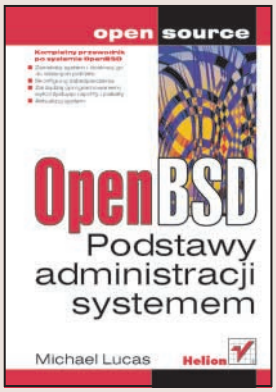
OpenBSD. Podstawy administracji systemem



Autor: Michael W. Lucas
Tłumaczenie: Michalski Mateusz, Moch Wojciech
ISBN: 83-7361-603-9
Tytuł oryginału: Absolute OpenBSD
Format: B5, stron: 552

OpenBSD jest to system operacyjny powszechnie uznawany za najbezpieczniejszy wśród systemów dostępnych na zasadach jakiejkolwiek licencji. Zawiera mniej luk niż w zabezpieczeniach niż dowolna wersja Windows lub Linuksa. Jego twórcy postawili na niezawodność, stabilność i bezpieczeństwo. W wyniku ich prac powstał i nadal rozwijany jest system operacyjny dostępny nieodpłatnie i możliwy do uruchomienia zarówno na zabytkowych komputerach klasy 386, jak i na nowoczesnych serwerach. Jego stabilność doceniło wiele firm, w tym Adobe i Apple. Ta ostatnia oparła na jądrze BSD swój system Mac OS X.

„OpenBSD. Podstawy administracji systemem" to praktyczny przewodnik opisujący wszystkie elementy systemu OpenBSD. Dzięki zawartym w niej wiadomościom poznasz system, będziesz w stanie efektywnie nim administrować, wykorzystywać zaimplementowane w nim mechanizmy zabezpieczające i instalować nowe oprogramowanie. Dowiesz się, jak w pełni wykorzystać możliwości OpenBSD.

- Instalacja systemu
- Konfiguracja i pierwsze uruchomienie
- Administrowanie kontami użytkowników
- Mechanizmy sieciowe
- Zabezpieczenia
- Konfiguracja i rekompilacja jądra
- Administrowanie systemem plików
- Filtrowanie pakietów i system PF

Cennym uzupełnieniem wiadomości zawartych w książce jest lista opcji kompilacji jądra oraz zestawienie przykładowych konfiguracji systemu PF.

Jeśli chcesz poznaj najbezpieczniejszy system operacyjny na świecie – sięgnij po tę książkę.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 7.

Obsługa użytkowników

Ten może się zalogować,
inny może odbierać pocztę;
superużytkownika pozostaw dla siebie.

Ataki na komputery wykonywane poprzez internet są najszerzej rozgłaszanym rodzajem włamań sieciowych, jednak największe zagrożenie dla bezpieczeństwa najczęściej stanowią sami użytkownicy systemów. Firmy tracą więcej ważnych danych z powodu działań nielojalnych lub niekompetentnych pracowników (a prym wiedzie tutaj niekompetencja), niż z powodu włamań z zewnątrz. Zapewnienie użytkownikom pełnego dostępu do systemu jest błędem nie tylko z powodu naruszenia bezpieczeństwa; bardzo szybko powodując utratę stabilności przez środowisko, ponieważ użytkownicy będą wprowadzali konfliktujące ze sobą zmiany i zawłaszczali zasoby systemu na własne potrzeby.

Jednym z najpowszechniejszych zadań administratora systemu jest dodawanie, usuwanie i modyfikowanie kont użytkowników. Niezależnie od tego, co próbuje nam wmówić „Pieprzony Operator z Piekła Rodem", to system operacyjny istnieje dla użytkownika. Prawidłowe tworzenie i obsługa kont użytkowników jest czymś absolutnie niezbędnym. W tym rozdziale opisywać będziemy tworzenie, dodawanie i edycję kont użytkowników, sposoby na przyznawanie grupom użytkowników dostępu do różnych części systemu, prawidłowe korzystanie z hasła superużytkownika, a także sposoby pozwalające na unikanie korzystania z tego hasła.

Systemy dla pojedynczego użytkownika

Nawet jeżeli z systemu OpenBSD będziemy korzystać tylko my, to i tak konieczne jest utworzenie konta użytkownika, którego będziemy używać w codziennej pracy z systemem. Czytanie poczty, przeglądanie sieci WWW, tworzenie własnego oprogramowania, to zadania, które należy wykonywać w koncie zwykłego użytkownika, a nie w koncie

superużytkownika. Wykorzystywanie do takich zadań konta superużytkownika znacząco zwiększa ryzyko uszkodzenia systemu i może tworzyć luki w bezpieczeństwie systemu. Nieostrożne działanie może wywołać niestabilność całego systemu, choć ta sama operacja wykonana w koncie zwykłego użytkownika skończy się tylko komunikatem o odmowie dostępu (ang. *permission denied*).

Jeżeli włamywaczowi uda się skorzystać z konta użytkownika, będzie on mógł spowodować uszkodzenia jedynie w zakresie uprawnień użytkownika. Jeżeli to konto umożliwi użytkownikowi jedynie obsługę poczty i zakładek stron WWW, to możemy mieć kłopoty jedynie z tymi elementami. Jeżeli jednak włamywacz przejąłby konto superużytkownika, to miałby możliwość dokonania nieograniczonych zniszczeń, w efekcie musielibyśmy korzystać z przygotowanych wcześniej kopii bezpieczeństwa. Używanie na co dzień konta zwykłego użytkownika oznacza, że moglibyśmy podjąć dodatkowe kroki mające na celu zabezpieczenie konta superużytkownika. Jeżeli operację tę zaplanujemy właściwie, to możemy zupełnie wyeliminować konieczność korzystania z konta superużytkownika, tworząc w systemie dodatkową warstwę zabezpieczeń.

W skrócie, każda operacja powinna być wykonywana przy udziale minimalnych, wymaganych uprawnień. Jeżeli do wykonania jakiegoś zadania nie są konieczne uprawnienia superużytkownika, to lepiej ich nie używać. Właśnie dlatego serwer WWW w OpenBSD działa jako osobny użytkownik, a nie korzysta z konta superużytkownika. W ten sposób system chroniony jest nie tylko przed włamywaczami, ale również przed ewentualnymi błędami ukrytymi w programie.

Systemy operacyjne traktujące każdego użytkownika na równi z superużytkownikiem borykają się z całą masą problemów: efektywnością działania wirusów, nieoczekiwanymi błędami w konfiguracji. Odpowiedzialnością za większość załamań systemu można obarczyć taki właśnie sposób traktowania użytkowników. OpenBSD jest być może najbezpieczniejszym systemem operacyjnym na świecie, ale wszystkie wbudowane w niego systemy zabezpieczeń na nic się zdadzą, jeżeli będziemy korzystali z niewłaściwych praktyk administracyjnych.

Korzystanie z konta superużytkownika do wykonywania typowych zadań również jest złym nawykiem. W sytuacjach krytycznych, ludzie mają w zwyczaju wykonywać typowe zdania w sposób, do którego się przyzwyczaili. Jeżeli zwyczajowo, do zwykłej pracy na komputerze biurkowym, wykorzystujemy konto superużytkownika, to gdy kiedyś przyjdzie nam pracować w systemie produkcyjnym, będziemy musieli zmuszać się, aby wykonywać te zadania w sposób właściwy. Tego rodzaju nieprawidłowości są główną przyczyną wyłomów w bezpieczeństwie systemów. Na moim komputerze biurkowym, którego używam tylko i wyłącznie ja, wszystkie prace wykonuję jako zwykły użytkownik — właśnie po to, żeby wyrobić w sobie dobre nawyki administratora systemu.

To wszystko powinno być wystarczającym powodem, żeby w codziennej pracy używać wyłącznie kont zwykłych użytkowników.

Dodawanie użytkowników

OpenBSD udostępnia wiele standardowych w Uniksie programów do zarządzania hasłami, takich jak *passwd*(8) i *vipw*(8). W systemie dostępny jest również wygodny i interaktywny program do dodawania nowych użytkowników — *adduser*(8). Na początek opiszemy właśnie ten program, a następnie zajmiemy się innymi narzędziami przeznaczonymi do bardziej zaawansowanych zastosowań.

Interaktywne dodawanie użytkowników

Jeżeli program *adduser*(8) uruchomiony zostanie w wierszu poleceń, bez podawania żadnych opcji, to natychmiast udostępnia on użytkownikowi własną, interaktywną powłokę. Przy pierwszym uruchomieniu, program zada kilka pytań, w celu ustalenia jego domyślnych ustawień. Zostaną one zapisane, ale bez obawy, domyślne ustawienia można później zmienić. Aby uruchomić program *adduser*(8), konieczne jest posiadanie uprawnień superużytkownika.

```
# adduser
Use option ``-silent'' if you don't want to see all warnings and questions.
Reading /etc/shells❶
Check /etc/master.passwd❷
Check /etc/group❸
Ok, let's go.
Don't worry about mistakes. I will give you the chance later to correct any input.
```

Przy każdym uruchomieniu, program *adduser* sprawdza pliki konfiguracji użytkowników. Najważniejsze informacje przechowywane są w plikach ❶*/etc/shells*, ❷*/etc/master.passwd* i ❸*/etc/group*. Gdy program upewni się, że pliki konfiguracji nie są uszkodzone, pozwoli wprowadzić nazwę użytkownika, dla którego chcemy utworzyć konto. W nawiasach kwadratowych podane są znaki, jakie mogą znaleźć się w nazwie użytkownika systemu OpenBSD — są to litery, cyfry oraz znak podkreślenia i myślnika.

```
Enter username [a-z0-9_-]: grzesiek
```

Następnie należy wprowadzić pełną nazwę użytkownika.

```
Enter full name []: Grzegorz M.
```

Kolejnym krokiem jest wybranie powłoki dla użytkownika. Lista dostępnych powłok pobierana jest z pliku */etc/shells* i uzupełniana jest o opcję *nologin*. Domyślna powłoka podawana jest w nawiasach kwadratowych.

```
Enter shell csh ksh nologin sh [csh]: csh
```

Teraz możemy wybrać numer identyfikacyjny użytkownika (uid — ang. *User id*). Domyślnie OpenBSD numeruje użytkowników, zaczynając od wartości 1000 i używając pierwszego wolnego numeru. Można zmienić ten proponowany numer, ale najczęściej nie jest to konieczne.

```
Uid [1000]:
Login group grzesiek [grzesiek]:
```

Domyślnie, każdy użytkownik przypisywany jest do grupy o takiej samej nazwie jak nazwa użytkownika. Jeżeli zachodzi taka potrzeba, można konto użytkownika przypisać do innej grupy. Jeżeli w systemie przygotowane są inne grupy użytkowników, i chcielibyśmy przydzielić użytkownika do jednej z nich, można tutaj wpisać jej nazwę. Jeżeli danemu użytkownikowi chcemy umożliwić korzystanie z hasła superużytkownika, to należy przydzielić go do grupy *wheel*.

```
Login group is ``grzesiek''. Invite grzesiek into other groups: guest no [no]: wheel
```

Program *adduser*(8) poprosi teraz o podanie wstępnego hasła użytkownika i wypisze wszystkie wprowadzone dane, umożliwiając ich końcową kontrolę. Każde wprowadzone wcześniej pole, wyświetlane jest ponownie, do ostatecznego zaakceptowania.

```
Enter password []:
Enter password again []:
Name:     grzesiek
Password: ****
Fullname: Grzegorz M.
Uid:      1001
Gid:      1001 (grzesiek)
Groups:   grzesiek
HOME:     /home/grzesiek
Shell:    /bin/csh
OK? (y/n) [y]:
```

W tym miejscu, naciskając klawisz *n,* możemy anulować całą operację. Jeżeli jednak wszystkie dane konta wyglądają prawidłowo, to naciśnięciem klawisza *y* utworzymy nowe konto użytkownika.

```
Added user ``grzesiek''
Copy files from /etc/skel to /home/grzesiek
```

Na zakończenie, program *adduser* zapyta, czy chcielibyśmy utworzyć kolejne konto użytkownika. Możemy się na to zdecydować, ale nie ma przymusu.

```
Add another user? (y/n) [y]: n
Goodbye!
#
```

Gratulacje! Właśnie dodaliśmy nowego użytkownika do systemu. Skoro już wiemy, jak działa ten proces, przyjrzyjmy się teraz sposobom na takie skonfigurowanie programu *adduser*(8), aby podawał potrzebne nam wartości domyślne.

/etc/adduser.conf

Przy pierwszym uruchomieniu programu *adduser*(8) z naszych odpowiedzi na zadane pytania zbuduje on swój plik konfiguracyjny */etc/adduser.conf*. Wartości przypisywane w tym pliku kontrolują zachowania programu. Jeżeli jakiejś zmiennej można przypisać wiele różnych wartości, to są one podawane w nawiasach. Poniżej przedstawiamy listę standardowych zmiennych, jakie można spotkać w pliku */etc/adduser.conf*. Pełną listę zmiennych, jakie można wpisać do pliku */etc/adduser.conf* uzyskać można jedynie poprzez analizę samego skryptu *adduser*.

verbose = 1

Zmienna *verbose* ustala ilość szczegółów, jaką podaje program *adduser* w czasie swojego działania. Jeżeli przypiszemy jej wartość `0`, to program będzie wypisywał jedynie szczątkowe informacje. Będzie on zakładał, że użytkownik wie, jakie operacje są wykonywane (na przykład, sprawdzanie plików użytkowników w katalogu */etc*) i nie chce być informowany o nieistotnych szczegółach. Domyślną wartością jest `1`. Jeżeli chcielibyśmy wyszukiwać błędy w skrypcie, to przypisanie tej zmiennej wartości `2` spowoduje, że skrypt będzie wypisywał wszystkie informacje o podejmowanych operacjach. Osobiście, bez namysłu wpisuję tutaj wartość `0`.

encryptionmethod = "blowfish"

OpenBSD stosuje do kodowania haseł całą gamę różnych schematów kryptograficznych. Standardowo jednak stosowana jest metoda *blowfish*. Jeżeli hasła użytkowników mają być współdzielone z innymi uniksowymi systemami operacyjnymi, to wpisanie tu wartości `old`, spowoduje zastosowanie metody DES.

dotdir = "/etc/skel"

Do każdego konta użytkownika kopiowany jest zbiór plików konfiguracyjnych. Można stosować pliki udostępniane przez system OpenBSD w katalogu */etc/skel*, albo przygotować własny zestaw plików odpowiedni dla własnego środowiska. Do katalogów domowych kopiowane są wszystkie pliki z podanego tu katalogu, dlatego można go wykorzystać do rozprowadzania innych dowolnych plików. Należy się jednak upewnić, że zwykli użytkownicy nie mogą dodawać do podanego tutaj katalogu żadnych plików.

send_message = "no"

W wielu systemach operacyjnych nowi użytkownicy automatycznie otrzymują pocztą (e-mail) wiadomość z powitaniem lub pierwszymi instrukcjami. OpenBSD domyślnie tego nie robi. Jeżeli jednak tej zmiennej przypisana zostanie pełna ścieżka do pliku, to każdy nowy użytkownik dostanie wiadomość z zawartością tego pliku. Domyślna wartość `no` powoduje, że takie wiadomości nie są wysyłane. Zespół OpenBSD przygotował co prawda domyślną wiadomość powitalną dla użytkowników (znajduje się ona w pliku */etc/adduser.message*), ale nic nie stoi na przeszkodzie, żeby przygotować własną.

W tworzonej wiadomości można wykorzystać zmienne `$username` i `$fullname`, i w ten sposób nieco spersonalizować treść wiadomości. Osoby znające język Perl mogą dodawać kolejne zmienne, poprzez edytowanie pliku */usr/sbin/adduser*. Tak naprawdę, to utworzenie własnej wiadomości powitalnej jest zdecydowanie lepszym rozwiązaniem, niż stosowanie króciutkiej wiadomości domyślnej. Osobiście stosuję wiadomość zapisaną w pliku */etc/adduser.message.local*, wyglądającą mniej więcej tak:

```
$fullname

Witamy w Firma X.

Wszelka pomoc dostępna jest pod numerem telefonu 0800-555-555
albo na stronie http://pomoc.firmax.com.
```

```
Korzystanie z tego konta musi odbywać się zgodnie z zasadami
określonymi w dokumencie dostępnym na stronie
http://www.firmax.com/zasady.html lub w systemie, w katalogu
/usr/local/share/firma/zasady.

Dziękujemy za skorzystanie z naszych usług. Mamy nadzieję
na owocną współpracę.

Zespół administratorów Firmy X.
```

logfile = "/var/log/adduser"

W podanym tutaj pliku program *adduser* będzie zapisywał historię wykonywanych operacji.

home = "/home"

Ta zmienna określa katalog, w którym umieszczane są katalogi domowe użytkowników. Jest to pierwsza rzecz, jaką zajmuję się w każdym systemie OpenBSD. Jeżeli nie zostanie utworzona partycja */home*, to domyślnie katalog */home* zostanie utworzony na partycji podstawowej. Nie jest to dobre rozwiązanie z przynajmniej kilku powodów. Pierwszym z nich jest fakt, że rozmiar partycji podstawowej ograniczony jest do 8 GB, co znacznie ogranicza ilość danych, jakie mogą przechowywać użytkownicy systemu.

Jeżeli w systemie przewidywane jest utworzenie wielu kont użytkowników (na przykład w serwerze WWW), to z całą pewnością konieczne jest utworzenie osobnej partycji */home*, tak aby możliwe było montowanie jej z odpowiednio dobranymi uprawnieniami. Jeżeli jednak w systemie mają istnieć tylko konta administratorów, to można umieszczać je w katalogu */usr/home*, a w katalogu */home* umieszczać tylko dowiązania symboliczne. Oba rozwiązania są prawidłowe, ale trzeba wiedzieć, że takie możliwości istnieją.

path = ('/bin','/usr/bin','/usr/local/bin')

W tej zmiennej zapisywana jest lista katalogów, które mogą przechowywać powłoki uprawnione do działania w systemie. Domyślne ustawienia sprawdzają się w większości systemów, ale jeżeli dodatkowe powłoki będą instalowane w innych katalogach, to konieczne będzie uzupełnienie tej listy.

shellpref = ('csh','sh','ksh','nologin')

Jest to lista uprawnionych powłok. W czasie tworzenia nowego konta użytkownika, program *adduser* pozwoli wybrać jedną z powłok wymienionych w tej liście.

defaultshell = "csh"

Określa domyślną powłokę użytkownika. Domyślną powłoką może być dowolna powłoka spośród wymienionych w zmiennej *shellpref*.

defaultgroup = USER

Określa pierwszą grupę, której członkiem stanie się użytkownik. Tradycyjnie, systemy BSD przydzielają każdego użytkownika do grupy o takiej samej nazwie jak nazwa użytkownika. Na przykład, nasz użytkownik *grzesiek* automatycznie staje się członkiem grupy o nazwie *grzesiek*, która tworzona jest wyłącznie dla tego użytkownika. Jeżeli chcielibyśmy, żeby wszyscy użytkownicy systemu byli członkami jednaj grupy, na przykład, *studenci* albo *klienci*, to można w tym celu zmienić wartość przypisywaną tej zmiennej. Użytkownika można później ręcznie przypisywać do innych grup, ale tutaj definiowana jest jego grupa podstawowa.

uid_start = 1000
uid_end = 2147483647

Te zmienne określają zakres prawidłowych numerów identyfikacyjnych użytkowników (uid). Domyślne ustawienia sprawdzają się w większości przypadków, a ich zmiana może być konieczna tylko przy współpracy z innymi systemami uniksowymi.

Nieinteraktywne dodawanie użytkowników

Czasami jednak konieczne jest dodanie nowego użytkownika za pomocą jednego, długiego polecenia. Taki sposób często jest praktykowany, jeżeli w systemie są uruchamiane skrypty lub działają zadania *cron*, dodające użytkowników w równych odstępach czasu. Lubują się w nim również osoby łatwo zapamiętujące długie polecenia z wieloma opcjami. Tego rodzaju działania umożliwia zastosowanie opcji `-batch` programu *adduser*. Gdy program działa w tym trybie, pobiera jeszcze cztery parametry: nazwę użytkownika, nazwę grupy, pełną nazwę użytkownika i hasło w formie zaszyfrowanej, co wygląda mniej więcej tak:

```
# adduser -batch krzys wheel 'Krzysztof L.' loser1
```

Powyższym poleceniem tworzymy nowe konto dla Krzysztofa, przydzielamy go do grupy *wheel* i nadajemy mu hasło, które po zaszyfrowaniu tworzy tekst *loser1*.

Hasła i tryb wsadowy

Jeżeli dokładniej przyjrzymy się poprzedniemu przykładowi, to dojdziemy do wniosku, że właśnie utworzyliśmy konto, do którego nie znamy hasła! Jak zapewne pamiętamy, żaden z nowoczesnych systemów uniksowych nie przechowuje już haseł użytkowników w formie zwykłego tekstu, ale zapisuje je w formie zaszyfrowanej. Jeżeli weźmiemy zwykłe hasło i przeprowadzimy na nim pewne „straszliwe” operacje matematyczne, to powstanie w ten sposób szyfrowany skrót hasła (ang. *hash*). Wykorzystując tę metodę, system przetwarza wszystkie podane mu hasła i zapisuje je w pliku */etc/master.passwd*. Później, przy próbie zalogowania do systemu, pobierane jest hasło, generowany jest z niego zaszyfrowany skrót, a następnie porównywany jest on ze skrótem zapisanym w pliku. Jeżeli obie wartości będą identyczne, to hasło uznawane jest za poprawne i system pozwala na zalogowanie.

W powyższym przykładzie tworzone jest konto, w którym *loser1* jest wartością zaszyfrowanego skrótu hasła, a nie samego hasła! Tak naprawdę, to nie jest to nawet prawidłowa wartość skrótu, wobec czego żadne hasło nie będzie z nim zgodne. Większość z nas nie jest w stanie samodzielnie wyliczyć zaszyfrowanego skrótu algorytmem Blowfish i musi posiłkować się wcześniej przygotowanymi zaszyfrowanymi hasłami. Alternatywą jest wprowadzanie w wierszu poleceń niezaszyfrowanych haseł i zlecenie programowi *adduser* przeprowadzenie wszelkich wyliczeń lub ewentualne utworzenie konta bez podawania hasła.

Utworzenie takiego właśnie konta jest najprawdopodobniej najprostszym wyjściem. Co prawda konto będzie wyłączone do czasu wprowadzenia w jego konfiguracji prawidłowego hasła, ale jest to akceptowalne wyjście, jeżeli konto ma być stosowane do uruchamiania demonów i usług. Aby utworzyć konto bez hasła, wystarczy w trybie wsadowym nie podawać hasła wśród parametrów programu *adduser*.

```
# adduser -batch krzys wheel 'Krzysztof L.'
```

Jeżeli w wierszu poleceń chcemy wprowadzić niezaszyfrowane hasło, to możemy w tym celu skorzystać z opcji `-unencrypted`. Należy pamiętać o tym, że opcja ta musi być umieszczona jeszcze przed opcją `-batch`! Na przykład, jeżeli naprawdę chcielibyśmy, aby Krzysztof rzeczywiście miał konto z hasłem *loser1*, to należałoby wpisać:

```
# adduser -unencrypted -batch krzys wheel 'Krzysztof K.' loser1
```

Teraz użytkownik będzie mógł korzystać z konta, wprowadzając hasło `loser1`[1]. Tego sposobu podawania hasła można używać wewnątrz skryptów albo tylko wtedy, gdy mamy absolutną pewność, że nikt inny nie zagląda nam przez ramię.

Generowanie haseł zaszyfrowanych

Jeżeli powyższej metody używamy wewnątrz skryptu, to najprawdopodobniej chcielibyśmy wykorzystać w niej przygotowane wcześniej hasła zaszyfrowane. Przygotowanie takich haseł umożliwia program *encrypt*(1). Domyślnie, zaraz po uruchomieniu, program wyświetla tylko pusty wiersz, w którym możemy wpisać słowo do zaszyfrowania. Po podaniu słowa, program zwróci nam je zaszyfrowane algorytmem Blowfish. Podając kolejne słowa, będziemy otrzymywać ich zaszyfrowane wersje. Taką zabawę w szyfrowanie słów można zakończyć wciskając klawisze *Ctrl+C*.

```
# encrypt
loser1
$2a$06$RdxEtB0DNJ6MY6j77m/Bu.JYydNnErTo2cAV0InHg5gkCK1JrbBC
^C
#
```

Jeżeli potrzebne nam jest tylko jedno hasło albo używamy programu w sposób interaktywny, to należałoby użyć opcji `-p`. W ten sposób uzyskujemy znak zachęty, w którym znaki wpisywanego hasła nie są wypisywane na ekranie.

[1] Przypominam, że to hasło jest wyjątkowo paskudne, z naprawdę ogromnej ilości powodów. Osoby co nieco znające się na zabezpieczeniach z pewnością wiedzą, o co chodzi.

```
# encrypt -p
Enter string:
$2a$06$RdxEtB0DNJ6MY6j77m/Bu.JYydNnErTo2cAV0InHg5gkCK1JrbBC
#
```

Podane wyżej trzy możliwości powinny pozwolić każdemu na dowolne podawanie haseł w trybie wsadowym polecenia `adduser`.

Inne opcje trybu wsadowego polecenia adduser

Uruchamiając program *adduser*(8) w trybie wsadowym, możemy skorzystać z kilku innych opcji zmieniających domyślną konfigurację programu. Bardzo często, w jeden sposób zakładam konta administratorów, a w inny sposób — konta użytkowników, dlatego w obu przypadkach korzystam z odmiennych narzędzi. Konta administratorów najczęściej tworzę w trybie interaktywnym — w końcu nie ma ich zbyt wiele. Później już ktoś inny rutynowo tworzy konta użytkowników, korzystając z napisanego przeze mnie skryptu. Pełną listę opcji programu *adduser*(8) można znaleźć na stronie podręcznika opisującej ten program. Tutaj podam tylko te, które osobiście uważam za istotne.

Wszystkie podawane tutaj opcje, w wierszu poleceń muszą znaleźć się przed opcją `-batch`. Program *adduser* zakłada, że wszystko, co znajduje się za tą opcją, to dane nowego konta.

Opcja `-noconfig` informuje program *adduser*, że nie ma on odczytywać informacji z pliku */etc/adduser.conf*. Stosowanie tej opcji wewnątrz skryptów jest wygodnym sposobem na to, żeby ustawienia przygotowane dla administratorów nie „wyciekły” do kont zwykłych użytkowników.

Opcja `-dotdir` określa niestandardowy katalog, w którym przechowywane są ukryte pliki użytkowników. Wszystkie pliki znajdujące się w tym katalogu zostaną skopiowane do katalogu domowego nowego użytkownika.

Opcja `-home` przenosi informację, w którym katalogu należy umieścić katalog domowy nowego użytkownika. Nie należy podawać tutaj samej nazwy katalogu, ale raczej katalog, w którym utworzony zostanie katalog domowy użytkownika. Na przykład, jeżeli wszyscy klienci korzystający z serwera WWW posiadają swoje katalogi domowe na partycji */www*, to w trybie wsadowym programu *adduser* należy wpisać opcję `-home /www`.

Ograniczenia kont

Z kontem użytkownika wiążą się następujące ograniczenia:

- Nazwy użytkowników mogą składać się wyłącznie z małych liter, cyfr, myślników i znaków podkreślenia[2].

[2] Tak naprawdę, to możliwe jest tworzenie nazw użytkowników składających się z dowolnych znaków, a nawet tworzenie wielu identycznych nazw użytkowników. Jednak takie praktyki powodują innego rodzaju problemy, dlatego o ile nie jesteśmy całkowicie pewni, że możemy bez problemu zmierzyć się z konsekwencjami takiego działania, lepiej nie przekraczajmy domyślnych ograniczeń.

- Pełna nazwa użytkownika nie może zawierać w sobie dwukropka (:).
- Powłoka, z której korzysta użytkownik musi być ujęta w pliku */etc/shells*.

Usuwanie kont użytkowników

Usuwanie kont użytkowników w odpowiednim momencie jest częścią każdej polityki bezpieczeństwa. Konto można usunąć poleceniem `rmuser`(8). Po uruchomieniu poprosi ono o potwierdzenie chęci usunięcia konta o podanej nazwie i zadecydowanie, czy usunięty ma być również katalog domowy użytkownika. Polecenie `rmuser` usunie również wszystkie zadania *cron* należące do użytkownika, a także jego katalog pocztowy. Na przykład, jeżeli Krzysztof nie będzie już więcej korzystał z tego systemu, można usunąć jego konto w następujący sposób:

```
# rmuser krzys
Matching password entry:

krzys:*:1002:1002::0:0:Krzysztof L.:/klienci/krzys:/usr/local/bin/tcsh

Is this the entry you wish to remove? y
Remove user's home directory (/klienci/krzys)? y
Updating password file, updating databases, done.
Updating group file:Removing group krzys -- personal group is empty, done.
Removing user's home directory (/klienci/krzys): done.
#
```

Edycja danych użytkowników

System OpenBSD udostępnia narzędzie `vipw`(8), które pozwala administratorowi na bezpośrednią edycję pliku */etc/master.passwd*. Jednak w większości przypadków, program *chpass*(1) wykona wszystkie konieczne operacje w znacznie przyjaźniejszy sposób. Jedyną koniecznością użycia programu *vipw*(8) jest wykrycie uszkodzeń w pliku haseł.

Każdy użytkownik korzystający z powłoki może zmienić swoje dane, korzystając z programu *chpass*(1). Często jednak konieczne jest odebranie użytkownikom możliwości korzystania z tego programu, ponieważ wśród wielu swoich opcji, umożliwia on zmianę *zaszyfrowanego* hasła użytkownika. Większość użytkowników nie jest przygotowana do podawania haseł w tej postaci; widziałem już kilka osób, o których sądziłem, że **powinny wiedzieć**, próbujących wpisywać nowe hasło w polu przeznaczonym na hasła zaszyfrowane. W ten sposób użytkownik blokuje swoje konto do czasu, aż administrator skoryguje nieprawidłowe hasło. Można co prawda zakładać, że niedoświadczony użytkownik przestraszy się, gdy zobaczy tak długi ciąg liter i cyfr, i zdecyduje się go nie zmieniać, ale w praktyce okazuje się, że takie „drobnostki" nie odstraszają użytkowników. Poza tym, widział ktoś użytkownika, który cofnąłby się przed ponownym wprowadzaniem zmian do hasła? Mimo że program *chpass*(1) pozwala na zmianę tak przydatnych informacji jak numer telefonu użytkownika, czy umiejscowienie biura, to niestety często zablokowanie korzystania z niego okazuje się koniecznością.

Superużytkownik może zmieniać dane kont innych użytkowników, uruchamiając polecenie `chpass nazwa_użytkownika`. Uruchamiany jest w ten sposób edytor tekstowy, pozwalający na edycję danych o koncie zapisanych w pliku */etc/master.passwd*. Na przykład, jeżeli jako superużytkownik uruchomię polecenie `chpass krzys`, to na ekranie pojawi się:

```
Changing user database information for krzys.
Login: krzys
Encrypted password:$2a$06$RdxEtB0DNJ6MY6j77m/Bu.JYydNnErTo2cAV0InHg5gkCK1JrbBC
Uid [#]: 1002
Gid [# or name]: 1002
Change [month day year]:
Expire [month day year]:
Class:
Home directory: /home/krzys
Shell: /usr/local/bin/tcsh
Full Name: Krzysztof L.
Office Location:
Office Phone:
Home Phone:
```

Wszystkie wprowadzone w tym miejscu zmiany zostaną prawidłowo odzwierciedlone w plikach */etc/master.passwd* i */etc/passwd*. Polecenie `chpass`(1) wprowadza zmiany jedynie w tych dwóch plikach. To oznacza, że jeżeli za pomocą programu *chpass* zmienimy w danych konta lokalizację katalogu domowego użytkownika, to będziemy musieli później ręcznie przenieść ten katalog we właściwe miejsce. Jeżeli tego nie zrobimy, to na użytkownika po zalogowaniu może czekać niemiła niespodzianka!

Ostrzeżenia

W systemach OpenBSD plik */etc/passwd* jest automatycznie generowany programem *pwd_mkdb*(8) na podstawie wpisów w pliku */etc/master.passwd*. Narzędzia w rodzaju `chpass`(1) i `vipw`(8) wykonują tę operację w sposób automatyczny. Osoby przyzwyczajone do pracy z innymi systemami uniksowymi, pozwalającymi na bezpośrednie wprowadzanie zmian w pliku */etc/passwd* będą musiały się nieco powściągnąć. Wiąże się to nie tylko ze znacznym prawdopodobieństwem popełnienia błędu w czasie edycji, ale również z tym, że wszystkie wprowadzone w ten sposób zmiany zostaną anulowane, gdy tylko ktoś zmieni swoje informacje o użytkowniku za pomocą standardowych narzędzi.

Grupy użytkowników

Unix gromadzi użytkowników wewnątrz tak zwanych *grup*, z których każda gromadzi użytkowników wykonujących podobne zadania. Administrator systemu może zdefiniować grupę o nazwie *www*, przydzielić do niej osoby zajmujące się tworzeniem i edytowaniem stron WWW i przyznać tej grupie uprawnienia do odczytu i zapisu plików związanych ze stronami WWW. Może też utworzyć grupę o nazwie *e-mail*, dodać do niej osoby zajmujące się administrowaniem pocztą i nadać jej uprawnienia do modyfikowania plików związanych z obsługą poczty. Tego rodzaju wykorzystanie grup użytkowników to bardzo wydajna, choć często lekceważona metoda administrowania systemem.

W jakich grupach jestem?

Każdy użytkownik może sprawdzić, do jakich grup został przydzielony. Wystarczy, że wpisze polecenie *id*(1). Polecenie to wypisuje informacje o tym, jaki użytkownik jest aktualnie zalogowany i do jakich grup on należy. Poza tym, wypisuje też liczbowe identyfikatory użytkownika (`uid`), grupy pierwotnej (`gid`) i każdej grupy (`groups`), do jakiej został przydzielony użytkownik.

```
# id
uid=1000(mwlucas) gid=1000(mwlucas) groups=1000(mwlucas), 0(wheel)
#
```

Osoby cieszące się przywilejem korzystania z hasła superużytkownika, mogą poleceniem `id`(1) sprawdzić, czy właśnie nie są zalogowani w ten sposób. Korzystając z terminala serwera X Window bardzo łatwo można zapomnieć, w którym z okien terminala wpisaliśmy hasło superużytkownika.

```
# id
uid=0(root) gid=0(wheel) groups=0(wheel), 2(kmem), 3(sys), 4(tty), 5(operator),
20(staff), 31(guest)
#
```

Jak widać, superużytkownik jest domyślnym członkiem kilku grup. Polecenie `id`(1) posiada kilka opcji, ale służą one raczej ograniczaniu, niż rozszerzaniu ilości podawanych danych. Na przykład, jeżeli chcielibyśmy zobaczyć jedynie nazwy grup, których jesteśmy członkiem, możemy wpisać polecenie `id -Gn`. Dane, jakie zobaczymy doskonale nadają się do użycia w skrypcie, a większość osób i tak woli szybko przejrzeć wypisane pełne dane niż zapamiętywać opcje, jakie należy wpisać.

Polecenie `id`(1) pobiera dane z pliku */etc/group*.

/etc/group

Plik */etc/group* przechowuje większość informacji o grupach użytkowników. Składnia tego pliku jest wyjątkowo prosta, ale w OpenBSD i tak dostępnych jest kilka narzędzi wiersza poleceń, pozwalających na zarządzanie grupami. Osobiście uważam, że składnia pliku */etc/group* jest na tyle przejrzysta, że można obyć się bez pomocy tych narzędzi. Osobom zainteresowanym stosowaniem tych narzędzi podaję ich nazwy — `groupadd`(8), `groupdel`(8), `groupinfo`(8) i `groupmod`(8). W większości przypadków ich użytkowanie jest równie łatwe jak edycja samego pliku */etc/group*. W każdym wierszu tego pliku znajdują się cztery pola rozdzielane dwukropkiem: nazwa grupy, hasło grupy, identyfikator i lista członków. Oto przykładowy wpis z tego pliku:

```
❶wheel:❷*:❸0:❹root,mwlucas,krzys
```

Nazwa grupy ❶ przedstawiana jest w formie czytelnej dla człowieka. W naszym przykładzie przedstawiliśmy grupę `wheel`. Nazwy grup mogą być zupełnie dowolne. Jeżeli mielibyśmy taką ochotę, to jedną z grup moglibyśmy nazwać na przykład *zajeczysmrod*. Dobrym zwyczajem jest jednak takie dobieranie nazw grup, aby odzwierciedlały

one ich przeznaczenie. Być może sami zapamiętalibyśmy, że grupa *zajeczysmrod* zajmuje się obsługą poczty, ale czy wiedzieliby o tym nasi współpracownicy? Nazwy grup muszą mieć pewne znaczenie.

W ❷ drugim polu zapisane zostało zaszyfrowane hasło grupy. Hasła grup zachęcały do stosowania złych praktyk przy zabezpieczeniach systemu, dlatego większość nowych systemów uniksowych ich nie stosuje. Sam system OpenBSD w ogóle nie korzysta z tych haseł, jednak niektóre ze starszych programów zakładają, że w pliku */etc/group* będzie znajdował się wpis z hasłem grupy. Z tego powodu, w systemie pole to jest stosowane, ale wypełniane jest tylko znakiem gwiazdki (*).

W ❸ trzecim polu przechowywany jest unikalny, liczbowy identyfikator grupy (`gid`). W wielu programach grupy identyfikowane są raczej za pomocą identyfikatorów, a nie nazw. Grupa *wheel* posiada identyfikator `0`.

Na końcu ❹ znajduje się, rozdzielana przecinkami, lista użytkowników przydzielonych do tej grupy. Członkami grupy `wheel` są użytkownicy `root`, `mwlucas` i `krzys`.

Grupa pierwotna

Gdy tworzony jest nowy użytkownik, system tworzy dla niego grupę o tej samej nazwie, co nazwa użytkownika, której jest on jedynym członkiem. Taka grupa nazywana jest *pierwotną grupą użytkownika*. Każdy użytkownik automatycznie staje się członkiem swojej pierwotnej grupy, co jest odzwierciedlane w pliku */etc/passwd*. Niektóre programy mogą zostać skonfigurowane w ten sposób, że rozróżniają uprawnienia użytkowników na podstawie ich grupy pierwotnej, a nie członkostwa w innych grupach.

Zmiana przynależności do grup

Jeżeli chcielibyśmy dodać użytkownika do grupy, to wystarczy dopisać jego nazwę użytkownika na końcu wiersza opisującego tę grupę. Na przykład, jeżeli chciałbym dodać użytkownika `grzesiek` do grupy `wheel`, to musiałbym dopisać go na końcu opisu grupy `wheel`[3]:

```
wheel:*:0:root,mwlucas,krzys,grzesiek
```

Tworzenie grup

Do utworzenia nowej grupy potrzebna będzie nam jej nazwa i numer identyfikacyjny. Z technicznego punktu widzenia, dla danej grupy nie potrzeba nawet przydzielać jej choć jednego członka. Niektóre programy uruchamiane są jako członkowie grupy, i wtedy system kontroluje ich poczynania w sposób podobny do kontroli użytkowników.

[3] Muszę zaznaczyć, że Grzesiek musiałby mnie zupełnie spić, żebym zdecydował się dodać go do grupy *wheel*. Ale to jest już decyzja administracyjna, a nie techniczna.

Tradycyjnie już grupy wypisywane są w kolejności numerów identyfikacyjnych. Identyfikator grupy może być dowolnym numerem z zakresu od 0 do 32 767, jednak numery poniżej 1000 zarezerwowane są dla celów administracji systemu. Najczęściej z tych numerów korzystają programy, które muszą działać jako członek dedykowanej grupy. Konta użytkowników wiązane są z grupami, których identyfikatory zaczynają się od wartości 1000. Niektóre grupy specjalne identyfikowane są numerami malejącymi od wartości 32 767.

Oczywiście można korzystać z zupełnie dowolnych numerów identyfikacyjnych, jednak stosowanie się do tych zasad znacznie ułatwi życie naszym współpracownikom i następcom.

Dodajmy teraz nową grupę. Ta przykładowa grupa przeznaczona będzie do obsługi programu bazodanowego, dlatego nazwiemy ją `db`. Tego rodzaju grupy celowo zaczynam numerować od wartości 5000 i kolejnym nadaję numery rosnące. Na koniec dodamy do grupy administratora bazy — Grześka.

```
db:*:5000:grzesiek
```

To wszystko!

Klasy użytkowników

Każdy z użytkowników OpenBSD posiada pewną klasę logowania, która określa ograniczenia, jakimi ten użytkownik objęty jest przy dostępie do zasobów systemu, zachowanie jego środowiska, a nawet rodzaj uwierzytelniania użytkowników z tej klasy. Zmieniając charakterystykę klasy, nakładane ograniczenia będą miały wpływ na wszystkich użytkowników z tej klasy. Klasy użytkowników zdefiniowane są w pliku */etc/login.conf*.

Klasę użytkownika można zmienić, wywołując polecenie `chpass` *`nazwa_użytkownika`* z uprawnieniami superużytkownika. W miejscu oznaczonym słowem `class` (można je zobaczyć w podrozdziale „Edycja danych użytkowników") należy wprowadzić nazwę klasy użytkownika.

Klasa domyślna

Za każdym razem, gdy tworzyć będziemy nowe konto poleceniem `adduser`(8), jego użytkownik automatycznie przydzielany jest do klasy „domyślnej". Najprostszym sposobem na łatwą obsługę klas użytkowników, jest takie skonfigurowanie systemu, aby klasa domyślna była najczęściej stosowaną klasą w tym systemie. Na przykład, jeżeli nasz komputer działa jako serwer poczty z garstką administratorów i setkami użytkowników, korzystających z usług pocztowych, to domyślną klasę należy skonfigurować w sposób odpowiadający najczęściej występującemu rodzajowi użytkownika — użytkownika poczty. Dla administratorów można już ręcznie zmieniać klasę na bardziej odpowiadającą ich wymaganiom. Jest to znacznie prostsze niż edycja klasy setek użytkowników.

Definicje klas

Każda definicja klasy składa się z zestawu przypisań wartości zmiennym. W momencie gdy użytkownik loguje się do systemu, program *login*(1) używa tych wartości do określenia limitów zasobów dostępnych dla użytkownika i konfiguracji środowiska. Każdy wpis w definicji klasy rozpoczyna się i kończy znakiem dwukropka (:), mimo że technicznie rzecz biorąc, każdy wpis mieści się w jednym wierszu. Znak lewego ukośnika (\) jest znacznikiem kontynuacji, informującym komputer, że powinien zignorować następujący po nim znak końca wiersza. Ludzie po prostu nie lubią czytać jednego wiersza tekstu o długości 500 znaków!

Standardowy plik */etc/login.conf* rozpoczyna się definicją klasy domyślnej (ang. *default*). Umożliwia ona typowemu użytkownikowi całkiem szeroki dostęp do systemu. Osobom uruchamiającym system OpenBSD na nowoczesnych komputerach wyposażonych w gigabajty pamięci RAM, te ograniczenia mogą wydawać się restrykcyjne, ale jeżeli ktoś uruchamia ten system na komputerze z procesorem Pentium 166, te same ustawienia dają użytkownikowi praktycznie nieograniczony dostęp do zasobów systemu. Jeżeli głównym problemem zaczynają się stawać użytkownicy pochłaniający wielkie ilości zasobów systemu, to edycja tych ustawień może znacząco poprawić sytuację. Oto przykładowy początek definicji klasy użytkownika:

```
default:\
        :path=/usr/bin /bin /usr/sbin /sbin /usr/X11R6/bin /usr/local/bin:\
        :umask=022:\
        :datasize-max=256M:\
...
```

W opisie klasy definiowanych jest znacznie więcej zmiennych, ale już ten wycinek powinien dać pewne wyobrażenie o strukturze pliku. Poprzez przydzielenie użytkownika do klasy konfigurującej jego środowisko zgodnie z naszymi wymaganiami, możemy całkowicie zmienić styl pracy tego użytkownika.

Niektóre ze zmiennych z pliku *login.conf* nie posiadają wartości; zmieniają one zachowanie konta jedynie swoją obecnością. Na przykład, zmienna *requirehome* zaczyna mieć wpływ na klasę, gdy tylko zostanie umieszczona w jej definicji.

```
:requierhome:\
```

Prawidłowe wartości zmiennych z pliku /etc/login.conf

Do zmiennych wymienianych w pliku *login.conf* można przypisywać jedną z poniższych wartości:

- pełną ścieżkę do pliku tekstowego,
- listę wartości rozdzielaną przecinkami,
- liczbę,

- listę ścieżek rozdzielanych spacjami; jeżeli pierwszym znakiem ścieżki jest znak tyldy (~), jest on zastępowany ścieżką katalogu domowego użytkownika,
- pełną ścieżkę do programu,
- rozmiar podawany w bajtach (domyślnie), kilobajtach (k) lub megabajtach (m),
- czas podawany w sekundach (domyślnie), minutach (m), godzinach (h), dniach (d), tygodniach (w) lub latach (y).

Oczywiście, niektóre ze zmiennych wymagają podania wartości o konkretnym typie. Ścieżka do katalogu domowego musi być pełną ścieżką, natomiast ilość pamięci, jaką użytkownik może wykorzystać — ścieżką być nie może. W większości przypadków właściwe wartości są dość oczywiste.

W wielu systemach BSD, aby zmiany wprowadzone do pliku */etc/login.conf* miały jakikolwiek wpływ na system, konieczne jest zastosowanie programu *cap_mkdb*(8) do utworzenia pliku bazy danych zawierającej wartości pobrane z tego pliku. W OpenBSD nie jest to konieczne, gdyż programy same odczytują dane z pliku */etc/login. conf*. Jeżeli jednak choć raz uruchomiony zostanie program *cap_mkdb*(8), to od tego momentu trzeba zacząć używać go na stałe albo usunąć plik bazy danych.

W OpenBSD, w domyślnym pliku */etc/login.conf* zdefiniowanych jest kilka różnych klas użytkowników. Przeglądając go, można poznać różne sposoby nakładania ograniczeń na użytkowników, stosowane w zależności od aktualnej sytuacji. Poniżej opiszemy zaledwie kilka najczęściej zmienianych pozycji.

Ograniczenia zasobów

Ograniczenia zasobów pozwalają kontrolować, z jakiej części systemu może w danym momencie korzystać użytkownik. Jeżeli do komputera zalogowanych jest kilkuset użytkowników, a jeden z nich będzie chciał skompilować 30 MB kodów źródłowych, to bardzo prawdopodobne jest, że zużyje on nieproporcjonalnie dużą część pamięci i czasu procesora. Wprowadzając ograniczenia ilości zasobów systemowych, jakie jeden użytkownik może wykorzystać na własne potrzeby, możemy spowodować, że system zacznie lepiej i szybciej odpowiadać na żądania mniej „zasobożernych" użytkowników. Każdej z klas użytkowników można przydzielić inny limit zasobów.

Limity zasobów bardzo często wiązane są z procesami. Jeżeli jednemu procesowi pozwolimy zużyć do 20 MB pamięci, a użytkownikowi pozwolimy uruchomić maksymalnie 20 procesów, to jeden użytkownik będzie mógł teoretycznie korzystać z 400 MB pamięci. Oto kilka najpopularniejszych zmiennych ograniczających zużycie zasobów, wpisywanych do pliku *login.conf*.

`coredumpsize` Maksymalny rozmiar dowolnego zrzutu pamięci.

`cputime` Maksymalna ilość czasu procesora przeznaczona dla jednego procesu.

`datasize` Maksymalna ilość pamięci, którą jeden proces może przejąć na swoje dane.

`filesize`	Maksymalny rozmiar dowolnego pliku.
`stacksize`	Maksymalna ilość pamięci stosu, jaką może wykorzystać pojedynczy proces.
`memoryuse`	Maksymalna ilość pamięci, jaką może zablokować jeden proces.
`maxproc`	Maksymalna liczba procesów, jaką może uruchomić jeden użytkownik.
`openfiles`	Maksymalna liczba plików otwartych przez jeden proces.

Aktualne i maksymalne ograniczenia zasobów

Mechanizm ograniczania zasobów pozwala zdefiniować limity ostrzegawcze (lub *aktualne*) i maksymalne. Limity aktualne (`-cur`) powodują jedynie wyświetlenie ostrzeżeń, które użytkownik może zignorować. Ten mechanizm sprawdza się doskonale w systemach, w których użytkownicy chętnie dzielą się z innymi dostępnymi zasobami. Limity maksymalne (`-max`) są wartościami nieprzekraczalnymi i bezwzględnie ograniczają działania użytkownika.

Aby określić wartość limitu aktualnego, za jego nazwą należy umieścić przyrostek `-cur`. Limity maksymalne definiowane są przyrostkami `-max`. Na przykład, aby ograniczyć liczbę uruchamianych przez użytkownika procesów do 60, ale wyświetlać ostrzeżenie, gdy wykorzysta już połowę tego limitu, można zastosować następujące wpisy:

```
:maxproc-cur=30:\
:maxproc-max=60:\
```

Jeżeli do nazwy, którejś z wartości nie zostanie dodany przyrostek `-cur` ani `-max`, to definiowane w ten sposób limity traktowane są jak limity maksymalne.

Domyślne ustawienia środowiska

W pliku */etc/login.conf* można zdefiniować również domyślne ustawienia środowiska. Jest to zwykle lepsze rozwiązanie niż konfigurowanie domyślnych ustawień użytkownika w plikach *.cshrc* lub *.profile*, ponieważ wszystkie wprowadzone w ten sposób modyfikacje będą uwzględniane już przy następnym logowaniu dowolnego użytkownika. Oto kilka typowych wartości konfigurujących środowisko.

`hushlogin`	Jeżeli ta wartość jest obecna, to w czasie logowania nie są podawane żadne informacje o systemie.
`ignorenologin`	Jeżeli ta wartość jest obecna, użytkownik może się zalogować, nawet jeżeli istnieje plik */etc/nologin*.
`nologin`	Jeżeli ta wartość jest obecna, użytkownik nie może zalogować się do systemu.
`path`	Domyślna ścieżka wyszukiwania programów.
`priority`	Domyślny priorytet procesów.

`requirehome`	Jeżeli ta wartość jest obecna, użytkownik, aby mógł się zalogować, musi posiadać prawidłowy katalog domowy.
`setenv`	Lista domyślnych zmiennych środowiskowych.
`shell`	Powłoka udostępniana użytkownikowi; zastępuje powłokę ustaloną w pliku */etc/passwd*.
`term`	Jeżeli domyślny terminal nie jest ustalany gdzie indziej, definiuje domyślny typ terminala.
`umask`	Domyślna wartość `umask`.
`welcome`	Plik zawierający powitalny tekst wypisywany po zalogowaniu użytkownika.

Opcje FTP

Użytkownikom FTP można zmienić katalog podstawowy na ich katalog domowy (operacja *chroot*), wprowadzając odpowiednie wpisy w pliku */etc/ftpchroot*. Jednak jeżeli w systemie mamy wielu użytkowników korzystających wyłącznie z FTP, to lepiej jest zdefiniować dla nich osobną klasę; na dłuższą metę jest to znacznie wygodniejsze rozwiązanie. Oto zmienne wpływające na działanie serwera FTP.

`ftpchroot`	Jeżeli ta zmienna jest zdefiniowana, dla użytkowników katalogiem głównym staje się ich katalog logowania (domyślne jest to katalog domowy).
`ftp-dir`	Pełna ścieżka do katalogu logowania użytkowników serwera FTP; umożliwia ona utworzenie dla tych użytkowników wspólnego katalogu.

Jeżeli zmieniamy użytkownikom w ten sposób katalog podstawowy, to należałoby ich o tym poinformować w powitalnej wiadomości (opisana została w punkcie „Domyślne ustawienia środowiska”).

Opcje kontrolujące hasła i logowanie

W pliku */etc/login.conf* możliwe jest kontrolowanie wielu operacji związanych z hasłami. W przeciwieństwie do konfiguracji środowiska, wiele z nich może być konfigurowanych wyłącznie w tym pliku. OpenBSD posiada również wiele metod pozwalających na kontrolowanie sposobu autoryzacji użytkowników. Oto kilka opcji konfigurujących nudny proces autoryzacji haseł.

localcipher

Definiuje metodę szyfrowania haseł. Domyślnie stosowany jest algorytm blowfish, ale można przypisać tej zmiennej wartość `old`, aby włączyć 56-bitowy algorytm DES, zgodny ze stosowanym w starszych wersjach Uniksa.

login-backoff

Określa, jak szybko użytkownik może ponowić próbę zalogowania. Po podanej tutaj liczbie prób, program logujący zaczyna powiększać odstępy czasu między kolejnymi wyświetleniami znaku zachęty.

passwordcheck

Przechowuje pełną ścieżkę do zewnętrznego programu kontrolującego ważność hasła. OpenBSD zakłada, że program ten będzie pobierał hasło poprzez standardowe wejście i zwracał 0, jeżeli hasło jest prawidłowe lub 1, jeżeli hasło jest niewłaściwe.

passwordtime

Określa czas życia hasła. Zmienna ta nadaje się do wymuszania na użytkownikach regularnych zmian haseł.

minpasswordlen

Określa minimalną długość hasła.

Metody autoryzacji

W pliku */etc/login.conf* możliwe jest również wybranie metody autoryzacji. W OpenBSD wykorzystywana jest metoda *BSD Authentication*, która działa nieco inaczej niż stosowana w kilku innych otwartych systemach uniksowych metoda PAM (ang. *Pluggable Authentication Modules*). W pliku */etc/login.conf* wystarczy określić pożądaną metodę autoryzacji, a OpenBSD od razu zacznie w ten sposób autoryzować użytkowników. Prościej już się nie da!

Zwykłe wybranie metody autoryzacji nie powoduje jeszcze jej skonfigurowania — przekazuje tylko do systemu informacje, że z tej metody ma korzystać. Na przykład, wybranie dla pewnej klasy użytkowników systemu autoryzacji Kerberos V nie przygotuje w magiczny sposób konfiguracji domeny Kerberos. Jeżeli dana metoda autoryzacji nie będzie dostępna, to konta korzystające z tej metody zostaną zablokowane.

Niektóre metody autoryzacji nie są zgodne z wszystkimi protokołami. W związku z tym, nie wszystkie metody będą współpracowały z każdym programem umożliwiającym logowanie. Na przykład, protokół *ssh* współpracuje z metodą *cryptocards*, ale nie będzie współpracował z metodą zmieniania haseł *lchpass*. Przy wyborze metody autoryzacji konieczne okazuje się przeglądanie ich stron podręcznika i wyszukiwanie w nich informacji o błędach i ewentualnych nieprawidłowych kombinacjach.

Niektóre z metod autoryzacji wymagają wpisania dodatkowych zmiennych do pliku *login.conf*. Zmienne te najczęściej opisywane są na stronie podręcznika poświęconej danej metodzie. Na przykład, wybranie metody autoryzacji Radius wymusza umieszczenie w pliku *login.conf* informacji, gdzie można znaleźć serwer Radius. W poniższej tabeli podawana jest również strona podręcznika, opisująca konfigurację danej metody autoryzacji. Oto metody autoryzacji użytkowników obsługiwane w systemie OpenBSD.

`krb4-or-pwd`	Najpierw wykonywana jest próba autoryzacji Kerberos IV, a następnie sprawdzany jest lokalny plik haseł. (*kerberos*(1))
`krb5-or-pwd`	Najpierw wykonywana jest próba autoryzacji Kerberos V, a następnie sprawdzany jest lokalny plik haseł. (*kerberos*(1))
`passwd`	Wykorzystywany jest lokalny plik haseł.
`krb4`	Wykorzystywany jest system Kerberos IV. (*kerberos*(1))
`krb5`	Wykorzystywany jest system Kerberos V. (*kerberos*(1))
`chpass`	Użytkownik nie jest logowany, ale wykonywana jest zmiana hasła systemu Kerberos, a jeżeli nie jest on dostępny, to zmieniane jest hasło w lokalnym pliku haseł. (*login_chpass*(8))
`lchpass`	Użytkownik nie jest logowany, ale wykonywana jest zmiana hasła w lokalnym pliku haseł. (*login_lchpass*(8))
`radius`	Używana jest metoda autoryzacji Radius. (*login_radius*(8))
`skey`	Używana jest metoda autoryzacji S/Key. (*skey*(1))
`activ`	Używana jest, oparta na tokenach, metoda autoryzacji ActivCard X9.9. (*login_activ*(8))
`snk`	Używana jest metoda autoryzacji Digital Pathways SecureNet Key. (*login_snk*(8))
`token`	Używana jest metoda autoryzacji Generic X9.9 Token. (*login_token*(8))

Używanie metod autoryzacji

Metody autoryzacji wybierane są w pliku *login.conf* poprzez przypisanie ich do zmiennej `auth` w postaci listy rozdzielanej przecinkami.

```
:auth=skey,passwd:\
```

Bardzo ciekawą rzeczą jest możliwość określania różnych metod autoryzacji, w zależności od tego, z jakiej usługi korzysta użytkownik. Jeżeli po słowie kluczowym `auth` umieścimy „nazwę usługi”, to podany dalej zbiór metod autoryzacji będzie wykorzystywany tylko w odniesieniu do tej konkretnej usługi. Na przykład, aby dla usługi FTP włączyć tylko autoryzację hasła, należy wpisać następujący wiersz:

```
:auth-ftp=passwd:\
```

Oto najczęściej wyszczególniane usługi:

`auth`	Domyślna metoda autoryzacji stosowana jest dla wszystkich usług, które nie zostały wyszczególnione.
`auth-ftp`	FTP.
`auth-ssh`	SSH.
`auth-su`	Autoryzacja `su`(1).

Na przykład, można pozwolić użytkownikom na logowanie się poprzez swój lokalny plik hasła lub metodę S/Key. Jeżeli jednak któryś z użytkowników będzie chciał skorzystać z polecenia `su`(1) do uzyskania praw superużytkownika, to będzie musiał być autoryzowany wyłącznie metodą S/Key.

```
:auth=passwd,skey:\
:auth-su=skey:\
```

Domyślnie, w pliku */etc/login.conf* wykorzystywany jest format *termcap*(5). Ma on wielkie możliwości i jest bardzo elastyczny, ale dla nowicjuszy może być bardzo dezorientujący — na ten temat napisano już wiele książek. Wpisy dotyczące autoryzacji, jakie znajdują się w domyślnej klasie, korzystają z rozszerzeń formatu *termcap*, ale można je łatwo zastąpić deklaracjami, jakich używamy w tym rozdziale. Poświęcenie czasu na naukę formatu *termcap*(5) znacząco podnosi umiejętności administratora, ale nie wchodzi już w zakres naszej książki.

Hasło superużytkownika

W Uniksie funkcjonuje model bezpieczeństwa typu „wszystko albo nic". Superużytkownik (root) może zrobić w systemie dosłownie *wszystko*, ale pozostali użytkownicy mogą wykonywać tylko to, na co pozwoli im superużytkownik. Właśnie taki podział odpowiedzialny jest za ogromną ilość włamań do systemów. Co więcej, tego rodzaju podejście do bezpieczeństwa powoduje wiele problemów z administracją systemem. Co prawda można tworzyć grupy użytkowników i przydzielać im prawa do obsługi różnego rodzaju plików, ale grupy nie sprawdzają się przy zadaniach administracyjnych. Tylko superużytkownik może dodawać nowych użytkowników, tylko on może konfigurować działanie sieci albo instalować oprogramowanie wpływające na cały system. Tymi zadaniami można by obdzielić kilka osób, ale każda z nich potrzebuje do tego uprawnień superużytkownika. Musimy więc zaufać innym, że wykonując własne zadania, nie będą sobie wzajemnie przeszkadzać albo posłużyć się innym narzędziem kontroli dostępu. Hasło superużytkownika można przekazywać jedynie zaufanym osobom. Wszyscy pozostali, do wykonania swoich zadań powinni używać programu *sudo*(8).

Używanie hasła superużytkownika

Polecenie *su*(1) pozwala użytkownikowi na zmianę tożsamości na innego użytkownika; oczywiście, o ile zna jego hasło. Korzystając z hasła Krzyśka, mógłbym dostać się do jego konta, tak jakbym był nim. Znajomość hasła Grześka pozwoliłaby mi stać się na chwilę Grześkiem. W końcu, znajomość hasła superużytkownika umożliwiłaby mi pełną kontrolę nad systemem.

Program *su* jest wyjątkowo prosty w użyciu. Wystarczy wpisać polecenie `su`, a system poprosi o podanie hasła. Po wprowadzeniu hasła superużytkownika uzyskamy dostęp do jego powłoki!

```
# su
Password:
#
```

Należy pamiętać o tym, że polecenie `su` przekazuje nam powłokę użytkownika, na którego konto się przełączamy. Czasami może nam to nie odpowiadać — jeżeli pracujemy w systemie z wieloma administratorami, to zapewne komuś nie będzie odpowiadała powłoka przypisana superużytkownikowi. Nie należy jednak jej zmieniać, chyba że znamy wszystkie implikacje, jakie może wywołać taka operacja. Alternatywą jest wykorzystanie opcji `-m` programu *su*, która pozwala na zachowanie swojej aktualnej powłoki wraz z wszystkimi zmiennymi środowiskowymi. Na przykład, ja bardzo lubię pracę w powłoce *tcsh*, ale powłoka superużytkownika w OpenBSD to staromodna *csh*. Jeżeli użyję polecenia `su`, to otrzymam powłokę *csh*, jeżeli jednak wpiszę polecenie `su -m`, to nadal będę mógł pracować w mojej ulubionej powłoce *tcsh*. Stosując polecenie `su -m` należy się upewnić, że w naszej powłoce nie znajdują się żadne śmieci, które mogłyby zakłócić pracę programów uruchamianych z konta superużytkownika. Niestandardowe wartości zmiennych `$PATH` i `$LD_LIBRARY_PATH` w połączeniu z poleceniem `su -m` mogą wybitnie utrudniać pracę w systemie.

Kto może używać hasła superużytkownika?

Z hasła superużytkownika mogą korzystać wyłącznie osoby przydzielone do grupy *wheel*. Użytkownik, który nie został przydzielony do tej grupy, nie może skorzystać z tego hasła, nawet jeżeli je zna. Na przykład złóżmy, że stałem się wyjątkowo leniwy i hasło superużytkownika zapisałem na karteczce przylepionej do monitora. Grzesiek przechodził akurat obok, zobaczył to hasło i postanowił spróbować ze swojego konta dostać się do konta superużytkownika, mimo że nie jest członkiem grupy *wheel*.

```
# su
Password:
you are not in the group wheel
Sorry
#
```

Co więcej, ta próba zalogowania zostanie odnotowana w pliku */var/log/authlog*.

```
Jul  1 16:10:15 openbsd su: BAD SU grzesiek to root on /dev/ttyp1
```

Jako odpowiedzialny administrator powinienem codziennie przeglądać dziennik autoryzacji i wyszukiwać w nim tego rodzaju zapisów. Takie błędy przesyłane są codziennie pocztą do administratora, w ramach dziennego raportu bezpieczeństwa, więc bez żadnych wymówek trzeba je przeczytać.

Trzeba jednak pamiętać, że każdy, kto zna hasło superużytkownika, może podejść do konsoli i zalogować się bezpośrednio jako superużytkownik. Wtedy, jeżeli będzie chciał, będzie mógł dodać się do grupy *wheel*. To bardzo niedobra sytuacja. Można by zabronić logowania się z konsoli jako superużytkownik, ale czasami taka możliwość jest bardzo przydatna. Istnienie grupy *wheel* nie zwalnia nas, z obowiązku utrzymywania tajności hasła superużytkownika!

Jeżeli w grupie *wheel* nie mamy żadnych użytkowników, to uprawnienia superużytkownika można uzyskać tylko z konta *root* (pominąwszy oczywiście wykorzystanie luk w zabezpieczeniach). Jeżeli zapomnimy dodać do grupy *wheel* nasze podstawowe konto, to będziemy musieli zalogować się z konsoli do systemu jako superużytkownik

i wprowadzić odpowiednie zmiany w pliku */etc/group*. Jeżeli dodatkowo wyłączona została możliwość zalogowania się z konsoli jako superużytkownik, to konieczne będzie uruchomienie systemu w trybie jednoużytkownikowym i wprowadzenie odpowiednich modyfikacji.

Unikanie używania hasła superużytkownika poprzez stosowanie grup

Hasło superużytkownika jest nie tylko problemem bezpieczeństwa systemu, ale może też doprowadzić do różnego rodzaju tarć w organizacji. Administratorzy zwykle bardzo nie lubią przekazywania innym osobom hasła superużytkownika, nawet tym odpowiedzialnym za obsługę części systemu. Jeżeli taki administrator dodatkowo nie umie prawidłowo zarządzać komputerem, tego rodzaju niechęć może skutecznie uniemożliwić innym wykonywanie ich zadań. Wielu jest też administratorów, rozdających hasło superużytkownika praktycznie każdemu, kto o nie poprosi, a następnie narzekających, że system staje się coraz bardziej niestabilny. Oba zachowania nie sprawdzają się na dłuższą metę, szczególnie gdy okazuje się, że Unix posiada bardzo rozbudowane funkcje pozwalające na praktyczne wyeliminowanie potrzeby korzystania z hasła superużytkownika.

Jedną z typowych sytuacji jest odpowiedzialność młodego administratora za wydzieloną część systemu. Pracowałem już z wieloma administratorami DNS; oni nie instalują żadnego oprogramowania, nie kompilują jądra systemu, ani nie wykonują żadnych innych zadań niskiego poziomu. Odpowiadają tylko na listy, aktualizują pliki stref i przeładowują demona *named*. Nowi administratorzy bardzo często sądzą, że do wykonania tego rodzaju zadań potrzebują uprawnień superużytkownika. Tworząc specjalne grupy użytkowników wykonujących podobne zadania administracyjne, można umożliwić im ich wykonywanie bez konieczności korzystania z hasła superużytkownika. W tym podrozdziale zaimplementujemy grupowy dostęp do plików serwera nazw. Te same zasady dotyczą też wszystkich innych plików, jakie chcielibyśmy chronić. Tego rodzaju oddelegowania często stosowane są również dla plików konfigurujących serwery pocztowe i WWW.

W OpenBSD istnieją pewne zarezerwowane konta użytkowników, wykorzystywane przez system do uruchamiania zintegrowanych z nim programów. Na przykład, serwer nazw uruchamiany jest w koncie o nazwie *named* i w grupie o tej samej nazwie. Jak już wspominaliśmy wcześniej, jeżeli włamywaczowi uda się złamać zabezpieczenia serwera, to uzyska on dostęp do komputera jedynie tak szeroki jak uprawnienia użytkownika serwera nazw. Możemy utworzyć grupę o nazwie *dns* i dodać do niej wszystkie osoby zajmujące się konserwacją tego serwera. Do takich prac nie należy używać konta samego serwera! Chcemy uzyskać stan, w którym użytkownik *named* będzie mógł odczytywać pliki konfiguracji serwera, których właścicielem będzie grupa *dns*. W ten sposób odbieramy użytkownikowi *named* możliwość zapisywania jakichkolwiek danych do tych plików, a tym samym ograniczamy możliwości zniszczeń, jakie mógłby poczynić ktoś, kto złamałby zabezpieczenia demona.

Najprostszą metodą na utworzenie grupy będącej właścicielem tych plików jest utworzenie użytkownika, przekazanie mu plików na własność i późniejsze wykorzystywanie jego grupy pierwotnej jako grupy plików serwera. Poleceniem `adduser`(8) tworzymy nowego użytkownika, nazywając go `dns`, ponieważ nazwa *named* jest już wykorzystana przez serwer nazw. Sama nazwa nie jest ważna, ale dobrze jest wybierać takie nazwy, które łatwo dają się zapamiętać.

```
# adduser -silent
Enter username [a-z0-9_-]: dns
Enter full name []: Użytkownik-administrator serwera DNS
Enter shell csh ksh nologin sh [csh]: nologin
```

Przypisaliśmy nowemu użytkownikowi powłokę `nologin`, która to opcja wiąże go z programem powłoki */sbin/nologin*. Nikt nie będzie mógł się zalogować na to konto.

```
Uid [1001]:
```

Mamy możliwość nadania tego rodzaju użytkownikom specjalnych numerów identyfikacyjnych. Znany jestem z tego, że w takich sytuacjach lubię nadawać numery identyfikacyjne podobne do numerów użytkowników związanych z samym programem. Na przykład, użytkownik *named* ma identyfikator 70, w związku z czym możemy nadać użytkownikowi *dns* identyfikator 1070 i w ten sposób zaznaczyć związek łączący mojego prywatnego użytkownika z użytkownikiem systemowym. Należy pamiętać, że identyfikatory użytkowników mniejsze od 1000 zarezerwowane są na potrzeby systemu.

```
Login group dns [dns]:
Login group is ``dns''. Invite dns into other groups: guest no [no]:
```

Sensem zakładania takiego użytkownika jest to, że posiada on swoją własną grupę. Pod żadnym pozorem nie wolno użytkownika administracyjnego dodawać do innych grup!

```
Enter password []:
Set the password so that user cannot login? (y/n) [n]: y
```

Na pytanie o hasło użytkownika odpowiadamy naciśnięciem klawisza *Enter*, a program *useradd* pozwoli nam skonfigurować je tak, że użytkownik nie będzie mógł się zalogować. Oto nam właśnie chodzi; użytkownik administracyjny nigdy nie powinien mieć powodu, żeby zalogować się do systemu.

Skoro mamy już przygotowanego użytkownika administracyjnego i grupę, to możemy rozpocząć przekazywanie mu na własność plików. Każdy plik ma swojego użytkownika-właściciela i grupę-właściciela. Istniejące zapisy o właścicielach pliku można zobaczyć, wpisując polecenie `ls -l`[4]. Wielu nowych administratorów systemów zwraca baczną uwagę na właściciela pliku, ale uprawnienia grupy przegląda już tylko pobieżnie.

```
# ls -l
total 29
-rw-rw-r--  1 root  wheel    27136 Sep 14 09:36 plik1
-rwxrwxr--  1 root  wheel     1188 Sep 14 09:35 plik2
#
```

[4] W ramach przypomnienia sobie uprawnień uniksowych proponuję przejrzenie strony podręcznika *ls*(1).

W powyższym przykładzie `plik1` może być odczytywany przez superużytkownika i członków grupy `wheel`, a pozostali użytkownicy mogą go jedynie odczytywać. Podobnie, superużytkownik i członkowie grupy `wheel` mogą odczytywać `plik2`. Jeżeli bylibyśmy członkiem tej grupy, to nie musimy używać hasła superużytkownika, aby edytować lub odczytywać `plik2` — wystarczy otworzyć edytor i rozpocząć pracę!

Aby zmienić właściciela pliku, należy korzystać z polecenia `chown`(1), natomiast poleceniem `chgrp`(1) możemy zmienić grupę-właściciela pliku. Oba polecenia mają dokładnie tę sama składnię: pobierają nazwę nowego właściciela i nazwę pliku.

```
# chown dns plik1
# chgrp dns plik1
# ls -l plik1
-rw-rw-r--  1  dns  dns    27136 Sep 14 09:36 plik1
#
```

Teraz właścicielami pliku są użytkownik *dns* i grupa *dns*. Każdy członek tej grupy może odczytywać i zapisywać dane do tego pliku bez konieczności używania hasła superużytkownika. Co więcej, ten plik może być odczytywany przez serwer nazw. Teraz pozostało już tylko dodać młodszych administratorów do grupy *dns* w pliku */etc/group*, co pozwoli im na edytowanie plików konfiguracyjnych serwera bez znajomości hasła superużytkownika.

Jak na razie, jedyną czynnością, do jakiej administratorzy DNS musieliby używać hasła superużytkownika jest ponowne uruchamianie serwera nazw. Najprostszym rozwiązaniem byłoby skonfigurowanie zadania programu *cron*, które regularnie uruchamiałoby serwer. Jednak administratorzy mogą chcieć w niektórych sytuacjach mieć możliwość ręcznego uruchomienia serwera. A do tego najlepiej nadaje się polecenie `sudo`.

Ukrywanie superużytkownika za programem sudo

Właściwe wykorzystanie grup może niemal całkowicie wyeliminować konieczność korzystania z hasła superużytkownika, ale nie zda się to na wiele, jeżeli chodzi o polecenia, które może uruchamiać wyłącznie superużytkownik. Można co prawda tak skonfigurować zadania programu *cron*, żeby serwer nazw był ponownie uruchamiany codziennie o północy, ale zdarzają się też sytuacje, w których administrator DNS zmuszony jest ręcznie uruchomić ponownie serwer. Polecenie `ndc`(8) służące do administrowania serwerem nazw może być uruchamiane wyłącznie przez superużytkownika. Superużytkownik jest mechanizmem typu wszystko albo nic, dlatego tradycyjnie już, osoby wykonujące mało znaczące zadania w systemie musiały korzystać z hasła superużytkownika.

W OpenBSD dostępny jest program *sudo*(8) wraz ze związanymi z nim narzędziami umożliwiającymi bardzo dokładną kontrolę dostępu do poleceń, które mogą być uruchamiane wyłącznie przez podanych użytkowników. Po przygotowaniu odpowiedniej konfiguracji, administrator systemu może pozwolić innym użytkownikom na uruchamianie dowolnego polecenia jako dowolny inny użytkownik. Program *sudo*(8) to bardzo

rozbudowane narzędzie, które może być skonfigurowane tak, aby zezwalało lub zabraniało wykonywania dowolnej kombinacji operacji. Tak wielkie możliwości powodują, że dokumentacja programu jest bardzo rozległa, w efekcie nowi użytkownicy najczęściej stronią od tego programu. Przygotujemy teraz prostą konfigurację programu, która pokryje niemal wszystkie jego zastosowania. Należy jednak pamiętać o tym, że możliwych kombinacji konfiguracyjnych jest o wiele więcej, udokumentowanych na stronach podręcznika *sudo*(8) i *sudoers*(5).

Po co używać sudo?

Poza uzyskaniem możliwości dokładnej kontroli dostępu, używanie programu *sudo* wiąże się z jeszcze innymi korzyściami. Jedną z największych zalet programu jest prowadzenie dziennika poleceń. Każde polecenie programu *sudo*(8) zapisywane jest do dziennika, co bardzo ułatwia sprawdzanie, kto wykonał pewne operacje. Poza tym, po prawidłowym skonfigurowaniu programu *sudo*(8), główny administrator systemu może zmienić hasło superużytkownika i całkowicie je utajnić. Jeżeli pozostali administratorzy otrzymali prawidłowe uprawnienia programu *sudo*, to hasło superużytkownika nie powinno im być już nigdy potrzebne! Zmniejszenie ilości osób znających hasło superużytkownika może znacząco zmniejszyć zagrożenie bezpieczeństwa systemu.

Dodatkowo, program *sudo*(8) może być uruchamiany w niemal wszystkich systemach uniksowych. Co więcej, pojedynczy plik konfiguracyjny może być stosowany we wszystkich tych systemach, znacząco ułatwiając pracę administratora.

Wady programu sudo

Zdecydowanie największą wadą programu *sudo*(8) jest to, że bardzo nie lubią go młodsi administratorzy. Osoby, które tradycyjnie już miały dostęp do systemu poprzez hasło superużytkownika uważają, że utracą swoje uprawnienia w momencie, gdy główny administrator zaimplementuje w systemie program *sudo*(8). Kluczem do przezwyciężenia takich oporów jest upewnienie się, że osoby odpowiedzialne za przeprowadzanie pewnych prac w systemie mają właściwe uprawnienia do ich wykonania. Jeżeli młodszy administrator narzeka, że nie może wykonać jakiegoś zadania, to znaczy, że przekracza on swój zakres odpowiedzialności albo trzeba nadać mu więcej uprawnień.

Jeżeli nie znamy składni opisującej uprawnienie, to może się ona wydawać zawikłana. Uzyskanie właściwych rezultatów za pierwszym razem bywa bardzo trudne. Jednak gdy już poznamy sposoby zarządzania uprawnieniami przez program *sudo*(8), wszystkie ustawienia wykonywane są szybko i sprawnie.

Na koniec trzeba wspomnieć o tym, że nieprawidłowa konfiguracja programu *sudo*(8) może być powodem wyłomów w zabezpieczeniach. Konfiguracja przygotowana w sposób bezmyślny, wytworzy tyle dziur w zabezpieczeniach, że sprytny młodszy administrator będzie mógł wykorzystać je do uzyskania uprawnień superużytkownika[5].

[5] Mimo wielkich nadziei menedżerów z całego świata, rozwiązania techniczne sprawdzają się tak dobrze wyłącznie na poziomie administracji. Jeżeli ludzie nie będą chcieli zachowywać się przyzwoicie, czasami konieczne jest wyciągnięcie Wielkiego Kija i okładanie ich tak długo, aż zmienią swoje zachowanie.

Przegląd

W skrócie, *sudo*(8) jest programem opakowującym konto superużytkownika i pozwalającym dowolnemu użytkownikowi uruchamiać inne polecenia. Pobiera on polecenie, jakie chcemy wykonać i porównuje je ze swoją wewnętrzną bazą zezwoleń i przywilejów. Jeżeli pozwalają one uruchomić danemu użytkownikowi polecenie jako inny użytkownik, to `sudo` wykona to polecenie. Superużytkownik może uruchomić dowolne polecenie w systemie jako dowolny inny użytkownik, dlatego `sudo` również może uruchamiać polecenia jako dowolny użytkownik systemu. Ten mechanizm można wykorzystać do udzielenia danemu użytkownikowi uprawnień do wykonania pewnego polecenia jako superużytkownik, jako inny użytkownik albo w dowolnej innej kombinacji.

System *sudo* składa się z trzech elementów. Pierwszym z nich jest samo polecenie `sudo`(8), czyli program opakowujący konto superużytkownika. Drugim elementem jest plik konfiguracyjny */etc/sudoers*. Zapisane są w nim informacje o tym, kto może uruchomić jakie polecenie, podając się za jakiego użytkownika. Sam plik jest w pełni udokumentowany na stronie podręcznika *sudoers*(5). I w końcu trzeci element — polecenie `visudo`(8) pozwala administratorom na edycję pliku *sudoers* bez tworzenia ryzyka dla całego systemu. Po kolei będziemy opisywać każdy z tych elementów.

visudo

Program *sudo* nie uruchomi się, jeżeli składnia pliku *sudoers* nie będzie prawidłowa. Jeżeli dostęp do pliku *sudoers* kontrolowany jest przez system *sudo*, to uszkodzenie tego pliku może spowodować, że zostaniemy odcięci od możliwości pracy w systemie na poziomie superużytkownika, a jednocześnie uniemożliwić jakiekolwiek próby skorygowania uszkodzenia. Z pewnością jest to bardzo niepożądana sytuacja. Program *visudo*(8) zapewnia ochronę przed tego rodzaju wypadkami.

Podobnie jak program *vipw*(8), *visudo*(8) również blokuje plik konfiguracji, tak że jednocześnie może edytować go tylko jedna osoba. Następnie otwiera plik konfiguracji w edytorze tekstowym (domyślnie jest to *vi*(1), ale respektowane są zapisy ze zmiennej środowiskowej `$EDITOR`. Gdy wyjdziemy z edytora, *visudo* sprawdzi składnię zapisów w pliku i potwierdzi, że nie ma w nim żadnych błędów składniowych. Nie daje to żadnej gwarancji, że plik spełni nasze oczekiwania, a jest jedynie potwierdzeniem poprawności zapisów konfiguracyjnych. *Visudo* zaakceptuje nawet plik z konfiguracją mówiącą, że „nikt nie może nic zrobić poprzez *sudo*”, o ile zapisy zasad będą poprawne.

Jeżeli w pliku konfiguracyjnym zostanie znaleziony jakiś błąd, to *visudo* wypisze komunikat podający numer wiersza z błędem i zapyta, co chcemy dalej robić.

```
# visudo
>>> sudoers file: syntax error, line 44 <<<
What now?
```

Okazało się, że popełniliśmy błąd w 44 wierszu pliku. Mamy teraz trzy możliwości do wyboru: ponownie edytować plik, zakończyć pracę bez zapisywania wprowadzonych zmian albo wymusić na *visudo* zapisanie przygotowanego przez nas pliku *sudoers*.

Jeżeli naciśniemy klawisz *e*, to zostaniemy odesłani z powrotem do edytora, gdzie będziemy mogli przejść do wspomnianego wiersza i próbować skorygować błąd.

Jeżeli naciśniemy klawisz *x*, to *visudo* przywróci pierwotną postać pliku konfiguracyjnego. Wszystkie wprowadzone zmiany będą stracone, ale nie jest to najgorsze rozwiązanie. Lepsza jest stara, ale działająca konfiguracja, niż konfiguracja nowa, ale niedziałająca.

Naciśnięcie klawisza q wymusi na *visudo* zapisanie zawartości pliku zawierającego błędy. Jeżeli w pliku konfiguracyjnym znajdują się błędy, to *sudo*(8) się nie uruchomi. Oznacza to, że decydujemy się wyłączyć cały system zabezpieczeń do czasu aż zalogujemy się do systemu jako superużytkownik i skorygujemy błędy. To zdecydowanie nie jest zalecany sposób działania!

/etc/sudoers

Plik *sudoers* przechowuje informacje o tym, kto może uruchomić jakie polecenie, podając się za jakiego użytkownika. Osoby traktujące ten punkt jako źródło informacji o systemie *sudo*, działającym w innym systemie operacyjnym, będą musiały same znaleźć w nim plik *sudoers*. Tego pliku nigdy nie należy edytować bezpośrednio, nawet jeżeli dokładnie wiemy, jakie zmiany chcemy wprowadzić; zawsze trzeba używać `visudo`(8).

Różne przykładowe pliki *sudoers*, jakie można znaleźć w internecie, często wyglądają na przerażająco skomplikowane, ponieważ przedstawiają wszystkie przydatne funkcje, jakimi dysponuje *sudo*. Na tym etapie nie potrzebujemy wszystkich przydatnych funkcji, a tylko prostego sposobu na przyznanie różnym użytkownikom uprawnień do uruchamiania pewnych poleceń. Podstawowa składnia jest tutaj bardzo prosta. Każda zasada opisywana jest w pliku następującym formatem:

```
❶nazwa_użytkownika      ❷host=❸polecenie
```

Nazwa użytkownika ❶ jest nazwą użytkownika (lub jej aliasem), który może uruchomić polecenie.

Host ❷ określa nazwę komputera, na którym obowiązuje ta zasada. System *sudo* skonstruowany jest w ten sposób, że jednego pliku konfiguracyjnego można używać na wszystkich systemach, co pozwala na stosowanie zasad dotyczących różnych komputerów.

Polecenie ❸ jest listą poleceń, jakich dotyczy ta reguła. Do każdego polecenia trzeba podawać jego pełną ścieżkę, inaczej `sudo` nie rozpozna polecenia! Nie chcielibyśmy przecież, żeby użytkownicy mogli modyfikować swoją zmienną `$PATH` i uzyskiwać w ten sposób dostęp do zmienionych wersji różnych poleceń.

W każdym z podanych pól można wpisać słowo kluczowe `ALL`, zastępujące wszystkie możliwe do wpisania w danym miejscu opcje.

Na przykład załóżmy, że użytkownikowi `krzys` mogę zaufać tak bardzo, że pozwolę uruchamiać mu wszystkie polecenia na każdym komputerze.

```
krzys     ALL = ALL
```

Przekazanie młodszemu administratorowi pełnej kontroli nad jednym z podległych mi systemów jest bardzo mało prawdopodobne. Skoro Krzysztof pracuje dla mnie, to wiem, jakie przydzieliłem mu zadania i jakich będzie potrzebował poleceń, żeby się z tych zadań wywiązać. Załóżmy, że Krzysztof odpowiedzialny jest za część systemu związaną z serwerem nazw. Za pomocą uprawnień grupowych kontrolujemy możliwości edytowania plików stref, ale to nie wystarczy, jeżeli serwer będzie musiał być uruchomiony ponownie, przeładowany lub zatrzymany. Poniżej przekazuję mu uprawnienia do uruchamiania na dowolnym komputerze programu kontrolującego demona *named* — `ndc`(8).

```
krzys     ALL = /usr/sbin/ndc
```

Jeżeli ten plik rozprowadzę na kilka komputerów, to będzie bardzo prawdopodobne, że na kilku z nich serwer nazw w ogóle nie będzie działał. Poniżej ograniczę uprawnienia Krzysztofa do uruchamiania programu jedynie na serwerze o nazwie `dns1`.

```
krzys     dns1 = /usr/sbin/ndc
```

Z drugiej strony, Krzysztof jest też administratorem serwera poczty o nazwie `poczta1`. Ten serwer jest całkowicie pod jego opieką i może na nim uruchamiać wszystkie dostępne polecenia. Mogę przyznać mu zupełnie inne uprawnienia na serwerze pocztowym, ale nadal korzystać z jednego pliku *sudoers*.

```
krzys     dns1 = /usr/sbin/ndc
krzys     poczta1 = ALL
```

Wiele wpisów w jednym polu

W jednym polu można podawać wiele wpisów, o ile będą one rozdzielane przecinkami. Poniżej pozwolimy Krzysztofowi dodatkowo na montowanie stacji dyskietek poleceniem `mount`(8).

```
krzys     dns1 = /usr/sbin/ndc, /bin/mount
```

Uruchamianie poleceń jako zwykły użytkownik

Przed poleceniem można podać nazwę użytkownika i zdefiniować w ten sposób, że dany użytkownik może wykorzystać *sudo* do uruchamiania tego polecenia jako inny użytkownik. Na przykład załóżmy, że serwer nazw działa jako użytkownik `named`, a wszystkie polecenia kontrolujące ten serwer muszą być wydawane właśnie przez tego użytkownika.

```
krzys     dns1 = (named) /usr/sbin/ndc
```

Aliasy w pliku /etc/sudoers

Jak można sobie wyobrazić, gdy będziemy mieli już kilka różnych komputerów, nad którymi opiekę sprawować będzie wielu administratorów o różnym poziomie uprzywilejowania, to bardzo szybko sprawy zaczynają się komplikować. Jeżeli mamy kilku użytkowników z podobnymi przywilejami i wielkie listy poleceń, jakie chcemy pozwolić im uruchamiać, to obsługa pliku konfiguracyjnego zaczyna być prawdziwym wyzwaniem, ponieważ trzeba się przedzierać przez długie listy użytkowników, poleceń i komputerów. Aliasy znacząco upraszczają te zadania i doskonale czyszczą plik konfiguracji *sudo*(8).

Alias można najprościej opisać jako zbiór użytkowników, komputerów lub poleceń. Gdy zmieniają się obowiązki użytkownika, to można nadać mu odpowiednie uprawnienia, wpisując je do aliasu tego użytkownika. Jeżeli chcielibyśmy pozwolić administratorom systemu na tworzenie kopii bezpieczeństwa, ale odmówić im prawa do odtwarzania tych kopii, to wystarczy z aliasu poleceń administratorów usunąć polecenie `restore`(8). Gdy zainstalujemy nowy serwer, to przekazanie odpowiednich uprawnień dla administratorów obsługujących ten serwer będzie polegało wyłącznie na dodaniu nazwy serwera do aliasu serwerów.

Alias musi zostać zdefiniowany jeszcze zanim pierwszy raz pojawi się w pliku *sudoers*. Z tego powodu wszystkie aliasy najczęściej grupowane są na początku tego pliku. Każda definicja aliasu składa się ze specjalnej etykiety określającej jego typ, nazwę i listy jego składowych.

Aliasy użytkowników

Aliasy użytkowników, oznaczane etykietą `User_Alias`, tworzą grupy użytkowników.

```
User_Alias    DNSADMINS = krzys, mwlucas
```

Alias użytkowników o nazwie `DNSADMINS` zawiera w sobie dwóch użytkowników — `mwlucas` i `krzys`.

Aliasy „uruchom jako”

Aliasy „uruchom jako” to wyjątkowy rodzaj aliasów. Przechowują one listę użytkowników, za których podając się inni użytkownicy mogą uruchamiać polecenia. Jak już wspomnieliśmy wcześniej, serwer nazw uruchamiany jest jako użytkownik o nazwie `named`. Może się zdarzyć, że administrator DNS będzie musiał uruchamiać polecenia jako ten użytkownik, dlatego dobrze byłoby zdefiniować w tym celu odpowiedni alias. Wiele aplikacji bazodanowych wymaga do działania własnego użytkownika, ponieważ w systemie działają jako właśnie ten użytkownik. W wielu przypadkach, administrator systemu odpowiedzialny za daną aplikację powinien mieć też możliwość tworzenia kopii bezpieczeństwa danych, korzystając z użytkownika `operator`. Na takie operacje pozwalają właśnie aliasy „uruchom jako”. Pozwalają one uruchamiać polecenia użytkownikowi podającemu się za innego użytkownika, oczywiście zgodnie z regułami zapisanymi w pliku *sudoers*. Nazwę użytkownika można podać w nawiasach przed samym poleceniem,

tak jak opisane to zostało w punkcie „Uruchamianie poleceń jako zwykły użytkownik”, alternatywą jest przygotowanie jednego aliasu „uruchom jako” grupującego wszystkie te polecenia. Aliasy „uruchom jako” oznaczane są etykietą `Runas_Alias`.

```
Runas_Alias    ADMINAP = uzytkbd,operator
```

Aliasy komputerów

Alias komputerów jest zwykłą listą komputerów, oznaczoną etykietą `Host_Alias`. Alias komputerów może być tworzony w postaci nazw komputerów, adresów IP albo bloków sieciowych. Trzeba pamiętać, że jeżeli będziemy stosować nazwy komputerów, to cała konfiguracja *sudo* będzie wrażliwa na wszystkie problemy z serwerami DNS! Poniżej podaję przykłady wszystkich trzech sposobów tworzenia aliasu komputerów:

```
Host_Alias    SERWERYDNS = dns1,dns2,dns3
Host_Alias    SERWERYBEZPIECZENSTWA = 192.168.1.254,192.168.113.254
Host_Alias    SIECFIRMOWA = 192.168.1.0/16
```

Aliasy poleceń

Alias poleceń jest listą różnych poleceń, oznaczanych etykietą `Cmnd_Alias`. Poniżej podaję polecenia konieczne do wykonania na taśmie kopii bezpieczeństwa systemu i jej późniejszego odtworzenia.

```
Cmnd_Alias    KOPIEBEZP = /bin/mt,/sbin/restore,/sbin/dump
```

Możliwe jest przygotowanie aliasu obejmującego wszystkie polecenia znajdujące się w podanym katalogu. Załóżmy, że pracujemy z pewną aplikacją, która wszystkie swoje polecenia umieszcza w jednym z podkatalogów katalogu domowego użytkownika. Zamiast wypisywać wszystkie te polecenia, wystarczy podać pełną nazwę katalogu i zastosować znak nazwy wieloznacznej (*), a aliasem objęte zostaną wszystkie polecenia z tego katalogu.

```
Cmnd_Alias    POLECENIABD = /usr/home/uzytkbd/bin/*
```

Długie wiersze

Każdy wpis w pliku */etc/sudoers* musi zmieścić się w jednym wierszu. Niestety w ten sposób mogą powstawać bardzo długie wiersze. Jeżeli musimy podawać długą listę składowych aliasów lub reguł, możemy przenieść się do następnego wiersza, umieszczając na końcu każdego niedokończonego wiersza znak ukośnika (\).

```
Cmnd_Alias    POWLOKI = /bin/sh, /bin/csh, /usr/local/bin/ksh, \
        /usr/local/bin/tcsh, /usr/local/bin/bash
```

Stosowanie aliasów w pliku /etc/sudoers

Aby użyć aliasu, wystarczy umieścić jego nazwę w miejscu, w którym normalnie podajemy użytkowników, komputery lub polecenia. Wcześniej już zdefiniowaliśmy alias `DNSADMINS`, a teraz użytkownikom wymienionym w tym aliasie pozwolimy uruchamiać wszystkie polecenia na każdym komputerze.

```
DNSADMINS    ALL = ALL
```

Załóżmy teraz, że Krzysztof musi zajmować się aplikacją, która uruchamiana jest jako jeden z użytkowników, dlatego musi on mieć możliwość uruchamiania poleceń jako ten właśnie użytkownik. Zdefiniowaliśmy już wcześniej alias „uruchom jako", wyszczególniający tego rodzaju użytkowników (ADMINAP), a także alias poleceń wymaganych do uruchamiania tej aplikacji (POLECENIABD).

```
krzys    ALL = (ADMINAP)POLECENIABD
```

Jako administrator aplikacji, Krzysztof będzie musiał też wykonywać kopie bezpieczeństwa. Nadaliśmy już aliasowi ADMINAP uprawnienia operatora, przygotowaliśmy także specjalny alias poleceń do wykonywania kopii bezpieczeństwa. Teraz możemy połączyć je w ten sposób:

```
krzys    ALL = (ADMINAP)POLECENIABD, (ADMINAP)KOPIEBEZP
```

Znacznie łatwiej odczytać taki zapis, niż ten, w który można by go rozwinąć.

```
krzys    ALL = (uzytkbd,operator)/usr/home/uzytkbd/bin/*,\
         (uzytkbd,operator)/bin/mt, (uzytkbd,operator)/sbin/restore,\
         (uzytkbd,operator)/sbin/dump
```

Niektóre z przyznanych tutaj uprawnień są zupełnie niepotrzebne — do wykonywania kopii bezpieczeństwa niepotrzebny jest alias uruchamiania jako użytkownik uzytkbd. Mimo to, jest to znacznie bezpieczniejsze rozwiązanie, niż przekazywanie hasła superużytkownika! Poza tym, można tak przebudować reguły, aby ograniczały uprawnienia użytkowników dokładnie zgodnie z naszymi wymaganiami.

Zagnieżdżanie aliasów

Wewnątrz aliasów możliwe jest podawanie innych aliasów. Spróbujmy, na przykład, połączyć aliasy POLECENIABD i KOPIEBEZP w jedną grupę poleceń.

```
Cmnd_Alias    ADMINDB = KOPIEBEZP,POLECENIABD
```

Wykorzystywanie grup systemowych jako aliasów

Program *sudo*(8) może pobierać informacje o grupach z systemu i włączać je w plik *sudoers* jako aliasy użytkowników. Zamiast ręcznie definiować alias użytkownika, można podać nazwę grupy systemu OpenBSD, poprzedzając ją znakiem procenta (%).

```
%wheel    ALL = ALL
```

Każdy członek systemowej grupy *wheel* będzie mógł od teraz na wszystkich komputerach uruchamiać dowolne polecenia jako superużytkownik.

Identyczne nazwy aliasów

Nazw aliasów można używać wielokrotnie. Alias użytkownika o nazwie ADMINBD nie jest równoznaczny z aliasem poleceń o nazwie ADMINBD. Nic nie stoi na przeszkodzie, żeby wprowadzić następujące zapisy.

```
Cmnd_Alias    APLIKACJABD = /usr/home/uzytkbd/bin/*
Host_Alias    APLIKACJABD = serwer8,serwer12,serwer15
Runas_Alias   APLIKACJABD = uzytkbd,operator
User_Alias    APLIKACJABD = krzys,mwlucas
APLIKACJABD   APLIKACJABD = (APLIKACJABD)APLIKACJABD
```

Jeżeli jednak zaczniemy tworzyć takie konfiguracje, to osoby, które będą musiały później wyszukiwać w nich błędów, będą przeklinać nas po wsze czasy. Nawet jeżeli przekleństwa nie robią na nas wrażenia, to takie zapisy najczęściej powodują, że telefon odzywa się w najbardziej nieludzkich godzinach, w których głównemu administratorowi normalnie wolno nieco przysnąć.

Używanie sudo

Skoro wiemy już, jak należy konfigurować system *sudo,* zobaczmy, jak można z niego korzystać. Poinformujemy *sudo*, że nasze konto uprawnione jest do uruchamiania wszystkich możliwych poleceń. Każdy czytelnik tej książki powinien mieć dostęp do konta superużytkownika na przynajmniej jednym komputerze — najlepiej byłoby, gdyby była to specjalna maszyna testowa — wobec czego nie będzie to stanowić zagrożenia dla bezpieczeństwa systemu.

Sudo i hasła

Przy pierwszym uruchomieniu programu *sudo*(8), poprosi on o podanie hasła. Należy wprowadzić hasło własnego konta, ale nie hasło superużytkownika. Jeżeli wprowadzimy nieprawidłowe hasło, to *sudo* wyrazi swoje wątpliwości co do naszych umiejętności pisania na klawiaturze i możliwości umysłowych, a na koniec pozwoli spróbować jeszcze raz. Po trzech nieudanych próbach, *sudo* zrezygnuje z dalszych prób, i będziemy musieli ponownie wprowadzić polecenie, aby uruchomić system *sudo*.

Gdy wprowadzimy w końcu właściwe hasło, *sudo*(8) zapisze sobie aktualny czas. Jeżeli program zostanie uruchomiony ponownie w ciągu pięciu minut, to nie zostaniemy zapytani o hasło. Jeżeli jednak nie będziemy korzystać z *sudo* przez dłuższy czas, to konieczna będzie ponowna autoryzacja. To bardzo ułatwia pracę, jeżeli uruchamiamy całą serię poleceń poprzez *sudo*, a jednocześnie pięć minut upływa wystarczająco szybko, aby zabezpieczyć system, gdy odejdziemy od komputera.

Kontrolowanie uprawnień sudo

Jeżeli jesteśmy użytkownikiem w systemie działającym z aktywnym *sudo*, pierwszą rzeczą, jakiej chcielibyśmy się dowiedzieć, jest lista poleceń, które pozwolił nam uruchamiać administrator. Taką listę otrzymamy, uruchamiając `sudo` z opcją `-l`:

```
# sudo -l
Password:
User mwlucas may run the following commands on this host:
    (root) ALL
#
```

Jeżeli ograniczenia są bardziej restrykcyjne, to zostaną one wypisane na ekranie.

Uruchamianie poleceń poprzez sudo

Uruchomianie polecenia poprzez *sudo*, uzyskujemy, wpisując w wierszu poleceń, przed właściwym poleceniem, słowo `sudo`. Na przykład, w ten sposób można uzyskać uprawnienia superużytkownika:

```
# sudo su
Password:
#
```

Używanie *sudo*(8) do uzyskania uprawnień superużytkownika pozwala głównemu administratorowi na utrzymanie tajności hasła superużytkownika. Nie jest to całkowicie bezpieczne, ponieważ młodszy administrator z nieograniczonym dostępem poprzez *sudo*, może łatwo zmienić hasło superużytkownika. Mimo to jest to początek, pozwalający na lepsze zabezpieczenie systemu.

Poprzez *sudo*(8) można uruchamiać też bardziej złożone polecenia, razem z wszystkimi ich parametrami. Na przykład, polecenie `tail -f` doskonale nadaje się do przeglądania ostatnich zapisów w pliku dziennika, z uzupełnieniem wyświetlania, gdy tylko do dziennika dopisane zostaną nowe informacje. Niektóre z plików dzienników może przeglądać wyłącznie superużytkownik — na przykład, dziennik zawierający zapisy o próbach dostępu zapisywany przez *sudo*. Czasami dobrze jest mieć możliwość przejrzenia tych dzienników bez konieczności logowania się jako superużytkownik.

```
# sudo tail -f /var/log/authlog
Jul 29 13:24:19 openbsd sudo: mwlucas : TTY=ttyp0 ; PWD=/home/mwlucas ; USER=root ;
COMMAND=list
Jul 29 13:30:03 openbsd sudo: mwlucas : TTY=ttyp0 ; PWD=/home/mwlucas ; USER=root ;
COMMAND=/usr/bin/tail -f /var/log/authlog
...
```

Uruchamianie poleceń jako inni użytkownicy

Możemy też chcieć uruchamiać polecenia jako użytkownik inny niż *root*, oczywiście jeżeli posiadamy odpowiednie uprawnienia. Na przykład, załóżmy, że w naszej aplikacji bazodanowej wszystkie polecenia musi wydawać ten sam użytkownik, w ramach konta którego uruchamiana jest sama aplikacja. Programowi *sudo* możemy przekazać, aby uruchomił polecenie jako inny użytkownik, podając parametr `-u` i nazwę tego użytkownika. Na przykład, użytkownik *operator* ma uprawnienia wystarczające do uruchomienia polecenia `dump`(8) i wykonania kopii bezpieczeństwa systemu.

```
# sudo -u operator dump /dev/sd0s1
```

Wykluczanie poleceń z grupy ALL

Znamy już podstawy działania *sudo*, przyjrzyjmy się więc typowej sytuacji spędzającej sen z oczu nawet doświadczonych administratorów. Czasami chcielibyśmy pozwolić użytkownikowi na uruchamianie wszystkich poleceń w systemie, z wyjątkiem kilku szczególnych. Można to zrealizować za pomocą operatora `!`. Nie jest to najefektywniejsze rozwiązanie, jednak ze względu na jego powszechność, opiszemy je tutaj, szczególnie zaznaczając wady.

Po pierwsze, należy przygotować aliasy zawierające wszystkie zakazane polecenia. Najczęściej wyłączane są polecenia uruchamiające powłoki (uruchamiając powłokę z danymi innego użytkownika, uzyskamy dostęp do systemu z jego uprawnieniami) oraz polecenie su(1). Następnie, dla użytkownika tworzymy regułę i wykluczamy z niej te polecenia, umieszczając przed aliasami znak operatora !.

```
Cmnd_Alias    POWLOKI = /bin/sh,/bin/csh,/usr/local/bin/tcsh
Cmnd_Alias    SU = /usr/bin/su
mwlucas       ALL = ALL,!POWLOKI,!SU
```

Wygląda świetnie… i wygląda na to, że działa!

```
opebnsd~;sudo sh
Password:
Sorry, user mwlucas is not allowed to execute '/bin/sh' as root on openbsd.
openbsd~;
```

Należy pamiętać, że *sudo* do uruchamiania poleceń podaje ich pełną ścieżkę. Pozwoliliśmy użytkownikowi na uruchamianie wszystkich poleceń, za wyjątkiem tych kilku, które wymieniliśmy, podając ich pełną ścieżkę. Jeżeli jednak użytkownik zmieni ścieżkę do jednego z tych poleceń, to utworzone przez nas ograniczenie przestanie obowiązywać! A najłatwiej zrobi to, kopiując plik polecenia w inne miejsce.

```
# id
uid=1000(mwlucas) gid=1000(mwlucas) groups=1000(mwlucas), 0(wheel)
# cp /bin/sh /tmp/sh
# sudo /tmp/sh
# id
uid=0(root) gid=0(wheel) groups=0(wheel), 2(kmem), 3(sys), 4(tty), 5(operator),
20(staff), 31(guest)
#
```

Witaj superużytkowniku!

Takie ograniczenia może bardzo łatwo obejść każdy, kto zna choć podstawy funkcjonowania systemu *sudo*. Ten problem został już dokładnie opisany w podręczniku *sudo* i innych źródłach. Jednak ludzie nadal bardzo chętnie korzystają z tego sposobu ochrony systemów!

Wypływa z tego następująca lekcja: jeżeli pracujemy z użytkownikami, którym nie możemy bez obaw przekazać nieograniczonego dostępu do systemu, to nie możemy wykluczać poleceń z ogólnych uprawnień. Niestety konieczne jest wymienienie wszystkich poleceń, jakie będzie mógł uruchamiać dany użytkownik. Jeżeli użytkownik będzie chciał uzyskać większe uprawnienia, to będzie musiał o nie poprosić. Wtedy, nie ufając mu całkowicie, będziemy mogli spytać, do czego chce je wykorzystać!

Dzienniki sudo

Taka możliwość kontroli uprawnień i śledzenia poleceń wygląda bardzo atrakcyjnie, ale gdzie zapisywane są te wszystkie informacje? Komunikaty *sudo* można znaleźć w pliku */var/log/secure*. Każdy z komunikatów składa się ze znacznika czasu, nazwy użytkownika, nazwy katalogu, w którym uruchomiono *sudo* oraz polecenia, jakie zostało uruchomione.

```
Jul 29 11:21:02 openbsd sudo:    krzys : TTY=ttyp0 ; PWD=/home/krzys ; USER=root ;
COMMAND=/sbin/mount /dev/fd0 /mnt
```

W najgorszych przypadkach, można przynajmniej sprawdzić, jakie polecenia doprowadziły do zakłóceń pracy systemu. Na przykład, jeżeli jeden z komputerów nie uruchamia się prawidłowo, ze względu na brak lub uszkodzenie pliku */etc/rc.conf*, to możemy sprawdzić w dzienniku *sudo*, kto ostatnio zmieniał coś w tym pliku.

```
Jul 29 11:34:56 openbsd sudo:    krzys : TTY=ttyp0 ; PWD=/home/krzys ; USER=root ;
COMMAND=/bin/rm /etc/rc.conf
```

Jeżeli każdy użytkownik zamiast korzystać z *sudo*(8), używałby poleceń *su*(1) albo `sudo su`, to nie mielibyśmy żadnych informacji o tym, dlaczego system przestał działać. Ale dzięki dziennikom *sudo*, natychmiast po naprawieniu usterki, będziemy wiedzieć, kogo należy obarczyć odpowiedzialnością. W tym przypadku, warto było zainstalować *sudo*, choćby tylko dlatego, że mogłem nawrzeszczeć na Krzysztofa.