Bezpieczny Linux – przegląd projektów

Michał Piotrowski

Artykuł opublikowany w numerze 2/2006 magazynu *hakin9*. Zapraszamy do lektury całego magazynu.

Magazyn *hakin9*, Software-Wydawnictwo, ul. Piaskowa 3, 01-067 Warszawa, *pl@hakin9.org*

Pod lupą

Bezpieczny Linux – przegląd projektów

Michał Piotrowski

stopień trudności

Systemy linuksowe są dość odporne na włamania. Jednak w pewnych sytuacjach, gdy zależy nam na wysokim poziomie bezpieczeństwa komputera, standardowe dystrybucje okazują się niewystarczające. Przyjrzyjmy się kilku najpopularniejszym mechanizmom zwiększającym bezpieczeństwo Linuksa na poziomie jądra.

Pojęcie *bezpiecznego* i *niebezpiecznego* systemu operacyjnego jest złudne. Poziom zabezpieczeń jest przede wszystkim uzależniony od tego, jak system został skonfigurowany, jakie środki ochrony zostały w nim zastosowane, i jak dobrze dba o niego administrator. Na to z kolei wpływ ma jego wiedza i doświadczenie

Nawet jednak najlepszy administrator nie jest w stanie w pełni ochronić systemu przed wszystkimi atakami, gdyż zawsze pozostaje zagrożenie związane z tak zwanymi eksploitami *zero day*, czyli z nowymi, nieznanymi jeszcze powszechnie podatnościami. Na szczęście istnieją metody, by zabezpieczyć system i przed nimi lub przynajmniej zmniejszyć szanse ich powodzenia.

Najwięcej włamań do systemów linuksowych wynika z błędów w oprogramowaniu. Intruzi najczęściej wykorzystują błędy umożliwiające przepełnienie bufora (stosu i sterty), nieprawidłowe przetwarzanie ciągów formatujących oraz występowanie sytuacji wyścigu. Rzadziej zdarzają się ataki wykorzystujące złe prawa dostępu do zasobów systemowych oraz błędy w mechanizmach i protokołach sieciowych.

Dodatkowe mechanizmy ochrony systemów linuksowych można podzielić na cztery grupy:

- chroniące pamięć w jądrze systemu,
- chroniące pamięć w kompilatorze,
- umożliwiające nadzór nad dostępem do systemu (kontrola dostępu),
- inne (stosujące randomizację, szczegółowe ograniczenia dostępu, zapis zdarzeń zachodzących w systemie itp.).

Z artykułu dowiesz się...

- jakie zabezpieczenia są stosowane do ochrony przed popularnymi typami ataków,
- czym możesz zabezpieczyć swojego Linuksa.

Powinieneś wiedzieć...

- powinieneś znać podstawy administracji systemami Linux,
- powinieneś znać podstawy bezpieczeństwa systemów operacyjnych i sieci komputerowych.

Tabela 1. *Rozwiązania zwiększające bezpieczeństwo systemu Linux*

Grupa	Openwall	PaX	Stack-Guard	SSP	grsecurity	LIDS	SELinux	RSBAC
Ochrona pamięci w jądrze	Tak	Tak			Tak (zawiera PaXa)			Tak (zawiera PaXa)
Ochrona pamięci w kompilatorze			Tak	Tak				
Kontrola dostępu					Tak	Tak	Tak	Tak

W Tabeli 1 przedstawiono omawiane rozwiązania oraz przydzielono je do wymienionych grup.

Openwall

Autorem projektu Openwall jest Alexander Peslyak znany też jako Solar Designer. Jest on twórcą narzędzia do łamania haseł John the Ripper oraz dystrybucji Owl.

Openwall początkowo nazywał się Secure Linux Patch. Jego pierwsza wersja pojawiła się już w styczniu 1998 roku i była stosowana w jądrach serii 2.0. Łata jest rozwijana do dziś, ale nie istnieje jeszcze wersja dla jąder serii 2.6. Autor uważa, że przeniesienie mechanizmu do nowego kernela będzie miało sens dopiero, kiedy pojawi się wersja 2.6.20.

Mechanizmy ochronne

Openwall oferuje kilka mechanizmów ochrony. Dwa najważniejsze (patrz Ramka *Ochrona pamięci w jądrze*) to:

- uniemożliwienie wykonywania kodu na stosie procesu (co powstrzymuje większość eksploitów opartych na przepełnieniu bufora),
- randomizacja adresów funkcji bibliotek dzielonych (co uniemożliwia ataki powrotu do biblioteki libc).

Pozostałe mechanizmy ograniczają tworzenie twardych dowiązań (ang. *hard links*) oraz pisanie do nazwanych potoków (ang. *named pipes*) w katalogach z ustawionym bitem `+t`, czyli katalogach tymczasowych (na przykład */tmp*). Zwykli użytkownicy nie mogą więc tworzyć dowiązań do plików, które do nich nie należą lub do których nie mają praw odczytu i zapisu. Nie mogą też zapisywać danych do plików FIFO, których nie są właścicielami.

Łata ogranicza też możliwość odczytu informacji o systemie z katalogu */proc* przez zwykłego użytkownika, o ile nie należy on do specjalnej grupy. Użytkownik może więc otrzymać tylko informacje o swoich własnych procesach. Uniemożliwia użytkownikom uruchamianie usług sieciowych pod wybranymi adresami IP (tylko w jądrze serii 2.0). Wymusza na programach z ustawionym bitem SUID lub SGID specjalny sposób korzystania z deskryptorów standardowego wejścia, standardowego wyjścia oraz wyjścia dla komunikatów o błędach (w jądrach serii 2.0 i 2.2). Umożliwia także zwalnianie nieużywanych obszarów pamięci współdzielonej.

Instalacja

Aby zainstalować Openwall należy:

- zaktualizować źródła kernela,
- nałożyć łatę,
- skonfigurować jądro,
- skompilować kernel.

Ochrona pamięci w jądrze

Do ochrony pamięci procesu w jądrze systemu wykorzystywane są dwa podstawowe mechanizmy. Pierwszy z nich umożliwia nadanie wybranemu obszarowi pamięci praw albo do zapisu albo do wykonywania kodu. Drugi wprowadza randomizację.

Pamięć programów uruchamianych przez system operacyjny jest zbudowana z kilku elementów zwanych segmentami:

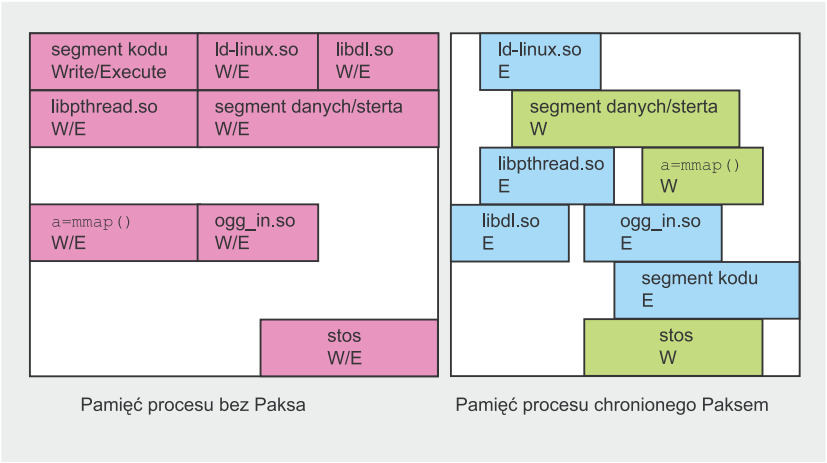
- segment kodu (zawiera kod wykonywalny programu),
- segment danych (zawiera zmienne statyczne i globalne),
- stos (zawiera dane tymczasowe – zmienne lokalne i parametry funkcji),
- sterta (zawiera dane tymczasowe, przydzielane i zwalniane przez program w sposób jawny za pomocą na przykład funkcji `malloc()`).

W pamięci umieszczone są również odwzorowania funkcji udostępnionych programowi, pochodzących z bibliotek dzielonych, na przykład `printf()`, `system()` itp.

Standardowe jądro Linuksa nadaje segmentom zbyt wysokie uprawnienia. Na przykład na stosie możliwe jest nie tylko zapisywanie danych, ale także wykonywanie kodu. Umożliwia to wykonywanie ataków przepełnienia bufora (ang. *buffer overflow*), które umieszczają i uruchamiają kod na stosie procesu. Aby ochronić system przed tego typu atakami, dodatkowe mechanizmy zmieniają domyślne uprawnienia poszczególnych segmentów pamięci. Proces otrzymuje albo prawo zapisu danych, albo wykonywania kodu który znajduje się w pamięci. Stos na przykład staje się niewykonywalny (ang. *non-executable*).

Niewykonywalny stos może jednak być wykorzystany do innego typu ataku, zwanego powrotem do biblioteki libc (ang. *return to libc*). Polega on na utworzeniu na stosie poprawnego wywołania współdzielonej funkcji – najczęściej funkcji `system()` z parametrem `/bin/sh`. W rezultacie włamywacz przekazuje sterowanie do wskazanej funkcji, która uruchamia powłokę systemową lub wykonuje inne operacje.

Aby atak się powiódł, napastnik musi jednak znać adres funkcji `system()` w pamięci procesu. Zapobiega temu drugi z wspomnianych mechanizmów – randomizacja. Dzięki niej, poszczególne segmenty pamięci i adresy funkcji dzielonych są ułożone inaczej przy każdym uruchomieniu programu. Intruz nie jest więc w stanie przewidzieć poprawnych adresów funkcji dzielonych, a tym samym nie może ich wywołać.



Rysunek 1. *Ochrona stosu za pomocą Paksa*

Aby korzystać z niektórych funkcji Openwalla w połączeniu z niektórymi wersjami biblioteki glibc oraz programów z pakietów procps lub psmisc, może być konieczne ich zaktualizowanie. Do łaty dołączono program *chstk*, który umożliwia wskazanie programów uprawnionych do wykonywania kodu na stosie. Ustawia flagę w nagłówku pliku ELF, a podczas tworzenia nowego procesu system sprawdza obecność flagi i nadaje odpowiednie uprawnienia.

PaX

Projekt PaX może być stosowany w systemach z jądrem serii 2.2, 2.4 i 2.6. Powstał w październiku 2000 roku, ale został zawieszony w kwietniu 2005 r., po ujawnieniu bardzo poważnej luki umożliwiającej ominięcie kluczowych mechanizmów ochronnych. Autorzy stwierdzili, że odbiorcy stracili zaufanie do Paksa i zrezygnowali z dalszych prac nad tym projektem (choć błąd został usunięty). Opiekę nad kodem przejął autor grsecurity, Brad Spengler.

Mechanizmy ochronne

PaX oferuje mechanizmy podobne do Openwalla: uniemożliwienie wykonywania kodu w pamięci oraz randomizację przestrzeni adresowej procesu. Różnią się one jednak trochę od oferowanych w konkurencyjnym projekcie.

Ochrona pamięci w Paksie obejmuje nie tylko stos. Każdy segment (patrz Ramka *Ochrona pamięci w jądrze*) ma oddzielne uprawnienia umożliwiające albo zapis, albo wykonywanie. Administrator może też wybrać technikę do zastosowania: PAGEEXEC (ochronę przez stronicowanie) czy SEGMEXEC (ochrona przez segmentację).

Drugi z wspomnianych mechanizmów umożliwia losowe rozmieszczenie wszystkich elementów pamięci procesu. Uniemożliwia to praktycznie przeprowadzenie skutecznego ataku powrotu do biblioteki libc. Rysunek 1 porównuje pamięć przykładowego procesu w systemie standardowym i w systemie chronionym Paksem.

Instalacja

Aby zainstalować Paksa, należy:

- zaktualizować źródła kernela,
- nałożyć łatę,
- skonfigurować jądro,
- skompilować kernel,
- skompilować i zainstalować narzędzia *chpax* i *paxctl*

Programy *chpax* i *paxctl* umożliwiają konfigurację programów wykonywalnych tak, aby korzystały z poszczególnych zabezpieczeń lub je omijały. Jest to bardzo przydatne, jeśli korzystamy z niestandardowych aplikacji, które źle działają po wprowadzeniu ograniczeń dostępu do pamięci.

Narzędzia stosują dwie różne metody nadzoru nad programami wykonywalnymi. Program *paxctl* stosuje metodę bardziej inwazyjną – modyfikuje nagłówek pliku wykonywalnego ELF dodając odpowiednie pola. Wiąże się to z koniecznością zmodyfikowania także narzędzi z pakietu binutils. Drugi sposób (narzędzie *chpax*) mniej wpływa na bieżącą konfigurację systemu i przypomina rozwiązanie stosowane w Openwallu. Wykorzystuje istniejące, zarezerwowane pole nagłówka ELF.

Pierwszy z wymienionych sposobów lepiej integruje Paksa z systemem operacyjnym. Umożliwia też korzystanie z tak zwanego *miękkiego* trybu pracy, w którym wszystkie funkcje zwiększające bezpieczeństwo są wyłączone i aby z nich skorzystać muszą być uruchamiane niezależnie w każdym programie.

StackGuard

StackGuard jest rozszerzeniem kompilatora GCC. Został opracowany przez firmę Immunix w 1997 roku. Techniki wykorzystane w tym projekcie były tak interesujące, że podczas konferencji GCC 2003 Summit zaproponowano, aby zastosować je w podstawowej wersji kompilatora.

Niestety, nie zrealizowano tego w wersjach 3.x ani 4.0. Wynika to z faktu, że w GCC 4.1 znajdą się mechanizmy ochronne zaczerpnięte z SSP (patrz dalej). Prace nad projektem przerwano, więc należy traktować go bardziej jako ciekawostkę (nie

Ochrona pamięci w kompilatorze

Ochrona pamięci procesu w kompilatorze jest bardzo prosta. Polega na dodaniu do kompilatora mechanizmów, które podczas budowy programu dodają do niego kod gwarantujący ochronę stosu przed uszkodzeniem.

Ochrona stosu opiera się na zmodyfikowaniu ramki stosu (ang. *stack frame*) czyli struktury używanej do przechowywania danych tymczasowych, związanych z wywoływanymi funkcjami. Zmiana polega na wprowadzeniu zmiennej kontrolnej zwanej kanarkiem (ang. *canary*), umieszczonej tuż przed adresem powrotu z funkcji. Uszkodzenie stosu i nadpisanie adresu powrotu spowoduje też zmianę wartości kanarka, co z kolei powoduje awaryjne zakończenie aplikacji i uniemożliwia wywołanie szelkodu.

Listing 1. *Program uwidaczniający ochronę pamięci w kompilatorze (patrz Rysunki 2 i 3)*

```
char *func(char *msg, int a)
{
  int var1;
  char buff[10];
  int var2;
...
}
int main(int argc, char *argv[])
{
  char *p;
  p = func(argv[1], argc);
  exit(0);
}
```

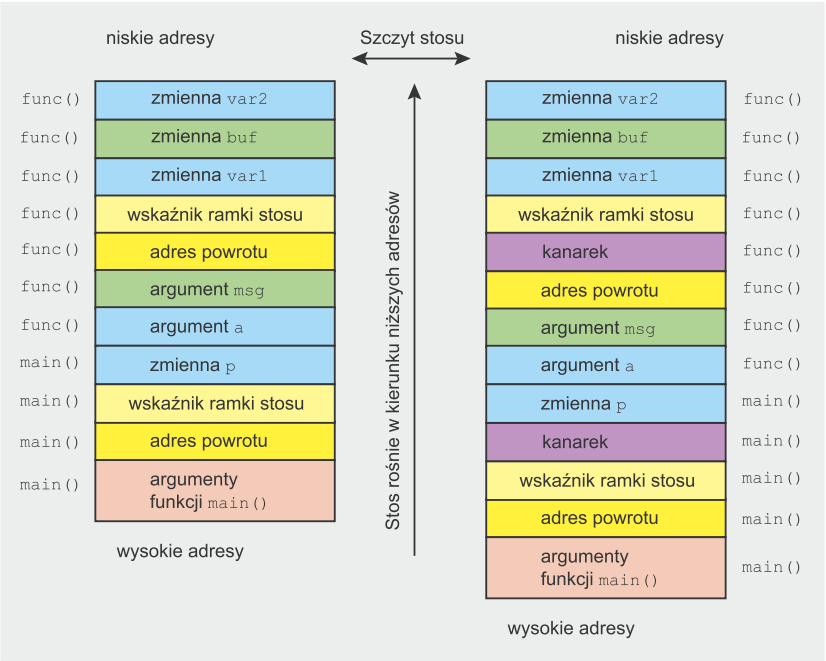
przedstawiamy więc procesu instalacji). Było to bowiem pierwsze narzędzie wprowadzające ochronę pamięci po stronie aplikacji, które stało się punktem wyjścia dla innych projektów tego typu.

Mechanizmy ochronne

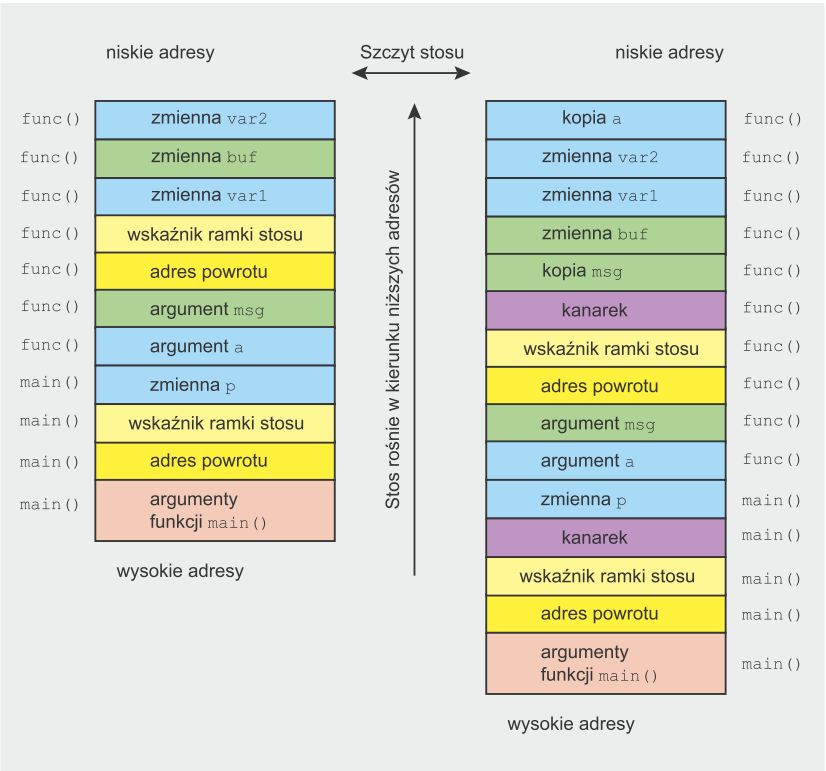
StackGuard wykorzystuje mechanizm kanarka (patrz Ramka *Ochrona pamięci w kompilatorze*) do ochrony pamięci procesu. Zastosowana technika nie gwarantuje jednak pełnej ochrony. Kanarek jest umieszczony w miejscu, które chroni tylko adres powrotu przechowywany w ramce stosu. Inne elementy ramki (lokalne zmienne funkcji oraz wskaźnik do poprzedniej ramki) nie podlegają ochronie.

Rysunek 2 prezentuje stos programu z Listingu 1 przed i po zastosowaniu StackGuarda. Jak widać w prawej części rysunku, przed każdym adresem powrotu umieszczono zmienną kontrolną. Atak polegający na przepełnieniu bufora `var2` nie uda się, albowiem spowoduje nadpisanie wszystkich pól w kierunku wyższych adresów pamięci. Zmiana adresu powrotu z funkcji `func()` zostanie wykryta, ale jeśli napastnik za cel przyjmie tylko zmienne `buf` i `var1` oraz wskaźnik ramki stosu, program będzie działał dalej.

W późniejszych wersjach StackGuarda kanarek został przesunięty przed wskaźnik ramki stosu. Umożliwiało to ochronę również tego pola,



Rysunek 2. *Ochrona stosu za pomocą StackGuarda*



Rysunek 3. *Ochrona stosu za pomocą SSP*

ale nadal nie gwarantowało integralności pozostałych lokalnych zmiennych funkcji.

SSP

Stack-Smashing Protector (SSP), wcześniej znany pod nazwą ProPolice, jest zbliżony koncepcyjnie do StackGuarda ale bardziej rozbudowany. Twórcą i opiekunem SSP jest Hiroaki Etoh. W chwili obecnej SSP jest dodatkiem do GCC, ale od wersji 4.1 kompilatora zostanie z nim zintegrowany.

Mechanizmy ochronne

Stack-Smashing Protector chroni stos procesu (adres powrotu, zmienne lokalne, argumenty funkcji i wskaźnik ramki) przy użyciu trzech technik. Każda z nich może być osobno włączana lub wyłączana podczas kompilacji programu:

- wartość kontrolna (kanarek),
- modyfikacja układu zmiennych,
- kopiowanie argumentów.

Pierwszy z nich działa tak samo, jak w StackGuardzie. Kanarek zostaje umieszczony przed wskaźnikiem ramki stosu. Drugi powoduje modyfikację kolejności umieszczania zmiennych lokalnych funkcji na stosie. Zmienne znakowe, podatne na przepełnienia, zostają umieszczone między zmiennymi liczbowymi a wartością kontrolną. Dzięki temu przepełnienie jednej z nich nie spowoduje uszkodzenia zmiennych lokalnych. Ostatni mechanizm chroni argumenty funkcji, których kopie są umieszczane na stosie obok zmiennych lokalnych i używane zamiast oryginalnych argumentów. Zastosowanie wszystkich trzech na raz znacznie utrudnia przepełnienie bufora w chronionym programie. Prawa część Rysunku 3 przedstawia stos programu po zastosowaniu SSP.

Instalacja

Instalacja SSP polega na dodaniu łaty do kompilatora GCC. Programista otrzymuje cztery dodatkowe opcje: `-fstack-protector`, `-fno-stack-protector`, `-fstack-protector-all` i `-fno-stack-protector-all`. Umożliwiają one włączenie lub wyłączenie ochrony stosu oraz włączenie lub wyłączenie ochrony wszystkich funkcji (nie tylko wykorzystujących tablice znakowe).

grsecurity

Grsecurity to projekt stworzony w lutym 2001 roku i rozwijany do dziś przez Brada Spenglera znanego pod pseudonimem Spender. Początkowo grsecurity było wersją Openwalla dla jąder serii 2.4, ale szybko zostało przekształcone w rozwiązanie niezależne. Dziś jest uznawane za jeden z najlepszych i najprostszych w instalacji mechanizmów do zabezpieczania systemów linuksowych.

Mechanizmy ochronne

Projekt łączy możliwości Openwalla i Paksa w zakresie ochrony pamięci z kontrolną dostępu opartą na rolach (patrz Ramka *Kontrola dostępu w systemach operacyjnych*). Wprowadza ograniczenia dla zwykłych użytkowników, randomizację mechanizmów obsługi procesów i sieci. Poprawia bezpieczeństwo funkcji `chroot` oraz chroni przed wyścigami w katalogach tymczasowych i rozszerza możliwości zapisu zdarzeń zachodzących w systemie.

Instalacja

Grsecurity jest łatą na jądro. Jego instalacja nie odbiega więc od sposobu opisanego we fragmentach o Openwall i PaX. Aby skorzystać z RBAC musimy jednak dodatkowo zainstalować narzędzie służące do konfiguracji systemu kontroli dostępu – *gradm*. Zarządzanie polityką kontroli dostępu jest bardzo proste, a *gradm* posiada możliwość uczenia się typowego zachowania użytkowników, co umożliwia automatyczne tworzenie minimalnych zasad dostępu.

LIDS

LIDS (*Linux Intrusion Detection System*) to kolejna łata na jądro, która rozszerza podstawowe mechanizmy kontroli dostępu. Autorami projektu są Xie Huagang i Philippe Bion-

Kontrola dostępu w systemach operacyjnych

W najpopularniejszych systemach operacyjnych stosowane są mechanizmy kontroli dostępu oparte na modelu nieobowiązkowym (ang. Discretionary Access Control, DAC). Każdy podmiot (użytkownik, proces) ma całkowity nadzór nad obiektami (pliki, katalogi, urządzenia) należącymi do niego. Może dowolnie zmieniać aktualne prawa dostępu do swoich zasobów, modyfikować je i usuwać.

Dodatkowo w systemie istnieje superużytkownik (*root*), który pełni rolę administratora. Ma nieograniczone prawa dostępu do wszystkich zasobów komputera i może zrobić praktycznie wszystko. Jeśli więc intruz uzyska dostęp do konta *roota*, uzyskuje nieograniczony dostęp do systemu.

W modelu obowiązkowej kontroli dostępu (ang. *Mandatory Access Control*, MAC) administrator nadal ma najwyższe uprawnienia w systemie operacyjnym. Jednak to on określa zasady dostępu wymuszane na wszystkich podmiotach. Model MAC wprowadza więc centralizację zarządzania kontrolą dostępu, w kontraście do zdecentralizowanego modelu DAC. Użytkownicy mają prawa ograniczone obowiązującą polityką i nie mają całkowitej kontroli nad swoimi plikami, katalogami itp.

Model MAC został opracowany na potrzeby systemów wymagających ścisłej ochrony poufności danych i jest wykorzystywany głównie w środowiskach wojskowych. Co ciekawe, polityka dostępu może również obejmować superużytkownika, który traci wtedy część swoich uprawnień. Jeśli więc intruz uzyska dostęp do jego konta, nie będzie mógł na przykład skopiować lub zmodyfikować części danych (chociażby strony WWW). Modele DAC i MAC przedstawiono po raz pierwszy w dokumencie TCSEC (*Trusted Computer Security Evaluation Criteria*), opublikowanym przez amerykański Departament Obrony w roku 1985.

Trzeci popularny model kontroli dostępu oparto na rolach (ang. *Role-Based Access Control*, RBAC). Został zaprezentowany w 1992 roku przez Davida Ferraiolo'a i Richarda Kuhna z Narodowego Instytutu Standardów i Technologii w USA. W tym modelu wszyscy użytkownicy otrzymują role ze ściśle określonymi uprawnieniami, które dotyczą też wszystkich uruchamianych przez danego użytkownika programów. Możliwości użytkowników mogą zostać ograniczone podobnie jak w modelu MAC, a zadania superużytkownika rozdzielone na kilku użytkowników.

W ten sposób model eliminuje niebezpieczeństwo związane z uzyskaniem przez napastnika dostępu do konta superużytkownika lub procesu działającego z jego prawami. Nawet jeśli atak będzie udany, intruz nie uzyska dostępu do całego systemu i przechowywanych w nim danych. Warto pamiętać, że RBAC jest szczególną formą MAC i oba modele umożliwiają uzyskanie podobnych efektów.

ACL kontra RBAC

Model obowiązkowej kontroli dostępu (MAC) można zrealizować w oparciu o dwa mechanizmy: listy kontroli dostępu (ang. *Access Control List*, ACL) i role (RBAC). Listy kontroli dostępu określają uprawnienia konkretnych użytkowników do konkretnych zasobów. Z punktu widzenia administratora konfiguracja ACL-i polega na przydzieleniu wszystkim użytkownikom odpowiednich praw dostępu.

Zarządzanie rolami polega na tym, że administrator definiuje grupy użytkowników według pełnionych przez nich obowiązków. Następnie określa uprawnienia dla każdej grupy i przydziela do niej wybranych użytkowników. Prosta implementacja ról nie różni się w praktyce od grup użytkowników w listach kontroli dostępu. Bardziej zaawansowane rozwiązania RBAC oferują jednak więcej możliwości.

di. Pierwsze wersje LIDS-a powstały jeszcze dla kerneli z serii 2.2, a obecnie stosować je można zarówno do serii 2.4 jak i 2.6.

Mechanizmy ochronne

LIDS wprowadza obowiązkową kontrolę dostępu (MAC z ACL). Uprawnienia definiujemy wykorzystując pakiet lidstools, w którego skład wchodzą narzędzia *lidsadm* i *lidsconf*. Konfiguracja polega na określeniu praw programów do zasobów (plików, katalogów, funkcji sieciowych itp.). Ponieważ ręczne ustawianie reguł jest trudne i czasochłonne, autorzy projektu zawarli w dokumentacji propozycje typowych konfiguracji dla niektórych aplikacji usługowych i narzędzi administracyjnych.

Instalacja

Instalacja LIDS-a nie odbiega od sposobu opisanego wcześniej. Należy dodatkowo skompilować i zainstalować narzędzia z pakietu lidstools oraz wstępnie skonfigurować reguły dostępu.

Przykładowe zastosowania

Serwer

Serwery najczęściej oferują określony zestaw usług, na przykład udostępniają pocztę elektroniczną, strony internetowe, przechowywanie plików lub bazy danych. W wielu przypadkach jeden system spełnia kilka ról na raz. Tylko w większych firmach możemy spotkać się z sytuacją, gdzie serwery są dedykowane wyłącznie jednej usłudze. Nierzadko administracją serwera zajmuje się kilka osób – kto inny administruje usługami pocztowymi, kto inny wprowadza zmiany na stronach WWW czy w bazach danych.

W takiej sytuacji dobrym rozwiązaniem jest wprowadzenie mechanizmu kontroli dostępu opartego na rolach i ścisły podział ról tak, by każdy administrator miał prawa nie większe, niż niezbędne do pracy. Administrator samego systemu operacyjnego nie powinien więc na przykład modyfikować treści stron internetowych.

Ze względu na fakt, że świadczenie usług wymaga udostępnienia ich wielu użytkownikom (często publicznie, w Internecie), bardzo istotne jest też wprowadzenie zabezpieczeń przed znanymi błędami: ochrona pamięci w jądrze i w kompilatorze. Poszczególne aplikacje usługowe powinny być zamknięte w oddzielnych środowiskach chroot. Najlepszymi rozwiązaniami do takich zastosowań wydają się więc projekty SSP i grsecurity lub RSBAC, gdyż oferują wszystkie wymagane w tej sytuacji środki ochrony.

Stacja robocza

Stacja robocza może być udostępniona więcej niż jednej osobie. Nie oferuje jednak usług użytkownikom zewnętrznym i podlega opiece jednego administratora. Dlatego też stację roboczą możemy zabezpieczyć stosując tylko ochronę pamięci w jądrze systemu (PaX lub grsecurity bez uruchamiania mechanizmu kontroli dostępu) i ograniczając prawa administratora za pomocą LIDS-a.

Ruter lub zapora

Wyspecjalizowane systemy takie jak rutery i zapory sieciowe oraz systemy IDS lub IPS są zazwyczaj administrowane przez jedną osobę. Nie oferują one żadnych usług, a często pracują tylko w drugiej warstwie TCP/IP i nie mają adresów IP, zaś dostęp do nich jest możliwy tylko z konsoli. W takiej sytuacji nie wymagają dodatkowych zabezpieczeń. Jeśli jednak systemowi przypisano adres IP i jest on zdalnie administrowany rozsądnym wyjściem będzie zastosowanie Paksa lub grsecurity. Można rozważyć też odebranie *rootowi* zbędnych uprawnień za pomocą LIDS-a.

SELinux

Projekt Security-Enhanced Linux (SELinux) stworzyła w 2000 roku Agencja Bezpieczeństwa Narodowego USA we współpracy z kilkoma firmami specjalizującymi się w ochronie informacji. Został oparty o techniki Flask, przeniesione do Linuksa z systemu Flux. SELinux był dodatkiem do jądra, ale w trakcie prac nad Linuksem 2.5 został z nim zintegrowany.

Mechanizmy ochronne

SELinux wprowadza obowiązkową kontrolę dostępu opartą na rolach. Jest bardziej rozbudowany niż grsecurity, ale też trudniejszy do skonfigurowania. Projekt jest doskonale udokumentowany, ale nie oferuje mechanizmów do automatycznego tworzenia polityki. Na szczęście istnieją osobne projekty udostępniające takie funkcje (na przykład polgen). Dystrybucje wykorzystujące SELinuksa mają pakiety z wstępnie zdefiniowanymi regułami dostępu dla najpopularniejszych aplikacji.

Instalacja

SELinux składa się z kodu w jądrze systemu, biblioteki libselinux oraz pakietów checkpolicy i policycoreutils. Kernele z serii 2.6 zawierają SELinuksa – wystarczy włączyć stosowne opcje w konfiguracji i skompilować jądro. Zastosowanie SELinuksa w jądrze serii 2.4 wymaga samodzielnego wprowadzenia łat. W chronionym systemie konieczne będzie korzystanie ze zmodyfikowanych wersji niektórych narzędzi, między innymi *ls*, *cp*, *ps* czy *login*.

RSBAC

RSBAC (*Rule Set Based Access Control*) to bardzo rozbudowany pro-

Tabela 2. *Porównanie projektów Openwall i PaX*

Funkcja	Openwall	PaX
Wersje jądra	2.0, 2.2, 2.4	2.2, 2.4, 2.6
Ochrona pamięci przed modyfikacją	Ochrona stosu procesu	Przydzielenie poszczególnym segmentom pamięci odpowiednich praw zapisu lub wykonywania
Randomizacja pamięci	Randomizacja adresów bibliotek dzielonych	Randomizacja wszystkich segmentów pamięci procesu
Ograniczenia dowiązań w katalogach tymczasowych	Tak	Nie
Ograniczenia potoków nazwanych w katalogach tymczasowych	Tak	Nie
Ograniczenia dostępu do katalogu /proc	Tak	Nie

Tabela 3. *Porównanie projektów grsecurity, LIDS, SELinux i RSBAC*

Funkcja	grsecurity	LIDS	SELinux	RSBAC
Kontrola dostępu	Role	ACL	Role	Role
Ochrona pamięci	Tak, zintegrowany z Paksem	Nie	Nie	Tak, zintegrowany z Paksem
Ochrona systemu	Ochrona katalogu */proc*, randomizacja obsługi sieci i procesów, wzmocniony `chroot`, ograniczenia dowiązań, ochrona przed sytuacjami wyścigu, rozszerzone logowanie zdarzeń	Ochrona systemu możliwa przez konfigurację list kontroli dostępu	Ochrona systemu możliwa przez konfigurację reguł kontroli dostępu	Ochrona katalogu */proc*, randomizacja obsługi sieci i procesów, wzmocniony `chroot`, ograniczenia dowiązań, ochrona przed sytuacjami wyścigu, rozszerzone logowanie zdarzeń, zarządzanie użytkownikami po stronie jądra
Automatyczne tworzenie reguł dostępu	Tak, wbudowane	Nie	Tak, programy zewnętrzne	Tak, wbudowane

Tabela 4. *Bezpieczne dystrybucje Linuksa*

	RedHat	Fedora	Hardened Gentoo	Adamantix	EnGarde	Adios	Owl
Openwall							Tak
PaX			Tak	Tak			
SSP			Tak	Tak			
grsecurity			Tak			Tak	
LIDS						Tak	
SELinux	Tak	Tak	Tak		Tak	Tak	
RSBAC			Tak	Tak		Tak	

jekt rozszerzający mechanizmy kontroli dostępu. Stabilna wersja istnieje od stycznia 2000 roku. Nie jest to jednolite rozwiązanie, lecz modułowy szkielet umożliwiający zastosowanie całej gamy zabezpieczeń.

Mechanizmy ochronne

RSBAC oferuje funkcje takie jak:

- Zarządzanie użytkownikami na poziomie jądra systemu. Informacje o kontach, grupach, hasłach i prawa dostępu są przechowywane w jądrze, a nie w plikach *passwd*, *shadow*, *group* i *gshadow*. Aby skorzystać z tej funkcji konieczna jest zmiana niektórych bibliotek systemowych i mechanizmu PAM.

O autorze

Michał Piotrowski, magister informatyki, ma wieloletnie doświadczenie w pracy na stanowisku administratora sieci i systemów. Przez ponad trzy lata pracował jako inspektor bezpieczeństwa w instytucji obsługującej nadrzędny urząd certyfikacji w polskiej infrastrukturze PKI. Obecnie pracuje jako specjalista do spraw bezpieczeństwa teleinformatycznego w jednej z największych instytucji finansowych w Polsce. W wolnych chwilach programuje i zajmuje się kryptografią.

- Ograniczenia funkcji `setuid()` – programy z ustawionym bitem SUID lub SGID muszą być autoryzowane.
- Obowiązkowa kontrola dostępu z możliwością oparcia o listy kontroli dostępu lub role.
- Ochrona pamięci procesów (integracja z Paksem), zwiększenie bezpieczeństwa mechanizmu `chroot`, limit wykorzystania zasobów komputera, specjalna ochrona plików, kontrola dostępu do interfejsów sieciowych, ochrona informacji w katalogu */proc* i inne.

Instalacja

Do wykorzystania możliwości oferowanych przez RSBAC konieczna będzie modyfikacja jądra systemu, a także zainstalowanie programów administracyjnych dostępnych na stronie projektu. Niezbędna będzie też zmiana niektórych narzędzi i bibliotek systemowych.

Trudny wybór

Wybranie najlepszego rozwiązania ochrony pamięci procesów w jądrze systemu lub po stronie kompilatora nie jest trudne. Jednak wybór mechanizmu rozszerzającego kontrolę dostępu do systemu można uznać za trudny.

SELinux jest zainstalowany w oficjalnych wersjach jąder serii 2.6, co gwarantuje dobrą integrację z systemem operacyjnym. Ponieważ rozwija go NSA, możemy się spodziewać, że prace nad nim nie zostaną zaniechane. To projekt dobrze udokumentowany, a także zainstalowany i wstępnie skonfigurowany w niektórych dystrybucjach (RedHat, Fedora). Jego wadą jest jednak konieczność konfiguracji dostępu od podstaw, zaś zastosowanie go w działającym, produkcyjnym systemie nie jest proste i wiąże się z dużą ilością pracy.

Grsecurity jest z kolei pakietem prostym w konfiguracji. Oferuje narzędzia automatycznie tworzące reguły dostępu i dodatkowe mechanizmy chroniące system (nie tylko MAC/RBAC). Niestety, system kontroli dostępu w tym projekcie nie jest zbyt dobrze udokumentowany i oferuje mniejsze możliwości niż SELinux i RSBAC. Grsecurity wydaje się jednak najlepszym wyjściem w sytuacji, gdy zależy nam na szybkiej i prostej instalacji oraz konfiguracji.

LIDS jest zbliżone funkcjonalnie do grsecurity, ale brakuje w nim ról i nie oferuje tylu mechanizmów chroniących system operacyjny. Jest jednak lepiej udokumentowany niż grsecurity, a na stronie domowej projektu możemy znaleźć sporo przykładowych ACL-i dla większości popularnych aplikacji. To dobry wybór, jeśli zależy nam na ograniczeniu praw superużytkownika, a jednocześnie szukamy narzędzia łatwego do zainstalowania.

Najbardziej kompleksowym rozwiązaniem jest RSBAC. Niestety, kompleksowość pociąga za sobą utrudnienia w instalacji i konfiguracji. Pomimo dobrej dokumentacji wprowadzenie go do systemu może nie być łatwe.

Dla każdego coś innego

Żadne z przedstawionych rozwiązań nie może być określone jako lepsze lub gorsze od innych. Wybór powinniśmy oprzeć na tym, czy projekt spełnia nasze potrzeby i jak dobrze rozumiemy stosowane w nim mechanizmy. Bezpieczeństwo naszego komputera będzie bowiem zależało nie tylko od efektywności narzędzi, ale w równie dużym stopniu od ich poprawnej i wydajnej konfiguracji.

W dokonaniu wyboru pomogą Tabele 2 i 3, zawierające podstawowe informacje o zaprezentowanych rozwiązaniach. Ramka *Przykładowe zastosowania* sugeruje rozwiązania w oparciu o typowe funkcje komputera.

Dla wielu użytkowników najprostszym wyjściem będzie wybór dystrybucji oferującej gotowe rozwiązania. W ten sposób unikniemy konieczności samodzielnej modyfikacji, konfiguracji i kompilacji jądra, wybierając przy instalacji systemu gotowy, wstępnie skonfigurowany pakiet. Tabela 4 informuje, jakie mechanizmy są dostępne w najpopularniejszych dystrybucjach Linuksa. ●

W Sieci

- *http://www.openwall.com/linux/* – Openwall,
- *http://pax.grsecurity.net/* – PaX,
- *http://www.trl.ibm.com/projects/security/ssp/* – SSP,
- *http://www.grsecurity.org/* – grsecurity,
- *http://www.lids.org/* – LIDS,
- *http://www.nsa.gov/selinux/* – SELinux,
- *http://www.mitre.org/tech/selinux/* – generator polityki dla SELinuksa,
- *http://www.rsbac.org/* – RSBAC,
- *http://www.gentoo.org/proj/en/hardened/* – Hardened Gentoo,
- *http://www.trusteddebian.org/* – Adamantix,
- *http://www.guardiandigital.com/products/software/community/* – EnGarde Linux,
- *http://dc.qut.edu.au/adios/* – Adios Linux,
- *http://www.openwall.com/Owl/* – Openwall GNU/*/Linux (Owl).