK
bar.tar: OK
```

Każdy kod jest reprezentowany przez 32-cyfrową liczbę szesnastkową. (Informacja dla mniej doświadczonych Czytelników: każda cyfra szesnastkowa zajmuje 4 bity). Można ten kod porównać z wartością otrzymaną z zaufanego źródła, aby sprawdzić, czy pobrane dane są prawidłowe. Jeśli kod MD5 pasuje do naszego wzorca, możemy być niemal pewni, że dany plik nie został zmieniony od chwili utworzenia wspomnianego kodu. Jedynym problemem jest to, czy źródło sumy MD5 rzeczywiście jest godne zaufania — z pewnością nie powinno to być to samo miejsce, z którego pobrano właściwy pakiet. Jeśli pobrany plik jest koniem trojańskim, a witryna źródłowa była celem skutecznego ataku, należy zakładać, że także kody MD5 oferowane przez tę witrynę zostały podmienione.

Przypuśćmy, że chcemy pobrać najnowszą i najdoskonalszą wersję pakietu biurowego OpenOffice z witryny internetowej *OpenOffice.org*. Skoro ta oficjalna witryna kieruje nas na jedną z wielu witryn lustrzanych, skąd pewność, że któryś z tych serwerów nie był celem ataku i że interesującego nas pakietu nie zastąpiono koniem trojańskim? W tym i podobnych przypadkach dobrym rozwiązaniem jest powrót na witrynę oficjalną i pobranie pliku z sumą MD5 właśnie za jej pośrednictwem. Witryna *OpenOffice.org* oferuje sumy MD5 dla wszystkich dostępnych plików, po pobraniu któregoś z tych plików możemy zatem bez trudu porównać jego kod z kodem dostępnym w witrynie oficjalnej. Zdarza się, że możemy pobrać odpowiedni plik w formacie właściwym dla danych wejściowych programu `md5sum`; jeśli nie, będziemy musieli przygotować odpo-

[4] **MD5** jest skrótem od **Message Digest algorithm number 5**.

wiedni plik samodzielnie przez skopiowanie i wklejenie kodu wyświetlonego przez przeglądarkę internetową. Poniżej przedstawiono sumę dla pobranego pliku *Ooo_1.1.4_LinuxIntel_install.tar.gz*, wklejoną do pliku *md5.sum*:

```
cf2d0beb6cae98acae81e4d690d63094 Ooo_1.1.4_LinuxIntel_install.tar.gz
```

Warto pamiętać, że program `md5sum` jest mało elastyczny w kwestii obsługi znaków białych. `md5sum` oczekuje, że kod wejściowy nie jest poprzedzony żadną spacją i jest oddzielony od nazwy pliku dokładnie dwiema spacjami. Każda inna składnia spowoduje zasygnalizowanie błędu.

Kiedy już będziemy dysponowali plikiem *md5.sum*, będziemy mogli sprawdzić pobrany plik pakietu za pomocą następującego polecenia:

```
$ md5sum --check md5.sum
OOo_1.1.4_LinuxIntel_install.tar.gz: OK
```

Przedstawiony przykład pokazuje, że w tym konkretnym przypadku program `md5sum` jednoznacznie potwierdza poprawność sumy MD5. Oznacza to, że plik pobrany z niesprawdzonej witryny pasuje do sumy MD5 dostępnej w oficjalnej witrynie internetowej *OpenOffice.org*. Teraz możemy być pewni, że dysponujemy uwierzytelnioną kopią danego pliku (zakładając, że sama witryna *OpenOffice.org* nie była przedmiotem skutecznego ataku).

1.6.3. Uwierzytelnianie pakietów z podpisami cyfrowymi

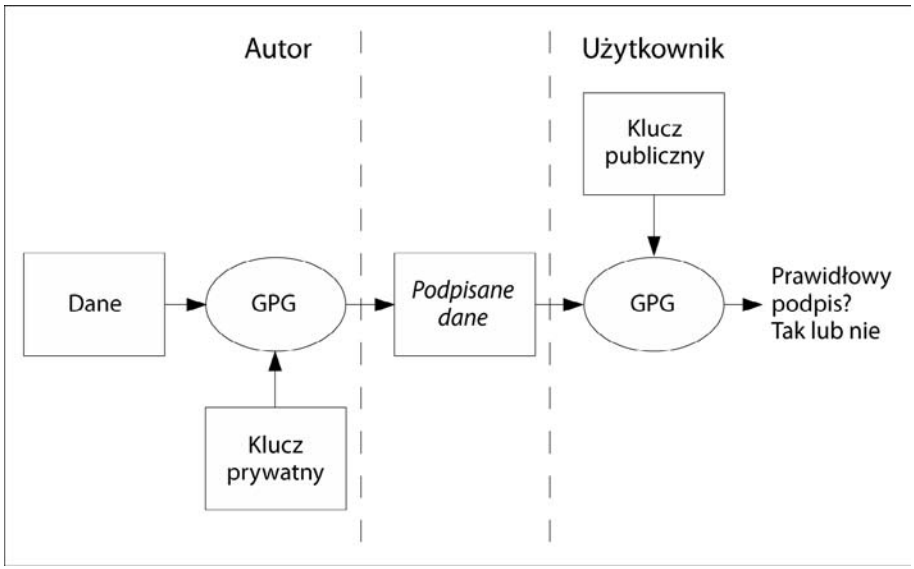
Innym rodzajem kodu (alternatywnym względem kodu MD5) jest podpis cyfrowy, który sprawdza się, gdy unikatowość uwierzytelnianych danych nie jest niezbędna. Uwierzytelnienie podpisu cyfrowego wymaga tylko pojedynczego klucza publicznego, uzyskanego od osoby bądź organizacji będącej przedmiotem uwierzytelniania. Kiedy już będziemy dysponowali takim kluczem, będziemy mogli uwierzytelniać dowolne dane podpisane przez określonego autora. Oznacza to, że nawet jeśli każdy z podpisów generowanych przez tego samego autora ma postać unikatowego kodu, uwierzytelnianie wszystkich produktów tego autora wymaga zaledwie jednego klucza publicznego.

Producent, który chce podpisać swoje dane, musi wygenerować dla nich dwa klucze: publiczny i prywatny. Wspomniane klucze są tworzone na podstawie długiego hasła (ang. *passphrase*) znanego tylko autorom oprogramowania. Zachowują oni klucz prywatny w tajemnicy i jednocześnie udostępniają klucz publiczny wszystkim zainteresowanym. Każda zmiana klucza prywatnego lub danych uniemożliwi uwierzytelnienie oprogramowania. Warto pamiętać, że prawdopodobieństwo wygenerowania prawidłowego podpisu z takim samym kluczem publicznym, ale innym

długim hasłem i kluczem prywatnym jest wyjątkowo niskie. Oznacza to, że szanse na podrobienie podpisu pasującego do istniejącego klucza publicznego są minimalne.

Opisywana metoda bazuje na zaufaniu. Musimy bowiem uwierzyć określonym programistom i organizacjom, że nie będą podpisywały danych zainfekowanych złośliwym oprogramowaniem. Musimy wierzyć, że podejmują kroki niezbędne do zachowania w tajemnicy kluczy prywatnych i długich haseł. Musimy też mieć zaufanie do źródeł, z których pobieramy wykorzystywane klucze publiczne. Jeśli wspomniane witryny internetowe są godne zaufania, możemy być pewni, że uwierzytelniamy dane z wykorzystaniem właściwych kluczy publicznych.

Najczęściej wykorzystywanym narzędziem do obsługi podpisów cyfrowych stosowanych dla kodu open source jest GNU Privacy Guard (GPG). Proces podpisywania danych za pomocą tego narzędzia przedstawiono na rysunku 1.1.



RYSUNEK 1.1. Proces podpisywania danych za pomocą narzędzia GPG

1.6.4. Podpisy narzędzia GPG, stosowane dla pakietów RPM

Format pakietów RPM umożliwia stosowanie podpisów narzędzia GPG z myślą o autoryzacji oprogramowania. Format RPM wykorzystuje też inne formy uwierzytelniania, w tym kody (np. kody MD5), w ramach poszczególnych pakietów. Można te kody z powodzeniem wykorzystywać zarówno do wykrywania ewentualnych uszkodzeń pakietów wskutek błędów pobierania,

jak i do potwierdzania ich prawidłowego stanu (już po instalacji), ale nie stanowią pomocy w procesie uwierzytelniania. Do uwierzytelniania pakietów w formacie RPM służą wyłącznie podpisy GPG. Alternatywnym rozwiązaniem jest ręczne uwierzytelnianie pakietów z wykorzystaniem kodów MD5, pobieranych z zaufanego źródła.

Narzędzie `rpm` obsługuje flagę `--checksig`, która — co jest pewnym utrudnieniem — powoduje umieszczenie wszystkich kodów w jednym wierszu. Oznacza to, że jeśli jakiś plik nie zawiera podpisu GPG, narzędzie `rpm` nadal będzie potwierdzało poprawność sumy MD5. W razie braku podpisu GPG otrzymamy we właściwym wierszu danych wynikowych komunikat `gpg ok`. Poniżej przedstawiono przykładowe dane wyjściowe tego narzędzia:

```
$ rpm -checksig *.rpm
abiword-2.2.7-1.fc3.i386.rpm: sha1 md5 OK
abiword-plugins-impexp-2.2.7-1.fc3.i386.rpm: sha1 md5 OK
abiword-plugins-tools-2.2.7-1.fc2.i386.rpm: sha1 md5 OK
firefox-1.0-2.fc3.i386.rpm: (sha1) dsa sha1 md5 gpg OK
dpkg-1.10.21-1mdk.i586.rpm: (SHA1) DSA sha1 md5 (GPG) NOT OK (MISSING KEYS: GPG#26752624)
```

Chociaż narzędzie `rpm` wyświetliło pięć plików pakietów RPM, tylko pakiety `firefox` i `dpkg` zawierają podpisy GPG. Ponieważ dysponujemy tylko kluczem publicznym pakietu `firefox` (nie mamy takiego klucza dla pakietu `dpkg`), właśnie `firefox` jest jedynym spośród tych pięciu pakietów, który można uznać za prawidłowo uwierzytelniony. Łatwo zauważyć, że narzędzie `rpm` w żaden sposób nie wyróżniło tego pakietu. Oznacza to, że nawet jeśli sprawdzany pakiet jest podpisany, nie dysponujemy rozwiązaniem umożliwiającym weryfikację tego podpisu.

W powyższym przykładzie sygnatura pakietu `firefox` została rozpoznana tylko dlatego, że opracowano ją w ramach projektu Red Hat i że narzędzie `rpm` uruchomiliśmy w dystrybucji Fedora. Wspomniana dystrybucja obejmuje wiele kluczy publicznych, wykorzystywanych przez firmę Red Hat do podpisywania udostępnianych przez nią pakietów. Klucze publiczne są kopiowane na dysk twardy w czasie instalacji dystrybucji systemu Linux. Jeśli pakiet RPM pobierzemy z internetu i jeśli jego sygnaturę będzie można zweryfikować pod kątem zgodności z którymś z tych kluczy, będziemy mieli pewność, że dany pakiet nie różni się od tego opracowanego przez firmę Red Hat (i że nie został zainfekowany żadnym złośliwym kodem). Odnalezienie klucza publicznego pakietu `dpkg` jest niemożliwe, ponieważ wspomniany pakiet pochodzi z dystrybucji Mandrake, odpowiedni klucz nie jest zatem dostępny w instalacji systemu Fedora. W tej sytuacji musimy odnaleźć ten klucz gdzie indziej.

Jeśli pobraliśmy pakiet z niewłaściwym podpisem lub wymagający nieznanego klucza publicznego (tak było w przypadku pakietu `dpkg` w powyższym przykładzie), narzędzie `rpm` wygeneruje stosowne ostrzeżenie już na etapie instalacji tego pakietu. Okazuje się, niestety, że nawet w razie

Odnajdywanie brakującego klucza publicznego

W internecie można bez trudu znaleźć witryny oferujące klucze publiczne narzędzia GPG, w wielu przypadkach jednak najprostszym rozwiązaniem jest pobieranie tych kluczy u źródeł. W prezentowanym przykładzie wykorzystamy klucz publiczny, zaczerpnięty z dystrybucji systemu Mandrake. Można to łatwo sprawdzić, wykonując następujące polecenie:

```
$ rpm -qip dpkg-1.10.21-1mdk.i586.rpm
Name : dpkg
Version : 1.10.21                      Vendor: Mandrakesoft
Release : 1mdk                         Build Date: Thu May 20 07:03:20 2004

Host: n1.mandrakesoft.com
Packager : Michael Scherer <misc@mandrake.org>
URL      : http://packages.debian.org/unstable/base/dpkg.html
Summary  : Package maintenance system for Debian
```

Poszukując zaufanych kluczy publicznych, warto się kierować kilkoma wskazówkami. W naszym przypadku warto rozpocząć poszukiwania od witryn internetowych *mandrakesoft.com* i *mandrake.org*. Z uzyskanych przed chwilą informacji o pakiecie wynika, że opracowano go w 2004 roku, co w dobie internetu jest czasem wyjątkowo odległym. Ponieważ dystrybucje Mandrake zastąpiono dystrybucjami Mandriva, wspomniane witryny już od dłuższego czasu są niedostępne. Sytuacja się komplikuje. Potrzebujemy jeszcze jednej informacji o poszukiwanym kluczu — jego identyfikatora:

```
$ rpm --checksig dpkg-1.10.21-1mdk.i586.rpm
dpkg-1.10.21-1mdk.i586.rpm: ...
... (GPG) NOT OK (MISSING KEYS: GPG 78d019f5)
```

W ten sposób otrzymaliśmy interesujący nas identyfikator.

W kolejnym kroku powinniśmy zajrzeć na oficjalną stronę internetową dystrybucji Mandriva. Wyszukiwarka Google kieruje nas pod adres *http://mandriva.com/*. W wyniku poszukiwania wyrażenia *public keys* w ramach tej witryny otrzymaliśmy stronę z kilkoma kodami i jednym kluczem publicznym (innym niż ten, którego szukamy). Także kilka kolejnych prób zakończyło się niepowodzeniem. Wydaje się, że utknęliśmy w martwym punkcie.

W tej sytuacji warto wpisać w wyszukiwarce Google wyrażenie *public keys*, aby uzyskać długą listę witryn oferujących klucze publiczne. Metodą prób i błędów odnalazłem interesujący nas klucz w witrynie *http://keys.pgp.net/* (musiałem się posłużyć identyfikatorem *0x78d019f5*):

```
Search results for '0x78d019f5'

Type bits/keyID cr. time exp time key expir

pub 1024D/78D019F5 2003-12-10
uid MandrakeContrib <cooker@linux-mandrake.com>
```

```
sig sig3 78D019F5 2003-12-10 _________ _________ [selfsig]
sig sig3 70771FF3 2003-12-10 _________ _________ Mandrake Linux
<mandrake@mandrakesoft.com>

sig sig3 26752624 2003-12-10 _________ _________ MandrakeCooker <cooker@linuxmandrake.
com>
sig sig3 45D5857E 2004-09-22 _________ _________ Fabio Pasquarelli (Lavorro)
<fabiopasquarelli@tin.it>
sig sig3 17A0F9A0 2004-09-22 _________ _________ Fafo (Personale)
<rec.r96@tin.it>

sub 1024g/4EE127FA 2003-12-10
sig sbind 78D019F5 2003-12-10 _________ _________ []
```

Poszukiwany klucz udało mi się odnaleźć na liście wygenerowanej dla wyrażenia *Mandrake*. Po kliknięciu hiperłącza *78D019F5* w oknie przeglądarki został wyświetlony klucz PGP (w formie zwykłego tekstu). Zaimportowanie tego klucza wymagało zapisania wyświetlonego tekstu w pliku *78D019F5.txt*. Sama operacja importowania klucza sprowadza się do wykonania następującego polecenia:

```
rpm --import 78D019F5.txt
```

Jeśli skopiowany tekst zawiera prawidłowy klucz publiczny, cały proces powinien przebiegać bezbłędnie. I wreszcie możemy zweryfikować poprawność oryginalnego pakietu za pomocą polecenia w postaci:

```
rpm --checksig dpkg-1.10.21-1mdk.i586.rpm
dpkg-1.10.21-1mdk.i586.rpm: (sha1) dsa sha1 md5 gpg OK
```

Z naszego punktu widzenia najważniejsze jest wyrażenie `gpg OK`.

Zanim zakończymy nasze rozważania poświęcone tej kwestii, warto poświęcić jeszcze chwilę problemowi zaufania. Sygnaturę pobrano z domeny *pgp.net*. Wierzymy, że właściciele tej bazy danych zadbali o to, by oferowane klucze publiczne pochodziły z uprawnionego źródła. Tak czy inaczej, wszystko jest kwestią zaufania.

braku możliwości uwierzytelnienia podpisu narzędzie `rpm` (przynajmniej w wersji 4.3.2) umożliwia nam instalację pakietu. Takie rozwiązanie jest o tyle niefortunne, że naraża cały system na niebezpieczeństwo związane z działaniem potencjalnie niebezpiecznych skryptów uruchamianych z uprawnieniami administratora. Narzędzie GPG nawet nie próbuje odróżniać sytuacji, w których klucz jest podrobiony, od sytuacji, w których w ogóle nie dysponujemy kluczem publicznym. Co więcej, takie rozróżnienie jest niemożliwe. Nie można wykluczyć, podobnie jak w przypadku zwykłych podpisów, że dwie osoby o takich samych nazwiskach podpisują się w identyczny sposób — trudno wtedy mówić o podrabianiu podpisów. Z analogiczną sytuacją mamy do czynienia

w przypadku kluczy publicznych, które nie gwarantują unikatowości. Narzędzie GPG ogranicza się więc do sygnalizowania niemożności uwierzytelnienia podpisu wskutek braku odpowiedniego klucza publicznego.

Należy szczególnie bacznie przyglądać się pakietom, które są podpisane, ale których podpisy nie są rozpoznawane przez nasz system (wskutek braku odpowiedniego klucza publicznego). Jeśli będziemy zmuszeni do samodzielnego odnalezienia klucza publicznego, powinniśmy go pobrać z innej witryny niż ta, z której pobrano właściwy pakiet RPM — najlepiej ze strony, która nie jest wskazywana z poziomu witryny oferującej ten pakiet. Nie powinniśmy ufać kluczom z witryn wskazywanych przez producenta, ponieważ takie klucze mogą być podmieniane przez autorów złośliwego oprogramowania. Z podobnymi sytuacjami mamy do czynienia w rzeczywistym świecie. Zwykle nie podejmuje się współpracy z kontrahentem wyłącznie na podstawie jego akcji promocyjnych — przed podjęciem ostatecznej decyzji warto zweryfikować jego dotychczasowe osiągnięcia w niezależnych źródłach, którymi z pewnością nie powinny być osoby z jego najbliższej rodziny.

1.6.5. Kiedy uwierzytelnienie pakietu jest niemożliwe

Użytkownicy powinni się dobrze zastanowić, zanim podejmą decyzję o instalacji nieuwierzytelnionego pakietu — byłbym jednak nieuczciwy, gdybym się nie przyznał do łamania tej reguły i instalowania rozmaitych pakietów w swoim komputerze domowym. Zupełnie inne środki ostrożności należy stosować w przypadku komputerów domowych, a inne podczas wdrażania pakietów w sieciach korporacyjnych. Utrata danych składowanych w moim komputerze domowym nie byłaby wielkim problemem, awaria systemu w wielkiej korporacji byłaby natomiast prawdziwą katastrofą.

Zdarza się, że autor oprogramowania w ogóle nie dołącza żadnych informacji uwierzytelniających. Mimo to prawdopodobieństwo pobrania pakietu systemu Linux z koniem trojańskim jest stosunkowo niskie, co jednak w przyszłości może ulec zmianie. Poniżej przedstawiono kilka praktycznych kroków, które warto podjąć przed instalacją pakietu pozbawionego podpisu cyfrowego:

- Kompilacja kodu źródłowego.

  Takie podejście nie gwarantuje pełnego bezpieczeństwa (warto sobie przypomnieć choćby historię pakietu OpenSSH, opisaną we wcześniejszej części tego rozdziału). Warto mieć na uwadze, że poziom trudności procesu kompilacji kodów źródłowych może być zarówno bardzo niski, jak i bardzo wysoki. Każdy projekt jest inny — jedynym sposobem sprawdzenia, jak jest w przypadku naszego projektu, jest przeprowadzenie stosownego eksperymentu. Nawet w przypadku stosunkowo niewielkich, ale nieznanych nam pro-

jektów próba kompilacji może wymagać pobrania dodatkowych pakietów. Problem kompilacji kodu źródłowego został szczegółowo omówiony w rozdziale 2.

- Analiza skryptów instalacyjnych.

  Skrypty instalacyjne stanowią największe zagrożenie, ponieważ są wykonywane z uprawnieniami administratora w trakcie instalacji pakietów (a więc jeszcze przed właściwym uruchomieniem instalowanego oprogramowania). W dalszej części tego rozdziału omówię sposoby analizy skryptów dla poszczególnych formatów pakietów.

- Analiza właściwej zawartości pakietu.

  Warto się uważnie przyjrzeć instalowanym plikom binarnym. Typowa aplikacja użytkownika nie powinna wymagać do prawidłowej pracy binariów z katalogów */usr/sbin* ani */sbin*, ponieważ wspomniane katalogi są zarezerwowane dla demonów systemowych i narzędzi administracyjnych. Należy unikać wszelkich plików z ustawionym bitem `setuid` lub `setgid`, szczególnie jeśli ich właścicielem jest administrator. Tego rodzaju pliki mogą być wykonywane z uprawnieniami swojego właściciela i jako takie stanowią doskonałą kryjówkę dla złośliwego oprogramowania. Ustawione bity `setuid` lub `setgid` są niezbędne zaledwie w przypadku kilku programów systemowych — w pozostałych sytuacjach należy je traktować z daleko idącą ostrożnością.

Niezależnie od wybranej techniki musimy pamiętać, że wciąż kluczowym czynnikiem jest zaufanie. Tym razem zamiast wykazywać zaufanie do uwierzytelnionego źródła, musimy zaufać własnym umiejętnościom w zakresie identyfikacji złośliwego oprogramowania.

1.7. Analiza zawartości pakietu

Istnieje kilka aspektów, które warto poddać analizie przed przystąpieniem do procesu instalacji pobranego pakietu. Większość formatów pakietów obejmuje następujące elementy:

- Archiwum plików, które zostaną zainstalowane w naszym systemie. Tego rodzaju archiwa z reguły mają postać plików narzędzia `tar`, plików narzędzia `cpio` lub plików dowolnego innego programu archiwizującego.
- Skrypty, które będą wykonywane w trakcie instalacji i usuwania danego pakietu.
- Informacje o zależnościach narzędzia instalacyjnego, na podstawie których można stwierdzić, czy nasz system spełnia wymagania danego pakietu.
- Dodatkowe informacje tekstowe o samym pakiecie.

Ilość informacji zawartych w pakiecie zależy przede wszystkim od woli osoby, która ten pakiet przygotowywała. Zwykle są to podstawowe dane autora, data sporządzenia pakietu oraz warunki umowy licencyjnej. Bardziej odpowiedzialni producenci umieszczają dodatkowo w pakietach informacje o faktycznym przeznaczeniu oferowanego oprogramowania (choć wciąż zbyt wiele pakietów nie obejmuje tego rodzaju danych opisowych).

Informacje o zależnościach pakietów mogą być — w zależności od specyfiki oprogramowania — dość rozbudowane, mogą być bardzo proste lub mogą w ogóle nie istnieć. Pakiety dla dystrybucji zbudowanych na bazie projektu Slackware w ogóle np. nie obejmują danych o zależnościach. Oznacza to, że instalując tego rodzaju pakiety, musimy trzymać kciuki za ich prawidłowe funkcjonowanie. Na drugim biegunie znajdują się pakiety RPM. Podczas konstruowania takiego pakietu możemy wymusić na stosowanych narzędziach automatyczne wykrywanie i uwzględnianie ewentualnych zależności w ramach gotowego pakietu. Okazuje się jednak, że nawet w przypadku pakietów RPM decyzja o zapisaniu informacji o zależnościach należy do ich twórców, którzy mogą — wzorem dystrybucji Slackware — zrezygnować z tej opcji.

Każdy format pakietów oferuje jakąś metodę wykonywania skryptów w czasie instalacji i usuwania oprogramowania. Skrypty instalacyjne powinniśmy analizować ze szczególną uwagą. Nawet jeśli ryzyko występowania złośliwego oprogramowania wydaje nam się niewielkie, musimy pamiętać, że mniej dojrzałe projekty mogą obejmować wadliwe skrypty instalacyjne, które — wbrew intencjom swoich autorów — mogą doprowadzić do uszkodzenia naszego systemu. Jeśli wspomniane skrypty są zbyt skomplikowane i niezrozumiałe, warto poszukać jakiegoś sposobu uwierzytelnienia pakietu przed instalacją.

Skrypty instalacyjne zwykle można zaliczyć do następujących kategorii:

- **Przedinstalacyjne** — tego rodzaju skrypty są wykonywane przed rozpakowaniem danych z archiwum.
- **Poinstalacyjne** — skrypty zaliczane do tej kategorii są wykonywane po wypakowaniu danych z archiwum. Tego rodzaju skrypty z reguły odpowiadają za mniej istotne zadania związane z dostosowywaniem instalacji do wymagań docelowego systemu, w tym za ustawianie ścieżek czy tworzenie plików konfiguracyjnych.
- **Przeddeinstalacyjne** — tego rodzaju skrypty są wykonywane bezpośrednio po podjęciu decyzji o odinstalowaniu pakietu, ale przed usunięciem z systemu jakichkolwiek plików.
- **Podeinstalacyjne** — skrypty z tej grupy są wykonywane już po usunięciu z systemu podstawowych plików pakietu.

Zakres informacji tekstowych dołączanych do pakietów różni się w zależności od stosowanych formatów. Takie dane często obejmują dodatkowe informacje uwierzytelniające, w tym adres strony projektu w witrynie internetowej *SourceForge.net*. Także ilość informacji o zależnościach jest ściśle związana z formatem pakietu. Takie dane mogą obejmować nazwy pozostałych pakietów lub nazwy wymaganych programów wykonywalnych.

1.7.1. Jak analizować pobrane pakiety

Pakiety wymagają analizy zarówno przed instalacją, jak i po niej. Przed instalacją należy się dokładnie przyjrzeć plikowi pakietu, który może mieć postać dowolnego, prawidłowego pliku systemu Linux. Nazwa tego pliku z reguły, choć nie zawsze, pochodzi od **oficjalnej** nazwy pakietu, czyli nazwy składowanej w bazie danych zainstalowanych pakietów. Nazwa pakietu jest kodowana w ramach jego pliku i powinna być widoczna po wykonaniu podstawowego zapytania na tym pliku. Mimo że twórcy pakietów starają się uwzględniać ich nazwy w nazwach oferowanych plików, nie powinniśmy oczekiwać pełnej zgodności obu nazw. Do zainstalowanego pakietu możemy się odwoływać wyłącznie z wykorzystaniem nazwy określonej w ramach pliku pakietu, dlatego warto korzystać z zapytań zwracających oficjalne nazwy na podstawie jeszcze niezainstalowanych plików. Może się okazać, że np. plik pakietu RPM kompilatora `gcc` z jakiegoś powodu nazwano *foo.rpm*. Aby sprawdzić zawartość tego pliku, należy wykonać następujące polecenie:

```
$ rpm -qip foo.rpm
```

Już po instalacji opisane zadanie można zrealizować, stosując zapytanie w formie:

```
$ rpm -qi gcc
```

Polecenie `rpm` zwykle kieruje zapytanie do bazy danych o pakietach RPM, jednak opcja `-p` określa, że właściwym przedmiotem zapytania ma być **plik** pakietu. Ten sam pakiet może być reprezentowany przez dowolnie nazwane pliki, już po instalacji w bazie danych jest jednak reprezentowany przez jedną nazwę.

Jak już wspomniano, podstawowe informacje włączane do pakietu obejmują następujące elementy: jego nazwę, numer wersji, dane autora, informacje o prawach autorskich oraz opis zależności. Informacje dodatkowe obejmują listę instalowanych plików, a także wykaz wszelkich skryptów wykonywanych w czasie instalacji i usuwania pakietu. Tego rodzaju dane powinniśmy poddawać analizie przed przystąpieniem do instalacji pakietu. Przykłady podstawowych zapytań kierowanych do plików pakietów w formatach RPM i Debian przedstawiono w tabeli 1.10.

TABELA 1.10. Przykładowe zapytania kierowane do plików pakietów

Zapytanie	RPM	Debian
Informacje podstawowe	`rpm -qpi` *`nazwa_pliku`*	`dpkg -s` *`nazwa_pliku`*
Lista instalowanych plików	`rpm -qpl` *`nazwa_pliku`*	`dpkg -L` *`nazwa_pliku`*
Skrypty instalacyjne lub deinstalacyjne	`rpm -qp -scripts` *`nazwa_pliku`*	`dpkg -e`
Weryfikacja informacji uwierzytelniających	`rpm –checksig` *`nazwa_pliku`*	niedostępny
Lista innych pakietów wymaganych	`rpm -qp --requires` *`nazwa_pliku`*	`dpkg -I`
Pakiet zawarty w ramach danego pliku (np. nazwa i wersja występująca w bazie danych o pakietach)	`rpm -qp --provides` *`nazwa_pliku`*	`dpkg -I`

Istnieje wiele powodów, dla których możemy się decydować na kierowanie zapytań do bazy danych o pakietach. Możemy np. potrzebować listy wszystkich pakietów zainstalowanych w systemie lub być zmuszeni do określenia zainstalowanej wersji interesującego nas pakietu. Innym przydatnym krokiem jest weryfikacja zawartości zainstalowanego pakietu celem sprawdzenia, czy żaden z zainstalowanych plików nie został przypadkowo usunięty. Format zapytań dotyczących pakietów już zainstalowanych jest inny niż w przypadku plików pakietów. Kilka przykładowych zapytań dotyczących zainstalowanych pakietów przedstawiono w tabeli 1.11.

TABELA 1.11. Zapytania dotyczące zainstalowanych pakietów

Zapytanie	RPM	Debian
Podstawowe informacje o określonym pakiecie	`rpm -qi` *`nazwa`*	`dpkg -s` *`nazwa`*
Lista wszystkich zainstalowanych pakietów	`rpm –qa`	`dpkg --list`
Lista wszystkich plików zainstalowanych przez dany pakiet	`rpm -ql` *`nazwa`*	`dpkg -L` *`nazwa`*

Weryfikacja plików zainstalowanych przez dany pakiet	`rpm -V` *`nazwa`*	`cd /;`
		`md5sum -c <`
		`/var/lib/dpkg/info/name.md5sums`

Określenie pakietu, do którego należy dany plik	`rpm -qf` *`nazwa_pliku`*	`dpkg -S` *`nazwa_pliku`*
Określenie aktualnie zainstalowanej wersji pakietu *X*	`rpm -q` *X*	`dpkg-query -W` *X*

1.7.2. Szczegółowa analiza pakietów RPM

RPM jest jednym z najbardziej dojrzałych i wszechstronnych formatów pakietów, stosowanych w systemach operacyjnych Linux. Pakiet RPM może obejmować mnóstwo przydatnych informacji, co wcale nie oznacza, że ich wyodrębnienie z tego pakietu musi być łatwe. Aby uprościć uzyskiwanie tych dodatkowych danych, narzędzie `rpm` oferuje opcję `--queryformat` (`--qf` w wersji skróconej). Większości znaczników stosowanych przez tę opcję nie udokumentowano w podręczniku użytkownika, ich listę można jednak uzyskać za pomocą polecenia w postaci:

```
$ rpm --querytags
HEADERIMAGE
HEADERSIGNATURES
HEADERIMMUTABLE
HEADERREGIONS
HEADERI18NTABLE
SIGSIZE
SIGPGP
SIGMD5
SIGGPG
PUBKEYS
...
```

Warto pamiętać, że w przypadku znaczników zapytań wielkość liter nie ma znaczenia, mimo że dane wynikowe, prezentowane przez narzędzie `rpm`, są pisane wielkimi literami. Gdybyśmy np. chcieli się dowiedzieć, kto opracował poszczególne pakiety dla naszej dystrybucji, powinniśmy wykonać następujące zapytanie:

```
$ rpm -qa --qf '%{vendor}' | sort | uniq -c
1 Adobe Systems, Incorporated
12 (none)
1 RealNetworks, Inc
838 Red Hat, Inc.
1 Sun Microsystems, Inc.
```

Powyższe zapytanie, wykonane w moim systemie Fedora Core 3, wykazało, że aż 838 pakietów opracowała firma Red Hat, a 12 pakietów pochodzi z niezidentyfikowanych źródeł. Okazuje się, że wspomniane pakiety z niezidentyfikowanych źródeł to tak naprawdę klucze publiczne GPG. Każdy taki klucz jest reprezentowany w bazie danych jako odrębny pakiet (zwykle bez ustawionego identyfikatora twórcy).

Innym ciekawym przykładem zapytania jest polecenie generujące listę skryptów instalacyjnych, dołączonych do wskazanego pakietu RPM. Poniżej przedstawiono jedno z takich zapytań:

```
$ rpm -qp --scripts gawk-3.1.3-9.i386.rpm

postinstall scriptlet (through /bin/sh):
if [ -f /usr/share/info/gawk.info.gz ]; then
    /sbin/install-info /usr/share/info/gawk.info.gz
/usr/share/info/dir
fi
preuninstall scriptlet (through /bin/sh):
if [ $1 = 0 -a -f /usr/share/info/gawk.info.gz ]; then
    /sbin/install-info --delete /usr/share/info/gawk.info.gz
/usr/share/info/dirfi
```

Powyższe dane wynikowe obejmują pojedynczy wiersz identyfikujący przeznaczenie skryptu (np. skryptu poinstalacyjnego) oraz typ skryptu (np. */bin/sh*). Takie rozwiązanie umożliwia nam wstępną analizę skryptów jeszcze przed ich wykonaniem.

Zawartość pliku archiwalnego, składowanego w ramach pakietu RPM, możemy uzyskać za pomocą polecenia `rpm2cpio`. Konwertuje ono wskazany plik pakietu RPM do postaci archiwum narzędzia `cpio` (czyli formatu wykorzystywanego wewnętrznie przez format RPM). `cpio` jest formatem archiwów, podobnym do formatu narzędzia `tar`, ale cechującym się nieco inną składnią. Inaczej niż w przypadku narzędzia `tar`, dane wynikowe narzędzia `rpm2cpio` są domyślnie kierowane do strumienia `stdout` (podobnie działa samo narzędzie `cpio`). Aby wyodrębnić pliki z pakietu RPM do katalogu bieżącego (bez instalowania tego pakietu), należy wykonać następujące polecenie:

```
rpm2cpio filename.rpm | cpio -i --no-absolute-filenames
```

Warto zwrócić uwagę na opcję `--no-absolute-filenames` polecenia `cpio`, która eliminuje ryzyko nadpisania cennych plików systemowych. W praktyce pakiety RPM nie dają możliwości stosowania ścieżek bezwzględnych w archiwach narzędzia `cpio`, co nie oznacza, że takie dodatkowe zabezpieczenie nie ma sensu.

1.7.3. Szczegółowa analiza pakietów Debiana

Pakiety Debiana cechują się prostszym formatem niż pakiety RPM, a narzędzie `dpkg` nie oferuje wielu spośród funkcji znanych użytkownikom narzędzia `rpm`. Oznacza to, że praca z tego rodzaju pakietami wymaga od użytkownika opanowania nowych umiejętności. Pliki pakietów Debiana z reguły wyróżnia się rozszerzeniem *.deb*, mimo że w rzeczywistości są to archiwa utworzone za pomocą programu `ar`. W tej sytuacji można oczywiście przeglądać zawartość tego rodzaju pakietów za pomocą polecenia `ar`, tak uzyskiwane informacje będą jednak niekompletne. Poniżej przedstawiono odpowiedni przykład:

```
$ ar -t cron_3.0pl1-72_i386.deb
debian-binary
control.tar.gz
data.tar.gz
```

Plik nazwany *debian-binary* zawiera pojedynczy wiersz tekstu ASCII, określający wersję pakietu i format wykorzystany do jego utworzenia. Plik nazwany *control.tar.gz* jest skompresowanym archiwum narzędzia `tar`, obejmującym zarówno skrypty instalacyjne, jak i kilka innych przydatnych informacji. Plik nazwany *data.tar.gz* jest skompresowanym archiwum narzędzia `tar`, obejmującym pliki instalacyjne programu. Wyodrębnienie tych plików celem dalszej analizy wymaga użycia następującego polecenia `ar`:

```
$ ar -x filename.deb
```

Spróbujmy teraz poddać ten przykładowy plik bardziej szczegółowej analizie. Plik *data.tar.gz* zawiera pliki wymagane do prawidłowego funkcjonowania danego programu. W niektórych przypadkach instalacja wymaga tylko wypakowania plików z pobranego archiwum, co jednak nie jest zalecane. W tym konkretnym przypadku lista plików zawartych w naszym archiwum ma następującą postać:

```
$ tar -tzf data.tar.gz
./
./usr/
./usr/bin/
./usr/bin/crontab
./usr/sbin/
./usr/sbin/cron
./usr/sbin/checksecurity
./usr/share/
./usr/share/man/
./usr/share/man/man1/
...
```

Plik *control.tar.gz* zawiera przede wszystkim pliki wymagane w procesach instalacji, usuwania i konserwacji pakietu. Można te pliki wypakować za pomocą polecenia `dpkg` z opcją `-e`:

```
$ dpkg -e cron_3.0pl1-72_i386.deb
$ ls ./DEBIAN/*
./DEBIAN/conffiles
./DEBIAN/control
./DEBIAN/md5sums
./DEBIAN/postinst
```

```
./DEBIAN/postrm
./DEBIAN/preinst
./DEBIAN/prerm
```

Jak łatwo się domyślić, pliki *preinst* i *postinst* to, odpowiednio, skrypty przed- i poinstalacyjne (opisane we wcześniejszej części tego rozdziału). Podobnie: pliki *prerm* i *postrm* to, odpowiednio, skrypty przed- i podeinstalacyjne.

Plik *md5sums* zawiera listę kodów MD5, które można wykorzystać do sprawdzenia integralności plików zawartych w archiwum *data.tar.gz*. Plik *md5sums* można wykorzystać w roli danych wejściowych dla programu `md5sum` — warto jednak mieć na uwadze, że zawarte w tym pliku kody mogą służyć wyłącznie weryfikacji, nie uwierzytelnianiu. Za pomocą narzędzia `md5sum` możemy sprawdzić, czy dany pakiet nie uległ uszkodzeniu przed instalacją i czy uszkodzeniu nie uległy pliki już zainstalowane, wspomniany program w żaden sposób nie ułatwia jednak zadania uwierzytelniania źródła weryfikowanych plików. Tak czy inaczej, okresowe sprawdzanie zainstalowanych pakietów za pośrednictwem tego narzędzia jest dobrą praktyką.

Plik *md5sums* nie obejmuje kodów wszystkich instalowanych plików, ponieważ pakiety często wymagają plików konfiguracyjnych, które powinny być modyfikowane już po instalacji. Oczekuje się, że po instalacji tego rodzaju pliki **nie** będą pasowały do oryginalnej zawartości pakietu. Pliki konfiguracyjne można wyłączyć z procesu weryfikacji, umieszczając ich nazwy w pliku *conffiles*. Wszystkie pliki wymienione w tym pliku są ignorowane w czasie weryfikacji integralności instalacji.

1.8. Aktualizowanie pakietów

Narzędzia aktualizujące pakiety ułatwiają śledzenie plików poszczególnych pakietów oraz wiążących je zależności. Przypuśćmy, że chcemy zainstalować pakiet *X*, który wymaga trzech innych, do tej pory niezainstalowanych pakietów. Oznacza to, że będziemy musieli zainstalować te trzy pakiety, zanim będziemy mogli zainstalować pakiet *X*. Musimy jednak pamiętać, że także te trzy pakiety mogą wymagać innych, również niezainstalowanych pakietów, które z kolei mogą wymagać jeszcze innych pakietów itd.

Właśnie w takich sytuacjach narzędzia aktualizujące pakiety okazują się szczególnie przydatne. Jeśli dysponujemy takim narzędziem, wystarczy, że wskażemy interesujący nas pakiet *X* — wspomniane narzędzie automatycznie określi, jakie inne pakiety są wymagane do instalacji, po czym wykona kroki niezbędne do ich pobrania i zainstalowania.

Działanie narzędzia aktualizującego polega na utrzymywaniu listy repozytoriów pakietów, niezbędnych podczas poszukiwania żądanych pakietów. Takie repozytoria są udostępniane w internecie, a za ich konserwację odpowiada dystrybutor (np. firma Red Hat). Za pośrednic-

twem repozytorium dystrybutor oprogramowania sygnalizuje dostępność poprawek i aktualizacji zabezpieczeń, a często także bardziej ogólne aktualizacje.

Repozytorium może być składowane także w lokalnym systemie plików, np. na płycie CD lub na innym komputerze oddzielonym (wraz z naszym komputerem) od sieci zewnętrznym firewallem. To drugie rozwiązanie sprawdza się, gdy musimy utrzymywać wiele komputerów w jednej sieci LAN. Możemy pobierać niezbędne pakiety z internetu i udostępniać je w sieci wewnętrznej z myślą o szybszej aktualizacji komputerów klienckich.

Do obsługi pakietów dystrybucji, korzystających z formatu Debiana, najlepiej stosować narzędzie APT (od ang. *Advanced Package Tool*). Przystosowano je także do pracy z dystrybucjami stosującymi format RPM. To, czy APT zyska popularność jako najlepsze narzędzie do zarządzania pakietami w formacie RPM, okaże się w bliższej lub dalszej przyszłości.

W kontekście dystrybucji, w których stosuje się pakiety w formacie RPM, warto wspomnieć o dwóch najważniejszych narzędziach. Pierwszym z nich jest program up2date, zaprojektowany Red Hat z myślą o dystrybucjach Enterprise Server i Fedora Core. Drugim takim narzędziem jest YUM (od ang. *Yellowdog Updater Modified*)[5].

Niektórzy twierdzą, że narzędzia APT i YUM mogą aktualizować całe instalacje — np. zaktualizować instalację Red Hat 8.0 do wersji 9.0 bez konieczności ponownego instalowania całego systemu operacyjnego. Byłbym bardzo ostrożny przed podjęciem decyzji o przetestowaniu tego rozwiązania na własnym systemie[6].

Nie należy oczekiwać od narzędzia aktualizującego pakiety realizacji wszystkich niezbędnych zadań. Ponieważ tego rodzaju narzędzia bazują na kilku wybranych repozytoriach i tam poszukują pakietów, zakres dostępnego oprogramowania i wersji jest ograniczony. Oficjalne repozytoria z reguły preferują dojrzałe narzędzia i sprawdzone, stabilne wersje. Okazuje się jednak, że pewne narzędzia lub wersje są umieszczane przez twórców repozytoriów wbrew tej regule, albo wskutek kaprysów, albo z uwagi na obsługę pewnych dystrybucji. W tej sytuacji nigdy nie należy brać poważnie słów autorów repozytoriów: „Jeśli nie ma czegoś w repozytorium, najpewniej nie będziesz tego potrzebował". Mnóstwo doskonałych produktów nie jest uwzględnianych w dystrybucjach ani repozytoriach. Jeśli chcemy korzystać z najnowszej wersji jakiegoś pakietu lub spróbować czegoś nowego, nietypowego, najprawdopodobniej będziemy musieli zrezygnować z usług narzędzia aktualizującego.

---

[5] Kiedy to samo narzędzie wchodziło w skład dystrybucji Yellowdog dla platformy PowerPC, nazywano je `yup`.

[6] Do najczęstszych skutków ubocznych tego rodzaju działań należy ból głowy, infekcja uszu, stany lękowe, mdłości i wymioty.

Wystarczy chwila poszukiwań, aby odnaleźć w internecie raporty o błędach i skargi użytkowników poszczególnych narzędzi aktualizujących pakiety. Dbałość o stałe aktualizowanie setek niezależnych pakietów oprogramowania bywa jednak bardzo trudne. Mimo że programiści stale podnoszą swoje umiejętności i nabierają doświadczenia, występowanie błędów jest nieuniknione. Wielu Czytelników zapewne ucieszy to, że opisywany obszar jest przedmiotem ciągłego rozwoju, odpowiednie rozwiązania w przyszłości będą więc zapewne lepsze od współczesnych.

1.8.1. APT — Advanced Package Tool

APT jest jednym z najbardziej dojrzałych narzędzi do zarządzania pakietami w dystrybucjach budowanych na bazie Debiana, a od jakiegoś czasu dodatkowo oferuje możliwość zarządzania pakietami w dystrybucjach wykorzystujących format RPM. Największą zaletą APT-a jest to, że w przeciwieństwie do narzędzia `dpkg`, umożliwia automatyczne uwierzytelnianie pakietów obejmujących podpisy cyfrowe GPG. Warto pamiętać, że podpisy GPG już teraz są obsługiwane przez format RPM. Co więcej, narzędzie APT pyta użytkownika o zgodę przed instalacją pakietu, którego uwierzytelnienie jest niemożliwe, nie musimy się zatem obawiać zmodyfikowanych repozytoriów ani koni trojańskich umieszczanych w pakietach.

Program APT, podobnie jak narzędzie `dpkg`, nie ma postaci pojedynczego polecenia, tylko zbioru poleceń. Do najczęściej stosowanych poleceń APT-a należą `apt-get` i `apt-cache`. Początkowo będziemy korzystali przede wszystkim z polecenia `apt-get`, czyli rozbudowanego mechanizmu obsługującego zadania związane z pobieraniem i instalacją pakietów. Polecenie `apt-cache` umożliwia nam wykonywanie zapytań na liście pakietów pobranych i umieszczonych w lokalnej pamięci podręcznej, co oznacza znaczne przyspieszenie względem wielokrotnego wykonywania zapytań na repozytoriach udostępnianych w internecie. Przetwarzana lista obejmuje nie tylko wszystkie dostępne pakiety, włącznie z tymi, których jeszcze nie zainstalowano, ale też zainstalowane aktualizacje.

Polecenie `apt-key` umożliwia dodawanie do bazy danych kluczy publicznych, uzyskiwanych z zaufanych źródeł. Za pomocą tego samego polecenia możemy przeglądać klucze już składowane w tej bazie danych. W ten sposób możemy uwierzytelniać pakiety podpisane przez twórców. Za pomocą polecenia `apt-setup` możemy wskazać repozytoria, w których narzędzie APT powinno poszukiwać pakietów. W mojej dystrybucji Ubuntu polecenie `apt-setup` nie umożliwia odwoływania się do repozytoriów Debiana, a jedynie do serwerów Ubuntu. W tej sytuacji warto rozważyć ręczną edycję pliku */etc/apt/sources.list* celem dodania niezbędnych repozytoriów. Elementy zdefiniowane w pliku */etc/apt/sources.list* mogą wskazywać na dowolne witryny internetowe, katalogi lokalne naszego systemu lub katalogi dostępne w naszej sieci lokalnej. APT wymaga tylko, by do repozytoriów odwoływać się za pośrednictwem adresów URL.

1.8.2. YUM — Yellowdog Updater Modified

YUM jest alternatywnym narzędziem aktualizującym pakiety w systemach stosujących format RPM. Ten program wiersza poleceń działa w sposób zbliżony do APT-a. Yum, podobnie jak APT, utrzymuje pamięć podręczną z informacjami o dostępnych pakietach. Inaczej jednak niż APT domyślnie kieruje zapytania do wszystkich repozytoriów podczas każdego uruchomienia. Takie rozwiązanie jest dużo bardziej czasochłonne niż odwołania do samej pamięci podręcznej (jak w przypadku narzędzia APT).

Za instalację, wykonywanie zapytań i aktualizowanie pakietów odpowiada polecenie `yum`. Opcja `-C` wymusza na programie użycie pamięci podręcznej na potrzeby żądania początkowego. Oznacza to, że jeśli na bazie tego żądania narzędzie YUM zdecyduje o instalacji oprogramowania, w pierwszej kolejności zaktualizuje pamięć podręczną.

Narzędzie YUM oferuje opcjonalną możliwość uwierzytelniania pakietów na podstawie podpisów GPG. Można tym procesem sterować (na poziomie poszczególnych repozytoriów) za pośrednictwem plików konfiguracyjnych */etc/yum.conf* i */etc/yum.repos.d*. Jeśli ustawimy flagę `gpgcheck=1`, polecenie `yum` nie będzie instalowało nieuwierzytelnionych pakietów. Podobnie jak w przypadku narzędzia APT istnieje możliwość tworzenia własnych repozytoriów w katalogach, do których można uzyskiwać dostęp za pośrednictwem adresów URL.

Opcje polecenia `yum` są dość intuicyjne. Aby uzyskać np. listę wszystkich aktualnie zainstalowanych pakietów, dla których istnieją aktualizacje, należy użyć polecenia w postaci:

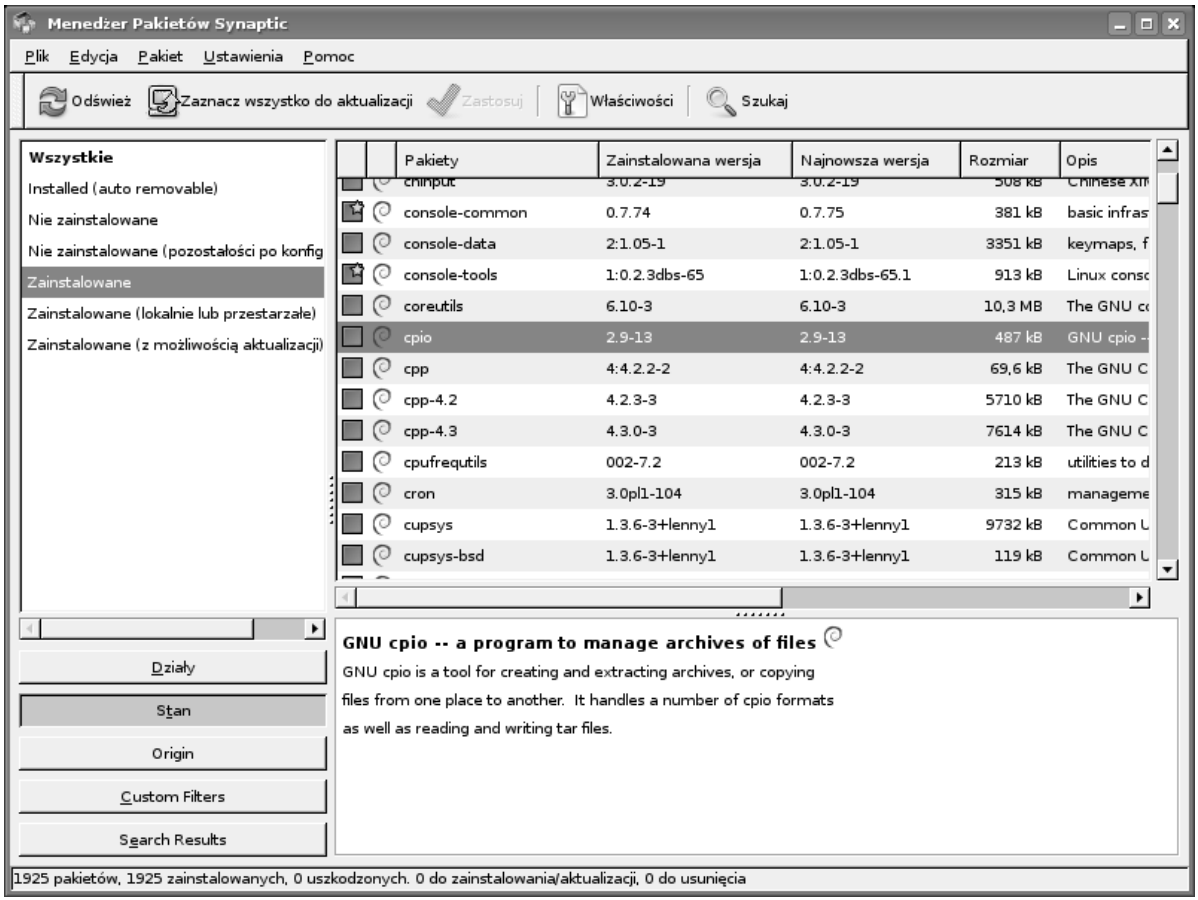
```
$ yum list updates
```

W ten sposób możemy wygenerować listę pakietów, które można zaktualizować. Najbliższym odpowiednikiem przytoczonego polecenia w świecie APT-a jest dużo mniej intuicyjna `apt-get --dry-run -u dist-upgrade` (która dodatkowo wyświetla mnóstwo nieprzydatnych danych dodatkowych).

1.8.3. Synaptic — nakładka narzędzia APT z graficznym interfejsem użytkownika

Kiedy pisałem tę książkę, projekt Synaptic nie osiągnął nawet wersji 1.0, a mimo to był już wyjątkowo wygodną nakładką z graficznym interfejsem użytkownika, ułatwiającym zarządzanie pakietami z wykorzystaniem narzędzia APT. W moim komputerze z dystrybucją Debiana było zainstalowanych 861 pakietów. Za każdym razem, kiedy korzystałem z narzędzia aktualizującego, okazywało się, że istnieją aktualizacje przynajmniej dla kilkudziesięciu spośród tych pakietów. W tym i podobnych przypadkach naturalnym rozwiązaniem jest użycie oprogramowania z graficznym interfejsem użytkownika (GUI). Synaptic grupuje pakiety według kategorii, co znacznie

ułatwia lokalizowanie i przeglądanie aktualizacji dla najczęściej stosowanych programów. Programiści najprawdopodobniej będą chcieli wiedzieć, kiedy kompilator `gcc` zostanie zaktualizowany z wersji 3.3 do wersji 3.4, ale raczej nie będą zainteresowani aktualizacją gry FreeCell z wersji 1.0.1 do wersji 1.0.2. Kategorie można wykorzystywać także do poszukiwania nowszego oprogramowania lub narzędzi, których do tej pory w ogóle nie instalowaliśmy. Jako programista regularnie przeglądam dostępne narzędzia programistyczne pod kątem nowych, często bardzo obiecujących projektów. Przykład działania nakładki Synaptic w praktyce przedstawiono na rysunku 1.2.



RYSUNEK 1.2. Przykład graficznego interfejsu użytkownika nakładki Synaptic

Fronton Synaptic, podobnie jak narzędzie APT, nie instaluje nieuwierzytelnionych pakietów bez wyraźnego potwierdzenia takiej woli przez użytkownika. Jedną z ciekawszych funkcji tego programu jest możliwość zapoznawania się z konsekwencjami planowanych działań bez konieczności długiego oczekiwania na pobranie niezliczonych pakietów, którymi często nie jesteśmy zainteresowani. Jeśli przejdę do kategorii *Games and Amusements* i nakażę instalację gry `kasteroids`, stanę przed pewnym problemem — dystrybucja Ubuntu domyślnie wykorzystuje środowisko Gnome, a gra `kasteroids` jest typową aplikacją środowiska KDE. Jeśli więc zdecyduję się na instalację tej gry, będę się musiał liczyć z koniecznością instalacji 10 dodatkowych pakietów. Synaptic może oczywiście zrealizować to zadanie bez mojego udziału, ale najpierw wygeneruje stosowne ostrzeżenie o potrzebie instalacji 10 dodatkowych pakietów wymagających pobrania wielu megabajtów. Takie rozwiązanie sprawia, że jeśli nie mamy zbyt wiele czasu, powinniśmy odłożyć instalację gry `kasteroids` na później.

Synaptic, niestety, nie oferuje funkcji identyfikacji pakietów, dla których są dostępne aktualizacje zabezpieczeń. Określenie, czy narzędzie `gcalctool` naprawdę wymaga aktualizacji z wersji 5.5.41-0 do wersji 5.5.41-1, jest niezwykle trudne. Na pierwszy rzut oka można by stwierdzić, że mamy do czynienia z drobną zmianą, eliminującą, być może, jeden błąd. Czy taki wniosek uprawnia nas jednak do stwierdzenia, że chodzi o aktualizację zabezpieczeń? Trudno ocenić. Wydaje się, że społeczność open source powinna się wreszcie zająć tym problemem.

Innym przydatnym mechanizmem oferowanym przez nakładkę graficzną Synaptic jest filtr, za pomocą którego możemy zarządzać ilością prezentowanych danych o aktualizacjach i — tym samym — ograniczać te informacje do tych, którymi naprawdę jesteśmy zainteresowani. Synaptic wciąż znajduje się we wczesnej fazie rozwoju (w czasie, kiedy pisano tę książkę, była to wersja 0.56), stąd konieczność udoskonalenia wspomnianego filtra. Obecnie nie ma możliwości eliminowania z prezentowanych danych np. mniej istotnych zmian w pakietach. Takie zmiany zwykle polegają na eliminowaniu pojedynczych błędów i wprowadzaniu poprawek w obszarze zabezpieczeń; poważne zmiany z reguły polegają na dodawaniu nowych funkcji.

Synaptic okazuje się wyjątkowo wygodny i jako taki powinien być wykorzystywany w roli nakładki GUI we wszystkich systemach na bazie Debiana. W przyszłości Synaptic zyska, być może, popularność także w dystrybucjach stosujących pakiety w formacie RPM (obecnie tylko niewielka część repozytoriów pakietów w tym formacie jest zgodna z narzędziami APT i Synaptic).

1.8.4. up2date — narzędzie aktualizujące pakiety dystrybucji Red Hat

Firma Red Hat opracowała narzędzie `up2date`, oferujące graficzny interfejs użytkownika, który można z powodzeniem stosować do przetwarzania repozytoriów narzędzia YUM. Program `up2date`, uruchomiony bez żadnych opcji, generuje i wyświetla listę dostępnych plików aktualizacji (często obejmującą setki pozycji), spośród których możemy wskazać pakiety do zaktualizowania.

Dane wynikowe domyślnie nie obejmują pakietów, których jeszcze nie zainstalowaliśmy, co oznacza, że w razie pojawienia się nowego narzędzia `up2date` nie poinformuje o tym użytkownika. `up2date`, podobnie jak Synaptic, uwierzytelnia pakiety na podstawie podpisów GPG i nie instaluje pakietów, w przypadku których takie uwierzytelnienie jest niemożliwe.

Graficzny interfejs użytkownika narzędzia `up2date` pozostawia wiele do życzenia. Oferowany zakres funkcjonalności jest minimalny i obejmuje wyłącznie aktualizację istniejących pakietów — nie prezentuje użytkownikowi nowych produktów, nie umożliwia też przeglądania i odinstalowywania pakietów. Brak odpowiednich mechanizmów jest sporą stratą, szczególnie jeśli mamy na uwadze przydatność i intuicyjność mechanizmów obsługiwanych z poziomu wiersza poleceń.

`up2date` próbuje w sposób możliwie transparentny zapewniać użytkownikowi dostęp do repozytoriów narzędzi YUM, APT i samego narzędzia `up2date`. Konfiguracja domyślna, dołączana do dystrybucji Fedora Core 4, kieruje nas do wyczerpującej listy repozytoriów narzędzia YUM. Takie rozwiązanie początkowo sprawia wrażenie przemyślanego, szybko jednak okazuje się, że tak realizowane aktualizacje wymagają dużo więcej czasu. Lepszym podejściem jest samodzielne operowanie na kilku repozytoriach i bezpośrednie dodawanie odpowiednich odwołań do pliku */etc/sysconfig/rhn/sources*. Ciekawą funkcją narzędzia `up2date` jest możliwość wskazywania katalogów pełnych pakietów RPM i automatycznego określania łączących je zależności. Dopóki wskazany katalog zawiera wszystkie wymagane pakiety RPM, proces przebiega znakomicie. W tej sytuacji warto rozważyć zamontowanie instalacyjnej płyty DVD jako */mnt/dvd* i umieszczenie w pliku */etc/sysconfig/rhn/sources* następującego wiersza:

```
dir fc-dvd /mnt/dvd/Fedora/RPMS
```

Od tej pory możemy instalować pakiety z płyty DVD i korzystać z mechanizmów narzędzia `up2date`, które będą automatycznie uwzględniały zależności. Poniżej przedstawiono prosty przykład wywołania tego narzędzia z poziomu wiersza poleceń:

```
$ up2date --install gcc
```

Nie miałem, niestety, zbyt wiele szczęścia, kiedy próbowałem sprawdzić graficzny interfejs użytkownika narzędzia `up2date` w praktyce. Sądziłem, że mogę wykorzystać to narzędzie w moim systemie Fedora Core 3 do zaktualizowania zbioru pakietów wyselekcjonowanych z listy ponad 200 programów, dla których istniały aktualizacje. Okazało się jednak, że kiedy kliknąłem przycisk *OK*, graficzny interfejs użytkownika nagle przestał odpowiadać. W czasie przesyłania danych za pośrednictwem mojego połączenia szerokopasmowego można było odnieść wrażenie, że system utknął w martwym punkcie — na ekranie nie pojawił się nawet pasek postępu. Po około 15

minutach narzędzie `up2date` poinformowało mnie o niezgodności jądra z dwoma spośród wybranych pakietów i niepowodzeniu aktualizacji wszystkich pozostałych pakietów.

Jednym z głównych powodów wspomnianego spowolnienia było wskazywanie za pośrednictwem pliku */etc/syscsonfig/rhn/sources* repozytoriów narzędzia `yum`, wymienionych w pliku */etc/yum.repos.d*, który zawierał sześć wpisów (każdy z listą kopii witryn). Wygląda na to, że właśnie lista kopii stron doprowadziła do opisanej przed chwilą sytuacji. Jedna z tych list obejmowała aż 65 witryn! Wydaje się, że w przeciwieństwie do programu uruchamianego z poziomu wiersza poleceń, prezentowany interfejs GUI wykorzystuje te informacje bardzo nieefektywnie. W tej sytuacji warto sprawdzić działanie narzędzia `up2date`, zanim graficzny interfejs użytkownika zdąży nas zniechęcić.

1.9. Podsumowanie

W tym rozdziale omówiono wybrane aspekty idei oprogramowania open source. Koncentrowaliśmy się przede wszystkim na formatach pakietów stosowanych w rozmaitych dystrybucjach oraz na narzędziach, które na tych formatach operują. Poświęciliśmy trochę czasu plikom archiwalnych, które są powszechnie stosowane w formatach wszystkich współczesnych dystrybucji, a w niektórych przypadkach wręcz **stanowią** te formaty.

Omówiłem kilka podstawowych rozwiązań w zakresie bezpieczeństwa, których stosowanie pozwala wyeliminować lub ograniczyć ryzyko pobierania i instalacji złośliwego oprogramowania. Przeanalizowałem podstawy techniki uwierzytelniania i ogólne techniki zabezpieczania systemów informatycznych.

Na końcu skupiliśmy uwagę na kilku narzędziach zbudowanych ponad popularnymi mechanizmami przetwarzania pakietów. Za pomocą tych narzędzi można z powodzeniem zarządzać wszystkimi pakietami zainstalowanymi w naszym systemie. Każde z tych rozwiązań ma swoje zalety i wady.

1.9.1. Narzędzia użyte w tym rozdziale

- `dpkg` — podstawowe narzędzie wykorzystywane do instalowania i przeglądania pakietów w formacie Debian (stosowanym przez dystrybucję Debian i jej dystrybucje pokrewne, w tym Ubuntu).
- `gpg` — narzędzie GNU, odpowiedzialne za szyfrowanie i podpisywanie danych. Ten uniwersalny program może z powodzeniem służyć do zabezpieczania pakietów przez dodawanie podpisów cyfrowych.

- `gzip`, `bzip2` — narzędzia GNU, umożliwiające kompresję plików (stosowane zwykle łącznie z plikami archiwalnymi).
- `rpm` — podstawowe narzędzie do instalacji i przeglądania pakietów w formacie Red Hat Package Manager (RPM). Format RPM jest wykorzystywany nie tylko w dystrybucji Red Hat, ale też w dystrybucji Suse i innych.
- `tar`, `cpio`, `ar` — narzędzia archiwizujące systemu Unix, wykorzystywane w większości formatów pakietów.

1.9.2. Materiały dostępne w internecie

- *http://www.debian.org/* — strona domowa dystrybucji Debian, obejmująca dział często zadawanych pytań (FAQ), w którym omówiono m.in. format pakowania.
- *http://www.gnupg.org/* — strona domowa projektu GNU Privacy Guard, w ramach którego opracowano narzędzie `gpg`.
- *http://www.pgp.net/* — repozytorium kluczy publicznych, stosowanych m.in. przez narzędzie `gpg`.
- *http://www.rpm.org/* — strona domowa projektu RPM.